Proceedings of the Eighth ESSLLI Student Session

August 2003, Vienna, Austria

Balder ten Cate (Ed.)

Preface

Ever since its 1996 edition, the annual European Summer School on Logic Language and Information has been accompanied by a separate student session. This student session performs an important role, by providing a forum where students can present their own work and get feedback from fellow students and experienced researchers. It welcomes submissions related to the familiar ESSLLI subject areas, from students at any level (graduate as well as undergraduate).

The popularity of the student session has been growing throughout the years, and this is reflected by the high number of submissions. This year, we received a total number of 69 papers, of which 18 were accepted for oral presentation and 14 as a poster. Of these 32 papers, 29 are included in this volume.

I'm very grateful to the program committee for all their contributions to the organization. I'd like to thank the co-chairs in particular for their efforts in coordinating the reviewing process, and the area experts for their continuous presence and helpful advice. Also, my gratitude goes to all the reviewers, whose detailed comments have not only proved invaluable during the selection procedure, but also provide useful feedback to the authors. As in previous years, Kluwer Academic Publishers have generously offered to support the ESSLLI student session with special awards for the Best paper presentation and Best poster. I'm very grateful for their support. Finally, there are a number of people who I would like to thank in particular, since they were always ready to give help and advice: Raffaella Bernardi, Paul Dekker, Darrin Hindsill, Ivana Kruijff-Korbayová, Malvina Nissim, Marie Šafářová, Kristina Striegnitz and Willemijn Vermaat.

I'm very much looking forward to this year's ESSLLI summer school, which will take place August 18–29, 2003 in Vienna, Austria. Hopefully, its student session will provide the stimulating and fruitful atmosphere that it did in the past.

Balder ten Cate — Stanford, June 2003
Chair of the ESSLLI 2003 Student Session

Program Committee

Logic & Language
Christian Retoré, INRIA Futurs, France
Roberto Bonato, University of Verona, Italy; University of Bordeaux I, France
Paul Égré, University of Paris I, France

Logic & Computation
Sergio Tessaris, University of Bolzano, Italy
Jakob Kellner, Vienna University of Technology, Austria
Favio Miranda-Perea, LMU München, Germany

Language & Computation
Dan Flickinger, Stanford University, USA
Laura Alonso i Alemany, University of Barcelona, Spain
Maria Fuentes Fort, University of Girona, Spain
Gabriel Infante Lopez, University of Amsterdam, The Netherlands

Reviewers

Marco Aiello, Natasha Alechina, Alex Alsina, Thorsten Altenkirch, J. Gabriel Amores, Carlos Areces, Pablo Ariel Duboue, Victoria Arranz Corzana, Nicholas Asher, Sergio Balari, Denis Bechet, Claire Beyssade, Nick Bezhanishvili, Patrick Blackburn, Eerke Boiten, Patrick Brezillon, Hans J. Briegel, Daniel Buring, Alastair Butler, Miriam Butt, Xavier Carreras Pérez, Robyn Carston, Montserrat Civit, Francis Corblin, Alexander Dekhtyar, Paul Dekker, Stéphane Demri, Alexandre Dikovsky, Myroslava Dzikovska, Jason Eisner, Noemie Elhadad, Martin Everaert, Annie Foret, Bernd Gärtner, Kim Gerdes, Jonathan Ginzburg, Radu Gramatovici, Phillippe de Groote, Kenneth Harris, Peter Harvey, James Henderson, Jan Johannsen, Makoto Kanazawa, Tracy Holloway King, Felix Klaedtke, Alexander Koller, Angelika Kratzer, Geert-Jan Kruijff, Alexander Kurz, Martin Lange, Mirella Lapata, Michael L. Littman, Lluís Màrquez, Carlos Martín-Vide, Louise McNally, Stephan Merz, Paola Monachesi, Richard Moot, Karin Müller, David Nicolas, Richard Oehrle Antoni Oliver, Lluís Padró, Rohit Parikh, Gerald Penn, Ahti-Veikko Pietarinen, Anna Pilatová, David Poole, Detlef Prescher, Maurizio Proietti, Alessandro Provetti, Josep Quer, Owen Rambow, Francesco Ricca, German Rigau, Horacio Rodriguez, Soyoung Roger-Yun, Robert van Rooy, Joana Rosselló, Jon Rowe, Marie Šafářová, Gabriel Sandu, Katsumi Sasaki, Ulrike Sattler, Uli Sauerland, Philippe Schlenker, Renate Schmidt, Chung-chieh Shan, Khalil Sima'an, Christoph Simon, Viorica Sofronie-Stokkermans, Martin Stokhof, Kristina Striegnitz, Vitezslav Svejdar, Kriszta Szendrői, Annette ten Teije, Hans-Jörg Tiede, Leon van der Torre, Angela Weiss, Dag Westerståhl, Stefan Woltran, Paul Wong, Alden Wright, Hi-Yon Yoo, Patrick Zabalbeascoa, Henk Zeevat

Contents

The Proper Treatment of *Your Ass* in English

John Beavers and Andrew Koontz-Garboden
Stanford University
{jbeavers,andrewkg}@csli.stanford.edu

Abstract. Reflexives (e.g. *himself*) and pronouns (e.g. *him*) are usually distinguished by categorical conditions on their binding domains. However, there is a class of English expressions of the form Possessive Pronoun+*ass* (e.g. *your ass*) which we demonstrate to have pronominal properties but which appear to have unrestricted binding domains. We explore the problems such expressions pose for different types of binding theories and how an appeal to the unique meaning of *ass*-pronouns can resolve potential domain specificity conflicts.

1 Introduction

In many colloquial dialects of English, there exist pronominal expressions of the form Possessive Pronoun+*ass*, which we collectively refer to as *your ass*. Examples of this pronominal expression are given in (1).[1,2]

(1) (a) Rundgren's shit is only fuckin' good when his ass sings pop....You and I see shit the fuckin' same way. I can dig partying with your ass. [rec.music.progressive, 03-12-98] (=he sings pop, I can dig partying with you)

(b) The poster claimed that HE paid for gas. In reality, every time his ass drives his car where he doesn't need to go, WE pay for it... [alt.fan.rush-limbaugh, 07-02-1997] (=He drives his car)

[1]We would like to thank David Beaver, Emily Bender, Lev Blumenfeld, Cleo Condoravdi, Iván García, Andrea Kortenhoven, Jacques Lafleur, John Rickford, Peter Sells, John Singler, and Arnold Zwicky for their useful comments and suggestions, and we'd especially like to thank Paul Kiparsky and Tom Wasow for their support and lively discussions. We'd also like to thank James Isaacs for first pointing out to us that *your ass* has binding properties. The ideas and discussion in this paper are as always our own responsibility. We really, really mean that. Both authors contributed equally to this paper; the order of authors is entirely alphabetical by last name.

[2]Although we both have native intuitions about *your ass*, we use naturally occurring examples wherever possible. These were collected via searches with *Google* of newsgroups and web text, and are listed with their original reference.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

(c) their asses sure know how to fuckin' jam. kick ass guitar, whaling keys, and fuckin' screetching ass voices! dig it. fuckin' a. after the fuckin' jam was over my ass handed the old chick her ten fuckin' bucks....his ass claimed that his old lady gave him the fuckin' bucks to fuckin' buy an ice cream sandwich....i told his ass i needed the fuckin' money in order to fuckin' buy some beer. shit. my ass ain't ready to rip off texaco quite yet. [alt.music.yes, 04-01-2000] (=They know, I handed, I told him, I'm not ready)

The expression is not simply a combination of possessive pronoun + *ass*; it is semantically non-compositional, since *your ass* can do things that *an ass* cannot do. This can be seen most clearly in examples such as (1c). It's not literally their *asses* that know how to jam, but rather the people in question. Likewise, the speaker did not literally hand ten dollars to the woman with his buttocks; the speaker merely means that he handed her the money, presumably with his hands. The non-compositionality of *your ass*, in addition to other data we consider below, leads us to conclude that *your ass* is not a simple possessive pronoun+NP expression (PossNP). Rather, we argue, it is a pronoun, of a somewhat peculiar type, since it appears in both reflexive and pronominal binding domains, contrary to the predictions of many binding theories, as shown in (2).

(2) (a) But most people do believe OJ_i bought his ass_i/himself_i/*him_i out of jailtime. [soc.culture.china, 01-28-2002]

(b) First Newton, Alexander, and Moore make an ass out of Pangborn_i. The more he_i whined about it, the more they nailed his ass_i/him_i/*himself_i. [soc.men, 04-23-99]

(c) his ass/he/*himself claimed that his old lady gave him the fuckin' bucks to fuckin' buy an ice cream sandwich.... [alt.music.yes, 04-01-2000]

In what follows, we first argue that *your ass* is a pronoun rather than simply a PossNP. We then consider its binding properties from the perspective of Kiparsky's (2002) binding theory, which actually predicts some of its peculiar properties, though also potentially causing problems for this theory as well. We examine further data showing both that *your ass* is not easily accommodated in alternative theories (particularly Reinhart and Reuland (1993)), and that it can be accommodated within Kiparsky's theory, once semantic factors are taken into account.

2 The Pronominality of *Your Ass*

In this section we present evidence that *your ass* is indeed pronominal rather than a PossNP. Unfortunately, there appears to be no standard non-theory-

internal definition for a pronoun; rather pronominals tend to be identified by clusters of distributional properties and then defined in theoretical terms (e.g. as referential entities that obey certain principles). We shall not attempt to offer any autonomous definition of a pronominal, but instead show that *your ass* patterns more like complex reflexives than PossNPs by a variety of syntactic and semantic criteria. We start by differentiating two readings: the non-literal (referring to a person) from the literal (referring to someone's backside). Focusing on the non-literal *your ass*, the main distinction between it and PossNPs is compositionality: *your ass* shares reference with its possessive determiner, unlike all other PossNPs, e.g. *your ear*, *your car*, *your preferred syntactic theory*, and *your mother* refer to ears, cars, theories, and mothers rather than the hearer. This can be seen in examples like (3) and (4):

(3) (a) John$_i$ bought [his$_i$ ear/mother/neck]$_{j\neq i}$ a car.
(b) John$_i$ bought [himself/his$_i$ ass]$_i$ a car.

In (3a) the recipient is the ear, mother, or neck, and never John, while it is only John in (3b) on the intended non-literal reading. If *your ass* were a PossNP this would be surprising since in no other PossNP does a verb predicate of the possessor rather than the possessed (i.e. assign a θ-role to the pronoun in [Spec,DP] rather than the DP itself).

Furthermore, *your ass* has unique properties when serving as the antecedent of other pronouns:

(4) (a) John$_i$, his ass$_i$ upset himself$_i$/*him$_i$.
(b) John$_i$, his$_i$ grade/mother/broken back upset him$_i$/*himself$_i$.

If *his ass* in (4a) were a PossNP then the purported possessive pronoun would be licensing the reflexive, something possessors in other PossNPs cannot do, as shown in (4b). This referential behavior is identical to reflexives, which are also non-compositional.

Another argument for the pronominality of *your ass* is that in general, PossNPs allow for a wide range of modification by adjectives, PPs, and relative clauses as in (5a), whereas *your ass*, like pronominals, shows a more limited modifiability, generally allowing pre-nominals (e.g. (5b) like *self* reflexives) but not relative clauses or PPs (e.g. (5c,d) also like *self* reflexives).

(5) (a) Your unkempt jacket on the coat rack that you got from your mother needs mending.
(b) Get your bad/ugly/own self/ass outta here.
(c) The doctor saw his finger with the ring/that he broke yesterday.
(d) *The doctor saw himself/his ass from Houston/that stopped by three times last week.

Finally, PossNPs can license $\bar{N}$-ellipsis whereas reflexives and *your ass* cannot (coindexation is intended to indicate "sense" coreference and not strict coreference):

(6) (a) Mary had her [car/house/office painted]$_i$, and Jane had hers e$_i$ entirely remodeled.

(b) *Mary had herself$_i$/her ass$_i$ committed, and Jane had hers e$_i$ released.

Given the evidence presented here, it is clear that *your ass* is a pronominal, not a regular PossNP. The superficial similarity between *your ass* and a PossNP is not surprising, however, since complex pronominals in a variety of languages (including English *himself*) are often grammaticalized PossNPs formed from a possessive pronoun+some body part (Faltz, 1985, Schladt, 2000). Typically these grammaticalize to reflexives as the PossNP type construction serves as a way of placing the pronominal (as a possessive) in a non-argument position and thus exempting it from binding constraints. In this sense it might also be best to view *your ass* as being on a cline of pronominal grammaticalization, patterning closer to pronominals than PossNPs.[3]

3 Pronoun Typology and the Elsewhere Principle

Kiparsky (2002, p.200ff) proposes a hierarchy of binding domains based on four increasingly specific criteria. The broadest criterion is referential dependence, wherein referentially dependent pronouns require the presence of a discourse antecedent, and referentially independent pronouns do not (cf. (7a)). Referentially dependent pronominals, in turn, are either non-reflexive, allowing for a syntactic or discourse-based antecedent, or reflexive, requiring a syntactic antecedent (cf. (7b)). Reflexive pronouns may be either finite-bound, requiring an antecedent in the same finite clause, or not finite-bound, allowing for the possibility of being bound by an antecedent outside of the finite clause (cf. (7c)). Finally, finite-bound pronominals may be either locally-bound, requiring an antecedent in the "first accessible subject domain", or not (cf. (7d,e)).

(7) (a) We need to talk about him$_i$/*himself$_i$, him$_j$/*himself$_j$, and her$_k$/*herself$_k$. [pointing] (Referential independence)

(b) John$_i$ is here. I saw him$_i$/*himself$_i$. (Referentially dependent, non-reflexive)

[3]Interestingly, according to Holm (2000, 226) the word for "buttocks" in several creole languages also shows at least reflexive uses, if not pronominal ones as well (Holm notes only a reflexive use).

(c) John$_i$ thought that I would criticize him$_i$/*himself$_i$. (Reflexive, non-finite-bound)

(d) John$_i$ asked me to criticize him$_i$/*himself$_i$. (Finite-bound, non-local)

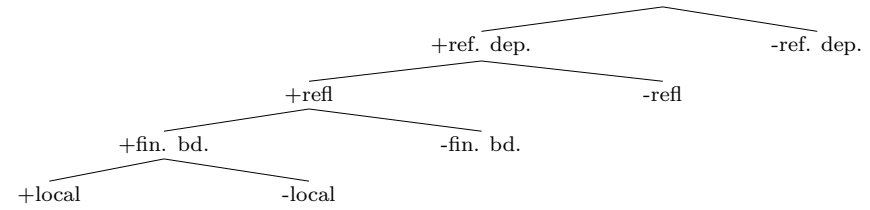
(e) John$_i$ criticized himself$_i$/*him$_i$. (Local) (Kiparsky, 2002, p.201)

Each domain is cross-classified for the property of 'obviation':

(8) Obviation
Coarguments have disjoint reference (Kiparsky, 2002, p.2)).

An obviative pronoun is one that must obey obviation, and a proximate pronoun is one that does not necessarily obey obviation. In English, the distinction between obviative and proximate pronominals is simply the distinction between pronouns and anaphors. However, as (Kiparsky, 2002, p.5ff) shows, there are languages with obviative/proximate pronominals (Swedish) and languages with obviative/proximate reflexives (Algonquian), making this distinction cross-linguistically valid. The increasing specificity of the various binding domains along with their interaction with obviation is illustrated in (9), with example pronominals satisfying most of the types.

(9) Kiparsky's pronominal typology



[O−]	*himself*	Russian *sebja*	Icelandid *sig*	Turkish *kendisi*	—
[O+]	—	Swedish *sig*	Marathi *aapaṇ*	Greek *o idhios*	*he*

The blocking relationship between the pronominals comes from their increasing specificity, wherein pronouns with more specific binding domains block the use of pronouns with less specific binding domains. For instance, *himself* has a more specific binding domain than *he* in English, as can be seen in (9). Likewise, if English had something like the Icelandic *sig*, *himself* would block use of it in finite bound local domains.

Notably absent in (9) are two types of pronominals, obviative locally bound pronominals (presumably a theoretical impossibility since obviation is by definition non-local) and a proximate referentially independent pronoun, i.e. a pronoun that, effectively, is compatible with any domain and is thus a sort of "universal pronoun" (Kiparsky, 2002, p.27). We believe that *your ass* fills the latter gap in the typology.

4 *Your Ass* and the Pronominal Typology

Of interest in the present context is the fact that *your ass* can apparently be used in all of the binding domains in (7), as shown in (10)-(14).

(10) Referential independence

(a) On the agenda for today is to talk about his ass_i, his ass_j, and her ass_k. [pointing]

(b) I mean her ass, over there.

(11) Referentially dependent, non-reflexive

(a) Please explain to me is Bobby V_i a good coach or not....His_i team has less infield errors than anyone else, give his ass_i some credit. [alt.sports.baseball.ny-mets, 08-25-99]

(b) I think if Mike and Buzz had their way, he'd_i be outta there. Mike hates his ass_i and Don knows it. The only think (sic) worse than listening to Dennis_i is listening to Bart and Freida. [alt.fan.don-n-mike, 06-16-2000]

(12) Reflexive, non-finite-bound

(a) I had one guy tell me the change was for gas, the box, and I bought his ass_i a coke while he waited in a long line.... [alt.toys.gi-joe, 05-11-02]

(b) First Newton, Alexander, and Moore make an ass out of Pangborn_i. The more he_i whined about it, the more they nailed his ass_i. [soc.men, 04-23-99]

(13) Finite-bound, non-local

(a) John_i asked me not to criticize his ass_i.

(b) Mary_i told me to buy her ass_i a diamond ring.

(14) Local

(a) You_i bought your ass_i a lap-dance? [alt.angst, 08-31-00]

(b) Don't give up! I am 30 and was ag. for a little over a year until I_i got my ass_i some help... [alt.support.agoraphobia, 06-15-99]

The fact that *your ass* can occur in contexts such as (14) shows that *your ass* is a proximate, and the fact that it can occur in contexts such as (10), with no linguistic antecedent, shows that it is referentially independent. Prima facie, these data appear to show that *your ass* is in fact a universal

pronoun, i.e. a referentially independent proximate. This type of pronominal, while theoretically possible, is otherwise unattested in Kiparsky's extensive survey, and proof of the existence of such a pronominal further validates this typology by filling in the final logically possible gap in the paradigm. However, *your ass* poses a serious problem for the theory in general since it seems to contradict the blocking principle, which incorrectly predicts that reflexives should block *your ass* in local binding domains (cf. (14)).

5 Semantics of *Your Ass* and Blocking

We argue, however, that examination of the meaning of *your ass* can account for its anomalous behavior. Specifically, it seems that *your ass* has two elements of meaning not found in other pronominals: (a) it can be used only in the proper social setting (acting as a marker of that setting), and (b) it carries additional semantics about relationships between participants and referents in the discourse.

5.1 On the social meaning of *your ass*

Although it may seem obvious, *your ass* can be used only in certain social settings; there are many social settings in which it is simply not appropriate, e.g. in a nice restaurant, at church, in a reputable conference proceedings, etc. This same point is made by Spears (1998, p.236) who argues that the meaning of *your ass* is "social and abstract" and that it "marks a discourse as being in U[ncensored] M[ode]", i.e. in a social context where expressions that would be inappropriate elsewhere (i.e. censored contexts) are neutral with respect to appropriateness (Spears, 1998, p.232).[4] Thus, *your ass* marks a discourse as being in a particular mode/social setting in a way that standard pronominals do not. This fact alone shows that there are more differences between *your ass* and other pronominals than simply domain specificity.

5.2 On The Non-Social Meaning of *Your Ass*

Although the facts are subtle and we have not conducted a full exploration, even when one is in the proper social context, *your ass* and other English pronominals are not simply interchangeable. All of the examples we have seen so far would be qualitatively different if a standard English pronominal were used in place of *your ass*. The first way in which it is different is that *your ass* can mark negative connotations of the *ass*-marked referent:

(15) (a) I am gonna knock your ass down the hill. [rec.climbing, 08-18-01]

[4]Spears' discussion was specifically concerned with use of the *ass* morpheme in African-American Vernacular English (AAVE), whereas our discussion concerns uses of just *your ass* by a wider set of speakers, including our own non-AAVE judgments.

(b) I am gonna knock you down the hill.

In (15a), use of *your ass* conveys the message that the patient is somehow subordinate to the agent, i.e. the speaker makes it explicit that she believes the patient to be of no match for her. When a regular pronoun is substituted, as in (15b), the same effect is not achieved. This negative use of *your ass* seems to be the most common, characterizing most of the examples we have given above. For such uses, the evaluation scale tends to be a relative scale, wherein the *ass*-marked referent is typically conveyed to be lower on some power-based hierarchy relative to another participant in the dialogue.

In addition to negative connotations, however, *your ass* can also mark positive connotations for the *ass*-marked referent:

(16) (a) brittney, you stupid....do you realy (sic) think my man mase is realy (sic) gonna reply to your stupid shit...mase is a horn dog, his ass fucks all his girls, ... [rec.music.hip-hop, 01-09-98]

(b) ...mase is a horn dog, he fucks all his girls...

In (16a), the writer uses *your ass* to convey a more positive message about his regard for the *ass*-marked participant, conveying envy or respect for Mase by referring to him with *your ass*. The parallel example in (16b) with a standard English pronoun is neutral regarding the writer's attitude towards Mase. These positive uses are rarer in the data we examined, and tend to involve generic scales: the *ass*-marked referent is typically conveyed in a generic positive light rather than relative to another discourse participant, unlike negative uses of *your ass*. A better understanding of the semantics of *your ass* will require much more examination of naturally occurring data. However it should be clear that *your ass* carries meaning that other English pronominals don't, at a social and linguistic level.[5]

5.3 The Interaction of Semantics and Blocking

It is not our goal to present a formal account of how semantics is incorporated into blocking, but the basic idea is that strict specificity is not enough, since once semantics are taken into account, no strict specificity relationship holds between the reflexive and *your ass*: one has a more specific domain and the other a more specific meaning. Instead the interaction must involve preserving semantics even when domain specificity is violated. This could be implemented in the OT account of Kiparsky by assuming that *your ass* overtly encodes additional meaning over other pronominals, and that there is

[5]Incidentally, these semantic facts also show that *your ass* cannot be the only pronominal form in the *ass*-register. Its unique meaning exists in contrast to alternative pronominals, and if these pronominals were not part of the register, then *your ass* could not contrast with them. Furthermore, *ass*-marked and non-*ass*-marked pronominals occur side-by-side in many of our naturally occurring examples (cf. (16a)).

a very highly ranked constraint, a sort of "semantic faithfulness" constraint, requiring this meaning to be overtly realized in the output if present in the input. With such a constraint, *himself* always loses on semantic grounds regardless of domain specificity since it never carries the more specific semantics. Presumably other approaches might accomplish the same thing, but the main point is that blocking must be sensitive to more than just binding domains and in a more complicated way than just specificity.

6 Non-Blocking Theories of Binding

Although our main focus is on blocking theories of binding, various alternatives may also have difficulties dealing with *your ass*. In particular, a class of theories such as those proposed in Chomsky (1981), Reinhart and Reuland (1993), Pollard and Sag (1994), which we will refer to as "partition" theories, have two parts. First, they assume a discrete partitioning of the space of pronominal types, and second, instead of blocking, they propose a (small) set of conditions that govern the distribution of different equivalence classes. *Your ass* potentially causes problems for partition theories since discrete partitioning is largely incompatible with its wide distribution. Taking as an example the extensive theory in Reinhart and Reuland (1993), they propose dividing the space of pronominals into four categories by two boolean features, R(eferential independence), roughly corresponding to whether something needs a linguistic antecedent, and Refl(exivizing function), corresponding to whether a pronominal requires a predicate to have coreference between coarguments. This yields the following typology:

(17)

	R+	R-
Refl+	∅	SELF
Refl-	*him*	SE

The three attested types are labeled SELF, for referentially dependent reflexives like English *himself* and Dutch *zichzelf*, SE ("simplex expression") for referentially dependent non-reflexives such as the Spanish *se* and Dutch *zich*, and referentially independent non-reflexives such as English *him*. The fourth type is a referentially independent reflexive, not discussed by Reinhart and Reuland (it's not clear whether they consider it a logical possibility or not). They propose three conditions governing the distribution of these pronoun types, the exact nature of which is largely irrelevant for present purposes. The relevant factor is the partitioning, since the data in (10)-(14) clearly show that *your ass* occurs in any argument position, and therefore does not fall into any of the four pronominal classes.

While still maintaining the partition approach, there are two obvious solutions to this problem. The first would be to maintain the strict partition but assume additional, presumably semantic, conditions governing the

distribution of *your ass*. This would still require, however, stipulatively assigning *your ass* to one class or another of pronouns. Alternatively, one could assume a polysemy of *your ass*, with one lexical entry for each possible pronoun type, an undesirable and unwarranted stipulation. The second solution is to assume that *your ass* is an instance of a generic pronominal underspecified for R and Refl. Allowing underspecification, though, expands the space of possible pronouns from four to nine:

(18)

	R+	R-	R+/-
Refl+	∅	SELF	∅
Refl-	*him*	SE	??
Refl+/-	∅	??	*your ass*

In this case, there are now several gaps in the paradigm. Depending on whether [R+,Refl+] pronouns are logically possible, there are now at least two and maybe five currently unattested pronominal types. The question of their existence is a largely empirical question, but relaxing the restrictions on partitioning makes strong predictions that may not be empirically validated.

Before concluding it is worth mentioning that there is a wealth of literature on various approaches to binding theory that we have not dealt with here. Much semantic work has dealt with anaphora (e.g. Reinhart (1983), *inter alia*, or Partee and Bach (1981) for a Montague-grammar style approach), and other strains have attempted to derive some (or all) binding facts from pragmatics (e.g. Levinson (1987), Huang (2000)). Semantic/Pragmatic approaches are often based on or quite compatible with other approaches to binding, and indeed our own account of *your ass* essentially assumes that semantics and pragmatics (contextual, social, and non-social meaning) are indeed relevant factors within a more syntactic account, although we have not addressed the details of accommodating *your ass* within these specific theories, since blocking has been our main concern.

7 Concluding remarks

Focusing primarily on the pronominal typology in Kiparsky (2002), we have shown first and foremost that *your ass* has pronominal uses, bringing new data to bear on binding theory. Secondly, *your ass* appears to fill in a hitherto unattested pronominal type, namely the most general category of "universal pronoun". However, it poses a problem for any theory of blocking of pronominals based on binding domain specificity since *your ass* can appear in any binding domain, immune to blocking. We argue that *your ass* contributes additional meaning that no other pronominal contributes and, in some fashion or another, the necessity of expressing this meaning must be taken into account by the blocking theory. That is, the facts of *your ass* suggest that blocking based purely on specificity of binding domains is too limited, and factors such as meaning must be taken into account as well.

References

Chomsky, N. (1981). *Lectures on Government and Binding*. Dordrecht: Foris.

Faltz, L. M. (1985). *Reflexivization: A Study in Universal Syntax*. New York: Garland Publishing, Inc.

Holm, J. (2000). *An Introduction to Pidgins and Creoles*. Cambridge, UK: Cambridge University Press.

Huang, Y. (2000). *Anaphora: a cross-linguistic approach*. New York: Oxford University Press.

Kiparsky, P. (2002). Disjoint reference and the typology of pronouns. In I. Kaufmann and B. Stiebels (Eds.), *More Than Words*. Berlin: Acadamie Verlag.

Levinson, S. C. (1987). Pragmatics and the grammar of anaphora: a partial pragmatic reduction of binding and control phenomena. *Journal of Linguistics* (23), 379–434.

Partee, B. and E. Bach (1981). Quantification, pronouns, and VP anaphora. In J. Groenendijk, T. Janssen, and M. Stokhof (Eds.), *Formal Methods in the Study of Language*, pp. 445–481. Amsterdam: Mathematisch Centrum.

Pollard, C. and I. A. Sag (1994). *Head-Driven Phrase Structure Grammar*. Chicago, IL: The University of Chicago.

Reinhart, T. (1983). *Anaphora and Semantic Interpretation*. London: Croom Helm.

Reinhart, T. and E. Reuland (1993). Reflexivity. *Linguistic Inquiry 28*.

Schladt, M. (2000). The typology and grammaticalization of reflexives. In Z. Frajzyngier and T. S. Curl (Eds.), *Reflexives: Forms and Functions*. Amsterdam: John Benjamins.

Spears, A. (1998). African-American language use: ideology and so-called obscenity. In S. Mufwene, J. Rickford, G. Bailey, and J. Baugh (Eds.), *African-American English: Structure, History, and Use*. New York: Routledge.

Algorithms for Combinatorial Optimization and Games Adapted from Linear Programming

HENRIK BJÖRKLUND
Uppsala University
henrikbj@it.uu.se

SVEN SANDBERG
Uppsala University
svens@it.uu.se

ABSTRACT. The problem of maximizing functions $f : \{0,1\}^d \to \mathbb{R}$ from the boolean hypercube to real numbers arises naturally in a wide range of applications. This paper studies an even more general setting, in which the function to maximize is defined on what we call a *hyperstructure*. A hyperstructure is a Cartesian product of finite sets with possibly more than two elements. We also relax the codomain to any partially ordered set. Well-behaved such functions arise in game theoretic contexts, in particular from parity games (equivalent to the modal μ-calculus model checking) and simple stochastic games (Björklund, Sandberg, and Vorobyov 2003b). We show how several subexponential algorithms for linear programming (Kalai 1992; Matoušek, Sharir, and Welzl 1992) can be adapted to hyperstructures and give a reduction to the abstract optimization problems introduced in (Gärtner 1995).

1 Introduction

We investigate the problem of optimizing certain well-behaved functions defined on combinatorial hyperstructures, generalizations of the boolean hypercube $\{0,1\}^d$. Such functions arise naturally in game-theoretic contexts. Their local maxima can be found with the following general local improvement scheme. Start at a vertex v of the hyperstructure. If v has a value greater or equal to all its neighbors, then return v; otherwise restart from one of the better neighbors. Generally, this algorithm can take exponential time (Tovey 1997). Since the result is a local maximum, it is natural to require every local maximum to be global. This condition, called *local-global* or *LG* (Tovey 1986), holds in the game-theoretic setting. Additionally, games satisfy the important property that local maxima are global *on every substructure*. We call such functions *recursively local-global* or *RLG* for short. As we will show, the extra structure of RLG-functions allows us to optimize them in subexponential time, roughly $2^{O(\sqrt{d})}$, where d is the dimension.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

The main motivation for our study is the application to parity and simple stochastic games. Parity games are two-player games played by moving a pebble along edges of a directed graph. The associated computational problem consists in finding the optimal strategies of both players. It is interesting from a complexity viewpoint as one of the few natural problems in NP $\cap$ CONP not known to be polynomial, especially after PRIMES was shown to be polynomial (Agrawal, Kayal, and Saxena 2002). It is practically significant, being polynomial time equivalent to model-checking for the modal μ-calculus (Emerson, Jutla, and Sistla 1993), one of the most expressive temporal logics of programs, subsuming logics like LTL, CTL, CTL*, etc. It is also polynomial time equivalent to Rabin chain tree automata nonemptiness (Emerson, Jutla, and Sistla 1993).

In a *binary* parity game, where the players have at most two choices in each move, a strategy of one player can be viewed as a corner of a hypercube. A strategy improvement algorithm finds the winner of a parity game by assigning values to each strategy of one player and then moving to better and better strategies until the best one is reached. The strategy evaluation has to be carefully chosen so that the resulting function is LG with any maximum corresponding to an optimal strategy of the game. This guarantees that the algorithm is correct. This was first done in (Vöge and Jurdziński 2000) and we later showed how to compress the value space to allow for a better upper bound on the number of iterations (Björklund, Sandberg, and Vorobyov 2003a). These functions are automatically also RLG, since any subgame is also a game. The same value functions also work for non-binary games, but the domain now becomes a more general *hyperstructure*.

A hyperstructure is a Cartesian product of finite sets with possibly more than two elements. Although non-binary parity games can be transformed to binary games, keeping them non-binary allows for better upper bounds because the transformation increases the number of vertices. Simple stochastic games is a related class of games where iterative improvement algorithms are easier to understand and have been known for a longer time; see, e.g., (Condon 1993). The values of strategies are real numbers and the resulting functions are also RLG.

Linear programming (LP) is the problem of finding the maximum of a linear functional on a convex polyhedron in $\mathbb{R}^d$. Although the problem statement looks very different from that of optimizing an RLG-function on a hyperstructure, we show how to adapt several LP algorithms to RLG optimization. These algorithms are *subexponential* in the dimension of the polyhedron. Translated to our terms, they are subexponential in the dimension of the hyperstructure (defined as the number of sets in the Cartesian product), and in terms of parity games they are subexponential in the number of vertices that the first player owns. The first algorithm shown to be subexponential was (Kalai 1992), adapted by us to parity games in (Björklund, Sandberg, and Vorobyov 2003a). The algorithm in (Sharir and Welzl 1992) was invented before Kalai's, but they proved it subexponential later (Matoušek, Sharir, and Welzl 1992); see also (Matoušek, Sharir, and Welzl 1996). It is easier to understand than Kalai's and the upper bound is similar. For a survey of these and other related algorithms, see (Goldwasser 1995). Ludwig was

first to adapt these algorithms to games; he shows how Matoušek–Sharir–Welzl's algorithm works on binary simple stochastic games (Ludwig 1995).

The success of transforming linear programming algorithms to hyperstructures and games can be understood through problem reductions. First, the strategy evaluation functions for parity games and simple stochastic games are RLG, as already discussed. Second, RLG-functions can be directly reduced to the *LP-type problems* introduced in (Sharir and Welzl 1992). LP-type problems is an abstract framework capturing the combinatorial structure of linear programming and numerous other problems in computational geometry (Matoušek, Sharir, and Welzl 1996). The algorithms discussed here and several others work on such problems. The reduction from RLG-functions to LP-type problems was shown for the more restricted class of CLG-functions in (Björklund, Sandberg, and Vorobyov 2003b), but works without modification on RLG-functions. The games actually have more structure, captured by CLG-functions, which allows for optimization algorithms that are intuitively 'more aggressive' (can take longer steps in the hyperstructure) but the best known upper bounds remain exponential.

The paper is organized as follows. In Section 2 we define the functions we optimize and the structures they are defined on. As an introduction to the concepts underlying later sections, we present a Ludwig-style optimization algorithm for hypercubes in Section 3. Sections 4 and 5 adapt the linear programming algorithms from (Matoušek, Sharir, and Welzl 1996) and (Kalai 1992), respectively. Finally, we show in Section 6 how our problem can be reduced to what (Gärtner 1995) calls an abstract optimization problem (AOP).

2 Hyperstructures

The set of all positional strategies in a parity game is isomorphic to a product of finite sets, each representing the choices in a vertex. Functions from strategies to values in such games motivate our study of functions on the *hyperstructures* defined here. They are generalizations of the boolean hypercube $\{0,1\}^d$. Optimization of well-behaved functions on hypercubes has been extensively studied in, e.g., (Hammer, Simeone, Liebling, and De Werra 1988; Wiedemann 1985; Williamson Hoke 1988; Tovey 1997; Björklund, Sandberg, and Vorobyov 2002).

Definition 2.1 (Hyperstructure) *For each $j \in \{1,\ldots,d\}$ let $\mathcal{P}_j = \{e_{j,1},\ldots, e_{j,\delta_j}\}$ be a finite nonempty set. Call $\mathcal{P} = \prod_{j=1}^{d} \mathcal{P}_j$ a d-dimensional* hyperstructure, *or* structure *for short.* □

A *substructure* of $\mathcal{P}$ is a product $\mathcal{P}' = \prod_{j=1}^{d} \mathcal{P}'_j$, where $\emptyset \neq \mathcal{P}'_j \subseteq \mathcal{P}_j$ for all j. A *facet* of $\mathcal{P}$ is a substructure obtained by fixing the choice in exactly one coordinate. Thus $\mathcal{P}'$ is a facet of $\mathcal{P}$ if there is a $j \in \{1,\ldots,d\}$ such that $|\mathcal{P}'_j| = 1$ and $\mathcal{P}'_k = \mathcal{P}_k$ for all $k \neq j$. We will sometimes identify a facet with its defining element $e_{j,i} \in \mathcal{P}_j$, assuming all sets P_k to be disjoint. If $|\mathcal{P}'_j| \leq 2$ for all j, then $\mathcal{P}'$ is called a *subcube* since it is isomorphic to a boolean hypercube. We will

consistently use the letter d for the dimension and $n := \sum_{j=1}^{d} |\mathcal{P}_j|$ for the number of facets. If $|P_j| = 1$ for some j, this coordinate can be disregarded since it is constant, and we will consider such structures as having smaller dimension.

An element of $\mathcal{P}$ is called a *vertex*. Two vertices $x, y \in \mathcal{P}$ are *neighbors* if they differ in only one coordinate. The neighbor relation induces a graph with elements of $\mathcal{P}$ as nodes, and allows us to talk about paths and distances in $\mathcal{P}$.

For a set $E \subseteq \bigcup_{j=1}^{d} \mathcal{P}_j$, define $struct(E)$ to be the substructure $\mathcal{P}' = \prod_{j=1}^{d}(\mathcal{P}_j \cap E)$. Thus, if E does not have elements from each $\mathcal{P}_j$, then $struct(E) = \emptyset$. For a set $\mathcal{F}$ of facets, we use $struct(\mathcal{F})$ for $struct(E)$ where E is the set of defining elements of the facets in $\mathcal{F}$.

Throughout this paper, let $\mathcal{D}$ be some partially ordered set. We are interested in finding optima of functions that map $\mathcal{P}$ to $\mathcal{D}$, such that any two neighbors have *comparable* values. A *local maximum* of a function $f : \mathcal{P} \to \mathcal{D}$ is a vertex v such that $f(v) \geq f(u)$ for all neighbors u of v. If $f(v) \geq f(u)$ for all vertices $u \in \mathcal{P}$, then v is a *global maximum* of f on $\mathcal{P}$. Local and global *minima* are defined symmetrically.

Arbitrary functions on $\mathcal{P}$ cannot be efficiently optimized; see, e.g., Corollary 19 of (Tovey 1997), but the prospects are better for some nontrivial and very interesting subclasses. The class of *completely unimodal* (CU) functions on hypercubes $\mathcal{H} = \{0, 1\}^d$ has been studied in, e.g, (Hammer, Simeone, Liebling, and De Werra 1988; Williamson Hoke 1988; Wiedemann 1985; Björklund, Sandberg, and Vorobyov 2002), and can be optimized in subexponential time (Björklund, Sandberg, and Vorobyov 2002). In (Björklund, Sandberg, and Vorobyov 2003b) we defined the class of *completely local-global* (CLG) functions, a generalization of CU-functions, defined on hyperstructures and capturing the properties of value-functions obtained from parity games. To demonstrate the full applicability of the algorithms in later sections, we here define the even wider class of *recursively local-global* (RLG) functions. The inclusions CU $\subset$ CLG $\subset$ RLG follow from the definitions. The strictness of these inclusions is easy to establish.

Definition 2.2 (Recursively Local-Global) *A function $f : \mathcal{P} \to \mathcal{D}$ for which all neighbors have comparable values is called* recursively local-global *(RLG) if for every substructure $\mathcal{P}'$ of $\mathcal{P}$, all local maxima of the restriction of f to $\mathcal{P}'$ are also global.* □

In the sequel, we discuss algorithms for maximizing RLG-functions. Such algorithms can also be used to maximize CLG-functions and CU-functions. We use *RLG-structure* to denote an RLG-function together with its underlying hyperstructure, and we use *RLG-cube* if the hyperstructure is a hypercube. An RLG-structure can be thought of as a hyperstructure with its vertices labeled by function values. Given an RLG-function $f : \mathcal{P} \to \mathcal{D}$, and a substructure $\mathcal{P}'$ of $\mathcal{P}$, let $w_f(\mathcal{P}')$ be the maximum value of f on $\mathcal{P}'$.

3 Ludwig-Style Algorithm for Hypercubes

The first subexponential randomized algorithm for solving simple stochastic games was presented in (Ludwig 1995). It uses the ideas of the linear programming algorithms in (Matoušek, Sharir, and Welzl 1992; Matoušek, Sharir, and Welzl 1996) and (Kalai 1992), which we will discuss in later sections. Unfortunately, the algorithm only applies to binary games (with vertex outdegree at most two), and reduction to such games may increase the number of vertices, undermining the subexponential analysis. Binary games give rise to RLG-cubes (Björklund, Sandberg, and Vorobyov 2003b).

As an introduction to the ideas underlying the algorithms presented later, and their adaptations to RLG-functions, the Ludwig-style Algorithm 1 maximizes any RLG-function $f : \mathcal{H} \to \mathcal{D}$, where $\mathcal{H} = \{0,1\}^d$.

Algorithm 1: Ludwig's Algorithm for RLG-functions
LUDWIG(RLG-cube $\mathcal{H}$, initial vertex v_0)
(1) **if** $\dim(\mathcal{H}) = 0$
(2) **return** v_0
(3) choose a random facet F of $\mathcal{H}$ containing v_0
(4) $v^* \leftarrow$ Ludwig(F, v_0)
(5) **if** the neighbor u of v^* on $\mathcal{H} \setminus F$ is better than v^*
(6) **return** Ludwig($\mathcal{H} \setminus F$, u)
(7) **else**
(8) **return** v^*

The key observation in the analysis of this algorithm is that, depending on the choice in line 3, the actual dimension of the remaining problem solved on line 6 may decrease with more than one. There are d equally likely choices; call them $F_1, F_2, \ldots, F_d$. Assume the set $\{w_f(F_1), \ldots, w_f(F_d)\}$ is linearly ordered so that

$$w_f(F_1) \leq \cdots \leq w_f(F_d).$$

If F_i is chosen for the first recursive call, none of the facets $F_1, \ldots, F_i$ will ever be visited by the algorithm again, since the value of the current vertex v^* is the biggest value on any of them. Let $T(d)$ be the expected number of times that the second recursive call will be made. Then $T(0) = 0$ and

$$T(d) \leq T(d-1) + 1 + \frac{1}{d}\sum_{j=0}^{d-1} T(j).$$

Solving the recurrence gives $T(d) = 2^{O(\sqrt{n})}$ (Ludwig 1995); see also (Gärtner 1995) for an elegant analysis. What if $\{w_f(F_1), \ldots, w_f(F_d)\}$ is not linearly ordered? This is no disadvantage to the algorithm, since it only visits vertices with *comparable and strictly bigger* values than the current vertex. Therefore, linearly ordered maximal values on the facets is the worst case.

4 Matoušek–Sharir–Welzl-Style Algorithm

In this section, we show how the LP-algorithm in (Matoušek, Sharir, and Welzl 1992; Matoušek, Sharir, and Welzl 1996) can be modified for RLG-functions, yielding a simple subexponential randomized algorithm for the problem (and thus also for parity games; see also (Björklund, Sandberg, and Vorobyov 2003b)). It has expected running time at most $2^{O(\sqrt{d\log(n/\sqrt{d})}+\log n)}$, where $n = \sum_{j=1}^{d} |\mathcal{P}_j|$ is the number of facets of $\mathcal{P}$.

Algorithm 2: MSW-Style Optimization Algorithm
MSW(RLG-structure $\mathcal{P}$, initial vertex v_0)
(1) **if** $\dim(\mathcal{P}) = 0$
(2) **return** v_0
(3) choose a random facet F of $\mathcal{P}$, not containing v_0
(4) $v^* \leftarrow$ MSW($\mathcal{P} \setminus F, v_0$)
(5) **if** the neighbor u of v^* on F is better than v^*
(6) **return** MSW(F, u)
(7) **else**
(8) **return** v^*

Note that this algorithm generalizes Ludwig's algorithm, discussed in Section 3. If $\mathcal{P}$ is a hypercube, the two algorithms are the same.

The algorithm always terminates since the recursive calls on lines (4) and (6) are made on strictly smaller substructures. It is correct since any vertex without better neighbors in an RLG-structure is globally optimal.

It remains to show that the bound from (Matoušek, Sharir, and Welzl 1996) holds in this setting. This is done by translating into the concepts of (Matoušek, Sharir, and Welzl 1996), thus showing that the same recurrence for the running time holds. The following definition will be useful.

Definition 4.1 (Extreme Facet.) *A facet of an RLG-structure is* extreme *if it contains all local maxima on $\mathcal{P}$.* □

Since all facets in one coordinate are disjoint, at most one of them can contain all maxima, so there are at most d extreme facets. The second recursive call on line (6) of the algorithm will be performed iff the chosen facet F is extreme on $\mathcal{P}$. If v_0 belongs to $d-k$ extreme facets, this happens with probability at most $\frac{k}{n-d}$, hence $\frac{\min(k,n-d)}{n-d}$. We now bound the subproblem solved in each of these cases; thus, assume that F is extreme on $\mathcal{P}$.

Definition 4.2 (Hidden Dimension.) *Given an RLG-structure $\mathcal{P}$ and a vertex $v \in \mathcal{P}$, the* hidden dimension *of the pair $(\mathcal{P}, v)$ is d minus the number of facets of $\mathcal{P}$ containing v and every $u \in \mathcal{P}$ with $f(u) > f(v)$.* □

Equivalently, $(\mathcal{P}, v)$ has hidden dimension k iff the smallest substructure containing v and all vertices with better values than v has dimension k. In particular, if the

hidden dimension is 0, then v is a maximum on $\mathcal{P}$. Let k be the hidden dimension and $\mathcal{P}'$ the corresponding substructure. The algorithm will only visit vertices that belong to $\mathcal{P}'$. There are $d-k$ facets, $F_1, \ldots, F_{d-k}$, that contain $\mathcal{P}'$; all these are extreme. In the worst case there are k more extreme facets, $F_{d-k+1}, \ldots, F_d$. For each such facet, consider the best value that does not belong to it. Assume first that these values are totally ordered. Enumerate the facets so that

$$w_f(\mathcal{P} \setminus F_{d-k+1}) \leq \cdots \leq w_f(\mathcal{P} \setminus F_{d-k+i}) \leq \cdots \leq w_f(\mathcal{P} \setminus F_d). \tag{1}$$

Suppose the algorithm chooses facet F_{d-k+i}. Then $f(v^*) = w_f(\mathcal{P} \setminus F_{d-k+i}) < f(u)$. Every vertex with a better value than v^* that belongs to F_{d-k+i} must also belong to F_{d-k+j} for all $1 \leq j < i$; the opposite would contradict (1). Thus u belongs to $(d-k+i)$ facets that contain all vertices with better values, and the hidden dimension of the pair (F_{d-k+i}, u) for the second call is at most $(k-i)$.

If the best values outside the remaining extreme facets are not totally ordered, this only benefits the algorithm. The values are partially ordered, and if facet F_{d-k+i} is chosen, u will belong to all facets that do not have a strictly bigger value for $w_f(\mathcal{P} \setminus F_j)$, and they will all contain all vertices with better values. This is because if $w_f(\mathcal{P} \setminus F_{d-k+i})$ and $w_f(\mathcal{P} \setminus F_j)$ are incomparable, then any value better than $w_f(\mathcal{P} \setminus F_{d-k+i})$ will be better than or incomparable to $w_f(\mathcal{P} \setminus F_j)$, since the order on $\mathcal{D}$ is transitive.

This discussion gives the same recurrences for the numbers of tests on line (5) and jumps to u on line (6) as (Matoušek, Sharir, and Welzl 1996) gets for the numbers of violation tests and basis computations, respectively, in the linear programming setting. They are both bounded by the recurrence $t_k(d) = 0$ (where $0 \leq k \leq d$) and

$$t_k(n) \leq t_k(n-1) + 1 + \frac{1}{n-d} \sum_{i=1}^{\min(k,n-d)} t_{k-i}(n), \qquad \text{for } n > d.$$

The recurrence is solved in (Matoušek, Sharir, and Welzl 1996), and from the solution it is easy to infer that the expected running time of the algorithm on RLG-structures is at most $2^{O(\sqrt{d \log(n/\sqrt{d})} + \log n)}$, as long as function evaluation and comparison of function values can be performed in polynomial time. As soon as n can be assumed to be polynomially bounded by d, this bound is subexponential in d. The reduction from parity games with d vertices to RLG-functions gives $n = O(d^2)$, so we can solve parity games in expected time $2^{O(\sqrt{d \log d})}$.

5 Kalai-Style Algorithm

We now describe another algorithm for optimizing RLG-structures, originally invented for linear programming by Kalai (Kalai 1992; Goldwasser 1995) and later adapted by us to parity games (Björklund, Sandberg, and Vorobyov 2003a). It is

also subexponential with complexity similar to the MSW-style algorithm of the previous section (but it is slightly more complicated).

Let $\mathcal{P}(d,n)$ denote the class of RLG-structures with dimension d and $n = \sum_{j=1}^{d} |\mathcal{P}_j|$. If v is a vertex in $\mathcal{P}$ then a facet F is *v-improving* if some *witness vertex* $v' \in F$ has a better value than v.

The Algorithm takes an RLG-structure $\mathcal{P} \in \mathcal{P}(d,n)$ and an initial vertex v_0 as inputs and returns the optimal vertex on $\mathcal{P}$. It uses the subroutine COLLECT-IMPROVING-FACETS, described later, that collects a set of pairs (F,v) of v_0-improving facets F and corresponding witness vertices $v \in F$.

Algorithm 3: Kalai-Style Optimization Algorithm
KALAI(RLG-structure $\mathcal{P}$, initial vertex v_0)
(1) **if** $\dim(\mathcal{P}) = 0$
(2) **return** v_0
(3) $M \leftarrow$ COLLECT-IMPROVING-FACETS($\mathcal{P}, v_0$)
(4) choose a random pair $(F, v_1) \in M$
(5) $v^* \leftarrow$ KALAI(F, v_1)
(6) **if** some neighbor u of v^* on $\mathcal{P} \setminus F$ is better than v^*
(7) **return** KALAI($\mathcal{P}, u$)
(8) **else**
(9) **return** v^*

How to Find Many Improving Facets. Now we describe the subroutine COLLECT-IMPROVING-FACETS. The goal is to find r v_0-improving facets, where r is a parameter (Kalai uses $r = \max(d, n/2)$ to get the best complexity analysis). To this end we construct a sequence $(\mathcal{P}^0, \mathcal{P}^1, \ldots, \mathcal{P}^{r-d})$ of substructures of $\mathcal{P}$, with $\mathcal{P}^i \in \mathcal{P}(d, d+i)$ and $\mathcal{P}^i \subset \mathcal{P}^{i+1}$. All the $d+i$ facets of $\mathcal{P}^i$ are v_0-improving; we simultaneously determine the corresponding witness vertices v^j optimal in $\mathcal{P}^j$. The subroutine returns r facets of $\mathcal{P}$, each one obtained by fixing one of the r choices in $\mathcal{P}^{r-d} \in \mathcal{P}(d,r)$. All these are v_0-improving by construction.

Let v^0 be a better neighbor of v_0 on $\mathcal{P}$. (If no better neighbor exists then v_0 is optimal in $\mathcal{P}$ and we are done.) Set $\mathcal{P}^0$ to the RLG-structure containing only v^0. Fixing any of the d coordinates of $\mathcal{P}$ as in v^0 defines a v_0-improving facet of $\mathcal{P}$ with v^0 as a witness.

To construct $\mathcal{P}^{i+1}$ from $\mathcal{P}^i$, let v' be a better neighbor of v^i. (Note that v^i is optimal in $\mathcal{P}^i$ but not necessarily in the full structure $\mathcal{P}$. If it is, we terminate.) Let $\mathcal{P}^{i+1}$ be the smallest substructure of $\mathcal{P}$ containing v' and all vertices in $\mathcal{P}^i$. Recursively apply the algorithm to find the optimal v^{i+1} in $\mathcal{P}^{i+1}$. Note that fixing a coordinate in $\mathcal{P}$ to any of the $d+i$ choices in $\mathcal{P}^i$ defines a v_0-improving facet. Therefore, the final $\mathcal{P}^{r-d}$ has r v_0-improving facets.

Analysis. First note that we can pick the random number before line 3, pass it to COLLECT-IMPROVING-FACETS, and only find that many facets. Now each solved subproblem starts from a strictly better vertex, so the algorithm clearly terminates.

It is correct because it can only terminate by returning an optimal vertex.

This algorithm yields a recurrence similar to those in previous sections. Kalai solves it and gets the subexponential running time $2^{O((\log n)\sqrt{d/\log d})}$.

It is interesting to note the similarities of Ludwig's, Matoušek–Sharir–Welzl's, and Kalai's subexponential algorithms. They all take a structure $\mathcal{P}$ and a vertex v_0 as parameters and recursively find the optimum v^* on a substructure $\mathcal{P}'$ of $\mathcal{P}$. The choice of $\mathcal{P}'$ is random, but it is guaranteed to contain vertices at least as good as v_0, and the recursive call starts from a witness of this fact. If the result is not an optimum on the entire structure, then we optimize the rest of the structure, starting from a better neighbor of v^*. Intuitively, the subexponential upper bound relies on the substructure being taken randomly from some set. Thus it is expected that the optima of many structures in the set are no better than v^*, and those structures will be ignored in the remainder of the algorithm.

6 Abstract Optimization Problems

The class of *abstract optimization problems* (AOPs) was introduced in (Gärtner 1995). He uses this generalization to show how the ideas from (Matoušek, Sharir, and Welzl 1996; Kalai 1992) can be used to obtain a subexponential randomized algorithm for a wider class of optimization problems. In this section we show that optimizing an RLG-function with totally ordered codomain can be reduced to solving an AOP.

Definition 6.1 (AOP (Gärtner 1995)) *An AOP is a triple $(A, <, \Phi)$, where A is a finite set, $<$ is a total order on 2^A, and $\Phi : \{(C, B) | C \subseteq B \subseteq A\} \to 2^A$ satisfies*

$$\Phi(C,B) = \begin{cases} C, & \text{iff } C = \max_<\{C' | C' \subseteq B\}; \\ C', \text{for some } C' \text{ s.t. } C < C' \subseteq B, & \text{otherwise.} \end{cases}$$

For $B \subseteq A$, let $opt(B)$ denote $\max_<\{C | C \subseteq B\}$. Solving an AOP means finding $opt(A)$. □

Intuitively, solving an AOP corresponds to maximizing some function on a boolean hypercube. The function may be unstructured, but there is an oracle capable of finding some better vertex than the current one on any subcube containing the origin. The algorithm in (Gärtner 1995) solves any AOP with $|A| = n$ using expected $e^{2\sqrt{n}+O(\sqrt[4]{n}\ln n)}$ calls to Φ. Maximizing an RLG-function f with n facets can be reduced to solving an AOP with $|A| = n$, if its codomain is totally ordered. This restriction is often satisfied, as for, e.g., parity games (Björklund, Sandberg, and Vorobyov 2003b).

Let $\mathcal{P}$ be a d-dimensional hyperstructure with n facets, and let $f : \mathcal{P} \to \mathcal{D}$ be an RLG-function with totally ordered codomain. Define an AOP $(\mathcal{F}, <, \Psi)$ as follows. Let $-\infty$ be an artificial value, smaller than all values of vertices of $\mathcal{P}$. Let

$\mathcal{F}$ be the set of all facets of p, and define $val : 2^{\mathcal{F}} \to \mathcal{D} \cup \{-\infty\}$ by

$$val(F) = \begin{cases} w_f(struct(F)), & \textit{iff } struct(F) \neq \emptyset; \\ -\infty, & otherwise. \end{cases}$$

Take $\prec$ to be any total order on $2^{\mathcal{F}}$ such that $F \prec F'$ whenever $|F| > |F'|$. For $F, F' \in \mathcal{F}$ let $F < F'$ iff 1) $val(F) < val(F')$, or 2) $val(F) = val(F')$ and $F \prec F'$. Now define $\Phi(F, G)$ as follows.

- Suppose $val(F) \neq -\infty$ and $struct(F)$ contains only one vertex v. If v is a local maximum on $struct(G)$, then $\Phi(F, G) = F$. Otherwise $\Phi(F, G)$ is a set defining a 0-dimensional substructure containing only a better neighbor of v on $struct(G)$.
- If $val(F) \neq -\infty$ but $|F| > d$ then $\Phi(F, G) = F'$ where F' defines a substructure containing only one vertex with maximal value on $struct(F)$.
- If $val(F) = -\infty$ and $struct(G) = \emptyset$ then $\Phi(F, G) = \emptyset$.
- If $val(F) = -\infty$ and $struct(G) \neq \emptyset$ then $\Phi(F, G) = F'$ for some $F' \in struct(G)$.

Now $(\mathcal{F}, <, \Phi)$ defines an AOP, and applying Gärtner's algorithm will yield a solution from which a maximal vertex of the original RLG-structure can be recovered.

Unfortunately, the bound this gives for maximizing RLG-functions is not very good. The previously discussed algorithms give bounds subexponential in d as long as n is polynomial in d, which is not the case here. If, for example, $n = \Theta(d^2)$, which is the worst case when reducing games to RLG-functions, we only get a $2^{O(d)}$ bound on the expected calls to Φ. The reason seems to be that the original dimension is lost in the reduction, and all facets are treated as independent.

Gärtner's algorithm, called with the appropriate parameters, will never call Φ with any set of size bigger than d as the first parameter. For such sets, Φ can be computed in polynomial time.

7 Conclusions

In (Björklund, Sandberg, and Vorobyov 2003a; Björklund, Sandberg, and Vorobyov 2003b) we showed how the well-known linear programming algorithms from (Kalai 1992; Matoušek, Sharir, and Welzl 1996) can be adapted to solving parity games. In this paper we showed that these algorithms work for maximizing recursively local-global functions on hyperstructures. Together with the reductions from games to RLG-functions from (Björklund, Sandberg, and Vorobyov 2003b), this stresses the combinatorial similarities between linear programming and solving games. It also provides a simple setting, stripped of cumbersome details, convenient for investigation of such similarities and for developing new algorithms for parity and simple stochastic games.

Acknowledgements We thank anonymous referees for valuable remarks, improvements and references.

References

Agrawal, M., N. Kayal, and N. Saxena (2002, August). Primes is in P. Unpublished manuscript, `http://www.cse.iitk.ac.in/news/primality.html`.

Björklund, H., S. Sandberg, and S. Vorobyov (2002, May). Optimization on completely unimodal hypercubes. Technical Report 018, Uppsala University / Information Technology. `http://www.it.uu.se/research/reports/`.

Björklund, H., S. Sandberg, and S. Vorobyov (2003a). A discrete subexponential algorithm for parity games. In H. Alt and M. Habib (Eds.), *20th International Symposium on Theoretical Aspects of Computer Science, STACS'2003*, Volume 2607 of *Lecture Notes in Computer Science*, Berlin, pp. 663–674. Springer-Verlag. Full preliminary version: TR-2002-026, Department of Information Technology, Uppsala University, September 2002, `http://www.it.uu.se/research/reports/`.

Björklund, H., S. Sandberg, and S. Vorobyov (2003b, January). On combinatorial structure and algorithms for parity games. Technical Report 2003-002, Department of Information Technology, Uppsala University. `http://www.it.uu.se/research/reports/`.

Condon, A. (1993). On algorithms for simple stochastic games. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 13*, 51–71.

Emerson, E. A., C. Jutla, and A. P. Sistla (1993). On model-checking for fragments of μ-calculus. In *Computer Aided Verification, Proc. 5th Int. Conference*, Volume 697, pp. 385–396. Lect. Notes Comput. Sci.

Gärtner, B. (1995). A subexponential algorithm for abstract optimization problems. *SIAM Journal on Computing 24*, 1018–1035.

Goldwasser (1995). A survey of linear programming in randomized subexponential time. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory) 26*, 96–104.

Hammer, P. L., B. Simeone, T. M. Liebling, and D. De Werra (1988). From linear separability to unimodality: a hierarchy of pseudo-boolean functions. *SIAM J. Disc. Math. 1*(2), 174–184.

Kalai, G. (1992). A subexponential randomized simplex algorithm. In *24th ACM STOC*, pp. 475–482.

Ludwig, W. (1995). A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation 117*, 151–155.

Matoušek, J., M. Sharir, and M. Welzl (1992). A subexponential bound for linear programming. In *8th ACM Symp. on Computational Geometry*, pp. 1–8.

Matoušek, J., M. Sharir, and M. Welzl (1996). A subexponential bound for linear programming. *Algorithmica 16*, 498–516.

Sharir, M. and E. Welzl (1992). A combinatorial bound for linear programming and related problems. In *9th Symposium on Theoretical Aspects of Computer Science (STACS)*, Volume 577 of *Lecture Notes in Computer Science*, Berlin, pp. 569–579. Springer-Verlag.

Tovey, C. A. (1986). Low order polynomial bounds on the expected performance of local improvement algorithms. *Mathematical Programming 35*, 193–224.

Tovey, C. A. (1997). Local improvement on discrete structures. In E. Aarts and L. J. K. (Eds.), *Local Search in Combinatorial Optimization*, pp. 57–89. John Wiley & Sons.

Vöge, J. and M. Jurdziński (2000). A discrete strategy improvement algorithm for solving parity games. In *CAV'00: Computer-Aided Verification*, Volume 1855 of *Lect. Notes Comput. Sci.*, pp. 202–215. Springer-Verlag.

Wiedemann, D. (1985). Unimodal set-functions. *Congressus Numerantium 50*, 165–169.

Williamson Hoke, K. (1988). Completely unimodal numberings of a simple polytope. *Discrete Applied Mathematics 20*, 69–81.

Building Sub-corpora Suitable for Extraction of Lexico-Syntactic Information

Ondřej Bojar
Institute of Formal and Applied Linguistics, ÚFAL MFF UK, Malostranské náměstí 25, CZ-11800 Praha, Czech Republic
obo@cuni.cz

Abstract.

Accuracy of automatic syntactic analysis of natural languages with relatively free word order (such as Czech) can be hardly improved without building large and precise lexicons of syntactic behavior of individual words (e.g. lexicons of verb valency frames). The current treebanks available do not cover enough words. Larger corpora lack the syntactic annotation and many sentences contained in them are too complex to extract the information easily or even automatically. The system AX (automatic extraction) was developed to perform selection of morphologically analyzed sentences by means of custom hand-written rules. The rules can be easily formulated to perform linguistic-motivated filtration. The system AX allows to perform partial syntactic analysis in order to check the occurrence of more complex linguistic phenomena.

1 Motivation

At the current stage of the development, the accuracy of automatic syntactic analyzers of natural languages (in particular Czech) is limited due to the lack of large and precise lexicons of syntactic behavior of individual words (verb valency frames are the most important example). Building such lexicons by hand is rather a time-consuming task and any kind of automatic preprocessing would help.

The available data sources include corpora annotated on different levels of linguistic description. Syntactic information can be easily extracted from corpora annotated on the syntactic level (such as the Prague Dependency Treebank, PDT[1], Böhmová et al. (2001)) and some attempts to extract for example verb subcategorization frames from treebanks were already performed[2]. However the number of occurrences of individual lexical items in

[1] `http://ufal.mff.cuni.cz/pdt/`
[2] Such as (Sarkar and Zeman, 2000).

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

such corpora is usually not sufficient. Bojar (2002) shows that only a few hundreds of verbs have enough occurrences in PDT and that more than 60% of 26,000 verbs found in the Czech National Corpus (CNC[3]) are not covered in PDT at all. Therefore, it is necessary to extract the lexico-syntactic information from larger corpora (such as the CNC) or any texts available.

However, not all sentences containing a given lexeme can serve as a good example to extract the syntactic information (for instance, if two verbs are present in one clause, their complements and adjuncts can be arbitrarily intermixed). And moreover, as the syntactic annotation in these corpora is missing, the sentences can often be too complex to be analyzed by any of the available parsers at a reasonable level of accuracy.

I developed the system AX to simplify the task of selecting feasible examples of sentences for extracting a specific lexico-syntactic information (not just the verb valency frames). The sentences can be easily selected on linguistically based criteria. Partial syntactic analysis of sentences is possible, in order to be able to answer more complex linguistic questions about the sentence.

In section 2 I describe the overall architecture of the system AX. Section 3 gives a brief description of the basic data structure, variant feature structure. Sections 4 and 5 describe the core of the scripting language AX: filters to reject sentences and rules to perform (partial) syntactic analysis. In the last section, I illustrate the use of the system to select sentences suitable for extracting valency frames of Czech verbs and document the improvement of accuracy of Czech parsers when applied only to the selected sentences.

2 The Architecture of AX

The system AX combines the idea of regular expressions and replacements (see (Karttunen et al., 1996; Aït-Mokhtar and Chanod, 1997) and others) with the idea of feature structures (see below) known from unification-based parsers. This combination leads to a formalism that is both, strong to describe complex linguistic properties of sentences of natural language and efficient in the process of parsing.

The user prepares a script of filters and rules to select sentences suitable for a specific purpose. The system AX loads the script and then expects sentences augmented with their morphological annotation (in format of the PDT[4]) on the standard input. The input sentences may or may be not morphologically disambiguated. For every input sentence, the system runs the script and checks, if the sentence passed all the filters or has been rejected. For sentences that pass (referred to with the term "selected sentences"), the output of the final phase (see below) is printed out. This output is for some

[3] `http://ucnk.ff.cuni.cz/`

[4] See `http://shadow.ms.mff.cuni.cz/pdt/Corpora/PDT_1.0/Doc/morph.html`

purposes already suitable for collecting the lexico-syntactic information so that no other parser to process the sentences is needed.

In the following, I describe the overall running scheme of AX. The input sentence is internally stored as a sequence of feature structures that correspond one to one to input word forms. (See section 3 for details.) The input sentence is then processed through a pipe of consecutive *blocks (phases) of operation*. Each of the blocks is either a *filter*, or a *set of rules*.

The input for each block is a set of sequences of feature structures (referred to with the term *the set of "input readings" of the sentence*). If the block is a filter, it checks all the input readings and possibly rejects some of them. If the block is a set of rules, it updates every input reading with all applicable rules and returns a larger set of new readings (it "generates" new readings).

Consecutive blocks are connected, so that the output set of readings from the former block is used as the input set of readings for the latter block. The first of the blocks receives as input the input sentence, the output from the last block is printed out. The order and type of the blocks is up to the author of the script. All the input sentences that were not rejected by any of the filters are *accepted* and the output produced for each of them can also be used to extract the lexico-syntactic information, if appropriate.

A sample flow of readings is demonstrated in figure 1.

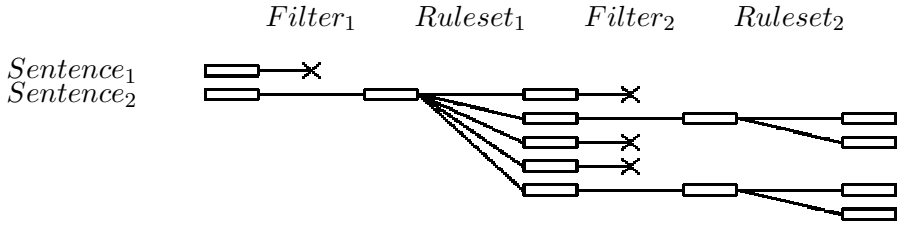


Figure 1: Progress of sentences through an AX script. The first input sentence was rejected by the first filter. The second sentence passed the filter and several readings were obtained by the rules in ruleset 1. Some of the readings were then rejected by filter 2 and some passed. Altogether four different readings were then produced by the last ruleset.

3 Feature Structures with Variants

Feature structures (also called attribute-value matrices) allow representing of various linguistic information in a compact and natural way.[5]. For the purposes of this work, untyped feature structures with alternatives (variants) of values are sufficient and serve well to represent very rich and often very

[5]For a detailed characteristic of typed feature structures see Penn (2000).

ambiguous morphological information[6] as well as arbitrary user flags useful in the process of filtering and generating new readings by rules.

The basic operation with two variant feature structures is *unification*. The output of the unification is a feature structure that holds information from both the input structures.[7] For instance:

$$\begin{bmatrix} \text{name} & \text{Kamil} \\ \text{surname} & \{\text{Horak, Klement}\} \end{bmatrix} \text{ and } \begin{bmatrix} \text{surname} & \text{Horak} \\ \text{age} & \text{int(32)} \end{bmatrix} \quad \begin{matrix} \text{unify and} \\ \text{the result is} \end{matrix} \quad \begin{bmatrix} \text{name} & \text{Kamil} \\ \text{surname} & \text{Horak} \\ \text{age} & \text{int(32)} \end{bmatrix}$$

Unification fails, if both the features contain an attribute of the same name but a non-unifying value.

For every input word in a sentence, the morphological analysis gives all possible lemmas and morphological attributes of the given word form. This ambiguous morphological information can be stored in a single feature structure with variants. See figure 2 on the facing page for an example. The whole sentence of word forms can therefore be stored as a list of feature structures of the same length.

4 Filters

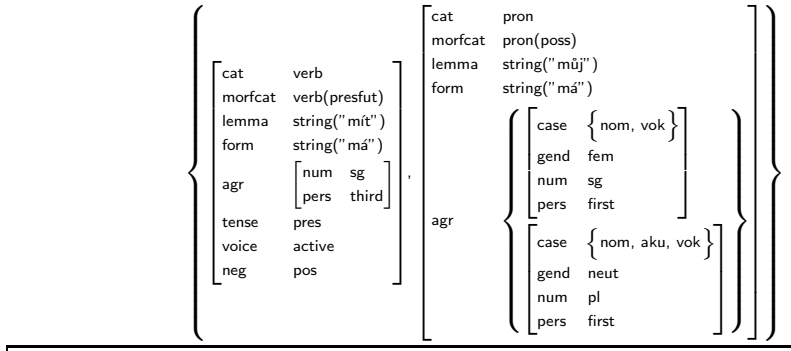
Filters in the language AX are expressed in the form of *regular expressions of feature structures*. The basic differences between common regular expressions (used for instance in many Unix tools) and regular expressions of feature structures used in the language AX are:

- The primitive element of regular expressions is no longer a character, but rather a feature structure. In scripting language AX, the feature structure can be expressed either explicitly or by a shortcut name[8].
- When searching for a subsequence of feature structures that matches a given regular expression, the system checks whether the input structure unifies with the structure in the expression. (Rather than checking two characters for equality.)

[6] In Czech, approx. 4,000 different tags are defined, half of which actually occurred in the Czech National Corpus. For many word forms several dozens of morphological tags are possible (7 different cases · 4 genders · 2 (sg/pl) = 56 possible tags).

[7] If more variants of a value are available, the output will carry out the intersection (more precisely the product of unification of all possible combinations of input variants).

[8] For example, specific types of pronouns can syntactically serve as nouns or adjectives. It is useful to define a shortcut of a feature structure that would match a noun or a noun-like type of pronoun etc. In filters and rules, it is then possible to introduce the whole structure only by its shortcut name.




```
[
  cat - verb, morfcat-verb(presfut), lemma - "mít", form - "má",
  agr - [ num-sg, pers-third ], tense - pres, voice - active, neg-pos
|
  cat - pron, morfcat - pron(poss), lemma - "můj", form - "má",
  agr - [ case - nom;vok, gend - fem, num - sg, pers - first
        | case - nom; aku; vok, gend - neut, num - pl, pers - first ]
]
```

Figure 2: This feature structure represents all the possible morphological analyses of the word form *má* which can be a word form of two different lemmas ($mít_{\texttt{verb}}$ and $můj_{\texttt{pronoun}}$ in different cases, numbers and genders). Below is the same feature structure expressed in syntax of the scripting language of AX.

Details of the syntax of the filters are described in Bojar (2002), here I give just a brief example of two different filters:

```
filter reject_more_than_two_verbs:
  .* verb .* verb .*
end

keep "Keep only sentences with exactly one verb
      or those not containing any conjunction":
  !verb* verb !verb* | !conj*
end
```

The keyword `filter` means: reject the sentence if it (as a whole) matches the given regular expression. The meaning of the keyword `keep` is: reject the sentence if it doesn't match the given expression.

5 Rules

Rules are used to modify the input readings of a sentence and generate new readings. Rules have always this form:

`rule` <rule name> :
<replacement> `--->` <input regular expression> `::`
<constraints>
`end`

The rule is applied as follows:

- The input sequence of feature structures is searched in order to find a subsequence that matches the <input regular expression> and the <constraints>.
- The obtained subsequence of feature structures is replaced with the <replacement>.

By default, the input sequence is searched for all possible subsequences matching the regular expression and constraints, therefore the rule can produce for one input reading several output readings. In many situations, this nondeterministic approach leads to more output readings than the user actually wants. For such cases, the user can write special keywords in the arrow in the rule to make the rule substitute for instance only the first matching subsequence of feature structures.

By default, the output of one rule within a ruleset is used as input for another rule in the same ruleset (possibly reusing the rule itself). All possible combinations of applying rules are attempted and all the possible outcomes are collected to build the final output set of possible readings for this block (phase) of operation. This nondeterministic behavior can be restricted in several ways: rules can be limited in number of allowed applications[9], an output from one rule can be included in the final set only if no other rule was able to change it, and others. A detailed description of all the options is out of the scope of this paper.

In order to "compute" the <replacement> from the subsequence found in the input, one can use *variables*. The variables can be used in all parts of rules: from the <input regular expression> they get their initial value. The value is then restricted or updated by the <constraints> and their final value is given to the output in the <replacement>.

All the variables can hold a feature structure. The scope of the variables is limited for one application of one rule, that is all the variables are local for the rule and within one application.

The <constraints> are expressed as an unordered list of requirements on variables values. All the requirements must be fulfilled for the rule to be applicable. The constraints can only require certain feature (sub)structures to

[9]In fact, the system AX will not start unless all non-shortening rules, i.e. the rules that are able to produce output not shorter than the input reading, have this maximum number of applications explicitly expressed as a function of the number of input structures. This effectively blocks out the possibility to loop ad infinitum.

unify. By means of these requirements, output variables also get their value. The following example shows a rule to perform a reduction: it combines an adjective and a noun together:

```
rule out_noun ---> adj noun ::
  adj.agr = noun.agr,
  out_noun = noun
end
```

The constraint `adj.agr = noun.agr` guarantees the congruence of the noun and the adjective in case, gender and number. The constraint `out_noun = noun` initializes the output variable with the feature structure of the noun *after* it was already restricted in case, number and gender due to the congruence requirement. In this way, the input morphological ambiguity is step by step solved.

The output <replacement> may copy parts of the input subsequence. This allows an easy formulation of rules that combine distant feature structures in the input sentence. The regular expression can then be used to restrict what can stay between the two (or more) feature structures. As a nice example I show the rule that combines two parts of a Czech verb – the auxiliary part ($být_{tobe}$, which can have several forms, such as $jsem_{\texttt{I}am}$) and the main verb (such as *zalít*, which also can have several forms). The rule also checks the parts for congruence[10]:

```
rule complex_past_tense:
  complex \gap trace
  ---->
     zalil {gap:!{verb,comma,conj}*} jsem
   | jsem {gap:!{verb, comma, conj}*} zalil
  ::
  zalil <- morfcat, voice -> ‘zalil‘,
  zalil = [cat-verb],
  jsem.cat = ‘jsem‘.cat,
  jsem <- morfcat, lemma, neg -> ‘jsem‘,
  jsem.agr = [pers-first;second],
  zalil <- agr.num, agr.gend -> jsem,
  complex = [cat-complexpast],
  complex <- lemma, form, neg, morfcat -> zalil,
  complex <- agr.pers, agr.num, agr.gend, mood -> jsem
  trace = [cat-trace, form-"XtraceX"]
end
```

[10]The scripting language AX has a shorter form of expressing several unification requirements on two variables at once. The constraint: “`usnul <- cat, agr.num, agr.gend -> jsem`” is equivalent with these three: “`usnul.cat = jsem.cat, usnul.agr.num = jsem.agr.num, usnul.agr.gend = jsem.agr.gend`”

The rules find such a subsequence of feature structures that begins with the auxiliary verb and ends with the main verb (or vice versa). The gap between the parts of verb must not contain any other verb, comma or conjunction (introduced by means of shortcuts, see above). The gap gets a label "`gap`", so that it can be copied to the output replacement. As the output, the rule produces a single feature structure representing the complex verb followed a copy of the `gap` section and an auxiliary trace at the place where the other part of the verb was found. Naturally, the trace can be omitted if not needed by any other rules of filters.

6 A Sample AX Usage and Results

The system AX was used to select sentences suitable for extracting typical complements of verbs, namely verb valency frames. The script for this purpose contained 15 filters and 21 rules and selected sentences where the complements of verbs would be easy to observe and excluded sentences that are too difficult to parse. Approximately 15 to 20% of sentences from the Czech National Corpus are selected by this script. I call them "very simple sentences".[11]

So far, the task of actually extracting verb valency frames from the selected sentences was not performed, neither manually nor automatically.[12] Anyway, the utility of the described sentence preselection can be illustrated by measuring the improvement of accuracy of parsers available for Czech (Collins et al. (1999), Zeman (1997, 2002) and a parser by Zdeněk Žabokrtský (unpublished)). All the parsers were tested on all the sentences in the evaluation part of the Prague Dependency Treebank and separately on the selected "very simple sentences" only.

The results show that the best parser available for Czech, the Collins parser, is able to correctly observe 55% of verb frames[13]. When used on very simple sentences only, this measure increases by 10%. A similar result can be achieved by using the parser on short sentences only: in sentences with at most ten words, the Collins parser correctly observes 68% of verb frames. A combination of both filters, short and "very simple" sentences, is however still 5% better, exceeding 73% of correctly observed verb frames.

[11] Optionally, sentences containing "suspicious word order patterns" (WOP) were also rejected. Straňáková-Lopatková (2001) cautiously analyzes the risks of syntactic ambiguity of noun and prepositional phrases in Czech. She defines those WOPs, where there it is not possible to decide whether a noun phrase depends on another noun phrase or on the verb itself. These examples would spoil the observation of verb frames, but it is easy to filter them out using the AX. Full description of the linguistically motivated filters exceeds the scope of this paper, see (Bojar, 2002) for details.

[12] In (Bojar, 2002) I propose an algorithm to extract surface verb frames and describe several open problems of the task of inferring verb valency frames from the surface ones.

[13] That is to correctly identify all immediate daughters of a verb in the dependency tree and not mark any extra nodes as daughters of the verb.

Also quite interesting is the improvement of the traditional accuracy measure, namely the number of correctly assigned dependencies. If used on very simple sentences, the parsers achieve an accuracy of 5 to 10 percent better, up to 88% for Collins. On short and "very simple" sentences, 91.4% for Collins is achieved.

7 Conclusions

In this paper I described a system to perform selection of sentences from corpora based on linguistically motivated criteria. The system allows an easy formulation of filters and also rules for partial syntactic analysis of sentences, if needed to check for more complex phenomena. The sentences can be selected for many purposes: as an automated preprocessing to supply lexicographers with more relevant examples of sentences as well as a first step in a fully automatic extraction of lexico-syntactic information. If the script of filters and rules is cautiously designed, already the output produced from the system AX can serve as raw input to build a lexicon. The expressive power of filters and rules of AX is better than any corpus searching tool known to me. The system was used to select sentences suitable to extract verb valency frames and the utility of this selection was illustrated by improvement of three parsers' accuracy.

In future work, I will use the described system in a large scale to extract valency frames of verbs from the whole Czech National Corpus. I will also try to design an AX script to select sentences suitable for extraction of valency frames of nouns.

8 Acknowledgments

The scripting language presented in this article was developed as a part of my master thesis, Bojar (2002). I would like thank to the supervisor of the thesis, RNDr. Vladislav Kuboň, Ph.D., as well as to all the researchers and staff at the Center for Computational Linguistics, Charles University, Prague. This work was partially supported by the grant GAČR 201/02/1456 and GAUK 300/2002/A INF-MFF.

References

Aït-Mokhtar, Salah and Jean-Pierre Chanod. 1997. Incremental finite-state parsing. In *Proceedings of ANLP'97*, pages 72–79, Washington, March 31st to April 3rd.

Böhmová, Alena, Jan Hajič, Eva Hajičová, and Barbora Hladká. 2001. The Prague Dependency Treebank: Three-Level Annotation Scenario.

In Anne Abeillé, editor, *Treebanks: Building and Using Syntactically Annotated Corpora*. Kluwer Academic Publishers.

Bojar, Ondřej. 2002. Automatická extrakce lexikálně-syntaktických údajů z korpusu (Automatic extraction of lexico-syntactic information from corpora). Master's thesis, ÚFAL, MFF UK, Prague, Czech Republic. In Czech.

Collins, Michael, Jan Hajič, Eric Brill, Lance Ramshaw, and Christoph Tillmann. 1999. A Statistical Parser of Czech. In *Proceedings of 37th ACL Conference*, pages 505–512, University of Maryland, College Park, USA.

Karttunen, Lauri, Jean-Pierre Chanod, Gregory Grefenstette, and Anne Schiller. 1996. Regular expressions for language engineering. *Natural Language Engineering*, 2(4):305–328.

Penn, Gerald. 2000. *The Algebraic Structure of Attributed Type Signatures*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University.

Sarkar, Anoop and Daniel Zeman. 2000. Automatic Extraction of Subcategorization Frames for Czech. In *Proceedings of the 18th International Conference on Computational Linguistics (Coling 2000)*, Saarbrücken, Germany. Universität des Saarlandes.

Straňáková-Lopatková, Markéta. 2001. Homonymie předložkových skupin v češtině a možnost jejich automatického zpracování (Ambiguity of prepositional phrases in Czech and possibilities for automatic treatment). Technical Report TR-2001-11, ÚFAL/CKL, Prague, Czech Republic. In Czech.

Zeman, Daniel. 1997. A Statistical Parser of Czech. Master's thesis, ÚFAL, MFF UK, Prague, Czech Republic. In Czech.

Zeman, Daniel. 2002. Can Subcategorization Help a Statistical Parser? In *Proceedings of the 19th International Conference on Computational Linguistics (Coling 2002)*, Taibei, Tchaj-wan. Zhongyang Yanjiuyuan (Academia Sinica).

Formalizing determination and typicality with LDO

Jérôme Cardot
Équipe LaLICC: Langages, Logiques, Informatique, Cognition, Communication
UMR 8139 - CNRS - Université Paris IV Sorbonne
http://www.lalic.paris4.sorbonne.fr
Jerome.Cardot@paris4.sorbonne.fr

Abstract. Determination is a fundamental operation in language, by which objects are built intensionally, more and more specified along the utterance. This paper presents LDO, a formalism in which objects are built upon concepts. Concepts are involved in incompatibility relations, used by LDO to give a representation of essence and intension of concepts. LDO allows inference on typicality, and representation of objects of language.

When we use a common noun to introduce a new object in discourse, this object is at first time unspecified, indeterminate. Then the utterance gives determinations to it, through different ways: determiners, quantifiers, modifiers such as adjectives, noun complements, relatives.

In this article, we present LDO[1], a logic to represent how objects are determinated. We show how the fundamental operation of determination helps in analyzing quantification and typicality, and in structuring concepts with intensional relations. We give a formalization and some theorems for LDO, then we point how this logic can represent objects, joining ability to structure the lexicon and ability to describe objects along utterances.

1 Overview of LDO

1.1 Sources of LDO

The main choice in LDO is to construct objects by their intension, that is, by ideas (or concepts) related with terms. This was already pointed out by (Arnauld and Nicole 1662)[2]:

[1] LDO: Logique de la Détermination d'Objets; the basic idea was originally presented in (Desclés 1986).

[2] Usually called *Logique de Port-Royal*.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

> « *La détermination est quand ce qu'on ajoûte à un mot général en restreint la signification, & fait qu'il ne se prend plus pour ce mot général dans toute son étendue, mais seulement pour une partie de cette étendue.* »[3]
>
> « *Determination is when something is added to a general word, restricts its meaning, so that it doesn't stand for the general word with all its scope, but only for a part of this scope.* »

We take concepts, as (Culioli 1999) does (he calls them *notions*), for primitive elements, nouns and verbs coming later, built upon these concepts.

More formally, our vision of concepts is inspired by Frege's one: in (Frege 1879), a concept is an « *insaturated expression* » (a function), the saturation of which (by an object) being a content of thought.

Inside the frame of GA&C[4], which is built on Curry's typed combinatory logic, a concept is an operator building a content of thought (its type being written H), from an object (of type J). The functional type of this operator is FJH[5].

1.2 Objects and determination in LDO

In LDO objects are built, may be more or less determinate, or still may be involved in predication, by use of operators τ and δ acting upon concepts:

- τ: builds, upon a notion f, τf, typical object f, the typical indeterminate object associated to notion f.

- δ: builds, upon a notion f, δf, a determination, that is, an operator acting upon an object x and building a more determinate object $(\delta f)x$. So an object may be built by several determinations acting one after the other, and we'll write Δ for a determination chain.

The class of concepts is denoted by $\mathcal{F}$, the class of objects by $\mathcal{O}$, among which $\mathcal{O}_{det}$ is the class of completely determinate objects, and $\mathcal{O}_{ind}$ the class of not completely determinate objects, so we have $\mathcal{O} = \mathcal{O}_{ind} \cup \mathcal{O}_{det}$

We represent objects determinated from τf (or from any other object x) with a cone, the vertex of which is τf (or x). In case of τf, all these objects constitute Étendue f (the scope of f). Among them, some objects are completely determinate: they constitute Extension f, represented by the base of the cone. Extension f = Étendue $f \cap \mathcal{O}_{det}$

[3](Arnauld and Nicole 1662, I, VIII, p. 66).

[4]GA&C: Grammaire Applicative et Cognitive – *cf.* (Desclés 1990).

[5]See (Curry and Feys 1958); if A and B are types, FAB denotes the type of an operator building a B-typed object upon an A-typed object.

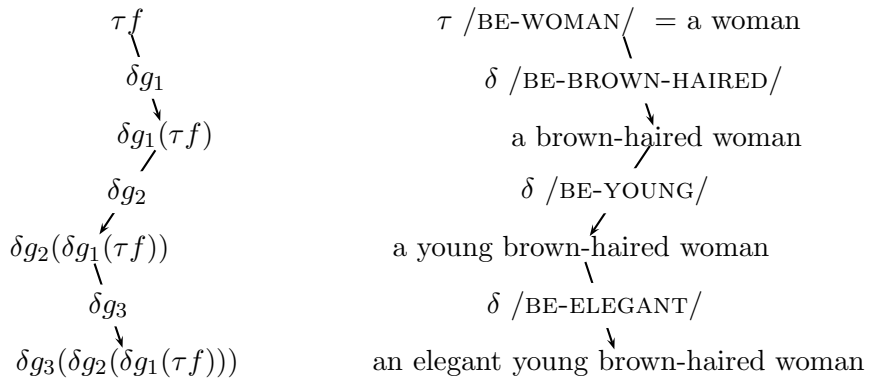


Figure 1: Determinations of objects.

2 Determination, compatibility and typicality

Some relations occur between concepts: LDO is concerned with relations of incompatibility, of comprehension, of typical-incompatibility, and of typical comprehension.

Not all determinations can be applied upon all objects for building other objects: some determinations are impossible, due to incompatibility between concepts. This is closely related to type incompatibility. For example, talking of « *a rainy dog* » doesn't make sense. But, as we intend to represent determinations in human language, there is no relevance in using here a type theory to forbid such a determination, too many phrases would then be forbidden, for example metaphors.

Comprehension of concepts shows an intensional relation between them. For Port-Royal, « f *encompasses* g » means that we can't subtract the idea g from the idea f without destroying the idea f.

This leads to a link between comprehension and incompatibility: f encompasses g iff f is incompatible with $\sim_1 g$ (where $\sim_1 g$ is the negation of g: $\sim_1 gx =\sim (gx)$, so $\sim_1 \equiv B \sim$)[6]. In other words, an object falling under f, falls necessarily under g. This is near to the building of quantifiers in (Schönfinkel 1924). Like Schönfinkel, we'll write $\mathcal{U}fg$ for « g *is incompatible with* f », and we'll assume that incompatibility is a symmetrical relation.

As we intend to capture in LDO the notion of typical object of a category, or typical representant of a concept, like in (Rosch 1975), we'll see the typical-incompatibility like a 'weak' incompatibility: if g is typically-incompatible with f, though it is possible that an object x falling under f falls under g, typical f-objects don't, so such an x is an atypical f-object.

[6]Here B is Curry's composition operator: $BXYZ \longmapsto X(YZ)$; later we'll use the other following Curry's combinators: $KXY \longmapsto Y$, $SXYZ \longmapsto XZ(YZ)$, $IX \longmapsto X$ and $C^*XY \longmapsto YX$.

We'll write $\mathcal{A}fg$ for « *g is typically-incompatible with f* », $\mathbb{A}fx$ for « *x is an atypical f-object* », and $\mathbb{T}fx$ for « *x is a typical f-object* », which means that x is an $f-$object, not atypical. Unlike $\mathcal{U}$, $\mathcal{A}$ is not a symmetrical relation; with canonical example, we can't infer from « *among birds, ostriches are atypical birds* », that « *among ostriches, birds are atypical ostriches* ».

In figure 2 we show the cone of objects determinated from τf: each Δ_i is a determination chain, determining $\Delta_i \tau f$, then $\Delta_j \circ \Delta_i \tau f$... In the cone, we represent a kernel of typical objects. If the determinated object is atypical because of one of the determinations, we represent this object near the surface of the cone. Otherwise, the object is in the typical kernel.

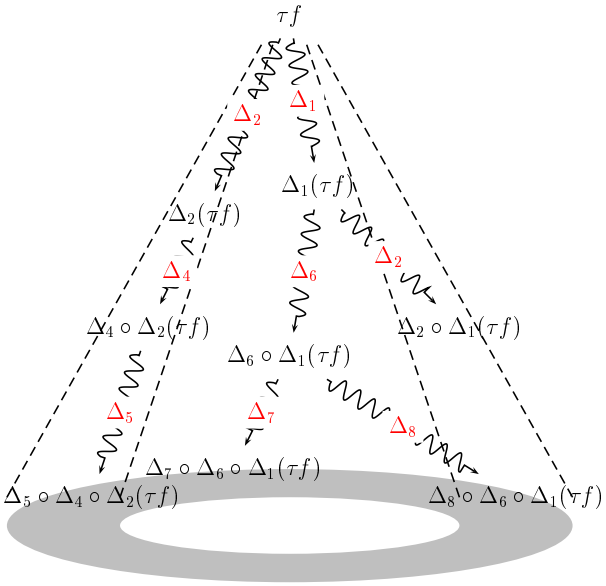


Figure 2: Typical kernel in a determination cone

In the base of the cone (the extension, *ie* determinate objects), we still see the typical kernel (in white) and the atypical part (in grey) of the extension.

In terms of TLA[7], the typical kernel is the interior of the determination cone, the atypical part is its internal thick border.

3 Quantification in LDO

Quantification in LDO is intended to give a uniform representation of various quantifiers in natural languages. The following sentences are usually represented in classical logic as shown thereafter:

[7]TLA: Théorie des lieux abstraits (Theory of abstract loci). See for example (Desclés, Gwiazdecka, and Montes-Rendon 2001).

1. « *Jean is mortal* » /BE-MORTAL/ $(Jean)$
2. « *A man is mortal* » $\exists x(/\text{BE-MAN}/\ (x) \wedge /\text{BE-MORTAL}/\ (x))$
3. « *Every man is mortal* » $\forall x(/\text{BE-MAN}/\ (x) \supset /\text{BE-MORTAL}/\ (x))$

LDO gives for them the following forms:

1. /BE-MORTAL/ $(Jean)$
2. /BE-MORTAL/ $(\Sigma^*/\text{BE-MAN}/\ )$
3. /BE-MORTAL/ $(\Pi^*/\text{BE-MAN}/\ )$

Σ^* and Π^* are linked to Curry's illative quantifiers Σ_2 and Π_2 through:

$$\Sigma_2 \equiv BC^*\Sigma^* \text{ and } \Pi_2 \equiv BC^*\Pi^*.$$

In (1), we could even write $\Delta(\tau/\text{BE-MAN}/\ )$ for $Jean$, to show that $Jean$ is determinated from the typical *man* through a determination chain Δ.

In each case, the LDO expression shows a determinated subject falling under a predicate, and meets the traditional medieval analysis, which is much closer to the language than the analysis of classical logic, or other formalisms like the so-called intensional logic.

Otherwise, from Curry's logic, LDO inheritates the ability of expressing quantification directly as a relation between concepts, without using bound variables (which don't appear in natural languages).

Here is to be noticed that quantification has the same effect than determination: it restricts the scope of a term. So quantification is to be analyzed like a particular determination, whereas other formalisms paraphrase determination with help of quantification.

4 Essence, intension

The class of concepts encompassed by f is called « *essence of* f », denoted by Ess f and characterized by: $g \in \text{Ess}\ f$ iff $\mathcal{U}(\sim_1 g)f$, or $\mathcal{U}f(\sim_1 g)$ (due to symmetry). We'll write $f \rightarrow g$ for « f *encompasses* g ». $\rightarrow$ is a partial order on $\mathcal{F}$.

But essence is not enough to capture all the meaning of a concept, especially when we consider typicality: though some birds don't fly, /BE-ABLE-TO-FLY/ is an important part of the concept /BE-BIRD/ ; we'll say « /BE-ABLE-TO-FLY/ *is in the intension of* /BE-BIRD/ ».

As essence is related to quantification by $g \in \text{Ess}\ f \supset \Pi fg$ (or $g(\Pi^* f)$), we define « *intension of* f », denoted by Intension f, based on the weakened comprehension relation (typical comprehension):

$f \overset{\mathbb{T}}{\to} g$ iff. $\Pi(\mathbb{T}f)g$ iff. $g(\Pi^*(\mathbb{T}f))$.

So: Intension $f = \{g \in \mathcal{F}; f \overset{\mathbb{T}}{\to} g\}$ and Ess $f = \{g \in \mathcal{F}; f \to g\}$.

$f \to g \supset f \overset{\mathbb{T}}{\to} g$. Hence: Ess $f \subset$ Intension f.

Like $\to$ w.r.t. $\mathcal{U}$, $\overset{\mathbb{T}}{\to}$ can be expressed with $\mathcal{A}$, so:

Intension $f = \{g \in \mathcal{F}; \mathcal{A}f(\sim_1 g)\}$

$f-$objects share all properties of Ess f, typical $f-$objects share all properties of Intension f; atypical $f-$objects are atypical because they lack of (at least) one property of the intension, but they still may have other properties of the intension.

This allows inference with inconsistent inheritance networks, like Nixon's diamond, as will be detailed in section 6.1.

One trouble is that, unlike $\to$, $\overset{\mathbb{T}}{\to}$ is not a partial order on $\mathcal{F}$. If we read $f \overset{\mathbb{T}}{\to} g$ as « *typical* $f-$*objects fall under* g », obviously $f \overset{\mathbb{T}}{\to} g$ doesn't imply that typical $f-$objects are typical $g-$objects. For example, /BE-BIRD/ is in the intension (and even in the essence) of /BE-OSTRICH/ , but an ostrich, even a typical one, is not a typical bird.

Of course we could have given Ess f and Intension f as primitives, rather than relations between concepts, but structure among concepts and coherence of the system wouldn't have appeared so clearly. Here essence and intension of different concepts are coherent because they are inferred from $\mathcal{U}$ and $\mathcal{A}$.

5 Formalization

In this section we give axioms for τ and δ operators, inference rules for LDO, and some theorems. As the formalism is based on functional calculus, we develop an applicative point of view and show results as fixed-point of some operator, when possible.

5.1 Axioms

$A\tau\delta 1$: For each concept f, there are an object τf and an operator δf.

Axiom $A\tau\delta 1$ shows clearly that in LDO an object exists as object in thought. It doesn't matter here whether a real object can be completely determinated from τf or not. The point is that τf exists in the logical system.

$A\tau\delta 2$: $(\forall f \in \mathcal{F})[(\delta f(\tau f)) = (\tau f)]$

τf is a fixed-point of δf; written in terms of Curry's combinators: $S\delta\tau f = \tau f$ so $S\delta\tau = \tau$; τ is a fixed-point of $S\delta$.

$A\tau\delta 3$: $(\forall f, g \in \mathcal{F})(\forall x \in \mathcal{O})[[(fx) = \top \wedge g \in \text{Ess } f] \Rightarrow ((\delta g)x) = x]$

An object falling under f is a fixed-point of δg, for all g in the essence of f.

$A\tau\delta 4$: $(\forall f \in \mathcal{F})[\delta f \circ \delta f = \delta f]$

Determination by a concept is *idempotent*. With applicational point of view, $WB(\delta f) = \delta f$, which leads to recognize δ as a fixed-point of $BBWB$.

$A\tau\delta 5$: $(\forall f \in \mathcal{F})(f(\tau f)) = \top$

With applicational point of view: $f(\tau f) = SI\tau f$, and the axiom consists in identifying $SI\tau$ to $K\top$, the constant-true function.

With this last axiom, τ is connected to the $\varepsilon-$function of Hilbert[8]; but Hilbert thought about extensional existence, whereas we think about intensional existence; former works on LDO had made here other choices, for example (Pascu 2001) identifies $f(\tau f)$ with Extension ${}_\tau f \neq \varnothing$ which means: $\exists x, x$ determinate and $f-$typical, such that $(fx) = \top$ (with extensional sense to existence).

5.2 Rules

LDO includes rules of Curry's combinatory logic and rules of inference for connectors of classical logic (as presented in (Gentzen 1955)). Here are specific rules for LDO:

Atypicality and inheritance

$$\frac{x = \delta g\ y \qquad y = \Delta(\tau f) \qquad \mathcal{A}fg}{\mathbb{A}fx} \qquad\qquad \frac{x = \delta g\ y \qquad \mathbb{A}fy}{\mathbb{A}fx}$$

These rules show the two ways for building an atypical $f-$object: by applying an $f-$typically-incompatible determination upon an $f-$object, or by determining an atypical $f-$object. Atypicality is inherited, and can't be lost through determination.

Typicality

$$\mathbb{T}f(\tau f) \qquad\qquad \frac{fx \qquad \sim(\mathbb{A}fx)}{\mathbb{T}fx}$$

τf is called « *typical* $f-$*object* », so at least we have to recognize it as typical! Besides, if we know that an $f-$object is not atypical, then it is typical.

[8] *Cf.* (Bernays 1935).

5.3 Some theorems

Axioms, rules (including rules of natural logic) and previous definitions allow to prove the following theorems; the next section will give a proof of theorems (3) and (4).

1. $\dfrac{h \in \text{Ess } f \qquad (f\ x)}{(h\ x)}$ — An object falling under f falls under the concepts in the essence of f.

2. $\mathcal{A} \equiv B\mathcal{U}\mathbb{T}$ — Both of these operators characterize the typical-incompatibility.

3. $\dfrac{fx \qquad \sim (\mathbb{T}fx)}{\mathbb{A}fx}$ — If we know that an $f-$object is not typical, then it is atypical; this is a lemma to prove the next theorem.

4. $\dfrac{g(\Pi^\star(\mathbb{T}f)) \qquad \sim_1 gx \qquad f\ x}{\mathbb{A}fx}$

   If typical $f-$objects fall under g, and x is an $f-$object which doesn't fall under g, then x is atypical (as an $f-$object).

5. Étendue f is the smallest class including τf and stable through all $f-$compatible determinations. This stability means, it is a fixed-point for $X \longmapsto \bigcup_{x \in X, g \in \mathcal{F}} \{\delta g\ x\}$

5.4 Consistency

The main result about consistency in LDO, is that no contradiction may occur through deduction rules about typicality.

Such a contradiction would appear as: $\mathbb{T}fx \wedge \mathbb{A}fx$. But none of the two rules introducing $\mathbb{T}fx$ may lead to this: one supposes that $\sim \mathbb{A}fx$, the other one supposes that $x = \tau f$, so that no determination has made x be an atypical $f-$object.

Another point to be stressed is that incoherent data about $\mathcal{U}$ would not lead to contradictions but would prevent the existence of some objects.

6 Applications

LDO is intended to study determinations in a logical point of view and in natural languages. We show in this section some inferences, with the proof of two theorems listed above, and with inference on typicality. Then we explain how LDO is helpful for representing language.

6.1 Inferences

Proof of theorems 3 and 4.

Here are proof-trees of theorems 3 (at left) and 4 (at right) from the previous section.

$$\dfrac{f x \quad \dfrac{f x \quad \sim(\mathbb{A} f x)}{\mathbb{T} f x} \quad \sim(\mathbb{T} f x)}{\dfrac{\bot}{f x;\sim(\mathbb{T} f x);\sim(\mathbb{A} f x) \vdash \bot}} \quad \sim(\mathbb{T} f x)}{\mathbb{A} f x} \qquad \dfrac{\dfrac{\dfrac{g(\Pi^\star(\mathbb{T} f))}{\Pi_2(\mathbb{T} f) g} \quad \dfrac{\sim_1 g x}{\sim(g x)}}{(\mathbb{T} f x) \supset (g x)}}{\sim(\mathbb{T} f x)} \quad f\ x}{\mathbb{A} f x}$$

Inferring on typicality

The canonical example of atypicality: « *typical birds fly, ostriches are birds but an ostrich doesn't fly* » is solved by theorem 4, with $f = /\text{BE-BIRD}/$, $g = /\text{BE-ABLE-TO-FLY}/$, and $x = \tau/\text{BE-OSTRICH}/$. So the theorem tells: $\mathbb{A}/\text{BE-BIRD}/\ (\tau/\text{BE-OSTRICH}/\ )$, which is to be read « *An ostrich is an atypical bird* ».

Let's go on with a transparent adaptation of Nixon's diamond: « *(a) Bush is methodist – (b) methodists are pacifist – (c) Bush is republican – (d) republicans are not pacifist.* » From (a) and (b), Bush is pacifist, from (c) and (d) he isn't. To solve the inconsistence, we have to formalize at least one of (b) or (d) with $\xrightarrow{\mathbb{T}}$ rather than with $\rightarrow$.

For example, $/\text{BE-METHODIST}/ \xrightarrow{\mathbb{T}} /\text{BE-PACIFIST}/$, which could also have been written $/\text{BE-PACIFIST}/\ (\Pi^*(\mathbb{T}/\text{BE-METHODIST}/\ ))$: typical methodists are pacifist; so, if Bush is not pacifist (doesn't matter here whether all or only typical republicans are not pacifist), he is an atypical methodist. And if Bush were pacifist, we would conclude that he is an atypical republican.

6.2 LDO and language

Lexicon can be structured with use of determination: for example a definition such as « *a manor is a big luxurious house* » could be represented by:

$\tau/\text{BE-MANOR}/\ := \delta/\text{BE-BIG}/\ (\delta/\text{BE-LUXURIOUS}/\ (\tau/\text{BE-HOUSE}/\ ))$

$\mathcal{U}$ and $\mathcal{A}$ relations (or their complement) allow to represent compatibility and typicality between concepts. A structured lexicon using LDO should give these relations.

Words refer to objects, and by saying « *a car* », we're speaking about an object, even if we don't know which one, but we're not speaking about the class of cars. Natural language deals with more or less determinated objects, and determinates them along the discourse. LDO gives a way to represent that part of semantics.

In (Cardot 2003), we showed how determination is involved in semantic representation of verbs, mainly for state clauses and event clauses. Our aim in forthcoming development is to represent how new determinations are given to an object along a text.

Moreover, LDO allows to represent *identification* of an object to another. For example, in « *John is (like) a lion* », the speaker selects in Intension /BE-LION/ the part he thinks relevant, and asserts this part about John.

Conclusion

Determination is a fundamental operation in language, by which objects are built intensionally, more and more specified along the utterance.

LDO, built with an intensional point of view, deals with this operation, by which objects, rather than classes, are specified. Formalization, based on $\mathcal{U}$ and $\mathcal{A}$ relations of (typical-)incompatibility, allows inference on typicality and representation of objects along the discourse. These incompatibility relations give access to essence and intension of concepts.

LDO contributes to the study of categorization, and can be included in a general formalism for representing language, such as GA&C.

References

Arnauld, A. and P. Nicole (1662). *La logique ou l'art de penser* (1993 ed.). Paris: Vrin.

Bernays, P. (1935). *Hilberts Untersuchungen über die Grundlagen der Arithmetik*, pp. 196–216. dans (Hilbert 1935).

Cardot, J. (2003). Logique de la détermination d'objets: un formalisme pour la représentation des objets du discours. In *Actas del VIII^{vo} symposio internacional de comunicación social*, Santiago de Cuba, pp. 115–120.

Culioli, A. (1999). *Pour une linguistique de l'énonciation – Tome 3: Domaine notionnel*. Paris: Ophrys.

Curry, H. B. and R. Feys (1958). *Combinatory Logic*, Volume I. North Holland.

Desclés, J.-P. (1986). Implication entre concepts: la notion de typicalité. *Travaux de linguistique et de littérature* (XXIV).

Desclés, J.-P. (1990). *Langages applicatifs, Langues naturelles et Cognition*. Paris: Hermès.

Desclés, J.-P., E. Gwiazdecka, and A. Montes-Rendon (2001). Towards invariant meanings of spatial preposition and preverbs. In *Workshop on Spatial and Temporal Information Processing, ACL-2001*, Toulouse.

Frege, G. (1879). *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle. Version française par Corine Besson, *l'Idéographie*, Vrin, Paris, 1999.

Gentzen, G. (1955). *Recherches sur la déduction logique - Untersuchungen über das logische Schließen*. Paris: Presses Universitaires de France.

Hilbert, D. (1935). *Gesammelte Abhandlungen – tome 3* (1970 ed.). Berlin: Springer Verlag.

Pascu, A. (2001). *Logique de Détermination d'Objets: concepts de base et mathématisation en vue d'une modélisation objet*. Ph. D. thesis, Université Paris IV.

Rosch, E. (1975). Cognitive representations of semantic categories. *Journal of Exprimental Psychology* (104), 192–233.

Schönfinkel, M. (1924). Über die Bausteine der mathematischen Logik. *Mathematische Analen 92*, 305–315.

Non-Redundant Scope Disambiguation in Underspecified Semantics

Rui Pedro Chaves
Centro de Linguística da Universidade de Lisboa (CLUL), Av. Gama Pinto, n°2, Lisboa, Portugal
rui.chaves@clul.ul.pt

Abstract. This paper presents a strategy that aims to efficiently avoid the generation of logically equivalent formulas that arise in scope processing applications. Constraint based semantic underspecification formalisms may be extended to include an additional scoping restriction that constrains the set of possible disambiguations, straightforwardly avoiding the generation of redundant quantifier scopings. Such a restriction, in principle valid to any logic, is formalized within Hole Semantics (Bos 1996), a general semantic underspecification framework.

1 Introduction

Semantic Underspecification frameworks (such as QLF (Alshawi and Crouch 1992), UDRSs (Reyle 1993), UMRS (Egg and Lebeth 1995), Hole Semantics (Bos 1996) and CLLS (Egg et al. 1998)) are able to cope efficiently with the combinatorial explosion of highly ambiguous Natural Language phenomena such as scope ambiguity.

Classic approaches to scope processing (such as Cooper (1983), Hobbs and Shieber (1987) and Keller (1988)) are inefficient in the sense that an exponential number of formulas is always generated (worse case entails generating $n!$ formulas, for n quantifiers). In underspecified semantics, the set of possible readings is described via a unique, compact partial representation which may be reduced in a straightforward fashion, simply by the incremental specification of additional constraints (e.g. from prosody, discourse context or world knowledge). Furthermore, underspecification is able to deal with other scope bearing constituents other than quantification.

However, the existing methods of scope processing typically overgenerate in the sense that some or even all generated formulas may be logically equivalent. We present a new scoping restriction capable of reducing the set of scopal disambiguations in underspecified semantics, avoiding the generation of logical redundancies that result from quantifier scope (a subset of formulas with the same *prenex normal form*) and therefore greatly reducing

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

the need for theorem provers. Section 2 briefly reviews the existing methods for the elimination of redundant formulas. Section 3 formalizes a scope disambiguation restriction within Hole Semantics and applies it to Discourse Representation Theory (Kamp and Reyle 1993) formulas. Finally, practical implementation results and some extensions are discussed.

2 Logical Overgeneration: Redundant Scopings

It is well known that the relative orderings of identical first order quantifiers do not result in distinct truth conditions. For instance, sentence (1) only has a single reading, yet two equivalent formulas are typically generated:

(1) Every boy saw every girl.
(1a) $(\forall x)(boy(x) \rightarrow (\forall y)(girl(y) \rightarrow saw(x, y)))$
(1b) $(\forall y)(girl(y) \rightarrow (\forall x)(boy(x) \rightarrow saw(x, y)))$

A more complex case is visible in sentence (2), which typically receives up to 60 equivalent formulas[1] but only has a single reading:

(2) A boy that sings in a choir gave a flower with a velvet lace to a girl.

The formulas generated for each of the examples above have the same *prenex normal form*[2] and hence are equivalent. We are presently concerned with equivalences that result from the Laws of Quantifier Movement (pulling nested quantifiers out of formulas: $(\varphi \rightarrow (\forall x)\psi(x)) \Leftrightarrow (\forall x)(\psi \rightarrow \psi(x))$ and $(\varphi \rightarrow (\forall x)\psi(x)) \Leftrightarrow (\forall x)(\psi \rightarrow \psi(x))$ where x is not free in φ and from the Laws of Quantifier Independence (interchanging quantifiers: $(\forall x)(\forall y)\varphi(x, y) \Leftrightarrow (\forall y)(\forall x)\varphi(x, y)$ and $(\exists x)(\exists y)\varphi(x, y) \Leftrightarrow (\exists y)(\exists x)\varphi(x, y)$).

Much more subtle cases involve (possibly many) different sets of logically redundant scopings, as sentence (3) below illustrates. Here, a distinct scopal operator (negation) induces partial non-redundancy: 6 possible permutations but only 4 logically distinct readings.

(3) A cat doesn't like a dog.
(3a) $(\exists y)(dog(y) \wedge (\exists x)(cat(x) \wedge \neg like(x, y)))$
(3b) $(\exists y)(dog(y) \wedge \neg(\exists x)(cat(x) \wedge like(x, y)))$
(3c) $\neg(\exists y)(dog(y) \wedge (\exists x)(cat(x) \wedge like(x, y)))$
(3d) $(\exists x)(cat(x) \wedge (\exists y)(dog(y) \wedge \neg like(x, y)))$
(3e) $(\exists x)(cat(x) \wedge \neg(\exists y)(dog(y) \wedge like(x, y)))$
(3f) $\neg(\exists x)(cat(x) \wedge (\exists y)(dog(y) \wedge like(x, y)))$

[1] The total number of scopings is not factorial (i.e. $5! = 120$) because nested NPs cannot have arbitrary semantic scope beyond their syntactical local domain, e.g. '*a choir*' may not simultaneously outscope '*a velvet lace*' and be outscoped by '*a flower*'.

[2] $(\mathrm{Q}_0\ \mathrm{x}_0) \ldots (\mathrm{Q}_{n-1}\ \mathrm{x}_{n-1})\psi$ where Q_i $(0 < i < n)$ is a quantifier and ψ is quantifier-free.

Clearly, formula (3a) is equivalent to (3d) and formula (3c) is equivalent to (3f). One solution to this problem is to use a theorem prover with a search time limit to help decide the equivalence of each pair of formulas generated: formulas P and Q are equivalent *iff* (P⇔Q) is provable. For n generated formulas a theorem prover requires $n(n-1)$ proofs in the worse case (i.e. equivalent formulas do not exist), for pairwise choices of P and Q.

Vestre (1991) presents a method that avoids quantifier scope redundancy by limiting the selection of determiners in an enumerative algorithm for scope processing that exhaustively generates and evaluates all possible scopings for a given quantifier. In contrast, we present a method that does not search an exponential number of scopings. Rather, the proposed strategy strictly generates irredundant quantifier scopings by keeping track of the scope-bearing constituent undergoing disambiguation and prohibiting certain disambiguation patterns. Because this method is formalized within a semantic underspecification framework it is able to consider partially disambiguated structures and the interaction of other scope-bearing structures besides quantification, such as negation, modality and indirect discourse.

Gabsdil and Striegnitz (1999) proposes and implements a general method that orders all the formulas outputted by a scoping algorithm into a graph structure in which logically equivalent readings are collapsed into a single structure. This strategy requires the exhaustive enumeration of possible formulas, the very problem that underspecification originally aimed to solve. The next section formalizes a method, independent from the logic of choice, that aborts (sets of) disambiguations that describe redundant scopings.

3 Non-Redundant Scopings in Hole Semantics

Hole Semantics (Bos 1996) is a general underspecification framework where there is a clear distinction between the underspecification metalanguage and the object language. In this framework, scopal ambiguity is represented via partial subordination constraints in an upper-semilattice. What follows is a brief excursion to Hole Semantics.

An Underspecified Representation (UR) is a triple ⟨H,L,C⟩ where H is a set of metavariables (*holes*) over formulas; L is a set of labelled formulas; and C is a set of subordination constraints expressed via a relation "$\leq$ " that establishes a partial order (i.e. is reflexive, transitive and antisymmetric) over labels and metavariables. This partial order is also an upper-semilattice given that a special element, the *supremum*, subsumes any pair of elements in the structure. An admissible disambiguation (henceforth a *plugging*) is a bijection from holes to labelled formulas that does not violate the set of constraints, operating recursively from a special *top* hole h_0. When a hole is plugged (i.e. P(h)=l), the hole variable is substituted by the formula identified by the correspondent label (e.g. "l:φ"). Bos (1996) shows that both

Dynamic Predicate Logic and Discourse Representation Structures (DRSs (Kamp and Reyle 1993)) can be used as object language in Hole Semantics.

We now formalize a general method within Hole Semantics that consists in a restriction that acts to constrain the set of *possible pluggings* (i.e. disambiguations that do not violate any subordination constraints), and is therefore in principle applicable to any logic of choice. In essence, pluggings shall be associated to a special unrestricted scope operator, initialised as "$\top$", and as the disambiguation proceeds this operator may be *updated*, *percolated* or forced to *abort* ("$\bot$") the plugging (in the latter case, plugging a given hole with a given formula would result in a non-empty set of formulas with logically equivalent scopings, if disambiguation is completed).

It must be expressly noted that we assume the numerical indexes associated to each label are unique and lexically attributed according to *surface order*. The *Distinct Scope* restriction will be endowed with a 'short-term memory' of the disambiguation process and shall impose decreasing order on the label indexes between formulas that may induce logical redundancy. More generally, scope disambiguation in constraint based underspecification formalisms can be further restricted in order to efficiently avoid the symmetric scoping counterpart between specific operators capable of inducing logical redundancy.

3.1 Metalanguage Definitions

Firstly, we shall define the set of metalanguage formula schemata that can result in redundant disambiguations. Secondly, we provide a basis for detecting formulas with identical outermost scopal operators and finally, we formalize a general *Distinct Scope* function and the restrictions therein.

Definition 1: Scopal Schemata
Let S be the set of formula schemata $\{\mathrm{Op}(\psi)\colon\ \mathrm{Op}(\psi) \in \mathrm{U}\}$, defined in a given metalanguage U, under the scope of an operator Op that can potentially induce redundant disambiguations. For instance, in an underspecified account of predicate logic such as Bos (1996), one would have $\mathrm{S} = \{(\exists x)(k), (\forall y)(k)\}$; where k is either a metavariable or a metalanguage formula.

Definition 2: Scopal Operator Equivalence
Formulas φ and ψ have an *equivalent syntactical scope* ($\varphi \equiv \psi$) *iff* φ and ψ are of the form $\mathrm{Op}_i(\varphi')$ and $\mathrm{Op}_j(\psi')$ respectively, where Op_i and Op_j are identical scope-bearing operators. E.g. : '$(\exists x)(\alpha)$' $\equiv$ '$(\exists y)(\beta)$' $\not\equiv$ '$(\forall \mathrm{z})(\tau)$'.

Definition 3: Distinct Scope Restriction
Let D be the function which is defined as follows $D\colon \Gamma \times \Gamma \rightarrow \Gamma$, where Γ is a set of $\phi ::= (l : \varphi) | \top | \bot$. This function establishes a mapping between a ordered pair of *main scope operators* (either some labelled metaformula 'l:φ',

or the special symbols "$\top$" (*verum*) and "$\bot$" ($falsum$)) and a new *main scope operator*. For n equivalent readings, the ordering constraints enforced by D are able to interrupt pluggings linearly, *at most* n times (i.e. depending on the plugging strategy). Below, φ and ψ are metalanguage formulas:

$$D(\top, (l_i : \varphi)) = (l_i : \varphi) \qquad (initial\ scope\ domain\ update)$$

$$D((l_i : \varphi), (l_j : \psi)) = \begin{cases} (l_i : \varphi) & if\ \ \psi \notin S & (percolate) \\ (l_j : \psi) & if\ \ \psi \in S\ \wedge\ \varphi \not\equiv \psi & (update) \\ (l_j : \psi) & if\ \ \varphi, \psi \in S\ \wedge\ \varphi \equiv \psi\ \wedge\ i > j & (update) \\ \bot & otherwise & (abort) \end{cases}$$

The crucial step takes place when both syntactically equivalent formulas are in set S and have *decreasing* labelling indexes. Function D aborts this specific plug, not allowing orderings where the numerical indexes increase, thus eliminating the full permutation of logically identical quantificational operators. For instance "every$_1$ student$_2$ read$_3$ every$_4$ poem$_5$ to$_6$ every$_7$ girl$_8$" would only have the disambiguation with the scoping order: 7>4>1.

Next, the main scope operator slot and D must be embedded into the plugging procedure itself. To illustrate this we shall adopt *unplugged* DRSs (Bos 1996) as a description language, extended in order to explicitly consider duplex conditions and a basic account of indirect discourse.

3.2 Extended Plugging Procedure for DRTU

Typical disambiguation algorithms for DRT formulas will also be able to overgenerate by building identical DRSs, given different scoping choices:

Syntax of DRTU formulas (adapted from Bos (1996))

1. If h_i and h_j are holes, p is a propositional discourse marker and Q is a generalized quantifier, then $\langle\{\},\{h_i \Rightarrow h_j\}\rangle$, $\langle\{\},\{\neg h_i\}\rangle$, $\langle\{\},\{h_i \vee h_j\}\rangle$, and $\langle\{p\},\{p : h\}\rangle$ are DRTU formulas.

2. If h is a hole, x is a discourse marker and $k_1 \ldots k_n$ are holes or DRTU formulas then $\otimes\{k_1, \ldots, k_n\}$ and $\langle\{\},\{Q_x(k, h)\}\rangle$ are DRTU formulas.

3. If $x_1 \ldots x_n$ are discourse markers and P is a symbol for an n-place predicate, then $\langle\{x\},\{\}\rangle$ and $\langle\{\},\{P(x_1, \ldots, x_n)\}\rangle$ are DRTU formulas.

4. Nothing else is a DRTU formula.

The *merge* operator "$\otimes$" denotes the union of DRSs (Universes and Conditions), e.g. $\otimes\{\langle U_1, C_1\rangle, \ldots, \langle U_n, C_n\rangle\} = \langle U_1 \cup \ldots \cup U_n,\ C_1 \cup \ldots \cup C_n\rangle$. By definition, the label of the formula subordinates every *hole variable* introduced by that same formula. In these terms, a single underspecified representation is assigned to an ambiguous sentence such as (4):

(4) A boy didn't read a book.

$$(5)\quad \mathrm{UR} = \left\langle \left\{ \begin{array}{l} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{array} \right\}, \quad \left\{ \begin{array}{l} l_1 : \otimes\{\langle\{x\},\{\}\rangle, h_1, h_2\} \\ l_2 : \langle\{\},\{boy(x)\}\rangle \\ l_3 : \langle\{\},\{\neg h_3\}\rangle \\ l_4 : \langle\{\},\{read(x,y)\}\rangle \\ l_5 : \otimes\{\langle\{y\},\{\}\rangle, h_4, h_5\} \\ l_6 : \langle\{\},\{book(y)\}\rangle \end{array} \right\}, \quad \left\{ \begin{array}{l} l_1 \leq h_0 \\ l_3 \leq h_0 \\ l_5 \leq h_0 \\ l_2 \leq h_1 \\ h_1 \leq l_2 \\ l_4 \leq h_2 \\ l_4 \leq h_3 \\ l_6 \leq h_4 \\ h_4 \leq l_6 \\ l_4 \leq h_5 \end{array} \right\} \right\rangle$$

Graphically, subordination constraints are represented by arrows:

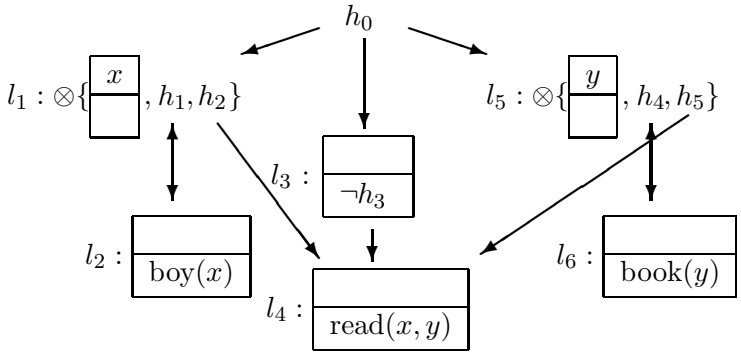


Figure 1: Graphical representation of UR (5).

Note that the scope between the metavariables in the subject NP indefinite (h_2), negation (h_3) and complement NP indefinite (h_5) is unspecified, and therefore a total of 6 pluggings are possible:

$$\begin{array}{l} \mathrm{P}_1 : \{h_0 = l_1, h_1 = l_2, h_2 = l_5, h_4 = l_6, h_5 = l_3, h_3 = l_4\} \\ \mathrm{P}_2 : \{h_0 = l_1, h_1 = l_2, h_2 = l_3, h_3 = l_5, h_4 = l_6, h_5 = l_4\} \\ \mathrm{P}_3 : \{h_0 = l_3, h_3 = l_1, h_1 = l_2, h_2 = l_5, h_4 = l_6, h_5 = l_4\} \\ \mathrm{P}_4 : \{h_0 = l_3, h_3 = l_5, h_4 = l_6, h_5 = l_1, h_1 = l_2, h_2 = l_4\} \\ \mathrm{P}_5 : \{h_0 = l_5, h_4 = l_6, h_5 = l_3, h_3 = l_1, h_1 = l_2, h_2 = l_4\} \\ \mathrm{P}_6 : \{h_0 = l_5, h_4 = l_6, h_5 = l_1, h_1 = l_2, h_2 = l_3, h_3 = l_4\} \end{array}$$

Note also that P_1 and P_6 are redundant (i.e yield the same DRS), and so are P_3 and P_4. In order to avoid this overgeneration, the labels of the formulas lexically introduced will be used to limit the range of possible pluggings as outlined by function D. An extended plugging algorithm that incorporates this function is defined below as a partial function $Plug$, restricting the set of possible disambiguations. Firstly, assume $\mathrm{S} = \{\otimes\{k, h, h\}, \langle\{\},\{Q_x(k, h)\}\rangle\}$ for the schemata of DRTU formulas, where Q is a given generalized quantifier such as '*every*' or 'few' (but not like '*most*' or '*many*', for the consecutive

interchanging of such quantifiers does not always preserve the same truth conditions; see McCawley (1981: 53-54) for a brief discussion).

Definition 4: Extended Plugging Procedure
Plugging corresponds to the function $Plug : (H \cup L) \times \Gamma \times \text{SR} \rightarrow \Sigma$, where H is the set of holes and L the set of labelled metalanguage formulas, Γ is the set of *main scope operators* ϕ as defined before, SR is the set of *solved* Underspecified Representations $\langle H, L, C \rangle$, and finally, Σ is the set of disambiguated object-language formulas.

The set of constraints C in a solved Underspecified Representation explicitly describes a solution to the original underspecified structure. In other words, the constraints in C describe a tree of the formulas in L (i.e. subordinations $l \leq h$ and $l \leq h'$ where $h \neq h'$ do not exist). Althaus et al. (2003) presents an efficient method for enumerating the described solutions as *forests* in constraint graphs (a general framework for the partial description of trees) and these results can be used in several other underspecification formalisms, including Hole Semantics (Koller et al. 2003).

The initial call to this function is $Plug(h_0, \top, \langle H, L, C \rangle)$ where H is a set of holes, L is a set of labelled unplugged formulas and C a set of solved subordination constraints. The return shall be a disambiguated formula in the object-language. The crucial step of avoiding spurious scopings takes place when a consistent plug $\text{P}(h) = l$ occurs and function D is applied:

$$Plug(h, \phi, \langle\, H \cup \{h\}\,,\, L \cup \{l : \psi\}\,,\, C \cup \{(l \leq h)\}\,\rangle) =$$
$$\quad Plug(\psi, D(\phi,\, l : \psi), \langle H, L, C \rangle)$$

Above, hole h is identified with a given outscoped formula ψ. Next, a recursive call attempts to plug formula ψ *iff* $D(\phi, l : \psi) \neq \perp$. More explicitly, $Plug(\psi,\, \perp,\, \langle H, L, C \rangle)$ always fails. Note that D is compatible with more straightforward, though less efficient, disambiguation strategies: in Blackburn and Bos (1999) the original set C of the UR is updated after each plug $\text{P}(h) = l$ (i.e. every occurrence of h in C is replaced by l) and is checked for consistency. Similarly, D applies to the next plug and the remaining cases are identical to the presented below (Chaves 2002:325-330).

In duplex conditions the *restrictor* hole remains unconstrained ("$\top$") while the *scope* hole does not, because nested generalized quantifiers within a restrictor yield distinct readings, as illustrated below in sentence (6):

$$Plug(\langle\{\}, \{Q_x(k, h)\}\rangle, \phi, \langle H, L, C \rangle) =$$
$$\quad \langle\{\}, \{\, Q_x(\, Plug(k, \top, \langle H, L, C \rangle),\, Plug(h, \phi, \langle H, L, C \rangle)\,\}\rangle$$

(6) Every representative of every company protested.

At least two readings are available for (6): companies with possibly different representatives protested; representatives that simultaneously represent all companies protested. Note that if D imposed an *increasing* order to the numerical indexes, nested quantifiers would not get wide scope readings over the main quantifier (perhaps indexes might be able to reflect scopal preferences via an underspecified partial order). Conditionals are similar:

$$Plug(\langle\{\}, \{\, h_i \Rightarrow h_j \,\}\rangle, \phi, \langle H, L, C\rangle) = \\ \langle\{\}, \{\, Plug(h_i, \top, \langle H, L, C\rangle) \Rightarrow Plug(h_j, \phi, \langle H, L, C\rangle) \,\}\rangle$$

$$Plug(\otimes\{k_1, \ldots, k_n\}, \phi, \langle H, L, C\rangle) = \\ \otimes\{Plug(k_1, \phi, \langle H, L, C\rangle), \ldots, Plug(k_n, \phi, \langle H, L, C\rangle)\}$$

Indirect discourse, disjunction, negation (as well as modal operators) and n-place predicates do not induce *scopal* logical redundancy, and have unrestricted ("$\top$") pluggings (where α is a propositional discourse referent):

$$Plug(\langle\{\alpha\}, \{\alpha : h\}\rangle, \phi, \langle H, L, C\rangle) = \langle\{\alpha\}, \{\alpha : (Plug(h, \top, \langle H, L, C\rangle))\}\rangle$$

$$Plug(\langle\{\}, \{\, h_i \vee h_j\}\rangle, \phi, \langle H, L, C\rangle) = \\ \langle\{\}, \{Plug(h_i, \top, \langle H, L, C\rangle) \vee Plug(h_j, \top, \langle H, L, C\rangle)\}\rangle$$

$$Plug(\langle\{\}, \{\neg h\}\rangle, \phi, \langle H, L, C\rangle) = \langle\{\}, \{\neg\, Plug(h, \top, \langle H, L, C\rangle) \,\}\rangle$$

$$Plug(\langle\{\}, \{R(x_1, \ldots, x_n)\}\rangle, \phi, \langle H, L, C\rangle) = \langle\{\}, \{R(x_1, \ldots, x_n)\}\rangle$$

Take for instance sentence (7) and the corresponding UR depicted in (8) below, which has 6 distinct readings but a total of 18 possible pluggings:

(7) A girl that a teacher mentioned didn't read a book.

(8)
$$\left\langle \left\{ \begin{array}{l} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{array} \right\}, \left\{ \begin{array}{l} l_1 : \otimes\{\langle\{x\}, \{\}\rangle, h_1, h_2\} \\ l_2 : \langle\{\}, \{girl(x)\}\rangle \\ l_3 : \otimes\{h_3, h_4\} \\ l_4 : \otimes\{\langle\{y\}, \{\}\rangle, h_5, h_6\} \\ l_5 : \langle\{\}, \{teacher(y)\}\rangle \\ l_6 : \langle\{\}, \{mentioned(y, x)\}\rangle \\ l_7 : \langle\{\}, \{\neg h_7\}\rangle \\ l_8 : \langle\{\}, \{read(x, z)\}\rangle \\ l_9 : \otimes\{\langle\{z\}, \{\}\rangle, h_8, h_9\} \\ l_{10} : \langle\{\}, \{book(z)\}\rangle \end{array} \right\}, \left\{ \begin{array}{l} l_1 \leq h_0 \\ l_4 \leq h_0 \\ l_7 \leq h_0 \\ l_9 \leq h_0 \\ l_3 \leq h_1 \\ h_1 \leq l_3 \\ l_2 \leq h_3 \\ h_3 \leq l_2 \\ l_6 \leq h_4 \\ l_8 \leq h_2 \\ l_5 \leq h_5 \\ h_5 \leq l_5 \\ l_6 \leq h_6 \\ l_{10} \leq h_8 \\ h_8 \leq l_{10} \\ l_8 \leq h_9 \end{array} \right\} \right\rangle$$

The 6 available readings are obtained through the following pluggings:

$\{h_0{=}l_4, h_5 = l_5, h_6 = l_1, h_1 = l_3, h_3 = l_2, h_4 = l_6, h_2 = l_7, h_7 = l_9, h_8 = l_{10}, h_9 = l_8\}$
(8a) $\langle\{y, x\}, \{teacher(y), girl(x), mentioned(y, x), \neg\langle\{z\}, \{book(z), read(x, z)\}\rangle\}\rangle$

$\{h_0{=}l_4, h_5 = l_5, h_6 = l_7, h_7 = l_9, h_8 = l_{10}, h_9 = l_1, h_1 = l_3, h_3 = l_2, h_4 = l_6, h_2 = l_8\}$
(8b) $\langle\{y\}, \{teacher(y), \neg\langle\{z, x\}, \{book(z), girl(x), mentioned(y, x), read(x, z)\}\rangle\}\rangle$

$\{h_0{=}l_9, h_8 = l_{10}, h_9 = l_4, h_5 = l_5, h_6 = l_1, h_1 = l_3, h_3 = l_2, h_4 = l_6, h_2 = l_7, h_7 = l_8\}$
(8c) $\langle\{z, y, x\}, \{book(z), teacher(y), girl(x), mentioned(y, x), \neg\langle\{\}, \{read(x, z)\}\rangle\}\rangle$

$\{h_0{=}l_9, h_8 = l_{10}, h_9 = l_4, h_5 = l_5, h_6 = l_7, h_7 = l_1, h_3 = l_3, h_3 = l_2, h_4 = l_6, h_2 = l_8\}$
(8d) $\langle\{z, y\}, \{book(z), teacher(y), \neg\langle\{x\}, \{girl(x), mentioned(y, x), read(x, z)\}\rangle\}\rangle$

$\{h_0{=}l_9, h_8 = l_{10}, h_9 = l_7, h_7 = l_4, h_5 = l_5, h_6 = l_1, h_1 = l_3, h_3 = l_2, h_4 = l_6, h_2 = l_8\}$
(8e) $\langle\{z\}, \{book(z), \neg\langle\{y, x\}, \{teacher(y), girl(x), mentioned(y, x), read(x, z)\}\rangle\}\rangle$

$\{h_0{=}l_7, h_7 = l_9, h_8 = l_{10}, h_9 = l_4, h_5 = l_5, h_6 = l_1, h_1 = l_3, h_3 = l_2, h_4 = l_6, h_2 = l_8\}$
(8f) $\langle\{\}, \{\neg\langle\{z, y, x\}, \{book(z), teacher(y), girl(x), mentioned(y, x), read(x, z)\}\rangle\}\rangle$

The remaining 12 pluggings are aborted by the ordering constraints in D. For instance, plug $\mathrm{P}(h_6) = l_9$ visible below (where $\otimes\{\langle\{z\}, \{\}\rangle, h_8, h_9\}$ is identified with h_6 in $\otimes\{\langle\{y\}, \{\}\rangle, h_5, h_6\}$) is unsuccessful because $4 \not> 9$:

$\{h_0 = l_4,\ h_5 = l_5,\ h_6 = l_9\}$ (*plugging aborted*)

$D(\top,\ \ l_4{:}\ \otimes\{\langle\{y\}, \{\}\rangle, h_5, h_6\}) = l_4{:}\ \otimes\{\langle\{y\}, \{\}\rangle, h_5, h_6\}$
$D(l_4{:}\otimes\{\langle\{y\}, \{\}\rangle, h_5, h_6\},\ \ l_5{:}\ \langle\{\}, \{teacher(y)\}\rangle) = l_4{:}\ \otimes\{\langle\{y\}, \{\}\rangle, h_5, h_6\}$
$D(l_4{:}\otimes\{\langle\{y\}, \{\}\rangle, h_5, h_6\},\ \ l_9{:}\ \otimes\{\langle\{z\}, \{\}\rangle, h_8, h_9\}) = \bot$

If continued, this plugging would be equivalent to either (8c) or (8d) above:

$$\{h_0{=}l_4, h_5{=}l_5, \mathbf{h_6 = l_9},\ h_8{=}l_{10},\ \begin{array}{l}\nearrow h_9{=}\ l_1, h_1{=}\ l_3, h_3{=}\ l_2, h_4 = l_6, h_2 = l_7, h_7 = l_8\}\\ \searrow h_9{=}\ l_7, h_7{=}\ l_1, h_1{=}\ l_3, h_3 = l_2, h_4 = l_6, h_2 = l_8\}\end{array}$$

Similarly for (8a) and (8c), $\{h_0 = l_1,\ h_1 = l_3,\ h_3 = l_2,\ h_4 = l_4\}$ is aborted:

$D(\top,\ \ l_1{:}\ \otimes\{\langle\{x\}, \{\}\rangle, h_1, h_2\}) = l_1{:}\ \otimes\{\langle\{x\}, \{\}\rangle, h_1, h_2\}$
$D(l_1{:}\ \otimes\{\langle\{x\}, \{\}\rangle, h_1, h_2\},\ \ l_3{:}\ \otimes\{h_3, h_4\}) = l_1{:}\ \otimes\{\langle\{x\}, \{\}\rangle, h_1, h_2\}$
$D(l_1{:}\ \otimes\{\langle\{x\}, \{\}\rangle, h_1, h_2\},\ \ l_2{:}\ \langle\{\}, \{girl(x)\}\rangle) = l_1{:}\ \otimes\{\langle\{x\}, \{\}\rangle, h_1, h_2\}$
$D(l_1{:}\ \otimes\{\langle\{x\}, \{\}\rangle, h_1, h_2\},\ \ l_4{:}\ \otimes\{\langle\{y\}, \{\}\rangle, h_5, h_6\}) = \bot.$

$$\{h_0{=}\ l_1, h_1{=}\ l_3, h_3{=}\ l_2, \mathbf{h_4{=}l_4},\ h_5{=}\ l_5, h_6{=}\ l_6,\ \begin{array}{l}\nearrow h_2{=}\ l_7, h_7{=}\ l_9, h_8{=}\ l_{10}, h_9{=}\ l_8\}\\ \searrow h_2{=}\ l_9, h_8{=}\ l_{10}, h_9{=}\ l_7, h_7{=}\ l_8\}\end{array}$$

In sum, the UR in (8) has a total of 18 disambiguations out of which 12 correspond to equivalent readings. The *Distinct Scope* restriction licenses only a subset of 6 pluggings, all corresponding to logically distinct scopings.

A Prolog implementation indicates that D speeds up the disambiguation of highly redundant URs by 10% to 25% (e.g. 1048 equivalent pluggings in 7.2 secs. *vs* a unique reading in 5.8 secs. total on a PIII 866MHz 256Mgs) and that no noticeable computational delay is induced by irredundant URs (CGI at `http://www.clul.ul.pt/clg/scope.html`). Although Chaves (2002) uses a less efficient plugging method overall, maximum speed up is of 91%.

3.3 Double Negation

The wide/narrow scopings of the indefinite "a woman" in (9a) and (9b) respectively, are equivalent because of double negation (Corblin (1995, 1996)):

(9) It is not true that John did not see a woman.
(9a) $\langle\{x,y\},\{John(x),woman(y),\neg\langle\{\},\{\neg\langle\{\},\{see(x,y)\}\rangle\}\rangle\}\rangle$
(9b) $\langle\{x\},\{John(x),\neg\langle\{\},\{\neg\langle\{y\},\{woman(y),see(x,y)\}\rangle\}\rangle\}\rangle$

This is also true of two of the readings of (10)(Gabsdil and Striegnitz (1999)):

(10) No criminal does not love a woman.
(10a) $\langle\{y\},\{woman(y),\neg\,\langle\{\},\{\neg\langle\{x\},\{criminal(x),love(x,y)\}\rangle\}\rangle\,\}\rangle$
(10b) $\langle\{\},\{\neg\,\langle\{\},\{\neg\langle\{x,y\},\{criminal(x),woman(y),love(x,y)\}\rangle\}\rangle\,\}\rangle$

These interpretations differ in dynamic potential (e.g. in the *specific* readings (9a) and (10a) the referent for "woman" is anaphorically available for continuations), and the resolution of such equivalences belongs to a different processing stage. In spite of this, the weaker, *non-specific* readings could in principle be dispensed with by extending S with '$\langle\{\},\{\neg\, h\}\rangle$' and adapting *Plug* to prohibit indefinites under double negation (since generalized quantifiers cannot have arbitrary wide scope and thus do not induce such equivalences). Informally: $D(\langle\{\},\{\neg\langle\{\ldots\},\{\neg\ldots h\ldots\}\rangle\}\rangle,\otimes\{k,h_1,h_2\})=\bot$.

4 Conclusion

A general restriction is formalized within Hole Semantics with the aim of efficiently avoiding the generation of equivalent quantifier scopings. The proposed scopal ordering constraint interrupts redundant disambiguations, being therefore able to significantly reduce the set of generated formulas.

Acknowledgements I am grateful to the anonymous referees for helpfull comments and to Johan Bos for providing a version of the Althaus et. al. (2003) constraint solver. The errors that I failed to correct are mine.

References

Alshawi, H. and R. Crouch (1992). Monotonic semantic interpretation. In *Proceedings of the 30th ACL*, pp. 32–39. Newark, Delaware.

Althaus, E., D. Duchier, A. Koller, K. Mehlhorn, J. Niehren, and S. Thiel (2003). An efficient graph algorithm for dominance constraints. *Journal of Algorithms*, To appear.

Blackburn, P. and J. Bos (1999). Representation and Inference for Natural Language - A First Course in Computational Semantics. `http://www.coli.uni-sb.de/~bos/comsem`.

Bos, J. (1996). Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pp. 133–142. ILLC / Department of Philosophy, University of Amsterdam, The Netherlands.

Chaves, R. P. (2002). *Fundamentos para uma Gramática Computacional do Português – Uma abordagem lexicalista*. Master's dissertation, University of Lisbon, Lisbon.

Cooper, R. (1983). *Quantification and Syntactic Theory*. Reidel Publications, Dordrecht.

Corblin, F. (1995). Compositionality and complexity in multiple negation. *IGPL Bulletin 3, 3-2*, 449–473.

Corblin, F. (1996). Multiple negation processing in Natural Language. In *Theoria*, pp. 214–260.

Egg, M. and K. Lebeth (1995). Semantic underspecification and modifier attachment ambiguities. In J. Kilbury and R. Wiese (Eds.), *Integrative Ansätze in der computer linguistik*, pp. 19–24. Düsseldorf, Seminar für Allgemeine Sprachwissenschaft.

Egg, M., J. Niehren, P. Ruhrberg, and F. Xu (1998). Constraints over lambda-structures in semantic underspecification. In *Joint COLING/ACL*, pp. 353–359. Montreal, Canada.

Gabsdil, M. and K. Striegnitz (1999). Classifying scope ambiguities. In C. Montz and M. de Rijke (Eds.), *ICOS-1, Institute for Logic, Language and Computation (ILLC)*, Amsterdam, pp. 125–131.

Hobbs, J. R. and S. M. Shieber (1987). An algorithm for generating quantifier scopings. *Computational Linguistics 13(1-2)*, 47–55.

Kamp, H. and U. Reyle (1993). *From Discourse to Logic: An Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and DRT*. Dordrecht, Holland: Kluwer.

Keller, W. R. (1988). Nested Cooper storage: The proper treatment of quantification in ordinary noun phrases. In U. Reyle and C. Roher (Eds.), *Natural Language Parsing and Linguistic Theories*, Dordrecht: Reidel, pp. 432–447.

Koller, A., J. Niehren, and S. Thater (2003). Bridging the gap between underspecification formalisms: Hole semantics as dominance constraints. In *Proceedings of the 11th EACL*, Budapest.

McCawley, J. D. (1981). *Everything that Linguists always wanted to know about logic... but were ashamed to ask*. Chicago: The University of Chicago Press.

Reyle, U. (1993). Dealing with Ambiguities by Underspecification: Construction, Representation and Deduction. *Journal of Semantics 102*, 123–179.

Vestre, E. J. (1991). An algorithm for generating non-redundant quantifier scopings. In *Fifth Conference of the European Chapter of the ACL*, pp. 251–256. Berlin, Germany.

The Beth Property for the Modal Logic of Graded Modalities, with an Application to the Description Logic $\mathcal{ALCQ}$.

Willem Conradie
Department of Mathematics and Statistics,
Rand Afrikaans University,
P O Box 524, Auckland Park, 2006, South Africa
wec@rau.ac.za

Abstract. **Abstract:** We prove the Beth definability-property for a modal logic with graded (or "counting") modalities. The result has an important consequence for the description logic known as $\mathcal{ALCQ}$, as it implies that every definitorial $\mathcal{ALCQ}$-terminology is equivalent to an acyclic $\mathcal{ALCQ}$-terminology. It is shown that the Beth property fails for the logics obtained by adding the difference operator to the logic of graded modalities or to basic modal logic.

1 Introduction

In section 1.1 we introduce the Beth property - a property standardly investigated for logics, which also happens to have a nice intuitive application to description logics. These are introduced in section 1.2. In section 2 we extend modal logic by adding to it the difference operator and graded modalities. Certain combinations of these operators give rise to logics in which the Beth property fails - two of these are illustrated is section 4, making use of a characterization of the Beth property given in section 3. Section 5 centers around the proof of our main result - the Beth property for the logic of graded modalities, and also makes use of the characterization provided in section 3.

We assume basic knowledge of the syntax and semantics of modal logic, as well the notions of *bisimulation*, *generated submodel*, the *unravelling* of a model and the *standard translation* of a modal formula. A state of the art reference here is [Blackburn et al., 2001]. From first-order model theory we assume the notions of *(countably) saturated model*, *type* and *ultraproducts* as well as related theorems such as that of Łos (see [Hodges, 1993]). All work will be purely semantical, languages will always be interpreted over the class of all possible models. With this in mind we will mostly identify

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

logics with their languages for the sake of brevity. We shall work with a global consequence relation $\Vdash$.

1.1 The Beth-Property

We parameterize a language $\mathcal{L}$ with a set of proposition letters Π, as follows $\mathcal{L}(\Pi)$ is we want to be explicit about the proposition letters used in the language. Similarly for sets of formulas: $\Gamma(\Pi)$. When $\Pi = \{q_1, q_2, \ldots, q_i, p\}$ we shall write $\Gamma(\overline{q}, p)$.

Suppose that we are working in a logic S in a language $\mathcal{L}$. Before introducing the Beth-property for propositional (modal) logics, we need the notions of explicit- and implicit definitions:

Definition 1.1 Let $\Gamma(\overline{q}, p)$ be a set of formulas in $\mathcal{L}$ and p' a proposition letter not occurring in any formula in $\Gamma(\overline{q}, p)$. Let $\Gamma(\overline{q}, p')$ be the result of uniformly substituting p' for every occurrence of p in Γ. Γ is called an *implicit definition* of p, if $\Gamma(\overline{q}, p), \Gamma(\overline{q}, p') \Vdash_S p \leftrightarrow p'$. $\lhd$

The intuitive meaning of this rather technical-looking definition is the following: Given an interpretation of the symbols in Γ except p, the interpretation of p is fixed. Or stated another way, given any model for the language $\mathcal{L}(\Gamma) - \{p\}$, this model can be extended in at most one way to a model of Γ, by giving a valuation for p. Showing that a set of formulas Γ is *not* an implicit definition of p then boils down to demonstrating the existence of two models of Γ, based on the same frame and having the same $\overline{q}$-valuation, but differing on their p-valuation. This was first observed by Padoa in 1900 in the context of first order logic, and the method has become known as *Padoa's method*.

Definition 1.2 Let p be a proposition letter and $\Gamma(\overline{q}, p)$ a set of formulas in $\mathcal{L}$. A formula $\phi \in \mathcal{L}(\overline{q})$ is called an *explicit definition* of p relative to $\Gamma(\overline{q}, p)$ if $\Gamma(\overline{q}, p) \Vdash_S p \leftrightarrow \phi$. $\lhd$

Definition 1.3 A logic S is said to have the *Beth-property* if for any implicit definition Γ, of a proposition letter p, there exists an explicit definition ϕ of p relative to Γ. $\lhd$

The Beth-property has been established for quite a number of (propositional) modal logics, including the logic K. One particularly interesting result here is due to [Maksimova, 1993]. This states that all normal extensions of the logic $\mathsf{K4}$ have the Beth-property. One should also mention the relationship between interpolation and the Beth property. Stated simply, interpolation implies the Beth-property. For a detailed discussion of these matters the reader is referred to [Hoogland, 2001].

1.2 Description Logic

An extensive hierarchy of knowledge representation formalisms, known as description logics, exists in the literature (see [Baader et al., 2003]). Many of these logics turn out to be syntactic variants of (extended) multi-modal logics. For example, the description language $\mathcal{ALC}$ corresponds to a propositional modal language with (possibly) multiple diamonds. $\mathcal{ALCQ}$, a much more expressive description logic, corresponds to the modal logic with counting modalities $\mathcal{L}_{\leq}$, which we define below, but possibly with many unary modalities. For an introduction to description logics the reader is referred to [Baader and Nutt, 2003].

Certain sets of formulas in description logics are known as *terminologies*. In such a terminology *concepts* (corresponding to our proposition letters) are defined in terms of other *base concepts* (corresponding to a designated subset of the proposition letters in the language). The definition of a concept may also (directly or indirectly) make use of the concept itself. A terminology in which this happens is said to by *cyclic*, and *acyclic* otherwise.

A terminology, $\mathcal{T}$, is said to be *definitorial* if any interpretation of the base concepts completely determines the interpretation of the defined concepts on any model for $\mathcal{T}$. This is clearly the same as saying that $\mathcal{T}$ is an implicit definition of the defined concepts. The Beth property for a description logic then says that on any model for a definitorial terminology, for every defined concept there is a formulas equivalent to it, only making use of base concepts. This implies that every definitorial terminology in the logic is equivalent to a acyclic terminology. From a computational viewpoint this is very important, as reasoning with acyclic terminologies is much easier that doing so with cyclic ones.

2 The Difference Operator and Graded Modalities

We work on Kripke models with one binary relation (as for the basic modal language). Let $\mathcal{M} = (W, R, V)$ be such a model and $m \in W$. The *difference operator*, D, has the following semantics

$\mathcal{M}, m \Vdash \mathsf{D}\phi$ iff there exists a point $n \in M$ such that $n \neq m$ and $\mathcal{M}, n \Vdash \phi$.

Note that the difference operator allows us to define the universal modality (cf. [Blackburn et al., 2001], p. 415)

$$\mathsf{E}\phi \equiv \phi \vee \mathsf{D}\phi$$

and

$$\mathsf{A}\phi \equiv \neg(\mathsf{E}\neg\phi).$$

We are also able to count states in a limited way. Define

$$\mathsf{E}!\phi \equiv \mathsf{E}(\phi \wedge \neg \mathsf{D}\phi)$$

which holds on a model precisely when exactly one state where ϕ holds.

Taking the idea of counting (successor) states more seriously leads us to the idea of *graded modalities*: For each $n \in \omega$ we add the modality $\lhd_{\leq n}$ to our language, with the following interpretation

$$\mathcal{M}, m \Vdash \lhd_{\leq n}\phi \text{ if and only if } |\{y \in M : Rmy \text{ and } \mathcal{M}, y \Vdash \phi\}| \leq n.$$

One can now define

$$\lhd_{>n}\phi \equiv \neg \lhd_{\leq n} \phi, \;\; \lhd_{\geq n} \phi \equiv \lhd_{>n-1}\phi, \;\; \lhd_{=n} \phi \equiv (\lhd_{\leq n}\phi \wedge \lhd_{\geq n}\phi), \;\; \Diamond\phi \equiv \lhd_{\geq 1}\phi$$

We shall be considering the languages $\mathcal{L}_{\Diamond,\mathsf{D}}$ defined as

$$\phi ::= p \mid \top \mid \neg\phi \mid \phi \vee \psi \mid \Diamond\phi \mid \mathsf{D}\phi$$

and $\mathcal{L}_{\leq}$ given by

$$\phi ::= p \mid \top \mid \neg\phi \mid \phi \vee \psi \mid \lhd_{\leq n} \phi$$

Note that, even though it does not contain the $\Diamond$-operator, $\mathcal{L}_{\leq}$ subsumes the basic modal language in terms of expressivity.

We extend the notion of a modal bisimulation to accommodate the difference operator and graded modalities respectively, obtaining the notions of *difference bisimulation* and *counting bisimulation*. If w is a point in a model we write $R[w]$ for the set its R-successors in that model.

Definition 2.1 Let $\mathcal{M} = (M, R, V)$ and $\mathcal{M}' = (M', R', V')$ be models. A relation $Z \subseteq M \times M'$ is called a *difference bisimulation* if it satisfies the following.

bisimulation Z is a bisimulation

difference forth if uZu', $v \in M$ and $v \neq u$, then there exists a point $v' \in M'$ such that $v' \neq u'$ and vZv'

difference back if uZu', $v' \in M$ and $v' \neq u'$, then there exists a point $v \in M$ such that $v \neq u$ and vZv'

We write $\mathcal{M}, m \leftrightarrows_{\mathcal{L}_{\Diamond,\mathsf{D}}} \mathcal{M}', m'$ to indicate that there exists a difference bisimulation between m and m', or $\mathcal{M}, m \leftrightarrows_{\mathcal{L}_{\Diamond,\mathsf{D}}(\Pi)} \mathcal{M}', m'$ if Π is the set of proposition letters with respect to which the local harmony clause of the bisimulation is taken, i.e. the clause specifying that bisimilar states satisfy the same proposition letters. $\lhd$

The proof of the next proposition is standard.

Proposition 2.2 *Let $\mathcal{M} = (M, R, V)$ and $\mathcal{M}' = (M', R', V')$ be models, $m \in M$ and $m' \in M'$. If $\mathcal{M}, m \leftrightarrows \mathcal{L}_{\Diamond,D}\mathcal{M}', m'$ then $\mathcal{M}, m \leftrightsquigarrow \mathcal{L}_{\Diamond,D}\mathcal{M}', m'$.*

Definition 2.3 Let $\mathcal{M} = (M, R, V)$ and $\mathcal{M}' = (M', R', V')$ be models. A relation $Z \subseteq M \times M'$ is called a *counting bisimulation* if the following conditions are satisfied

local harmony if uZu' then u and u' satisfy the same proposition letters

counting forth if uZu' and $S \subseteq R[u]$ is a finite or countably infinite set, then there exists a set $S' \subseteq R'[u']$ such that (i) for each $s \in S$ there exists an $s' \in S'$ such that sZs', (ii) for each $s' \in S'$ there exists an $s \in S$ such that sZs' and (iii) $|S| = |S'|$.

counting back A similar condition for points in $\mathcal{M}'$ and sets of their successors.

$\lhd$

We write $\mathcal{M}, m \leftrightarrows_{\mathcal{L}_\leq} \mathcal{M}', m'$ to indicate that there exists a counting bisimulation linking m and m', or $\mathcal{M}, m \leftrightarrows_{\mathcal{L}_\leq(\Pi)} \mathcal{M}', m'$ if Π is the set of proposition letters with respect to which the local harmony clause is taken.

Proposition 2.4 *Let $\mathcal{M} = (M, R, V)$ and $\mathcal{M}' = (M', R', V')$ be two (not necessarily distinct) models, $m \in M$, $m' \in M'$, and $Z : \mathcal{M}, m \leftrightarrows_{\mathcal{L}_\leq(\Pi)} \mathcal{M}', m'$ a counting bisimulation linking m and m'. Then $m \leftrightsquigarrow_{\mathcal{L}_\leq(\Pi)} m'$.*

The logic $\mathcal{L}_{\leq,\mathsf{D}}$ is obtained by adding the difference operator to $\mathcal{L}_\leq$. The notion of a *difference counting bisimulation* is obtained by adding difference clauses to a counting bisimulation. The corresponding preservation result is easily proved.

3 The Beth Property as the Extendability of Equivalences

In this section we formulate and prove a necessary and sufficient condition for a logic with Kripkean semantics to have the Beth property. This will be used in the proof of our main result, theorem 5.5.

If m_1 and m_2 are points from (not necessarily distinct) Kripke models $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively, we write $\mathcal{M}_1, m_1 \leftrightsquigarrow_{\mathcal{L}(\Pi)} \mathcal{M}_2, m_2$ if m_1 and m_2 satisfy exactly the same formulas from the language $\mathcal{L}(\Pi)$. We say m_1 and m_2 are equivalent with respect to $\mathcal{L}(\Pi)$.

Theorem 3.1 *Let S be a compact logic in a language $\mathcal{L}$, and $\Gamma(\overline{q}, p)$ a (possibly empty) set of formulas. There exists an explicit definition of p in the language $\mathcal{L}(\overline{q})$ relative to $\Gamma(\overline{q}, p)$ if and only if for any models $\mathcal{M}_1$ and $\mathcal{M}_2$ of Γ and points $m_1 \in \mathcal{M}_1$ and $m_2 \in \mathcal{M}_2$,*

$$m_1 \leftrightsquigarrow_{\mathcal{L}(\overline{q})} m_2 \text{ only if } m_1 \leftrightsquigarrow_{\mathcal{L}(\overline{q},p)} m_2.$$

Proof: Suppose there exists an explicit definition of p relative to $\Gamma(\overline{q}, p)$, say $\phi \in \mathcal{L}(\overline{q})$. Specifically $\Gamma \Vdash p \leftrightarrow \phi$, hence if $\psi \in \mathcal{L}(\overline{q}, p)$ and $\psi[p/\phi]$ is the result of uniformly replacing p with ϕ in ψ, we have that $\Gamma \Vdash \psi \leftrightarrow \psi[p/\phi]$. Let $\mathcal{M} = (W, R, V)$ be any model of Γ and $m_1, m_2 \in W$. Suppose $m_1 \leftrightsquigarrow_{\mathcal{L}(\overline{q})} m_2$. If $\psi \in \mathcal{L}(\overline{q}, p)$ then $\mathcal{M}, m_1 \Vdash \psi$ iff $\mathcal{M}, m_1 \Vdash \psi[p/\phi]$ iff $\mathcal{M}, m_2 \Vdash \psi[p/\phi]$ iff $\mathcal{M}, m_2 \Vdash \psi$. Hence $m_1 \leftrightsquigarrow_{\mathcal{L}(\overline{q},p)} m_2$.

To prove the right-to-left direction we proceed by contraposition. Suppose that there exists no explicit definition of p relative to $\Gamma(\overline{q}, p)$. Let $\psi_0, \psi_1, \psi_2, \ldots$ be an enumeration of the formulas in $\mathcal{L}(\overline{q})$. For each ψ_i let $\psi_i^1 = \psi_i$ and $\psi_i^0 = \neg\psi_i$.

Claim 1: For each $n \in \omega$ there exists a function $\chi : \{0, 1, \ldots, n\} \to \{0, 1\}$ such that both $p \wedge \bigwedge_{i=1}^n \psi_i^{\chi(i)}$ and $\neg p \wedge \bigwedge_{i=1}^n \psi_i^{\chi(i)}$ are satisfiable on models of Γ.

Proof of Claim 1: Let $n \in \omega$. Note that there are 2^n functions $\chi : \{0, 1, \ldots, n\} \to \{0, 1\}$, call them $\chi_1, \chi_2, \ldots, \chi_{2^n}$. Suppose to the contrary that for no function $\chi : \{0, 1, \ldots, n\} \to \{0, 1\}$ both $p \wedge \bigwedge_{i=1}^n \psi_i^{\chi(i)}$ and $\neg p \wedge \bigwedge_{i=1}^n \psi_i^{\chi(i)}$ are Γ-satisfiable. Hence for each χ, either $\Gamma \Vdash \bigwedge_{i=1}^n \psi_i^{\chi(i)} \to p$ or $\Gamma \Vdash \bigwedge_{i=1}^n \psi_i^{\chi(i)} \to \neg p$. Let $\Psi_j = \bigwedge_{i=1}^n \psi_i^{\chi_j(i)}$ for each $1 \leq j \leq 2^n$. Note that $\Vdash \bigvee_{j=1}^{2^n} \Psi_j \leftrightarrow \top$. Let $J_1 = \{j : \Gamma \Vdash \Psi_j \to p\}$ and $J_2 = \{j : \Gamma \Vdash \Psi_j \to \neg p\}$. Then clearly $\Gamma \Vdash \bigvee_{j \in J_1} \Psi_j \to p$ and $\Gamma \Vdash \bigvee_{j \in J_2} \Psi_j \to \neg p$, and $J_1 \cup J_2 = \{1, 2, \ldots, 2^n\}$. Hence $\Gamma \Vdash \bigvee_{j \in J_1} \Psi_j \leftrightarrow p$, yielding an explicit definition and a contradiction. $\dashv$

Claim 2: There exists a chain of functions $\rho_1 \subseteq \rho_2 \subseteq \rho_3 \subseteq \cdots$ where for each i, $\rho_i : \{1, 2, \ldots, i\} \to \{0, 1\}$, such that if $\Phi_i = \bigwedge_{j=1}^i \psi_j^{\rho_i(j)}$ then both $\Phi_i \wedge p$ and $\Phi_i \wedge \neg p$ are Γ-satisfiable.

Proof of Claim 2: To see that this is the case, let as represent the set of functions $\mathcal{F} = \{\chi : \{0, 1, \ldots, n\} \to \{0, 1\} : n \in \omega\}$ as a binary tree. We call this tree $\mathcal{T}$. More precisely, a function $\chi : \{0, 1, \ldots, n\} \to \{0, 1\}$ is represented by a path from the root to level $n + 1$, where, if the left branch is taken at level i, it means that $\chi(i) = 0$, and if the right is taken that

$\chi(i) = 1$. Clearly there is a one-to-one correspondence between functions in $\mathcal{F}$ and finite paths in $\mathcal{T}$.

We call a function $f \in \mathcal{F}$ acceptable for $n \in \omega$ if $f : \{0, 1, \ldots, m\} \to \{0, 1\}$ for some $m \geq n$, and both $p \wedge \bigwedge_{i=1}^{n} \psi_i^{f(i)}$ and $\neg p \wedge \bigwedge_{i=1}^{n} \psi_i^{f(i)}$ are satisfiable on models of Γ. Clearly, if f is acceptable for n, then f is acceptable for all $j \leq n$. We call a path of length n acceptable, if it corresponds to some function in $\mathcal{F}$ which is acceptable for n.

By claim 1 there are arbitrarily long acceptable paths from the root of $\mathcal{T}$. Thus by König's Lemma there must be an infinitely long path from the root of $\mathcal{T}$, call it P, such that every finite sub-path of P, starting at the root is acceptable. The desired chain of functions is then the chain corresponding to the finite sub-paths of P, starting at the root. $\dashv$

Let $U = \{p\} \cup \{\Phi_i : i \in \omega\}$ and $T = \{\neg p\} \cup \{\Phi_i : i \in \omega\}$. Every finite subset of U is Γ-satisfiable, as is every finite subset of T, hence U and T are both Γ-satisfiable, hence by compactness (recall that by assumption we are dealing with a compact modal logic) both U and T are satisfiable on models of Γ. Moreover U and T are maximal satisfiable sets in the language $\mathcal{L}(\overline{q})$, as for each ψ_i either ψ_i or $\neg\psi_i$ is a conjunct of Φ_i. This means there are models of Γ, $\mathcal{M} = (W, R, V)$ and $\mathcal{M}' = (W', R', V')$ say, and points $w \in W$ and $w' \in W'$ such that every formula in U is satisfied at w and every formula in T is satisfied at w'. As U and T are maximal in $\mathcal{L}(\overline{q})$, w and w' agree on all formulas in $\mathcal{L}(\overline{q})$, but differ on their valuation of p, i.e $w \leftrightsquigarrow_{\mathcal{L}(\overline{q})} w'$ but $w \not\leftrightsquigarrow_{\mathcal{L}(\overline{q},p)} w'$. $\dashv$

4 Two Negative Beth Results

Theorem 4.1 *$\mathcal{L}_{\Diamond,D}$ and $\mathcal{L}_{\leq,D}$ do not have the Beth property.*

Proof: Let p, q and r be proposition letters and let

$$\Gamma(q, r, p) = \{\mathsf{E!}(\mathsf{p} \wedge \mathsf{q})), \neg\mathsf{q} \to \mathsf{p}, \mathsf{Er}, \mathsf{r} \to \Diamond\top, \mathsf{r} \to \Box(\mathsf{p} \wedge \mathsf{q})\}$$

The intuitive content of this proof is the following. On any model of Γ all the $\neg q$-points are p-points, and further there is precisely one $(p \wedge q)$-point. But at which q-point must p hold? This is pointed out by some r-point: the unique point which is the successor of all the r-points. The death-blow to any aspiring explicit definition however, is the fact that a p-point might not be able to "see" if it is being pointed at by an r-point. It is not difficult to prove that if $\mathcal{M} = (M, R, V)$ and $\mathcal{M} \Vdash \Gamma$, then

$$[\![p]\!]_{\mathcal{M}} = [\![\neg q]\!]_{\mathcal{M}} \cup \{m \in M : \mathcal{M}, m \Vdash q \text{ and there is a } w \in M \text{ such that } \mathcal{M}, w \Vdash r \text{ and } Rwm\}$$

So Γ implicitly defines p. But now consider the model in figure 1. It is easy to check that it indeed models Γ. Note that $\alpha \leftrightarrows_{\mathcal{L}_{\Diamond,\mathsf{D}}(q)} \beta$. It is also the case that $\alpha \leftrightarrows_{\mathcal{L}_{\leq,\mathsf{D}}(q)} \beta$. (The relation relating the top point to itself and α and β to each other will suffice in both cases.) This means $\alpha \leftrightsquigarrow_{\mathcal{L}_{\Diamond,\mathsf{D}}(q)} \beta$ and that $\alpha \leftrightsquigarrow_{\mathcal{L}_{\leq,\mathsf{D}}(q)} \beta$. But yet $\alpha\neg \leftrightsquigarrow_{\mathcal{L}_{\Diamond,\mathsf{D}}(q,p)} \beta$ and that $\alpha\neg \leftrightsquigarrow_{\mathcal{L}_{\leq,\mathsf{D}}(q,p)} \beta$. Hence there can be no explicit definition of p relative to Γ. $\dashv$

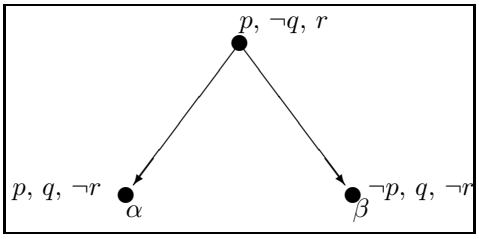


Figure 1: Theorem 4.1

5 The Beth-property for $\mathcal{L}_{\leq}$ or $\mathcal{ALCQ}$

Before being able to prove theorem 5.5 we need the following remark, propositions and lemma:

Remark 5.1 Note that the truth of formulas of $\mathcal{L}_{\leq}$ is invariant under the taking of generated submodels. Specifically, if $v \in \mathcal{M}$, then $\mathcal{M}, v \leftrightsquigarrow_{\mathcal{L}_{\leq}} \mathcal{M}(v), v$, where $\mathcal{M}(v)$ denotes the submodel generated by $\{v\}$.

Proposition 5.2 *If $\mathcal{M} = (M, R, V)$ is rooted at r, then $\mathcal{M}, r \leftrightarrows_{\mathcal{L}_{\leq}} \vec{\mathcal{M}}, (r)$ where $\vec{\mathcal{M}}$ is the unravelling of $\mathcal{M}$.*

The following two proposition are not difficult to prove. For a complete proofs see [Conradie, 2002].

Proposition 5.3 *Let $\mathcal{M}$ and $\mathcal{N}$ be countably saturated models. Then the relation $\leftrightsquigarrow_{\mathcal{L}_{\leq}}$ is a counting bisimulation between $\mathcal{M}$ and $\mathcal{N}$.*

Lemma 5.4 *Let $\mathcal{T} = (T, R, V)$ and $\mathcal{T}' = (T', R', V')$ be two at most countable tree models, rooted at r and r' respectively. If $\mathcal{T}, r \leftrightarrows_{\mathcal{L}_{\leq}} \mathcal{T}', r'$ then $\mathcal{T}, r \cong_{\mathcal{L}_{\leq}} \mathcal{T}', r'$.*[1]

Theorem 5.5 *$\mathcal{L}_{\leq}$ has the Beth-property.*

[1]The symbol $\cong$ is taken to mean, as usual, isomorphism, while the subscript means that the isomorphism respects the relations used in for the interpretation of the language indicated.

PROOF: Suppose as set of formulas $\Gamma(\overline{q}, p)$ implicitly defines p. Let $\mathcal{M}$ and $\mathcal{M}'$ be any models of $\Gamma(\overline{q}, p)$ of *of at most countable cardinality.* Let m and m' be points in $\mathcal{M}$ and $\mathcal{M}'$ respectively such that $\mathcal{M}, m \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q})} \mathcal{M}', m'$. By theorem 3.1 it is sufficient to show that $\mathcal{M}, m \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q},p)} \mathcal{M}', m'$. This is what we now proceed to do.

Let $\mathcal{UM}$ and $\mathcal{UM}'$ be ultrapowers of $\mathcal{M}$ and $\mathcal{M}'$ respectively over some countably incomplete ultrafilter $\mathcal{U}$. Let id_m and $id_{m'}$ be the points a/U and a'/U in $\mathcal{UM}$ and $\mathcal{UM}'$ respectively where $a(i) = m$ and $a'(i) = m'$ for all i. Note that these models are still at most countable. By Łos's theorem (and the fact that $\mathcal{L}_{\leq}$-formulas may be translated into first-order formulas preserving equivalence on the level of models) it is the case that $\mathcal{M}, m \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q},p)} \mathcal{UM}, id_m$ and $\mathcal{M}', m' \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q},p)} \mathcal{UM}', id_{m'}$ and, further, that $\mathcal{UM}$ and $\mathcal{UM}'$ are models of $\Gamma(\overline{q}, p)$. Hence we have $\mathcal{UM}, id_m \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q})} \mathcal{UM}', id_{m'}$. But since $\mathcal{UM}$ and $\mathcal{UM}'$ are countably saturated it follows from lemma 5.3 that in fact $\mathcal{UM}, id_m \leftrightarrows_{\mathcal{L}_{\leq}(\overline{q})} \mathcal{UM}', id_{m'}$.

Now, unravel $\mathcal{UM}$ and $\mathcal{UM}'$, around id_m and $id_{m'}$ respectively (by remark 5.1 we may assume that the they are rooted at id_m and $id_{m'}$) to obtain two tree models $\mathcal{T}$ and $\mathcal{T}'$ rooted at (id_m) and $(id_{m'})$ respectively. Note that the tree models are at most countable and models of $\Gamma(\overline{q}, p)$. By proposition 5.2 we have $\mathcal{UM}, id_m \leftrightarrows_{\mathcal{L}_{\leq}(\overline{q},p)} \mathcal{T}, (id_m)$ and $\mathcal{UM}', id_{m'} \leftrightarrows_{\mathcal{L}_{\leq}(\overline{q},p)} \mathcal{T}', (id_{m'})$ and hence $\mathcal{T}, (id_m) \leftrightarrows_{\mathcal{L}_{\leq}(\overline{q})} \mathcal{T}', (id_{m'})$. But it then follows from lemma 5.4 that $\mathcal{T}, (id_m) \cong_{\mathcal{L}_{\leq}(\overline{q})} \mathcal{T}', (id_{m'})$. It must be the case that $\mathcal{T}, (id_m) \cong_{\mathcal{L}_{\leq}(\overline{q},p)} \mathcal{T}', (id_{m'})$, for otherwise, by Padoa's method it would follow that $\Gamma(\overline{q}, p)$ did not implicitly define p - a contradiction. But this implies that $\mathcal{T}, (id_m) \leftrightarrows_{\mathcal{L}_{\leq}(\overline{q},p)} \mathcal{T}', (id_{m'})$, hence that $\mathcal{T}, (id_m) \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q},p)} \mathcal{T}', (id_{m'})$. Now we have

$$m \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q},p)} id_m \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q},p)} (id_m) \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q},p)} (id_{m'}) \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q},p)} id_{m'} \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q},p)} m'$$

i.e. $m \leftrightsquigarrow_{\mathcal{L}_{\leq}(\overline{q},p)} m'$ as desired.

This proves the Beth property for the logic over the class of all models of at most countable cardinality. We may now apply the Löwenheim-Skolem theorem upwards, to obtain the result for the logic over the class of all models. ⊣

6 Concluding Remarks

To conclude we mention some recent results and some open questions.
[Ten Cate, 2003] has recently proven that the smallest extension of the $\mathcal{L}_{\Diamond,\mathsf{D}}$ that has the interpolation property is the first order correspondence language. It would be interesting to find a similar result for the smallest extensions of $\mathcal{L}_{\Diamond,\mathsf{D}}$ and $\mathcal{L}_{\leq,\mathsf{D}}$ that have the Beth-property. As far as $\mathcal{ALCQ}$-terminologies are concerned, it would be interesting to find an upper bound on the size of the acyclic terminology corresponding to a cyclic terminology, the existence of which is guaranteed by our result. Lastly, techniques for

finding such an acyclic terminology constructively (analogous to those developed in [Blackburn and Marx, 2002] for finding interpolants) do not seem to exist as yet.

References

[Baader and Nutt, 2003] F. Baader, W.Nutt. Basic Description Logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider (editors). *The Description Logic Handbook: Theory, Implementation and Applications,* Cambridge University Press, 2003.

[Baader et al., 2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider (editors). *The Description Logic Handbook: Theory, Implementation and Applications,* Cambridge University Press, 2003.

[Beth, 1953] E. W. Beth. On Padoa's method in the theory of definition. *Indagiones Math.,* 15:330-339, 1953.

[Blackburn et al., 2001] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic.* Cambridge University Press, 2001.

[Blackburn and Marx, 2002] P. Blackburn and M. Marx. Constructive interpolants for every bounded fragment definable hybrid logic. To appear in *Journal of Symbolic Logic.*

[Conradie, 2002] W. E. Conradie. *Definability and Changing Perspectives - The Beth Property for Three extensions of Modal Logic.* MOL Thesis, Institute for Logic, Language and Computation, Universiteit van Amsterdam, 2002.

[D'Agostino, 1998] G. D'Agostino. *Modal Logic and non-well-founded Set Theory: translation, bimsimulation, interpolation.* PhD Thesis, Institute for Logic, Language and Computation, Universiteit van Amsterdam, 1998.

[Fine, 1972] K. Fine. In so many possible worlds. *Notre Dame Journal of Formal Logic,* 13(4):516-520, 1972.

[Hodges, 1993] W. Hodges. *Model Theory.* Cambridge University Press, 1993.

[Hoogland, 2001] E. Hoogland. *Definability and Interpolation.* PhD Thesis, Institute for Logic, Language and Computation, Universiteit van Amsterdam, 2001.

[Kurtonina and de Rijke, 1999] N. Kurtonina and M. de Rijke. Expressiveness of concept expressions in first-order description logics. *Artificial Intelligence,* 107:303-333, 1999.

[Maksimova, 1993] L. L. Maksimova. An analogue of Beth's theorem in normal extensions of the modal logic K4. *Siberian Mathematical Journal,* 33(6):1052-1065.

[Ten Cate, 2003] Ten Cate, B. Personal communication.

[Van der Hoek and de Rijke, 1995] W. van der Hoek and M. de Rijke. Counting Objects. *Journal of Logic and Computation,* 5(3):325-345, 1995.

Alternations, monotonicity and the lexicon: an application to factorising information in a Tree Adjoining Grammar

BENOIT CRABBÉ

LORIA

Campus Scientifique

B.P. 239

F-54506 Vandoeuvre-lès-Nancy Cedex

Benoit.Crabbe@loria.fr

ABSTRACT. For reasons of maintenance, economy and consistency, devising ways of factorising the information contained in a grammar is particularly important. This paper proposes a factorising system for TAG (Tree Adjoining Grammar) taking its sources in Beth Levin's alternations and thereby remains monotonic. It explains why monotonicity is an important caracteristic of such a system and compares the proposed approach to existing, related proposals.

1 Introduction

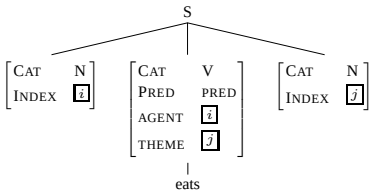
Tree Adjoining Grammar (TAG)[1] is a tree composition system whose units are elementary trees. Two operations are defined for tree composition: adjunction and substitution. TAG is mostly used in its lexicalized version (LTAG). The lexicalization condition constrains each elementary tree to be anchored by a terminal symbol. In this context the lexicon of a TAG is a set of elementary trees.

We further assume a syntax-semantics interface as illustrated by the following tree[2]:

[1] See (Joshi et al. 1975) for an historical definition and (Joshi and Schabès 1997) for a more recent presentation.

[2] The underlying version of TAG suggested here is Feature Based TAG (Vijay-Shanker 1987), where nodes are associated with a top and a bottom feature structure. Though, for readability, top and bottom features are not explicitly stated on the figures. Moreover, the following figures abbreviate the notation: feature structures from left to right will be abridged respectively as N_i, $V_{pred(i,j)}$ and N_j.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

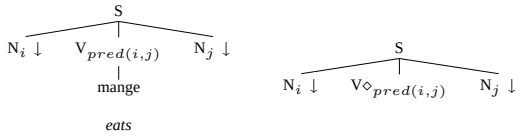


Here the association between tree nodes and unification variables encodes the syntax/semantics interface – it specifies which node in the tree provides the value for which variable in the final semantic representation (Gardent and Kallmeyer 2003)

Our goal is to automate the production of such elementary trees and more generally, to give means of designing a TAG lexicon for French[3]. We argue (Section 3) that this automation process can benefit from Beth Levin's ideas on alternations (Levin 1993) and in particular, that alternations support the monotonic formulation of a tree generation system. Section 4 provides a detailed example of how such a system can be used in practice and Section 5 compares the proposed approach to related proposals namely, Becker's metarule based approach (Becker 1993), Evans et al.'s DATR (Evans et al. 2000) treatment, Candito's metagrammar (Candito 1999) and Xia's LexOrg system (Xia 2001). We conclude with pointers for further research.

2 Metagrammar and the lexicon

Computational implementations of TAGs (XTAG Research Group 2001; Abeillé 2002) abstract templates from elementary trees. A template is an elementary tree where the anchor is left unspecified (marked as $\diamond$). Elementary trees are dynamically build at parse-time by adding the lexical anchor.



Factorising the grammar using only templates has to be improved. Indeed, templates are massively sharing sub-structures: it is straightforward to identify among templates common sub-trees as depicted in Figure 1 (all trees share the S-N subtree).

Since a realistic TAG grammar comprises several hundreds or thousands of templates, it is absolutely crucial to design trees using a factorizing system. Otherwise such grammars quickly become difficult to maintain. To introduce any modification, each template has to be manually checked. This led to the idea of metagrammar: a system that semi-automates the task of tree description (Vijay-Shanker and Schabes 1992; Candito 1996).

[3]This work focuses on French syntax and takes most of its justifications regarding structural descriptions in the work of (Abeillé 2002).

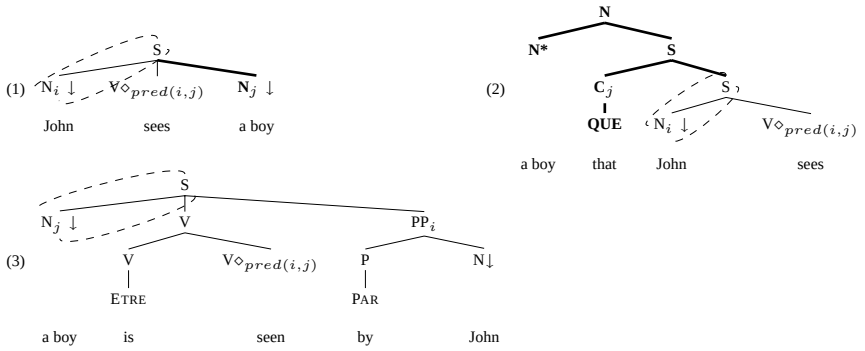


Figure 1: Structure sharing among templates

Moreover in a TAG, shared sub-trees often represent general linguistic properties. Thus for instance, the sub-trees encircled in Figure 1 all encode the structural information associated with a subject NP (in English). Additionally a grammatical function such as subject or object represents an abstraction over different possible realizations: sub-trees in bold face are different realizations of a direct object (canonical (1) and relativized (2)). We call **grammatical alternations** the possible alternative realisations associated with a given grammatical function.

Besides this first kind of alternations, the tree (3) in Figure 1, which represents a tree for a passive. Again the subject is encircled but a comparison with the two other trees reveals that the semantic index in subject position has actually changed. Alternations that specify the link between semantic arguments and grammatical functions are named **lexical alternations**.

More generally, it is important to note that a TAG lexicon encodes both grammatical and lexical information. A main feature of TAG is that a given lexical item is linked to *all* elementary trees representing the possible syntactic environments of that word, involving grammatical information. TAG differs in this from phrase-structure grammars where the lexicon involves only lexical alternations whilst grammatical alternations are captured by means of rewriting rules.

3 Alternations: form and meaning

The novelty in our proposal lies in the way lexical alternations are handled. The idea is inspired from (Levin 1993): rather than linking a word's meaning to a canonical subcategorization frame and then deriving additional frames, we associate a meaning to its full set of frames, canonical and non-canonical.

Much work in computational linguistics[4] associates a meaning (SEM) with a syntactic canonical frame (SYN). For instance, the meaning of the verb EAT might be linked to a syntactic canonical frame as follows[5]:

[4]See e.g. (Flickinger 1987; Sag and Wasow 1999; Evans et al. 2000). The thematic roles used here are stated after (Saint-Dizier 1996).

[5]The first line indicates the part of speech of the word considered, the third (SEM) the thematic

(1)
CAT	V
SYN	$\langle$SUBJ(NP)$_i$, OBJ(NP)$_j\rangle$
SEM	$\langle$AGENT$_i$, THEME$_j\rangle$

In a second step, syntactic variants of this canonical frame are derived by the application of lexical rules[6]. For instance, the application to frame (1) of a lexical rule mapping active to passive might yields the following additional frame:

(2)
CAT	V
SYN	$\langle$SUBJ(NP)$_j$, BY-OBJ(PP)$_i\rangle$
SEM	$\langle$AGENT$_i$, THEME$_j\rangle$

However, it is well known that lexical rules encounter many exceptions. Consider for instance, the following examples:

(a) Peter broke the window
(b) The window broke
(c) Peter ate an apple
(d) *An apple ate

Whereas *break* allows the ergative rule application, *eat* doesn't. This kind of observation leads to a Bloomfield like conclusion: lexical rules encounter arbitrary exceptions.

In other words, classical work associates a meaning only with its canonical frame and then mechanically derives additional frames using lexical rules, which requires in practice to specify many exceptions.

Here we want to identify a meaning with its full set of syntactic frames, canonical as non-canonical, that is with its full set of alternations. Thus we deny any status to lexical rules.

Indeed, that approach reflects lexical idiosyncrasy and removes the need to specify exceptions. But, an obvious drawback of such an approach is the loss of major syntactic generalisations expressed by lexical rules.

An alternation class is a set of alternations which are common to a significant number of verbs: according to the syntactic point of view, these verbs behave exactly the same way. Furthermore, it appears that verbs belonging to a given class share the same kind of meaning: for English, B. Levin identifies classes such as VERBS OF PUTTING, VERBS OF COLOURING, VERBS OF PERCEPTION... As for a given language, the set of classes is finite, it follows that establishing a lexicon using this approach only requires to identify a given lexeme with its alternation class.

To sum up, we get a different degree of generality: instead of expressing generality through the use of lexical rules, it is done through the use of alternation classes. It seems to be a better generalization in that it does not require to state any exceptions.

---

roles realised by the syntactic arguments of this word, and the second (SYN) the grammatical function and the category of the arguments. The linking between syntactic arguments and thematic roles is given by the shared indices (here i and j). Thus frame (1) indicates that EAT is a verb that can take a subject and an object NP as arguments, whereby the subject provides the agent and the object the theme index.

[6]Such an application can be expressed as a procedural $if \ldots then$ statement, as in early LFG, or as an inheritance relation, see e.g. (Bouma et al. 1998; Sag and Wasow 1999).

3.1 A formal corollary: monotonicity

We have just seen the linguistic interest of alternations, but a further interest lies on the formal side: using alternations permits us to express lexical knowledge in a monotonic (hence declarative) way.

Systems for lexical representation are usually non-monotonic (Flickinger 1987; Sag and Wasow 1999; Evans et al. 2000). A major source of non-monotonicity comes from the use of lexical rules that erase arguments. A striking example comes from a rule such as *agentless passive*. Given the two place subcategorization frame of the canonical frame (1), this rule outputs a one place subcategorization frame:

CAT V
SYN $\langle \text{SUBJ(NP)}_j \rangle$
SEM $\langle \text{AGENT}_i, \text{THEME}_j \rangle$

Using erasing rules licenses the removal of information, here the subject expressed in the canonical frame has been removed. Erasing is related to sequencing: allowing to remove information requires defining different states in the system. Hence the resulting system is procedural .

In contrast, using alternations does not require to apply erasing rules since these rules are not considered anymore. To handle agentless passive, we simply associate with a given verb the agentless passive alternation as well as any other alternation that is possible for that verb.

Besides erasing, there are cases were multiple lexical rules may be applied (in a given order) to a canonical structure. Here is an example taken from French:

Marie lave Marie
Mary washes Mary
Mary washes Mary (canonical)
Marie se lave
Mary herself wash
Mary washes herself (+ reflexive)
Jean fait se laver Marie
John causes herself wash Mary
John causes Mary to wash herself (+ causative)

To account for this, we must ensure that the set of possible alternations also include these alternations that corresponds to each possible sequence of rules. However these cases are, in French, quite limited. Indeed, (Candito 1999) identifies only ten such sequences, all involving the causative rule.

4 Formalization

This section concentrates on the use of alternations to factorise a TAG lexicon and automate its production. The basic idea is to describe a TAG by means of a (very big) disjunctive tree description. The TAG described is then the set of trees satisfying this description.

We proceed in three steps. First, we show how to describe a single TAG tree. Then, we show how the approach can be generalized to describing the full set

of trees related to an alternation. Finally, we show how the formulas used for descriptions can be integrated in a monotonic inheritance network.

4.1 Description of a single tree

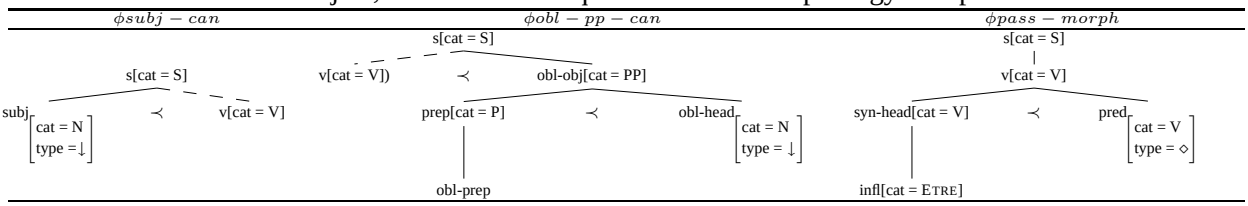
The information contained in a tree is made both of a lexical alternation and of some grammatical alternations. The description of a single tree illustrates this distinction. We first start with a lexical alternation, say *passive-with-2-arguments*, as stated by (Saint-Dizier 1996)[7].

CAT V
SYN $\langle$SUBJECT$_j$(NP),BY-OBJECT$_i$(PP)$\rangle$
SEM $\langle$AGENT$_i$, THEME$_j\rangle$

Trees are described with a language taking its origin in (Rogers and Vijay-Shanker 1992; Rogers and Vijay-Shanker 1994). Nodes are denoted by constants. The connectives are $\wedge$ and $\vee$. Relations are expressed with three binary predicates: dominates ($\overset{\star}{\triangleleft}$), immediately dominates ($\triangleleft$) and precedes ($\prec$). Additionally, each constant may be associated to a feature structure (noted between square brackets). For convenience, we also use here a graphical version of the language where the dominance relation is indicated with a dashed edge and the immediate dominance relation with a solid edge. Thus the *passive-with-2-arguments* lexical alternation is expressed in the language as:

$\phi pass - alt$
pred [cat = V; agent = i; theme = j]
$\wedge$ subj [cat = N; index = j]
$\wedge$ obl-obj [cat = PP; index = i]
$\wedge$ obl-prep [cat = by]
$\wedge$ $\phi pass - morph$
$\wedge$ $\phi subj - can$
$\wedge$ $\phi obl - pp - can$

This formula representing the lexical alternation describes four nodes, each of them associated with a feature structure and states three macros (noted in italics), the grammatical alternations: as the names suggest $\phi subj - can, \phi obl - pp - can, \phi pass - morph$, abbreviates respectively appropriate tree descriptions for a canonical subject, a canonical oblique PP and the morphology of a passive verb.



Relations between the lexical and grammatical alternations are performed through a naming convention. As constants's names are global to all the macros, some of

[7]Though to a lesser extent, (Saint-Dizier 1996; Saint-Dizier 1999) aims to be an adaptation to French of (Levin 1993). He has added further information and in particular he associates a thematic role to each argument.

them are used both in the lexical alternation and in the grammatical alternations. It is apparent here with the use of the *subj*, *obl-obj* and *pred* constants. In the lexical part we specify semantic information that concerns each of these nodes without paying attention to their position in a tree. The grammatical part mentions again the same nodes but, their position is specified in a tree without taking care of semantics.

As ϕpass-alt is now fully defined, its interpretation through a tree structure is given as tree (3) in Figure 1. The process of tree generation consists of building every (minimal) model satisfying the formula.

4.2 Generalization

The next step is to describe the full set of trees related to the passive alternation. Other possible trees for passive alternations can be described through the introduction of disjunctions e.g.:

$\phi pass - alt - gen$
pred[cat = V; agent = i; theme = j]
$\wedge$ subj[cat = N; index = j]
$\wedge$ obl-obj[cat =PP; index = i]
$\wedge$ obl-prep[cat = par]
$\wedge \phi pass - morph$
$\wedge(\phi subj - can \vee \phi subj - cltq \vee \phi subj - extr)$
$\wedge(\phi obl - pp - can \vee \phi obl - pp - extr)$

Where we further assume that $\phi subj - cltq, \phi subj - extr$ and $\phi obl - pp - extr$ abbreviates tree descriptions for a clitic subject, an extraposed subject and an extraposed oblique PP.

The translation to disjunctive normal form (DNF) show that intuitively, the trees described by $\phi pass - alt - gen$ are the trees for a passive with a canonical subject and an oblique canonical PP, a passive with a canonical subject and an extraposed oblique PP etc.[8]

(ϕpass-alt-core$\wedge\phi$subj-can$\wedge\phi$obl-pp-can$\wedge\phi$pass-morph)
$\vee$(ϕ pass-alt-core$\wedge\phi$subj-can$\wedge\phi$obl-pp-extr$\wedge\phi$pass-morph)
$\vee$(ϕ pass-alt-core$\wedge\phi$subj-cltq$\wedge\phi$obl-pp-can$\wedge\phi$pass-morph)
$\vee$(ϕ pass-alt-core$\wedge\phi$subj-cltq$\wedge\phi$obl-pp-extr$\wedge\phi$pass-morph)
$\vee$(ϕ pass-alt-core$\wedge\phi$subj-extr$\wedge\phi$obl-pp-can$\wedge\phi$pass-morph)
$\vee$(ϕ pass-alt-core$\wedge\phi$subj-extr$\wedge\phi$obl-pp-extr$\wedge\phi$pass-morph)

More generally, a wide set of trees can be described by using a single disjunctive formula.

It follows from that observation and our preceding considerations on monotonicity that we are now theoretically able to describe a full TAG lexicon using a single logical description: such a formula, describing the whole grammar, is a disjunction of all the possible lexical alternations defined for a given language.

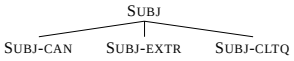
[8] In the following description, ϕpass-alt-core abbreviates the first four lines of ϕpass-alt-gen.

4.3 Lexical knowledge representation

The implementation used to test our ideas (Gaiffe et al. 2002) allows a structuring of the lexical knowledge encoded in tree descriptions. It relies on two devices: a monotonic inheritance hierarchy and a cancelling mechanism of resources and requirements.

Inheritance hierarchies have been used extensively to structure the lexicon of computational grammars and thus factorize information. Similarly, the tree descriptions used in our approach can be organised in an inheritance hierarchy where the nodes of the hierarchy are classes associated with tree descriptions and inheritance is just conjunction of the tree descriptions associated with the parents of the hierarchy node being considered.

For instance, a class expressing a subject in canonical position inherits information from a superclass such as subject[9], thus conforming to the semantics of an inheritance relation: it is natural to state that a canonical subject is a particular kind of subject.



The second device used in the implementation is a cancelling mechanism of resources and requirements. It is used to combine information from different hierarchies (one for lexical alternations, one for grammatical alternations). This is used for instance, to account for the relation between PASS-ALT on the one hand, the passive morphology and the disjunctions of all the possible realisations of subjects and oblique objects on the other hand. Intuitively, we want a mean to capture the fact that the class PASS-ALT describes a set of trees where realisations of subjects and oblique objects may vary.

4.4 An open question: principles

The developing framework has been used to produce TAG grammars that count around 500 trees. Closer examination of the grammars output by the system reveals an over-generation problem however in that some of the output trees are not linguistically correct. There are two reasons for this.

First given the complexity of even a medium size grammar, it becomes quickly very difficult to be certain that the stated descriptions are linguistically adequate and that the classes used in a given conjunction of lexical descriptions interact appropriately.

Second, existing tree generators build tree structures through a semantics taking its sources in (Rogers and Vijay-Shanker 1994). A formula interpreted in that way ensures that the generated trees are a set of finite trees satisfying this input formula and nothing more. But nothing in this language ensures that we are actually

[9]To get a comparison with the macros previously used, the notation in small caps represents a class that embeds the content of its corresponding formula: so PASS-ALT is the class that embeds ϕpass-alt.

describing elementary trees. To do this we should indeed state more constrained models, expressing additional interpretative conditions. Such models would give us insurance that a tree contains at least one anchor (lexicalization), that a root and foot node share the same label...

One way to overcome this over-generation problem is to use general linguistic principles to filter out linguistically illegitimate trees. Recall that in TAG, grammar rules are very empoverished (Indeed tree composition is done with only two operations: adjunction and substitution) and that subsequently, most of the linguistic content is described within elementary trees. From a theoretical perspective, it would thus be of major interest to define principles of well formation of trees that go far beyond the sole guarantee of describing finite trees. Target principles are, for instance, LFG's completeness-unicity, or a Predicate Argument Coocurrency Principle (PACP).

Besides universal constraints such as completeness-unicity, a TAG lexicon has to verify additional constraints that are language specific. For French, (Candito 1999) identifies constraints that are TAG counterparts to (Ross 1985) constraints on transformations or to (Perlmutter 1970) surface constraints on clitic ordering in Romance languages. These constraints are related to the interaction between independently described phenomenons. Such a constraint is for instance: *if there is an extracted argument and if there is a sentential argument then the latter has to be represented as a substitution node*, which is a standard manner to account for wh-islands constraints in TAG (Abeillé 2002; Frank 2002).

5 Comparison with related work

The idea of automating tree generation for TAG is not new. We review briefly the four existing systems.

The first two systems are built upon an non monotonic inheritance network to express structure sharing, patterned after the work of D. Flickinger (Flickinger 1987)[10] for phrase structure grammar. Metarules (Becker 1993; Becker 2000) are a way of emulating transformations over elementary trees. A set of base elementary trees is manually defined and a set of metarules is applied to them in order to output an extended set of elementary trees[11]. In the same register, (Evans et al. 2000) show that the problem of compact lexical representation in TAG can be handled using DATR, that is using a non-monotonic system designed for lexical representation. This proposal has lead to the compact encoding of a DTG lexicon (Smets and Evans 1998). Both approaches differ from ours in that they take their foundations in the use of lexical rules allowing to express non-monotonicity (erasing, rule ordering...), thus reflecting the usual structuring of the lexicon.

[10] See also (Sag and Wasow 1999; Bouma et al. 1998) for an up to date presentation in HPSG.

[11] The reader is reported to (Prolo 2002) for an extensive evaluation with XTAG. See also (Kinyon and Prolo 2002) for a more detailed comparison between metarules and metagrammar.

The two remaining systems are based on a tree description language. However both systems include extra layers of representation allowing to express non-monotonicity. Candito's metagrammar is the closest work to ours (Candito 1996; Candito 1999). First thought to be monotonic (Candito 1996), it eventually turned out not to be (Candito 1999), for above mentioned reasons reflecting the use of erasing rules. Her algorithm is procedural: first describing argument structure, then diathesis lexical rules and finally grammatical alternations. She catches many principles, but unfortunately they are partly mixed with her algorithm. This system has been proven to generate wide coverage grammars but add-ons that adjust the initial intuition make a formal system with unknown properties. Xia's LexOrg (Xia 2001) is two sided: a lexical side where lexical rules are applied on subcategorization frames and a grammatical side responsible for describing sub-trees. The two parts are linked by a translation module. For a classical, lexical-rule based approach, her system seems to be a good compromise. But she has underestimated the importance of language specific well-formation principles. As for Candito's system, her algorithm directly allows her to account for some constraints such as *there cannot be two extracted arguments in an elementary tree*. Unfortunately, there exists counter-examples to such a principle. (Abeillé 2002) mentions cases of double extraction in French.

6 Conclusion and Perspectives

Our proposal aims to specify a system of lexical knowledge representation for TAG. We have seen that alternations enable to state the lexicon using a monotonic framework.

Technically, monotonicity allows to use only a logical language dedicated to tree description for describing a TAG lexicon. For practical purposes, we add an inheritance hierarchy, defined as a macro layer, allowing to structure the organisation of the lexicon.

On the linguistical side, monotonicity casts a different perspective on lexical organisation. It specifically favours an in-depth exploration of Levin's alternations through its integration in syntax.

Subsequent work demands to define a formal system specifically crafted for tree generation where the goal is to give a satisfactory account for the representation of principles.

7 Acknowledgements

The author would like to thank Claire Gardent and two anonymous reviewers. Their support and their comments contribute to clarify and improve this work.

References

Abeillé, A. (2002). *Une grammaire d'arbres adjoints pour le français*. Paris: Editions du CNRS.

Becker, T. (1993). *HyTAG : A new Type of Tree Adjoining Grammars for Hybrid Syntactic Representation of Free Word Order Language*. Ph. D. thesis, Universität des Saarlandes.

Becker, T. (2000). Patterns in metarules for tag. In A. Abeillé and O. Rambow (Eds.), *Tree Adjoining Grammars. Formalisms, Linguistic Analysis and Processing*. Stanford: CSLI.

Bouma, G., F. van Eynde, and D. Flickinger (1998). Constraint-based lexicons. In F. van Eynde, D. Gibbon, and I. Schuurman (Eds.), *Lexicon Development for Speech and Language Processing*, Dordrecht. Kluwer. 1999.

Candito, M.-H. (1996). A principle based hierarchical representation of LTAGs. *Proceedings of COLING 96*.

Candito, M.-H. (1999). *Organisation Modulaire et Paramétrable de Grammaires Electroniques Lexicalisées*. Ph. D. thesis, Université de Paris 7.

Evans, R., G. Gazdar, and D. Weir (2000). 'lexical rules' are just lexical rules. In A. Abeillé and O. Rambow (Eds.), *Tree Adjoining Grammars. Formalisms, Linguistic Analysis and Processing*. Stanford: CSLI.

Flickinger, D. (1987). *Lexical Rules in the Hierachical Lexicon*. Ph. D. thesis, Stanford University.

Frank, R. (2002). *Phrase Structure Composition and Syntactic Dependencies*. Boston: MIT Press.

Gaiffe, B., B. Crabbé, and A. Roussanaly (2002). A new metagrammar compiler. *Proceedings of TAG+6*.

Gardent, C. and L. Kallmeyer (2003). Semantic construction in feature-based tree adjoining grammar. *Proceedings of the 10th conference of the European Chapter of the Association for Computational Linguistics*.

Joshi, A. K., L. S. Levy, and M. Takahashi (1975). Tree adjunct grammars. *Journal of the Computer and System Sciences 10*, 136–163.

Joshi, A. K. and Y. Schabès (1997). Tree adjoining grammars. In G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*. Berlin: Springer Verlag.

Kinyon, A. and C. Prolo (2002). A classification of grammar development strategies. *Proc. GEE 2002*.

Levin, B. (1993). *English Verb Classes and Alternations*. The University of Chicago Press.

Perlmutter, D. (1970). Surface structure constraints in syntax. *Linguistic Inquiry 1*, 187–255.

Prolo, C. (2002). Systematic grammar development int the XTAG project. *Proceedings of COLING'02*.

Rogers, J. and K. Vijay-Shanker (1992). Reasoning with descriptions of trees. *Proceedings of ACL 92*.

Rogers, J. and K. Vijay-Shanker (1994). Obtaining trees from their descriptions: an application to tree-adjoining grammars. *Computational Intelligence 10*, 401–421.

Ross, J. (1985). *Infinite Syntax !* Dordrecht: Reidel.

Sag, I. and T. Wasow (1999). *Syntactic Theory. A Formal Introduction*. Stanford: CSLI Publications.

Saint-Dizier, P. (1996). Verb semantic classes in french. version 2. Technical report, IRIT – CNRS, Toulouse.

Saint-Dizier, P. (1999). Alternations and verb semantic classes for french : Analysis and class formation. In P. Saint-Dizier (Ed.), *Predicative Forms in Natural Language and Lexical Knowledge Bases*. Dordrecht: Kluwer.

Smets, M. and R. Evans (1998). A compact encoding of a dtg grammar. *Proc. TAG+4*.

Vijay-Shanker, K. (1987). *A study of Tree Adjoining Grammar*. Ph. D. thesis, Department. of computer and information science, University of Pennsylvania.

Vijay-Shanker, K. and Y. Schabes (1992). Structure sharing in lexicalized tree-adjoining grammars. *COLING 92*.

Xia, F. (2001). *Automatic Grammar Generation from two Different Perspectives*. Ph. D. thesis, University of Pennsylvania.

XTAG Research Group (2001). A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

On a Unified Semantic Treatment of Donkey Sentences in a Dynamic Framework

FABIO DEL PRETE
University of Milan
fabio_del_prete@hotmail.com

ABSTRACT. In this paper I take into account some irregularities concerning the distribution of so called ∀-readings and ∃-readings on the set of relative clause donkey-sentences. Dynamic semantics is assumed to provide the correct analysis of donkey dependencies. I defend a theory of the distributional facts based on a notion of dynamic monotonicity, and introduce for this end an algorithmic form of quantifier domain restriction, whose representation at LF provides a key to explain away counter-examples. No ambiguity is introduced in the semantics either of pronouns, or of quantificational determiners.

1 The problem

Commonly held semantic judgments concerning truth-conditions of some donkey-sentences pose a serious challenge to any theory of donkey-anaphora in which uniform semantic representations are construed for such sentences. Take the following representative pair:

(1) Every person who has a credit card pays a heavy charge for it.

(1') $\forall x[(person(x) \wedge \exists y(c\text{-}card(y) \wedge have(x,y))) \rightarrow \forall y((c\text{-}card(y) \wedge have(x,y)) \rightarrow pay\text{-}charge\text{-}for(x,y))]$

(2) Every person who has a credit card pays the bill with it.

(2') $\forall x[(person(x) \wedge \exists y(c\text{-}card(y) \wedge have(x,y))) \rightarrow \exists y(c\text{-}card(y) \wedge have(x,y) \wedge pay\text{-}with(x,y))]$

According to intuition, (1) shows a preference for a ∀-reading, formally expressed in (1'), whereas (2) seems to require an ∃-reading construal, given in (2'). The latter sentence is problematic for every classical approach to donkey-anaphora: Kamp's DRT (1981) and Groenendijk&Stokhof's DPL (1991), on the dynamic side, as well as the variants of the e-type approach proposed by Heim (1990) and Neale (1990), on the static side, would all indeed converge in predicting a ∀-reading for both (1) and (2). All these approaches lack a formally

Proceedings of the Eighth ESSLLI Student Session.
Balder ten Cate (editor)

implemented explanation of how ∃-readings of sentences with 'every' come about.[1]

In the case at hand, the logical formulas hypothetically assumed as adequate semantic representations are not isomorphic, in spite of the structural similarity between the two sentences. The question thus arises, whether a uniform semantic representation of (1) and (2) is conceivable, which has the potentiality of accommodating both readings (1') and (2'). More generally, we are faced with the question whether a principled explanation of how the two readings are distributed can be provided. Hereafter, I will speak of 'distribution' *simpliciter*, meaning the distribution of ∀/∃-readings on the set of donkey-sentences.

2 A monotonicity-based theory of distribution

I assume here that dynamic semantics provides the correct approach to the analysis of donkey sentences; I stick further to a theory of distribution based on extension of monotonicity to dynamic contexts[2]. According to Kanazawa (1994), the distributional phenomena should be expected to correlate with monotonicity patterns of determiners. In general, whether a donkey-sentence $[_{det}\delta][_{N'}\gamma][_{VP}\chi]$ has a ∀-reading or an ∃-reading is relevant for the quantifier which interprets $[_{det}\delta]$ to preserve the usual monotonicity behaviour of the determiner itself. Take for instance (3), which has intuitively a ∀-reading. If one accepts Kanazawa's theory, one has principled reasons to expect that (3) should have such a reading. Roughly, his argument runs as follows: suppose (3) had the ∃-reading; as a consequence, the downward monotonicity inference from (3) to (3c), which is intuitively felt to hold, could not be valid:

(3) Every man who has a son loves him.
(3b) Every man who has a teen-age son is a man who has a son.
(3c) Every man who has a teen-age son loves him.

Suppose that in the discourse domain there be a man *x* who has just two sons *y*, *z* : *y* is a teen-ager, *z* is six years old, and *x* loves only *z*. In a situation so characterized, fixed an ∃-reading construal for both (3) and (3c), (3) could be true, but (3c) could not; but if we assume that the base generated reading for a sentence with 'every' is the universal one, we are in a position to explain the validity of a monotonicity inference like the one exemplified by (3)-(3c): under such an assumption, we could not have anymore the premiss true and the conclusion false, since *x* would be a counter-instance for (3) too.

[1] The extended e-type approach put forward by Lappin&Francez (1994) can indeed cope with these data; but it seems to me that it gains empirical adequacy at the price of positing a semantic ambiguity in donkey-pronouns, whose representations at LF happen to involve semantically different types of functions (*maximal* functions vs. *choice* functions).
[2] See Kanazawa (1993, 1994).

The simple example I have thus given shows that there are important reasons for expecting that sentences with 'every' have the ∀-reading, and more generally that the distributional phenomena be not random, but conform to an ideal pattern correlating with the exact distribution of monotonicity properties on the set of quantificational determiners. These reasons concern preservation of a particular semantic structure in the transition from simple (non-donkey) quantificational contexts to quantified sentences with donkey-anaphora. The interesting point is that semantic properties of quantifiers such as monotonicity have been assuming a central role in contemporary semantic theory, so that the requirement that these properties hold of quantifiers even when they occur in donkey-contexts appears as perfectly natural and coherent with current assumptions concerning the role of so-called semantic universals. However, an objection against the possibility of extending the concept of monotonicity to donkey-sentences (hence to a concept of *dynamic monotonicity*) may come from consideration of the following invalid syllogism[3]:

(A) Every man who has a house is a man who has a garden.
(B) Every man who has a garden sprinkles it on Sundays.
(C) So(???), every man who has a house sprinkles it on Sundays.

Dynamic semantics allows one to explain the inferential failure in (A)-(C) in a natural, non-*ad hoc* way, without giving up the idea that determiners maintain their usual monotonicity behaviour in donkey-contexts. The reason of the failure is that the premiss (A) expresses a simple inclusion between the set of house-owners and the set of garden-owners, the latter being the static denotation of (B)'s N'-restriction 'man who owns a garden', whereas a proper semantic analysis of (B) would require to take into account the anaphoric potential of such constituent, hence its *dynamic* denotation. To put it in an informal way, one can say that in (B) the N'-restriction on 'every' sets up pairs of referents $\langle x, y \rangle$, such that x is a man and y is a garden that x has, whereas each of such pairs is relevant for the evaluation of the VP. Hence, for the purpose of drawing (left) monotonicity inferences from donkey-sentences, one should consider inclusions between given sets of pairs of this kind. In what follows, I will provide a formalization of left dynamic monotonicity principles, hence a more precise characterization of the condition thus informally expressed.

3 The Monotonicity Principle in a dynamic setting

Following Kanazawa (1994), I assume a meta-semantic principle whose function is that of regulating the distribution of the two readings on the pure semantic ground represented by monotonicity of quantificational determiners. To formulate the principle in a semi-formal manner, we need firstly to introduce

[3] This syllogism is due to van Benthem.

some logical notation. In our language we want symbols for both static and dynamic generalized quantifiers. We will use dynamic generalized quantifiers to model donkey-sentences, so as to represent donkey-dependencies as binding relations. The main characteristics of a DGQ is indeed that of allowing variables occurring free in its right argument to be bound to existential quantifiers in its left argument. For each SGQ **Q**, our dynamic logic has a corresponding DGQ $\boldsymbol{Q}$; to say that $\boldsymbol{Q}$ corresponds to **Q** as its dynamic counterpart means that Qx{Φ}{Ψ} and Qx{Φ}{Ψ} get the same truth-conditions when Ψ does not contain free occurrences of a variable which is accessible to existential quantifiers in Φ. Logical signs comprehend static and dynamic connectives: on the static side, we have the usual connectives ¬, ∧, →,…, with their usual interpretations; amongst dynamic connectives we have **;** and ⇒ for dynamic conjunction and dynamic implication respectively. Dynamic existential quantifier is symbolized as $\boldsymbol{E}_{\cdot}$. Definitions of **;** , ⇒ and $\boldsymbol{E}$ are as in Groenendijk&Stokhof (1991); given the relevance of these logical operators in modeling donkey sentences, I report below their semantic clauses, besides the one for the meta-connective ≤, which is put for dynamic meaning inclusion and play a crucial role in the formalization of left dynamic monotonicity:

(a) $[[E\text{x}\varphi]] = \{\langle g,h\rangle: \exists k\ k[x]g\ \&\ \langle k,h\rangle \in [[\varphi]]\ \}$
(b) $[[\varphi;\psi]] = \{\langle g,h\rangle: \exists k\ \langle g,k\rangle \in [[\varphi]]\ \&\ \langle k,h\rangle \in [[\psi]]\}$
(c) $[[\varphi \Rightarrow \psi]] = \{\langle g,h\rangle: h=g\ \&\ \forall k\ \langle g,k\rangle \in [[\varphi]] \rightarrow \exists m \langle k,m\rangle \in [[\psi]]\}$
(d) $[[\varphi \leq \psi]] = \{\langle g,h\rangle: g=h\ \&\ \forall s\ \langle g,s\rangle \in [[\varphi]] \Rightarrow \langle g,s\rangle \in [[\psi]]\}$

The combined effect of (a)-(c) is that the following (static) equivalences hold:

$$\left.\begin{array}{l}(E\text{x}\varphi) \Rightarrow \psi \equiv \forall \text{x}(\varphi \Rightarrow \psi) \\ \\ (E\text{x}\varphi);\psi \equiv E\text{x}(\varphi;\psi)\end{array}\right\} \text{ whether x doesn't occur free in } \psi \text{ or not}$$

As a purely language-internal matter, one finds the question of how to define the semantics of a DGQ $\boldsymbol{Q}$ so as to create the possibility of indirect binding relations between its two arguments. There are a lot of definitional options from a technical point of view, but having to express either ∀-readings or ∃-readings, we can restrict ourselves to two possibilities:

(Def.1) Qx{Φ}{Ψ} $=_{\text{def}}$ Qx{Φ}{Φ⇒Ψ}
(Def.2) Qx{Φ}{Ψ} $=_{\text{def}}$ Qx{Φ}{Φ;Ψ}

Assuming definition (Def.1) for DGQs, we are able to represent the ∀-reading of (1) by means of the following logical structure:

(1°) *EVERY* x {person(x) ; *E*y c-card(y) ; have(x,y)}{pay-charge-for(x,y)}
≡ EVERY x {person(x) ∧ ∃y(c-card(y) ∧ have(x,y))} {∀y((c-card(y) ∧ have(x,y)) → pay-charge-for(x,y))}

If we opt to define DGQs as in (Def.2), we can model a sentence like (2) so as to represent its apparent ∃-reading. The linguistic structure is the same as in the preceding case, but it will have structurally different semantic aspects:

(2°) *EVERY* x{person(x) ; *E*y credit-card(y) ; have(x,y)}{pay-with(x,y)}
≡ EVERY x {person(x) ∧ ∃y(c-card(y) ∧ have(x,y))}{∃y((c-card(y) ∧ have(x,y) ∧ pay-with(x,y))}

General availability of the two kinds of reading imposes the following condition: for our logical system to be materially adequate, it must contain both types of DGQs. Indeed, given the apparent availability of both reading-types for sentences with equi-monotone determiners, one may be induced to admit, for each static quantifier **Q**, two dynamic counterparts: $Q_{\forall}$ (introduced through (Def.1)) and $Q_{\exists}$ (introduced through (Def.2)). But this would reduce to introduce a lexical ambiguity in every determiner. In particular, one would have two different dynamic interpretations for the same item 'every', only one of which (namely $EVERY_{\forall}$) would match with the familiar quantifier EVERY with respect to monotonicity (this point has been established in discussing the inference from (3) to (3c); for the inference to go through, the reading of the premiss must be the universal one, what comes down to the claim that only $EVERY_{\forall}$ preserves the familiar monotonicity properties of EVERY).

We are now in a position to formulate the meta-semantic principle which we want to take as a rule governing the distribution of the two reading-types. We state first the dynamic monotonicity principles (DMP1) and (DMP2), as well as the semantic fact (F) and the definition (D):

(DMP1) $\boldsymbol{Q}$ is ↓DMON iff $\Phi' \leq \Phi$ & $\boldsymbol{Q}$ x {Φ}{Ψ} ⊨ $\boldsymbol{Q}$ x {Φ′}{Ψ}
(DMP2) $\boldsymbol{Q}$ is ↑DMON iff $\Phi \leq \Phi'$ & $\boldsymbol{Q}$ x {Φ}{Ψ} ⊨ $\boldsymbol{Q}$ x {Φ′}{Ψ}[4]
(F) Given a SGQ **Q** which is left monotone and right monotone in either direction, just one of the two DGQs $\boldsymbol{Q}_{\forall}$, $\boldsymbol{Q}_{\exists}$, which can be defined on the basis of **Q**, preserves the full monotonicity pattern of **Q**
(D) '$\boldsymbol{Q}$ is *monotonically congruent* with **Q**' ≡ '$\boldsymbol{Q}$ preserves the full mp of **Q**'

MONOTONICITY PRINCIPLE (MP)

Be $\boldsymbol{Q}_{\forall}$ ($\boldsymbol{Q}_{\exists}$) the DGQ monotonically congruent with **Q**: it is stipulated that $\boldsymbol{Q}_{\forall}$ ($\boldsymbol{Q}_{\exists}$) is the only dynamic counterpart of **Q** allowed to enter semantic computation for a donkey-sentence [δ N' VP] whose main determiner δ is conventionally mapped onto **Q**.

This formulation corresponds very closely to the one given by Kanazawa (1994), but I state the principle more explicitly as a choice-criterion.

[4] The labels ↓DMON and ↑DMON are to be taken as abbreviations for 'dynamically downward monotone on the left' and 'dynamically upward monotone on the left', respectively.

4 Quantifier Domain Restriction as a way out from anomalies

In virtue of its simplicity and principled character, the monotonicity-based theory meets empirical difficulties. If the instance of preserving monotonicity inferences is taken seriously, one has to provide an explanation of distributional anomalies such as one can find in sentences like (2). The explanation I propose is based on the hypothesis that a certain form of *quantifier domain restriction* (QDR) has come into play in the transition from the base-generated meaning of α to the output of α's interpretation in context (where α is whatever sentence for which an unexpected reading has been observed). Following Stanley and Szabó (2000), I assume that quantified sentences (including the donkey cases) are mapped onto LFs characterized by free domain variables of the form 'f(x)' ('f' and 'x' being variables of type **<e,<e,t>>** and **e**, respectively), and show that, given special assumptions on the binding of 'x' and on the evaluation of 'f', the deviant readings can be represented within the dynamic framework referred to above, without positing any change in the type of the DGQs involved. For the purpose of introducing my proposal in an informal way, let me focus on sentence (2):

(2) Every person who has a credit card pays the bill with it.

The most salient reading of this sentence is paraphrasable as 'every person having a credit card uses one credit card she has to pay the bill'. My idea is that we are disposed to accept such a paraphrase because of our *world knowledge*, which excludes as a rule the fact of a person paying a single bill with more than one card. An utterance-context for (2) is thus supposed to include the assumption (call it **A**) that a person pays a given bill with *at most one* credit card, whereas **A** must be thought as something having the status of a presupposition publicly accessible; it is properly **A**'s public accessibility what determines a hearer to interpret (2) as stating something about no more than one object of the relevant kind. The intuition underlying my proposal is that **A** interacts with the structural elements in (2)'s LF, and that such interaction takes the form of a domain restriction accomplished through accommodation of the content of **A**.

I proceed now to show how the explanatory proposal thus illustrated can be formally implemented. Consider firstly the following rough LF for (2):

(2_{LF}) EVERY x {person(x) $\wedge$ $\exists$y(c-card(y) $\wedge$ have(x,y))} {$\exists$!y(c-card(y) $\wedge$ have(x,y) $\wedge$ pay-with(x,y))}

(2_{LF}) represents the intuitively right reading of (2), but it seems completely unrelated with the LF one would obtain if one used the DGQ $EVERY_{\forall}$. Anyhow, if we take into account the accommodation of presupposition **A** referred to above, and further conceptualize the process at issue as one of QDR, we shall be

in a position to capture the apparent ∃-reading of (2) within the DGQ-analysis that MP sanctions. A natural way of accomplishing accommodation of the referred presupposition at the level of the restriction 'person who had a credit card', is as follows: in the underlying logical structure of such N', two variables *x*, *y*, are introduced; given that *x* is fixed to be a person and *y* a credit card owned by *x*, presupposition **A** may be accommodated by supposing that *x* does not pay her bill with any of her possibly multiple credit cards *z*, where *z* is distinct from the card *y* previously introduced; this latter condition comes down to the proviso that *x* pays her bill with no more than one credit card. A sentence we may think to be derivable from (2) via a process of accommodation taking the form above indicated is (2*), given along with the standard DGQ-analysis:[5]

(2*) Every person who has [a credit card]$_i$ and who doesn't pay the bill with [any [other]$_j$ of her credit cards]$_j$ pays the bill with [it]$_i$.

(2*$_{LF}$) $EVERY_{\forall}$ x { person(x) ; Ey card(y) ; have(x,y) ; $\forall$z((card(z) ; have(x,z) ; z≠y)⇒¬pay(x,z)) } {pay(x,y)}

As can be easily checked, (2*) has a ∀-reading, represented in (2*$_{LF}$) (whatever may be the credit card *y* introduced by the indefinite NP, it is true that the owner of *y* pays her bill with *y*, if she doesn't use any other of her cards). (2*$_{LF}$) can be claimed to bear a particular relation of equivalence to the naïve LF (2$_{LF}$): let's call it *pragmatic equivalence*, to distinguish it from strict semantic equivalence. The two logical structures cannot be said to be semantically equivalent in a strict sense, (2$_{LF}$) being about persons having credit cards, while (2*$_{LF}$) being about persons having credit cards and using at most one to pay their bills. Nonetheless, I say they are *pragmatically equivalent*, because of a background assumption according to which any person pays a single bill with at most one credit card: because of this assumption, (2$_{LF}$) can be seen as pragmatically restricted to persons who pays with at most one credit card, when they pay at all). Given the pragmatic equivalence between (2*$_{LF}$) and (2$_{LF}$), the former, as well as the latter, can be assumed to fix (2)'s semantic content (with respect to a context characterized as containing presupposition **A**). I propose to take the former as an LF-representation of (2), modulo QDR. What we gain if we accept this option is to keep our LF-analysis consistent with MP.

Consider now sentence (4), which is another problematic case for the monotonicity-based account.

(4) No person who has an umbrella leaves it home on a day like this.

[5] In the structural description and in the logical formulas I underline the adjoined predicate, in order to mean that it is contextually integrated; I also give the relevant LFs in a non-linear notation, for the sake of perspicuity; in such a notation, the lines in the restrictive part of a DPL-formula are to be intended as (dynamically) conjoined in the top-down order.

(4_{LF}) NO x {person(x) ∧ ∃y[umbrella(y) ∧ have(x,y)]} {∀y[(umbrella(y) ∧ have(x,y)) → leave'(x,y)]}

(4) seems to require the ∀-reading construal specified above, in spite of the initial determiner 'no'. Nonetheless, it is straightforward to show that the same QDR-based strategy applied to (2) allows us to represent the intuitive reading of (4) by means of the DGQ $NO_{\exists}$. In the present case, we focus on the existence of a presupposition to the effect that a person takes at most one umbrella with her on a rainy day. The presupposition gets accommodated via QDR in the following way: given that in the restrictive part of the main quantifier *x* is fixed to be a person and *y* an umbrella owned by *x*, the referred presupposition may be integrated by supposing that *x* leaves *z* home, where *z* is any of *x*'s umbrellas distinct from the umbrella *y* previously introduced.

(4*) No person who has [an umbrella]$_i$ and who leaves [each [other]$_j$ of her umbrellas]$_j$ home, leaves [it]$_i$ home on a day like this.

(4^*_{LF}) $NO_{\exists}$ x { person(x) ; *E*y umbrella(y) ; have(x,y) ; ∀z((umbrella(z) ; have(x,z) ; z≠y)⇒ leave'(x,z)) } {leave'(x,y)}

An ∃-reading construal for (4*) is materially adequate; indeed, the LF-representation (4^*_{LF}) expresses its intuitive reading, and since it is pragmatically-equivalent to (4_{LF}) – under the background assumption that any man takes at most one umbrella with him when going out –, it can be used to fix the truth-conditions of (4). As for the preceding case, I propose to assign (4^*_{LF}) to (4) as a possible LF-representation, modulo QDR. The interesting thing in this connection is that (4^*_{LF}) is construed with the DGQ $NO_{\exists}$, which is the dynamic counterpart of **NO**, according to MP.

In the QDR-based analyses given thus far, I have been presupposing that (2) and (4) share an interpretational property, which can be highlighted by rephrasing the latter as an *every*-sentence, along the lines indicated below:

(2) Every person who has a credit card pays the bill with it.

(2_{LF}) EVERY x {person(x) ∧ ∃y[c-card(y) ∧ have(x,y)]} {**∃!y**[c-card(y) ∧ have(x,y) ∧ pay'(x,y)]}

(4) Every person who has an umbrella takes it with her on a day like this.

(4_{LF}) EVERY x {person(x) ∧ ∃y[umbrella(y) ∧ have(x,y)]} {**∃!y**[umbrella(y)∧have(x,y)∧take'(x,y)]}

The rough LFs (2_{LF}) and (4_{LF}) ascribe to the meaning of both sentences a 'uniqueness feature' (every individual in the domain bears a given relation to a *unique* object of a given kind). I have assumed that the existence of such interpretive feature depends on speakers' world-knowledge (normally, a person pays the bill with *at most one* credit card, and takes *at most one* umbrella with

her when she goes out on a rainy day; that's all). The complex predicates that I have supposed to be contextually integrated in the restrictions capture indeed the uniqueness feature, in virtue of their common logical structure (no individual in the locally determined domain *D* can bear the pertinent relation *R* to more than one object of the relevant kind *X*, since a contextual, non-local factor introduces *a priori* the constraint: *ceteris paribus*, an individual in *D* bears *R* to no more than one object *X*). The structure of those complex predicates can be expressed by means of the meta-formula (DR), where 'φ' stays for the N'-restriction of the antecedent indefinite NP, 'ψ' for the transitive verb of the antecedent containing clause, and 'χ' may stay alternatively either for the verb **V** of the main clause or for its negation **¬V**.

(DR) $\forall z((\varphi(z);\psi(x,z);z\neq y)\Rightarrow\chi(x,z))$

But if we applied (DR) as a rule whenever we found a sentence preferring the unexpected reading, we would meet some difficulties. Take an *every*-sentence with a surface ∃-reading, but lacking uniqueness implicatures of the sort considered above; (5), given along with the naïve representation (5_{LF}), is a case in point:

(5) Every person who had a dime put it in the parking meter.

(5_{LF}) EVERY x {person(x) ∧ ∃y[dime(y) ∧ have(x,y)]} {∃y[dime(y) ∧ have(x,y) ∧ put'(x,y)]}

If we derived now a sentence like (5*), through mechanical application of (DR) to (5), we would get an unlikely result:

(5*) Every person who had [a dime]$_i$ <u>and did not put [any [other]$_j$ of his dimes]$_j$ in the parking meter</u> put [it]$_i$ in the parking meter.

The problem is that there seems to be no background assumption according to which any person puts at most one dime into the parking meter on a given occasion. For this reason, a restriction like the one articulated in (5*), obtained by means of the rule (DR), results to be too strong, having the effect of narrowing down the quantifier domain too dramatically. This effect reduces to weakening the truth-conditions of the original sentence. Nonetheless, it would be very strange if the ∃-reading of (5) had nothing to do with a QDR-effect. After all, it seems plausible to assume the existence of a background assumption concerning normal behaviour of persons in a parking lot, where any of such persons is supposed to put in the parking meter just as many dimes as it needs (hence, not necessarily all of her dimes); moreover, we intuitively judge that an assumption like this *is* relevant to the ∃-reading of (5). I suggest that, as in the preceding cases we were able to account for the unexpected readings by allowing speakers' presuppositions relevant to the subject-matter to interact with grammatical structure, so one should be able to explain away (5)'s ∃-reading by giving a

suitable implementation at LF of the presupposition referred to above. It is indeed straightforward to provide such an implementation: given that in the restrictive part of the main quantifier *x* is fixed to be a person and *y* a dime owned by *x*, the presupposition may be integrated by supposing that *x* did not use any *quantity* of other dimes (each different from dime *y*) which were adequate for the purpose. In order to give a logical translation of the relevant verbal structure, one needs two ingredients: a) a notation for representing quantities of objects (predicates and variables over *i-sums*), b) a unary predicate applying to *i-sums*, which should express a general condition of pragmatic adequacy to whatever relevant purpose. I provide here the required notation:

(a) $\{\Sigma_i\}_{i\in N}$ is a set of variables for *i-sums*
(b) the binary predicate constant 'Π' denotes the *atomic-part* relation: 'Π(x,Σ)' means that the individual x is an atomic part of the *i-sum* Σ
(c) the unary operator '*' is prefixed to a distributive *n*-ary predicate over individuals, giving an *n*-ary predicate with one place for *i-sums*
(d) the interpretation of a formula '*R(..,Σ,..)' is fixed by the formula '$\forall$x[Π(x,Σ)$\rightarrow$R(..,x,..)]'
(e) the unary predicate constant 'Ω' denotes a context-sensitive property: with respect to a context **c**, 'Ω(Σ)' means that Σ is large enough for a **c**-salient purpose

In virtue of (a)-(e), we are in a position to construe an LF for the explicitly restricted version (5°):

(5°) Every person who had [a dime]$_i$ and did not put [any sufficient quantity of [other]$_j$ dimes]$_j$ in the parking meter put [it]$_i$ in the parking meter.

$$(5°_{LF}) \quad EVERY_{\forall}x \left\{ \begin{array}{l} \text{person}(x) \\ E\text{y d(y) ; h(x,y)} \\ \forall\Sigma((*d(\Sigma);*h(x,\Sigma);\Omega(\Sigma);\neg\Pi(y,\Sigma))\Rightarrow\neg*p(x,\Sigma)) \end{array} \right\} \{p(x,y)\}$$

As for the previous QDR-based analyses, I propose to consider ($5°_{LF}$) as a possible semantic representation of sentence (5), modulo QDR; I do so in virtue of its pragmatic equivalence to the immediate representation (5_{LF}).

The treatment proposed for sentence (5) can be generalized to the previous cases, where a uniqueness implicature was said to be present. The extension can be made by specifying an algorithmic procedure of QDR. The two algorithms given below provide a formalization of the accommodation processes which I take to be responsible for the apparent reading-shifts[6].

[6] I recall that I assume LFs with domain variables for quantified sentences; domain variables occupy a position in the restriction of quantifiers, and have the form f(x) – where 'f' and 'x' have semantic type **<e,<e,t>>** and **e**, respectively –.

$$Q_{\forall}\,x\ \{\varphi(x);E y_i\ \psi(x,y_i);f(y_j)(x)\}\{\chi(x,y)\}\ \Rightarrow_{RES\forall}$$

$$\Rightarrow_{RES\forall}\ Q_{\forall}\,x \left\{ \begin{array}{l} \varphi(x) \\ E y\ \psi(x,y) \\ \forall\Sigma((^*\psi(x,\Sigma);\ \Omega(\Sigma);\neg\Pi(y,\Sigma))\rightarrow\neg^*\chi(x,\Sigma)) \end{array} \right\}\ \{\chi(x,y)\}$$

$$Q_{\exists}\,x\ \{\varphi(x);E y_i\ \psi(x,y_i);f(y_j)(x)\}\{\chi(x,y)\}\ \Rightarrow_{RES\exists}$$

$$\Rightarrow_{RES\exists}\ Q_{\exists}\,x \left\{ \begin{array}{l} \varphi(x) \\ E y\ \psi(x,y) \\ \forall\Sigma((^*\psi(x,\Sigma);\ \Omega(\Sigma);\neg\Pi(y,\Sigma))\rightarrow{}^*\chi(x,\Sigma)) \end{array} \right\}\{\chi(x,y)\}$$

$RES_{\forall}$ and $RES_{\exists}$ describe a structural procedure for evaluating the function variable 'f' in the quantifier domain variable '$f(y_j)$'; moreover, both algorithms sanction the binding of the individual variable 'y_j' to the quantifier 'Ey_i', thus imposing the further condition i=j. More precisely, the derivations given above go through if we assume the following conditions:

(a) $[[f]]^{g_c} = \left\{ \begin{array}{l} \lambda y.\lambda x.\forall\Sigma((^*\psi(x,\Sigma);\ \Omega(\Sigma);\neg\Pi(y,\Sigma))\rightarrow\neg^*\chi(x,\Sigma)) \\ \text{if the sentence is of the form } Q_{\forall}\,x\ \Phi\ \Psi \\ \\ \lambda y.\lambda x.\forall\Sigma((^*\psi(x,\Sigma);\ \Omega(\Sigma);\neg\Pi(y,\Sigma))\rightarrow{}^*\chi(x,\Sigma)) \\ \text{if the sentence is of the form } Q_{\exists}\,x\ \Phi\ \Psi \end{array} \right.$

(b) i=j

I intend the two algorithms to describe a procedure of QDR which is contextual in nature, though it happens to be expressible in a form under which the covert restrictive predicates are reconstructed in a syntax-driven fashion. The algorithmic expression of the two processes mustn't make one think that all happens there within the realm of syntax, so to speak. The main factor calling for restrictions of this special kind resides typically in the structure of utterance-contexts. The fact that I recognize the restrictions at issue to have a contextual character, and in particular to depend on the existence of certain presuppositions, while describing at the same time a uniform procedure which happens to be sensitive to grammatical structure, should not be regarded as denoting incoherence: as I have shown with my sample-analyses, it is a matter of fact that the presupposition π charged with the apparent reading-shift of the sentence α bears such a relation to α's subject-matter, that it can be expressed under a form recoverable from α's linguistic structure; this fact concerning formal expressibility of π explains how accommodation of π can be represented in the final analysis via expansion of the quantifier restriction according to a syntactic algorithm.

Appendix

I provide here a sample-derivation of ($5°_{LF}$) as semantic representation of (5) with respect to a context satisfying the presupposition (π). The derivation is intended to show that there is no necessary contrast between the idea that accommodation of a contextually supplied assumption occurs and the idea that the resulting restriction can be equated to the output of a syntactic algorithm.

(5) Every person who had a dime put it in the parking meter.

(5_{LF}) $EVERY_{\forall}$ x {person(x) ; *E*ydime(y) ; have(x,y)} {put'(x,y)}

(π) Every person who has a dime put in the parking meter just as many of her dimes as it needs.

(π_{LF}) $\forall x\{[p(x)\ ;\ Ey\ d(y)\ ;\ h(x,y)] \Rightarrow Ey[d(y)\ ;\ h(x,y)\ ;\ \forall\Sigma((*d(\Sigma)\ ;\ *h(x,\Sigma)\ ;\ \Omega(\Sigma)\ ;\ \neg\Pi(y,\Sigma)) \Rightarrow \neg*put'(x,\Sigma))]\}$

($5_{LF}'$) $EVERY_{\forall}$ x { person(x); *E*y dime(y) ; have(x,y); (π_{LF}) } {put'(x,y)}

($5_{LF}''$) $EVERY_{\forall}$ x { person(x); *E*y dime(y) ; have(x,y); *E*y[d(y) ; h(x,y) ; $\forall\Sigma$((*d(Σ) ; *h(x,Σ) ; $\Omega(\Sigma)$; $\neg\Pi(y,\Sigma)) \Rightarrow \neg$*put'(x,$\Sigma$))] } {put'(x,y)}

($5°_{LF}$) $EVERY_{\forall}$x { person(x); *E*y dime(y) ; have(x,y); $\underline{\forall\Sigma((*d(\Sigma);*h(x,\Sigma);\Omega(\Sigma);\neg\Pi(y,\Sigma))\Rightarrow\neg*p'(x,\Sigma))}$ } {put'(x,y)}

References

Barker, C.: 1996, "Presuppositions for Proportional Quantifiers", in *Natural Language Semantics* **4**, 237-259.

Chierchia, G.: 1995, *Dynamics of Meaning: Anaphora, Presupposition, and the Theory of Grammar*, The University of Chicago Press, Chicago.

Groenendijk, J. and Stokhof, M.: 1991, "Dynamic Predicate Logic", in *Linguistics and Philosophy* **14**, 39-100.

Heim, I.: 1990, "E-type Pronouns and Donkey Anaphora", in *Linguistics and Philosophy* **13**, 137-177

Kamp, H.: 1981, "A Theory of Truth and Semantic Representation", in J. Groenendijk, T.M.V. Janssen, M. Stokhof (eds.), *Formal Methods in the Study of Language*, Foris, Dordrecht.

Kanazawa, M.: 1993, "Dynamic Generalized Quantifiers and Monotonicity", in M. Kanazawa and C. J. Piñón (eds.), *Dynamics, Polarity and Quantification*, Center for the Study of Language and Information, Stanford, California.

Kanazawa, M.: 1994, "Weak vs. Strong Readings of Donkey Sentences and Monotonicity Inference in a Dynamic Setting", in *Linguistics and Philosophy* **17**, 109-158.

Lappin, S. and Francez, N.: 1994, "E-type Pronouns, I-sums, and Donkey Anaphora", in *Linguistics and Philosophy* **17**, 391-428.

Neale, S.: 1990, *Descriptions*, Cambridge, Mass.: MIT Press.

Stanley, J. and Szabó, Z. G.: 2000, "On Quantifier Domain Restriction", in *Mind and Language* **15**, 219-261.

On the Categorization via Rank-Distance

Anca Dinu
Bucharest University, Faculty of Mathematics
anca_radulescu@yahoo.com

Liviu P. Dinu
Bucharest University, Faculty of Mathematics
ldinu@funinf.cs.unibuc.ro

Abstract. In this paper we present an unsupervised categorization method. The method uses the rank distance (Dinu, 2003a), a metric which measures the similarity between two classifications based on the ranks of objects. In the framework of natural language, the most important information is carried by the first part of the unit. By analogy, the difference on the first positions between two classifications is more important than the difference on the last positions. This was the starting point in the construction of the rank distance.

1 Introduction

When dealing with categorization and/or aggregation problems, there are three important decisions to make: choosing the metric, the aggregation method and the classifiers.

The long series of papers (Finch, 1993; Herdan, 1966; Kashyap and Oommen, 1983; Manning and Schütze, 1999; Marcus and others, 1971; Marcus, 1974; Mitchell, 1997; Păun, 1983; Schütze and others, 1995; Tin Kam Ho and others, 1994) and their references dedicated to this topic proves the theoretical and practical relevance of the problem. We mention here some metrics as *Euclidean metric, Manhattan metric, Spearman rank correlation coefficient, divergence, Hamming distance, Leuvenstein distance, etc.* and some classification and aggregation methods as *decision trees, k nearest neighbor classification, Bayesian classification, maximum entropy modeling, decision inference, logistic regression, neural networks based method, Borda aggregation method, etc.* encountered in the above works.

The classifications may differ not only by the position of the elements, but also by the elements themselves: some element in a classification may be absent in another classification. In this paper we present an unsupervised

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

categorization method. The method uses the rank distance (Dinu, 2003a), a metric which measures the similarity between two classifications based on the ranks of objects. We define the aggregation of n classifications as the classification for which the sum of distances from it to each of the n classifications is minimal.

2 Rank distance

In this section we shortly present the rank distance (Dinu, L.P., 2003a, 2003b).

Any classification can be regarded as a word of finite length, obtained from concatenation of the elements of the classification, in their appearance order, from the first to the last one. By analogy with natural language where the most important information is carried by the first part of the lexical unit (Marcus, 1971), the difference on the first positions between two classifications is more important than the difference on the last positions. This was the starting point in the construction of the rank distance. Thus, if the differences between two classifications are at the top (i.e., in essential points), the distance has a bigger value then when the differences are at the bottom of the classifications.

Let $L = (x_1, x_2, \cdots, x_n)$ be a classification of length n, such that x_i is in ith position ($1 \leq i \leq n$) in classification L. In this classification x_i means the element itself, not the classification criterion (for example, if we chose the frequency as classification criterion, x_i would not represent the frequency, but the element x_i which is in position i). We assume that the elements of the classification are mutually different. We shall attribute to the first element of the classification the highest rank, i.e., n, to the second one $(n-1)$, and so on, to the final element the lowest rank, i.e., 1 (a classification in Borda sense). It is possible that some elements require the same place (e.g., some elements have the same frequency). In this case we adopt the same strategy as in Spearman rank correlation coefficient method (Herdan, 1966; Finch, 1993): the rank of elements will be the mean of ranks they should receive.

Notation 1 *We denote the position of the element x in classification L with* `ord(x|L)`.

Notation 2 *We denote the vocabulary of the classification L with V(L), i.e.,* $V(L) = \{x_1, x_2, \ldots, x_{n-1}, x_n\}$.

Let L_1 and L_2 be two classifications of lengths n, m respectively. We may presume $m \geq n$.

Definition 1 *The rank distance between L_1 and L_2 is given by:*

$$\begin{aligned}\Delta(L_1, L_2) &= \sum_{x \in V(L_1) \cap V(L_2)} |ord(x \mid V(L_1)) - ord(x \mid V(L_2))| \\ &+ \sum_{x \in V(L_1) \setminus V(L_2)} ord(x \mid V(L_1)) + \sum_{x \in V(L_2) \setminus V(L_1)} ord(x \mid V(L_2)).\end{aligned}$$

Example 1 *Set $L_1 = (a, b, c, d)$ and $L_2 = (x, a, b)$, $V(L_1) = \{a, b, c, d\}$, $V(L_2) = \{a, b, x\}$. According to Definition 1, we have:*
$\Delta(L_1, L_2) = |ord(a|L_1) - ord(a|L_2)| + |ord(b|L_1) - ord(b|L_2)| + ord(c|L_1) + ord(d|L_1) + ord(x|L_2) = |4 - 2| + |3 - 1| + 2 + 1 + 3 = 10.$

Theorem 1 *Δ is a distance.*

Proposition 1 *The rank-distance between L_1 and L_2 is minimal when $V(L_1) \subseteq V(L_2)$ and $ord(x_i|L_1) + m - n = ord(x_i|L_2)$ for all $i = 1, 2, \ldots, n$, and it is maximal when $V(L_1) \bigcap V(L_2) = \emptyset$. In former case $\Delta(L_1, L_2) = n(m-n) + \sum_{i=1}^{m-n} i = \frac{1}{2}(m-n)(m+n+1)$ and in later $\Delta(L_1, L_2) = \frac{1}{2}n(n+1) + \frac{1}{2}m(m+1)$.*

Marcus (1974) has used a similar idea to measure the distance between two codons (sixty-four triplets of nucleotide). He has modified the Hamming distance (Hamming, 1950) by giving a different weight to each position of the three possible position in a codon, in the decreasing order from left to right, $p_1 > p_2 > p_3$. The distance between two codons differing on the first (second, third) position will be equal to p_1 (p_2, p_3 respectively);the distance between two codons differing on the first and the second (the first and the third, the second and the third) positions will be equal to $p_1 + p_2$ ($p_1 + p_3$, $p_2 + p_3$ respectively); the distance between two codons differing on each of their positions will be equal to $p_1 + p_2 + p_3$.

We must say that the existing distances are not appropriate yet for genomics. In (Karp, 2002) it is shown that *"the distance between genomes should be measured not only by counting mutations, but also by determining the number of large-scale rearrangements needed to transform one genome to another.*

3 An aggregation method based on rank distance

The aggregation of classifications means to obtain a classification of given objects that appear in the set of initial classifications. *Categorization* is the task of assigning objects from a universe to two or more classes or categories (Manning and Schütze, 1999).

In this section we present an aggregation method (Dinu, 2003b) based on rank distance which can aggregate classifications having different vocabularies. A particularity of this method is the non-determinist feature.

3.1 Rank-distance aggregation method

Definition 2 *Let $\mathcal{A}$ be a set with n elements. A classification L is of length m over $\mathcal{A}$ if it contains m elements ($|V(L)| = m$) and $V(L) \subseteq \mathcal{A}$. We write $L \in \mathcal{A}^m$.*

Definition 3 *Let $\mathcal{T} = \{L_1, L_2, \ldots, L_n\}$ be a sequence of n initial classifications. The vocabulary of the sequence $\mathcal{T}$ is the set $V(\mathcal{T})$:*

$$V(\mathcal{T}) = \bigcup_{i=1}^{n} V(L_i).$$

Definition 4 *Let $\mathcal{T} = \{L_1, L_2, \ldots, L_n\}$ be a sequence of n initial classifications having the same length m and let L be a classification of length m. We denote the distance between L and $\mathcal{T}$ with $\Delta(\mathcal{T}, L)$:*

$$\Delta(\mathcal{T}, L) = \sum_{i=1}^{n} \Delta(L_i, L).$$

Definition 5 *Consider $\mathcal{T} = \{L_1, L_2, \ldots, L_n\}$ as in Definition 4. The rank-distance aggregation (RDA) is the classification $L \in V^m(\mathcal{T})$ for which one obtain the minimum of the following expression:*

$$\min_{X \in V^m(\mathcal{T})} \frac{1}{n} \sum_{i=1}^{n} \Delta(L_i, X),$$

i.e., $\Delta(\mathcal{T}, L) = \min_{X \in V^m(\mathcal{T})} \Delta(\mathcal{T}, X)$.

Remark 1 *The aggregate classification is not unique. Thus, we introduce a new set of classifications, containing all aggregate classifications that satisfy the minimum condition from Definition 5.*

Notation 3 *Consider $\mathcal{T}$ as in Definition 4. We denote the set of RDA classifications with $agr(\mathcal{T})$.*

Remark 2 *The set $agr(\mathcal{T})$ is not empty. It contains at least one element.*

3.2 On the aggregation algorithm

We have introduced a new method for the aggregation of classifications. One of the problems is the computational aspect. Some constructions that produce a subset of *agr* $(\mathcal{T})$ were developed, but we have not obtained yet an efficient algorithm (i.e. polynomial) that produces all the RDA classifications for a given sequence $\mathcal{T}$.

3.3 The equivalence of elements

In a given classification, two or more elements are said to be equivalent if they have the same position. The RDA method cannot aggregate classifications that have equivalent elements. To overcome this problem we shall decompose each classification L that contains n equivalent elements $a_{i+1}, \ldots, a_{i+n}$ into $n!$ classifications. Each of this classifications will contain one of the $n!$ possible permutations of the elements $a_{i+1}, \ldots, a_{i+n}$.

3.4 Example

Example 2 *Set the following sequence of 5 initial classifications:* $\mathcal{T} = \{(a_1 > a_2 > a_3), (a_1 > a_2 > a_3), (a_3 > a_1 > a_2), (a_2 > a_3 > a_1), (a_2 > a_3 > a_1)\}$.
The vocabulary of this sequence is: $V(\mathcal{T}) = \{a_1, a_2, a_3\}$.
The set of classifications over $V^3(\mathcal{T})$ *contains 6 elements (permutations of 3 elements). We shall evaluate the distance between each classification to* $\mathcal{T}$. *We shall choose the classifications having the minimal distance.*

1. $L = (a_1 > a_2 > a_3);\ \Delta(L, \mathcal{T}) = \frac{12}{5}$
2. $L = (a_1 > a_3 > a_2);\ \Delta(L, \mathcal{T}) = \frac{14}{5}$
3. $L = (a_2 > a_1 > a_3);\ \Delta(L, \mathcal{T}) = \frac{12}{5}$
4. $L = (a_2 > a_3 > a_1);\ \Delta(L, \mathcal{T}) = \frac{12}{5}$
5. $L = (a_3 > a_2 > a_1);\ \Delta(L, \mathcal{T}) = \frac{14}{5}$
6. $L = (a_3 > a_1 > a_2);\ \Delta(L, \mathcal{T}) = \frac{16}{5}$

$$\min_{X \in V^3(\mathcal{T})} \Delta(X, \mathcal{T}) = \frac{12}{5},$$
$$agr(\mathcal{T}) = \{(a_1 > a_2 > a_3), (a_2 > a_1 > a_3), (a_2 > a_3 > a_1)\}.$$

We observe in Example 2 that $agr(\mathcal{T})$ can have the cardinal bigger then 1 and that a RDA classification is not necessary an initial classification($(a_2 > a_1 > a_3)$ is not in $\mathcal{T}$).

4 A categorization method based on the RDA method

As most of the products of human mind, the aggregation and categorization methods should have the characteristic of being *rational*. Arrow (1973) proved that not any rationality conditions can be satisfied simultaneously:*there is no aggregation method that simultaneously satisfies the relevance, Pareto optimality, independence and the absence of the dictator conditions, for all*

problems with at least 3 objects and at least 2 initial classifications, supposing any classification of those elements is admissible (can appear as initial classification).

Păun (1987, 1983) mentions 17 conditions of rationality that should be fulfilled by an aggregation method. Based on some subsets of this conditions, he proposes some variants of the impossibility theorem of aggregation (Păun, 1983, 1987).

4.1 Properties of RDA

Definition 6 *Pareto optimality. If a_i is preferred to a_j in all the initial classifications, then a_i is preferred to a_j in the aggregated classification as well.*

Proposition 2 *The rank-distance aggregation method satisfies the Pareto optimality conditions.*

Definition 7 *Reasonability. An aggregation method is reasonable if, when applied only for two elements, gives the same results as the simple majority method.*

Proposition 3 *The rank-distance aggregation method satisfies the reasonability condition.*

Definition 8 *Independence The final order of any two objects a_i and a_j only depends on a_i and a_j and it never depends on the presence of another element a_k.*

Proposition 4 *The rank-distance aggregation method does not satisfy the independence condition.*

4.2 Rank distance categorization method

An essential problem in text categorization is to choose the classifiers. Different classifiers may assign different classes to the same object. Decision methods based on the inference of classifiers can be used to predict the class of the object.

A method of inference of classifiers based on logistic regression is presented in (Ho et al., 1994). Each classifier receives a weight using a training set.

In the following we propose an unsupervised method of categorization based on rank-distance aggregation (called rank-distance categorization - RDC).

Consider $\mathcal{C}=(C_1, C_2, \ldots, C_n)$ a set of n classifiers and $\mathcal{S}=\{s_1,\ s_2, \ldots, s_m\}$ the possible classes of an object.

Each classifier gives a classification of classes; let $\mathcal{T}=\{L_1, Ł_2, \ldots, Ł_n\}$ be the set of this classifications.

Let $agr(\mathcal{T})=\{A_1,\ A_2, \ldots, A_k\}$ be the aggregation of the set $\mathcal{T}$.

The class of the object predicted by the RDC method is the one that occupies most frequently the first position in the classifications $A_1, .., A_k$.

Due to the Proposition 2, if all classifiers predict a class s on the first place, then the RDC method will predict the same s class on first place.

We tested the RDC method on the data base *World Now*. The method *Naïve Bayes* was used to choose four classifiers. The RDC method improved the mean of individual performance of classifiers with fifteen-twenty percent.

5 Final remarks and further investigations

We have presented the rank distance aggregation (RDA) method, that defines the aggregation of a sequence $\mathcal{T}$ of classifications as those classifications L for which $\sum \Delta(\mathcal{T}, L)$ has the minimum value, where Δ is the rank distance (Dinu, 2003a). The classification obtained by RDA method is not necessarily unique. Based on RDA, we have defined the rank distance categorization method (RDC).

As we have shown in section 3.2, the construction of an efficient algorithm to compute *agr($\mathcal{T}$)*, for a given sequence $\mathcal{T}$, remains an open problem. The investigation of rank distance in genome analysis applications and the relationship of the RDC with other existing technique (e.g. Crammer and Singer, 2003, 2001) are some future work which are in our attention.

Acknowledgements

We thank the anonymous referees for remarks permitting the improvement of the text and for suggestions regarding future works.

References

[1] Borelli, M. and A. Sgarro. A Possibilistic Distance for Sequences of Equal and Unequal Length. In *Finite VS Infinite: Contributions to an Eternal Dilemma*, C. Calude and Gh. Păun eds., Springer-Verlag, London, 2000

[2] Crammer, K. and Y. Singer. Pranking with ranking. In *Advances in Neural information Processing Systems 14*, 2001

[3] Crammer, K. and Y. Singer. A family of additive algorithms for category ranking. *Journal of Machine Learning Research*, 3, 1025-1058, 2003

[4] Dinu, L. On the similarity of the classifications (submitted), 2003a

[5] Dinu, L. On the Aggregation of Hierarchies with Different Constitutive Elements, *Fundamenta Informaticae*, 55, (1), 39-50, 2003b

[6] Finch, S.P. *Finding stucture in language*. PhD thesis, Univ. of Edinburgh, 1993

[7] Hamming, R.W. Error detecting and error correcting codes. *Bell System Technical Journal*, 29, 146-160, 1950

[8] Herdan, G. *The advanced theory of language as choice and chance*. Springer, New York, 1966

[9] Karp, R. Mathematical Challenges from Genomics and Molecular Biology. *Notices of the AMS*, 49, 544-553, 2002

[10] Kashyap, R.L. and B.J. OOmmen. Similarity Measure for Sets of Stringst. *Intern. J. Computer Math.* 13, 95-104, 1983

[11] Manning, C. and H. Schütze. *Foundations of statistical natural language processing*, MIT Press, 1999

[12] Marcus, S., E. Nicolau and S. Stati. *Introduzione alle linguistica matematica*. Casa editrice Riccardo Patron, Bologna, 1971

[13] Marcus, S. Linguistic structures and generative devices in molecular genetics. *Cahiers Ling. Theor. Appl.*, 11, 77-104, 1974

[14] Marcus, S. An emergent triangle: semiotics-genomics-computation. In *Proceedings of the Congress of the German Semiotic Society*, Kassel, July 2002 (to appear)

[15] Mitchell, T.M. *Machine Learning*. McGraw-Hill, New York, 1997

[16] Păun, Gh. An imposibility theorem for social indicators aggregation, *Fuzzy Sets and Systems*, 9, pp. 205-210, 1983

[17] Sgarro, A. A Fuzzy Hamming Distance, *Bull. Math. de la Soc. Sci. Math de la Roumanie*, Tome 21 (69), no. 1-2, 1977

[18] Schütze, H., D.A. Hull and J.O. Pedersen. A comparison of classifiers and document representations for the routing problem. In *SIGIR '95*, pp. 229-237, 1995

[19] Tin Kam Ho, J. Hull and S. Srihari. Decision Combination in Multiple Classifier Systems, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no 1, pp. 66-75, 1994.

Resumptive Elements: Pronouns or Traces?

JUDIT GERVAIN
University of Szeged and Scuola Internazionale di Studi Avanzati (SISSA)
gervain@sissa.it

ABSTRACT. Resumptives have been characterized in the literature both as resembling pronouns and traces. The aim of the present paper is to confront the two theories on the basis of their empirical predictions. A small-scale experimental survey was carried out to test speakers judgments with respect to several diagnostic constructions as they apply to focus-raising via resumption in Hungarian. It is concluded that resumptives have a dual nature, their syntactic behavior is ambiguous between pronouns and traces. This ambiguity is theoretically captured by an analysis in terms of vehicle change (Fiengo and May 1994; Safir 1999).

1 Resumptives in Hungarian Focus-Raising

Previous work (Gervain forthcoming) has found that focus-raising (FR) constructions ((1b) vs. (1a), an expletive construction), traditionally analyzed as instances of A'-movement (É.Kiss 1987; Kenesei 1994; Lipták 1998), are construed via resumption (focus-raising via resumption, FRR), at least for certain speakers of Hungarian (1c)[1].

(1) a *Azt mondtad, (hogy) GÁBOR síel jól.*
expl.acc said.2s that Gábor.nom ski.3s well
"You said Gábor can ski well."
b *(*Azt) GÁBOR$_i$ mondtad, hogy* e$_i$ *jól síel.*
expl.acc Gábor.acc said.2s that well ski.3s
c [$_{CP}$ *GÁBORT$_i$ mondtad,* [$_{CP}$ *hogy* pro$_i$ *jól síel*]].

The empirical evidence for this assumption comes from an experimental survey (for the methods and the statistical analyses, see Gervain 2002), where speakers of Hungarian were shown to cluster into three groups ("dialects"[2] or microvariants) with respect to their grammaticality judgment patterns for focus-raising sentences. Besides the irrelevant group of informants who rejected FR altogether, the other two dialects exhibited opposite patterns for subject focus-

[1] Small capitals indicate focus.
[2] No implication of a geographical or sociological distinction is intended here.

Proceedings of the Eighth ESSLLI Student Session.
Balder ten Cate (editor)

raising along two factors, namely the case of the focused DP and number agreement on the embedded verb (of which the focused DP is the subject).

Before the experimental findings are discussed, a peculiarity of Hungarian number morphology has to be introduced. Plurality is not marked on a noun that is preceded by a numeral or a (plural) quantifier (2a). Moreover, the whole DP agrees with its verb in the singular (2b), but it still clearly refers to plural entities, as cross-sentential anaphora illustrates (2c). So [Num/Quant$_{pl}$ [N$_{sg}$]]$_{sg,\ pl}$ type DPs[3] act as morphologically/syntactically singular, but semantically plural entities.

(2) a [$_{DP}$ [$_{NumP}$ *két* [$_{NP}$*fiú*]]]
two boy.**sg**
"two boys"

b *Két fiú jött/ *jöttek.*
two boy.**sg**.nom come.past.**3s**/ come.past.**3p**
"Two boys came."

c *Két fiú jött. Leültettem *őt/ őket.*
two boy.sg.nom come.past.3s seat.past.1s him/ them
"Two boys came. I offered them seats."

The results of Gervain (2002) show that when such a DP is in the matrix focus position, one dialect (3) tolerates both nominative and accusative case on the DP, but strictly rejects plural embedded verbs, whereas the other dialect (4) accepts only one of the cases, accusative, but both agreements.

(3) a ???*AZ ÖSSZES LÁNY mondtad, hogy jön.*
the all girl.sg.nom said.2s that come.3s
"You said that all the girls were coming."

b **AZ ÖSSZES LÁNY mondtad, hogy jönnek.*
the all girl.sg.nom said.2s that come.3p

c ?*AZ ÖSSZES LÁNYT mondtad, hogy jön.*
the all girl.sg.acc said.2s that come.3s

d *AZ ÖSSZES LÁNYT mondtad, hogy jönnek.*
the all girl.sg.acc said.2s that come.3p

(4) a ??*Az ÖSSZES LÁNY mondtad, hogy jön.*
the all girl.sg.nom said.2s that come.3s
"You said that all the girls were coming."

b ???*AZ ÖSSZES LÁNY mondtad, hogy jönnek.*
the all girl.sg.nom said.2s that come.3p

c ?*AZ ÖSSZES LÁNYT mondtad, hogy jön.*
the all girl.sg.acc said.2s that come.3s

d ??*AZ ÖSSZES LÁNYT mondtad, hogy jönnek.*
the all girl.sg.acc said.2s that come.3p

[3] Indices in italics refer to semantic, non italicized ones to syntactic properties.

These empirical generalizations were analyzed as reflecting two underlying strategies for FR. Speakers of the first dialect (3) display a pattern that is readily describable as resulting from movement. The raised DP is base-generated as the subject of the embedded verb, thus it agrees with it in the singular and receives nominative case. When it undergoes raising, it optionally takes up the accusative case of the matrix verb, which is normally assigned to the expletive in the non-raising expletive constructions (1a). Consequently, it can surface in either of the two cases (for the technical details of how double case is licensed and derived for chains, see Bejar and Massam 1999; Español-Echevarría and Ralli 2000).

The second dialect (4), however, doesn't lend itself to this account, since the embedded verb is in the plural, which is excluded under regular subject-verb agreement in simple clauses (2b). Rather, Gervain (forthcoming) proposes a resumptive strategy to account for this dialect. Under this view, the DP is base-generated directly in the matrix clause, in the position occupied by the expletive in the non-raising counterpart. Here, it receives accusative as its only case, in which it obligatorily surfaces. As for the embedded subject position, it is filled by a phonologically null[4] resumptive pronoun, which establishes a dependency with the antecedent DP. Through this dependency, the resumptive inherits among others the number feature of the DP. Just how this is done exactly will constitute the subject of this paper, but even without going into the details, it can be argued that because of the number ambiguity of the DP, the resumptive may inherit either of the number features. Depending on the value of the transmitted feature then, the resumptive can trigger singular or plural agreement on the embedded verb.

Gervain (2002) presents further empirical and theoretical evidence to ground the above distinction between a movement and a resumptive strategy of FR. Most of them are not discussed here in detail, however one is worth mentioning at this point and some others will also be evoked in later discussions. The most common diagnostic tool to tell apart movement and resumption is movement constraint violations, e.g. islands, as suggested by Chomsky (1981, 1982) and exemplified in (5).

(5) a *I wonder who$_i$ Mary marries (*him$_i$)*.
b *I wonder* [*who$_i$ they think* [*that* [*if Mary marries *(him$_i$)*] *then everybody will be happy*]]

Speakers of the two different FR dialects are expected to behave differently with respect to such diagnostics. Judgments on FR sentences containing complex NPs were measured and compared. Informants in the two dialects answer statistically differently both as a group and individually. Movement speakers

[4] That the resumptive pronoun is phonologically null comes as no surprise, since Hungarian is a *pro*-drop and an Avoid Pronoun language. Languages of this kind typically exhibit null resumptives (Engdahl 1985; Suñer 1998). A formal account of these observations is given by Montalbetti's (1984) Overt Pronoun Constraint.

fully reject, while resumptive speakers accept sentences like (6). This further confirms the movement-resumption dissociation.

(6) b *AZ ÖSSZES VENDÉGET mondtad, hogy hallottad*
the all guest.sg.acc say.past.2s that hear.past.2s
"You said that you heard the news that all the guests had arrived."
a hírt, hogy megérkeztek.
the news that arrive.past.3p

In light of the above considerations, the two strategies are well established. However, Gervain (2002) and Gervain (forthcoming) leave the issue about the nature of the resumptive dependency unresolved. Two alternatives are proposed, but theoretical considerations and empirical data are not conclusive. The first possibility outlined assumes that the dependency is entirely syntactic, i.e. operator-variable binding and coindexation between the antecedent and the resumptive. In such a scenario, the optionality of number results from the fact that the DP is transparent for coindexation, *à la* Longobardi (2001), so the resumptive may target either the plural Num/QuantP or the singular NP. The second option consists of positing a mixed type dependency, one that can be either syntactic or semantic. In the former case, the resumptive will inherit the feature of the whole DP, i.e. singular. In the latter, coreference obtains, which, as (2c) showed, is plural. Notice that the ultimate difference between the two possibilities hinges on the nature of the resumptive element. If resumptives are like variables, they always have to be bound, whereas if they resemble pronouns, they may be bound, but may also corefer.

The nature of resumptives appears to vary considerably across languages, and no theoretical consensus has been reached in the literature either. The present paper attempts to contribute to the debate by introducing experimental evidence, as well as by offering a theoretical account of Hungarian resumptives in FR constructions.

2 Theories of resumption

Resumptive elements come in various shapes and sizes in the world's languages. Their theoretical analyses are equally heterogeneous. In the following, I will give a brief overview of the resumptive landscape.

As referred to above, one of the first detailed theoretical treatments of resumptive pronouns was given in Chomsky (1981, 1982). The main assumption, based mostly on English data of the type (5), was that resumptives appear in positions from where gaps/traces are excluded by constraints on movement (e.g. in island contexts). The resumptive pronoun is base-generated in its surface position and is A'-bound at LF by its antecedent, with which it is coindexed. On this view, resumptive pronouns are expected to be in complementary distribution with traces, and they come as a kind of last resort device to save otherwise

disallowed movement configurations. Overall, they were thought of as a rare and marked strategy, with no specific UG constraints required to account for them. Rather their distribution was believed to fall out from independent UG principles.

This approach was later challenged on several grounds. Resumptive strategies turned out to be subject to considerable cross-linguistic variation, which led to the introduction of different typologies (Engdahl 1985; Demirdache 1991; Suñer 1998; Aoun et al. 2001). The last resort nature of resumption has also been questioned (see e.g. Shlonsky 1992 and Aoun et al. 2001 for strong last resort views; but Suñer 1998 and Willis 2000 for challenges).

In addition to the above cited works, some analyses specifically address the question whether resumptives are pronouns or traces. These will be presented in somewhat more detail.

Discussing relative clauses in Hebrew, Sharvit (1999) points out that some of the syntactically free and optional alternations between traces and resumptives actually produce interpretational differences. Pair-list/multiple individual readings are not available for resumpives, while they are possible with traces in non-equative relative clauses, while this asymmetry disappears in equative clauses. In his account, resumptives are licensed under two conditions: (i) they need a contextually salient (e.g. D-linked) antecedent, and (ii) they can only be assigned values that the given pronoun can take when it is A/A'-free. Pair-list readings generally violate the first condition, but this impairment is amended in equative clauses, where a highly salient antecedent is available. Sharvit further claims that resumptives have a dual nature. Like traces, they are A'-bound and are interpreted as bound variables, while their distribution (e.g. within islands) resembles that of ordinary pronouns.

Falk (2002) offers an LFG account of resumptives. He introduces the pronoun vs. variable debate, and summarizes some of the empirical evidence. As arguments for the trace hypothesis, he enumerates the following observations: (i) resumptives, just like gaps, are linked to some discourse function or operator (cf. also Erteschik-Shir 1992; Sharvit 1999), (ii) anaphora between a possessive reflexive in a fronted whP and its antecedent DP in an embedded subject position is allowed when the extraction site of the whP contains a trace or a resumptive (cf. Zaenen et al. 1981), (iii) like traces, resumptives are able to license parasitic gaps (cf. Engdahl 1985; Shlonsky 1992), (iv) both traces and resumptives show crossover effects (cf. Shlonsky 1992), and (v) they can be coordinated. On the other hand, as Falk underlines, resumptives are exempt in most (but not all) languages from the island constraints traces/movement obey (Chomsky 1981, 1982). Also, resumptives are associated with special morphology on the verb or the complementizer in some languages (cf. McCloskey 2001, Vailette 2002). In Falk's own analysis, which is cast in LFG terms, resumptives receive the same treatment as gaps, except for their licensing. Interestingly, on the basis of the same empirical evidence as Sharvit, Falk makes the additional claim that

resumptives are referential and *cannot* be bound variables. Rather, they are D-linked, and besides syntactic constraints, they also respect the principle of Sufficiency of Expression, which claims that syntactic elements providing clues for parsing are exceptions to (syntactic) considerations of economy.

As it is apparent from the above argumentations, the debate between the pronoun and the trace hypotheses is far from being resolved. In the following, I will examine resumptives in Hungarian FR, by applying some of the aforementioned empirical tests.

3 The experiment

Focus-raising in itself is not a good testing ground to distinguish between the trace and the pronoun hypotheses. Therefore, some of the diagnostics had to be applied to allow a better comparison between the predictions of the two approaches.

If resumptives behave like traces, i.e. bound variables, they are expected not to be able to corefer. They are supposed to license parasitic gaps and show crossover effects. Moreover, their coordination with another trace should be grammatical. If, on the other hand, they resemble ordinary pronouns, they can corefer, they don't license parasitic gaps or show crossover effects and it is impossible to coordinate them with traces.

To test these predictions, a paper-and-pencil survey was carried out, comprising the following diagnostics: i, parasitic gap licensing; ii, coordination with traces/pronouns; iii, crossover effects. Responses were given statistical treatment[5].

Material. Sentences were constructed to test the following structures:

(i) FR from embedded subject position: acceptable/ unacceptable
(ii) FR from embedded object position: acceptable/ unacceptable
(iii) coreference: acceptable/unacceptable
(iv) overt resumptive pronoun in FR of embedded subject: acceptable/unacceptable
(v) overt resumptive pronoun in FR of embedded object: acceptable/unacceptable
(vi) parasitic gaps in FR: licensed/not licensed
(vii) strong crossover effects: present/absent

[5] The use and interpretation of statistics carried out on grammaticality scales is not without its difficulties. Lack of space precludes any serious discussion here, but Schütze (1996), Cowart (1997), Keller (2000) and Gervain (2002) give ample treatment to the subject. One consideration is noteworthy here. The use of parametric statistics means that judgments are interpreted not only as absolute values, which is the general practice in (non-experimental) syntactic research, but also as relative ones.

(viii) weak crossover effects: present/absent

(ix) coordination of resumptive pronoun: with null pronoun/with overt pronoun

Factor (i) served as the criterion for inclusion; all subjects rejecting subject FR via resumption were automatically excluded from all further analysis. As discussed above, parasitic gap licensing and crossover phenomena have been proposed as diagnostics to distinguish between gaps and pronouns. As crossover phenomena are more readily testable with wh-movement than with other types of displacement, wh-raising (whR) constructions, rather than FR, were used in crossover sentences. This should introduce no confound in the test, given the fact that whR and FR are both instantiations of the more general phenomenon of operator-raising[6]. However, to exclude all potential biases, additional sentence types were constructed to serve as controls for (vii)-(viii) above.

(viic) whR from embedded subject position: acceptable/ unacceptable

(viiic) anaphor binding between raised subject and reflexive pronoun in the embedded object position: acceptable/ unacceptable

To avoid lexical biases and to enable the statistical treatment of judgments, each sentence type was instantiated by three sentence tokens. Thus, for example, (i) is represented by (7a-c).

(7) a *KÉT FIÚT mondtál, hogy jönnek.*
two boy.sg.acc say.past.2s that come.3p
"You said that two boys were coming."

b *NÉGY LÁNYT mondtál, hogy táncoltak.*
four girl.sg.acc say.past.2s that dance.past.3p
"You said that four girls had danced."

c *HÁROM DIÁKOT hiszel, hogy feleltek.*
three student.sg.acc believe.2s that reply.past.3p
"You believed that three students were interrogated."

Since plural verbal agreement with *két fiút* type NPs is a telltale sign of FR via resumption, sentences with raised subjects were construed with plural embedded verbs. In addition, for sentence types (vii), (viii), (viic) and (viiic), singular variants were also included. This adds up to 48 test and control sentences. Furthermore, habituation and repetition effects were counterbalanced by the insertion of 15 grammatical and ungrammatical filler sentences. These were

[6] In early treatments (e.g. Marácz 1987, É. Kiss 1987), the two kinds of opertaror-raising receive identical treatment. Moreover, Bródy (1995) argues for an essential underlying similarity between focus and questions in Hungarian. Recently, however the similarity between whR and FR has been challenged by Lipták (1998). Nevertheless, even if Lipták (1998) is right, the differences are very subtle and do not concern the properties under investigation in this study. For empirical confirmation of the similarity, see the current results.

never included in the data used for the statistical analyses. The complete survey thus contained 63 sentences.

Subjects. Twenty-two informants participated in the survey. Four of them rejected FRR completely (i.e. assigned a grammaticality value lower than 0 to at least two of the test sentences used as inclusion criteria (7), therefore they were excluded from the statistical analyses.

Procedure. A paper-and-pencil questionnaire was administered to informants through electronic mail or physically. Informants were given detailed instructions and examples. Subjects were asked to evaluate the sentences on a five-grade scale (from –2 to +2), ranging from totally unacceptable (-2) through three intermediate levels (-1, 0, 1) to fully acceptable (+2). The five-grade scale was adopted in order to allow comparison, as it is one of the most commonly used ratings and was also employed in Gervain (2002).

Results. First a baseline of comparison had to be established, i.e. the grammaticality of pure FRR had to be calculated. The mean grammaticality values of subject FRR were 1.167, 1.278 and –.556[7], their average being .630. Object FRR (values 1.056, –.222, 1.611) had an average of .815. A *t*-test showed no statistical difference between the two means ($t(17)=.777$, ns.), absolute grammaticality values were also very similar. Secondly, the availability of overt resumptive pronouns had to be evaluated. The means for overt subject resumptives were –1.056, 0.000 and –.278 for the individual sentences, with an overall average of –.444. Overt object resumptives obtained mean grammaticality values –1.056, –1.389 and –1.556, the overall average was –1.333. A repeated measures analysis of variance (ANOVA) was carried out with the factors Overtness (null vs. overt pronoun) and Constituent Type (subject vs. object). Constituent Type, just like in the previous *t*-test, had no significant main effect ($F(1, 17)=3.610$, ns.), whereas Overtness did ($F(1,17)=78.555$, $p< .0001$). In addition, the Constituent Type × Overtness interaction was also very significant ($F(1,68)=16.980$, $p<.001$). Results thus show that FRR resists overt resumptives.

Secondly, diagnostic sentences were examined in comparison to the established baselines. Test sentences combining parasitic gaps with object FR obtained sentence means of .167, -.556 and .444, with an average of .019. This was compared to object FRR in a *t*-test, which revealed a significant difference ($t(17)=–3.690$, $p<.05$). Nevertheless, note that the average grammaticailty of parasitic gap + object FRR sentences is still within the positive range of the –2 to +2 scale. Parasitic gaps do worsen grammaticality, but do not induce severe violations. These results are not decisive. Further evidence could be gained from

[7] The grammaticality of sentence 11 is considerably lower than that of the other two subject FRR sentences. A possible explanation, also suggested by previous data (Gervain 2002) is that the verb *hisz* „to believe" is not clearly a bridge verb in Hungarian. This case nicely shows the advantages of the experimental procedure. A lexical bias is compensated for by other, non-biased lexicalisations.

a comparison with simple parasitic gap constructions, i.e. those not containing an additional resumptive dependency.

Co-ordination with traces was designed as a second diagnostic tool. This measure, however, crucially depends on the availability of overt resumptives, which turned out not to be the case in Hungarian. Therefore, the analysis of this diagnostic test will not be further pursued.

Several instanciations of cross-over effects were also tested[8]. Strong cross-over (SCO) was lexicalized both with singular and plural embedded verbs (the latter being an overt sign of the presence of resumptives). The means of the singular sentences were –1.500, –1.500 and –1.500, those of the plurals were –1.722, –1.333 and –1.000 respectively. The overall means were –1.500 for singular, –1.352 for plural sentences. There was no significant difference between the grammaticality of the two types ($t(17)$=–1.512, ns.). The absolute values were very low, implying that SCO sentences are quite marginal, as expected. Weak-cross over (WCO) effects, on the other hand, appear to be finer diagnostic tools (Bissell 1999; Ruys 2000). Like SCO, these constructions were also lexicalized both with singular and plural morphology, but the relevant site of agreement is not the embedded verb, but the possessive suffix of the embedded subject DP[9]. The means of the sentences were .560, .278, .556 and –.500, –.333, –.167, with averages .296 and –.333. Controls for WCO were whR sentences in which the wh-constituent was in the subject, rather than the object position of the embedded clause, hence no crossover could obtain. Both singular and plural were tested. The means for the individual sentences were 1.222, 1.500, 1.444 and 1.000, 1.333, 1.222. The overall averages are 1.315 and 1.185. A *t*-test showed no difference between the two types ($t(17)$=1.236, ns.). To compare WCO sentences to their controls, a repeated measures ANOVA was performed with factors Crossover (WCO vs. control) and Number (singular vs. plural). The factor Crossover had a highly significant main effect ($F(1,17)$=47.361, $p<.0001$), indicating that WCO sentences are less grammatical than controls. The main effect of Number was also significant ($F(1,17)$=8.286, $p<.05$). There was no two-way interaction between the factors ($F(1,17)$=3.180, ns.).

---

[8] Several types of controls were introduced for cross-over sentences. Space considerations prevent me from giving an appropriate description of these, but some statistics will be mentioned where necessary.

[9] An example sentence is:

(i) ???*Hány* *férfit*$_i$ *gondolsz, hogy a feleség-e*$_i$*/*
how.many man.acc think.2s that the wife.poss3g
"How many men do you think his/their wife loves?"
feleség-ük$_i$ *szeret* t_i?
wife.poss3p love.3s

To sum up new results, the presence of crossover effects shows a variable-like behavior, while the marginality of the parasitic gap test points in the other direction.

Discussion. The present results attest to an ambiguity in the syntactic behavior of resumptives. Some previous evidence is worth evoking here that also shows the same ambiguity. Empirical results in Gervain (2002) suggest that the resumptive dependency is grammatical with a quantified DP antecedent, even through islands. This is indicative of a variable-like nature, at least at LF. On the other hand, reciprocals in the embedded object position improve or even force plural agreement (which is not the case in simple clauses, where agreement is strictly singular). This property goes in the direction of the pronoun hypothesis. An antecedent that is made contextually more salient is easier to establish coreference with. The most general conclusion on the basis of these results is that resumptives have both trace-like properties, for instance crossover effects and (possibly) parasitic gaps licensing, while they also exhibit traits characteristic of pronouns, e.g. they appear in islands and are sensitive to contextual salience. Therefore, I conclude, in accordance with Sharvit (1999) and Falk (2002), that resumptives are ambiguous in nature between traces and pronouns.

4 Towards a syntactic analysis

Some previous accounts, such as Sharvit (1999) and Falk (2002) do acknowledge the two-faceted nature of resumptives. However, they propose solutions, in which the ambiguity results from a tension between syntactic constraints and other considerations, e.g. semantic or processings (e.g. Falk 2002). Without denying the need for a complex, multilevel account (for some discussion, see section 5), I argue that the ambiguity of resumptives has to be captured on a syntactic level, as well. I propose that resumptives, at least in the examined FR context, are best analyzed as instances of vehicle change.

Vehicle change, as defined by Fiengo and May (1994) and Safir (1999), is a mechanism that allows copies/traces of names to be treated as pronouns by interpretive principles. It was originally proposed to explain the lack of Principle C effects in certain elliptic constructions (8).

(8) a ???*Lara loves Sol*$_{\text{i}}$ *and he*$_{\text{i}}$ *thinks that Sally loves Sol*$_{\text{i}}$ *too.*

b *Lara loves Sol*$_{i}$ *and he*$_{i}$ *thinks that Sally does too*.

Sentence (8a) violates Principle C on the reading that the indices define, because the second occurrence of *Sol* is not free. However, the same does not hold for the elliptic counterpart (8b). Fiengo and May (1994) argue that the first instance of *Sol* is not copied identically into its trace in the second VP. Rather, the trace changes into identically into its trace in the second VP. Rather, the trace changes into a pronominal element for purposes (and mechanisms) of interpretation, e.g. binding.

I claim that the same mechanism applies to FR in Hungarian. The resumptive pronoun behaves like a variable in many respects, e.g. crossover or parasitic gaps, but it can be treated as a pronoun for interpretive purposes, for instance when there is a contextually salient antecedent that facilitates coreference. A clear objection that can be made at this point is that vehicle change was proposed for names, i.e. non-quantificational DPs, while Hungarian FR is grammatical with quantified DPs as antecedents. The reason for this, I believe, has to do with the fact that resumptives in FR are linked with quantified DPs that are focused. Focus obviously comes with strong discursive/contextual relevance, and creates a set of possible interpretations. Thus I claim that focused quantifiers lose their real quantificational force, and behave like ordinary, non-quantified DPs. This is illustrated in (9).

(9) a **MINDEN LÁNY jött el.*
every girl.sg.nom come.past.3s PART
"It was every girl that came."

b *SOK LÁNY jött el*,
many girl.sg.nom come.past.3s PART
"It was many girls that came (not a few/a few boys)."
(nem KEVÉS/ KEVÉS FIÚ)
not few/ few boy

As (9a) shows, when there is nothing to contrast with the focused quantifier, the result is ungrammatical. When some contrastive interpretation is possible, the sentences is ruled in. Without a more elaborate theory of the semantics of focus, strong conclusions might appear far-fetched, but (9) suggests that when in focus, quantifiers suspend their usual function of quantifying over NPs, rather they denote contrastable elements with a set, for instance many girls as opposed to just a few girls, no girls or some boys (within the contextually relevant group of boys and girls). In this situation, quantified DPs are not different from ordinary ones, thus vehicle change is allowed to apply. Note how this approach parallels Sharvit's (1999) two constraints on resumptives, but offers a syntactic account at the same time. Focus provides a contextually salient antecedent, which can be further reinforced by other coreferent pronouns, e.g. a reciprocal. When in focus, quantified DPs act like ordinary ones, which makes them possible referents for the free counterpart of the pronoun, in other words, the difference between *két fiú* and *az összes fiú* is minimized.

Safir (1999) also raises the possibility that vehicle change is responsible for resumptives, and hypothesizes that restrictions on the type of the antecedent might be relaxed in resumptive contexts, as opposed to ellipsis and reconstructions. However, in the absence of empirical evidence, he elaborates the claim no further. I assume that the Hungarian data presented in this work offers exactly this evidence. Furthermore, restrictions do not need to be relaxed in an *ad*

hoc, thus unattractive way. The interaction of quantifiers and focus takes care of this issue.

A prediction of my proposal is that resumptives should not be able to link to their antecedents when those are quantified and nothing intervenes to modify their interpretation. This prediction seems to be borne out, for instance in Lebanese Arabic, where resumptives cannot be construed with QPs in certain contexts (see Aoun et al. 2001 for the data, although the account they give is different; see also Sharvit (1999) and Falk (2002) on some relevant Hebrew data).

In sum, it has been proposed that the syntactic duality of resumptives can be explained if we assume that they are mediated by vehicle change. The otherwise variable-like resumptives are seen as pronouns by interpretive mechanisms. To answer the original question left open in Gervain (2002), the two options that were put forward to describe the resumptive dependency do not represent an either/or choice. Rather, the mixed kind of chain (coindexation and coreference) is „seen" by interpretive mechanisms, while the „coindexation only" chain applied in the rest of the syntax.

5 Conclusion and perspectives

A proposal has been put forth claiming that the syntactic ambiguity of resumptives is best explained as a case of vehicle change. This account makes special reference to interpretive mechanisms. As mentioned earlier, this is not the only analysis of resumptives that links their syntactic properties to semantic (Sharvit 1999), pragmatic (Erteschik-Shir 1992) or even parsing (Falk 2002) considerations.

The questions that need to be addressed on these levels of description are somewhat similar to the one formulated in syntax above. What is the semantic type of resumptives? Are they bound variables or rather pronouns that refer to individuals (e-type entities)? If resumptives play a role in parsing, as some experimental results suggest (Alexopoulou and Keller 2002), what is the interaction between their syntax, semantics and psychology? In more general terms, what level of language is responsible for resumptives: is it possible that they constitute an „intrusion" into the autonomy of syntax?

References

Alexopoulou, T. and F. Keller. 2002. Resumption and locality. *Papers from the 38th Meeting of the Chicago Linguistic Society, Vol. 1: The Main Session*. Chicago.

Aoun, J., L. Chouieri, N. Hornstein. 2001. Resumption, Movement, and Derivational Economy. *Linguistic Inquiry* 32: 3.

Bissell, T. 1999. Further Evidence for Null Pronominal Variables. Unpublished MA Thesis. University of California at Santa Cruz. Downloadable at http://www.mit.edu/~bissell/NullPro.pdf

Bródy, M. 1995. Focus and checking theory. In *Levels and Structures (Approaches to Hungarian 5.)* edited by I. Kenesei. Szeged: JATEPress.

Chomsky, N. 1981. *Lectures on Government and Binding*. Dordrecht: Foris.

Chomsky, N. 1982. *Some Concepts and Consequences of the Theory of Government and Binding*. (*Linguistic Inquiry Monographs 6*.) Cambridge, MA.: The MIT Press.

Cowart, W. 1997. *Experimental Syntax: Applying Objective Methods to Sentence Judgments*. Thousand Oaks: Sage Publications.

Demirdache, H. 1991. Resumptive chains in restrictive relatives, appositives and dislocation structures. Doctoral dissertation. MIT.

É. Kiss, K. 1987. *Configurationality in Hungarian*. Budapest: Akadémiai Kiadó.

Engdahl, E. 1985. Parasitic gaps, resumptive pronouns, and subject extractions. *Linguistics* 23: 3-44.

Erteschik-Shir, N. 1992. Resumptive Pronouns in Islands. In *Island Constraints* edited by H. Goodluck. and M. Rochemont, 89-108. Dordrecht: Kluwer.

Español-Echevarría, M., A. Ralli. 2000. Case mismatches in Greek: Evidence for the autonomy of Morphology. *Acta Linguistica Hungarica* 47(1-4): 179-203.

Falk, Y. 2002. Resumptive Pronouns in LFG. In *Proceedings of the LFG 02 Conference* edited by M. Butt and T. Holloway King, 154-173. On-line publication at http://.cslipublications.standord.edu/LFG/7/lfg02.html

Gervain, J. forthcoming. Syntactic microvariation and methodology. *Acta Linguistica Hungarica*.

Gervain, J. 2002. Linguistic Methodology and Microvariation in Language. MAThesis. University of Szeged.

Horvath, J. 1995. Partial Wh-Movement and Wh 'Scope-Markers'. In *Levels and Structures (Approaches to Hungarian 5.)* edited by I. Kenesei, 89-124. Szeged: JATEPress.

Horvath, J. 1998. Multiple wh-pharses and the wh-scope-marker strategy in Hungarian Interrogatives. *Acta Linguistica Hungarica* 45(1-2): 31-60.

Keller, F. 2000. Gradience in Grammar. Ph.D. dissertation. University of Edinburgh, Edinburgh.

Kenesei, I. 1994. Subordinate Clause. In *The Syntactic Structure of Hungarian. Syntax and Semantics 27*, edited by F. Kiefer and K. E. Kiss, 275-354. New York: Academic Press.

Lipták, A. 1998. A magyar fókuszemelések egy minimalista elemzése. In *Proceedings of "A mai magyar nyelv leírásának újabb módszerei III."* L. Büky, M. Maleczki M., 93-115. Szeged: JATEPress.

Longobardi, G. 2001. NP structure. In *The Handbook of Current Syntactic Theory* edited by Ch. Collins and M. Baltin. Cambridge, Mass.: MIT Press.

Marácz, L. 1987. Wh-strategies in Hungarian: data and theory. In *Logic and Language* edited by I. Ruzsa and A. Szabolcsi. Budapest: ELTE.

McCloskey, J. 2001. The morphosyntax of wh-extraction in Irish. *Journal of Linguistics* 37: 67-100.

McDaniel, D., W. Cowart. 1999. Experimental evidence for a minimalist account of English resumptive pronouns. *Cognition* 70: B15-B24.

Montalbetti, M. 1984. After binding. Doctoral Dissertation. MIT.

Ruys, E.G. 2000. Weak Crossover as a Scope Phenomenon. *Linguistic Inquiry* 31(3): 513-539.

Safir, K. 1999. Vehicle Change and Reconstruction in A'-Chains. *Linguistic Inquiry* 30(4): 587-620.

Schütze, C. 1996. *The empirical base of linguistics. Grammaticality judgments and linguistic methodology*. Chicago: Chicago University Press.

Sharvit, Y. 1999. Resumptive Pronouns in Relative Clauses. *Natural Language and Linguistic Theory* 17: 587-612.

Shlonsky, U. 1992. Resumptive Pronouns as a Last Resort. *Linguistic Inquiry* 23(3): 443-468.

Suñer, M. 1998. Resumptive restrictive relatives: A cross-linguistic perspective. *Language* 74(2): 335-364.

Vailette, N. 2002. Irish gaps and resumptive pronouns in HPSG. In *Proceedings of the 8th International HPSG Conference* edited by F. van Eynde et al., 284-299. On-line publication: http://cslipublications.stanford.edu

Willis, D. 2000. On the distribution of resumptive pronouns and wh-traces in Welsh. *Journal of Linguistics* 36: 531-573.

Zaenen, A., E. Engdahl, J. Maling. 1981. Resumptive Pronouns can be syntactically bound. *Linguistic Inquiry* 12(4): 679-682.

Formalized Interpretability in Primitive Recursive Arithmetic

Joost J. Joosten
Department of Philosophy, University of Utrecht,
Heidelberglaan 8, 3584CS Utrecht, The Netherlands
http://www.phil.uu.nl/~jjoosten/
jjoosten@phil.uu.nl

Abstract. Interpretations are a natural tool in comparing the strength of two theories. In this paper we give a brief introduction to the topic of interpretability and interpretability logics. We will focus on the, so far, unknown interpretability logic of PRA. One research technique will be treated. This technique can be best described as restricting the realizations in the arithmetical semantics.

1 What are interpretations and why study them?

How to interpret "Eli, Eli, lama sabachtani"? Let us consider the concept of interpretation in the previous phrase[1]. What does it actually mean to interpret something. Or more specifically, what do we mean when we say that T interprets some utterance φ of S? Well, in this case T can first *translate* φ to its own language, then *place it in an adequate context* and then somehow *make sense* of it.

The mathematical notion of interpretation is somewhat similar. We say that a theory T interprets another theory S whenever there is some translation such that all translated theorems of S become provable in T. We give a precise definition. Throughout this paper we will stay in the realm of first-order logic.

Definition 1 *$\mathcal{K}$ is a relative interpretation of a theory S into a theory T, we write $\mathcal{K} : T \rhd S$, whenever the following holds. $\mathcal{K}$ is a pair $\langle \delta, F \rangle$. The first component, δ, is a formula in the language of T with a single free variable. This formula is used to specify the domain of our interpretation in a sense that we will see right now. The second component, F, is an*

[1] "Eli, Eli, lama sabachtani" were Jesus' last words. Some scholars translate this to "My God, my God, why hast thou forsaken me?". Others read it as "My God, my God, how thou dost glorify me!".

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

easy (primitive recursive) map that sends formulas ψ in the language of S, to formulas $F(\psi)$ in the language of T. We demand for all ψ that the free variables of ψ and $F(\psi)$ are the same. The map F should commute with the boolean connectives, like $F(\alpha \wedge \beta) = F(\alpha) \wedge F(\beta)$. Moreover F should relativize the quantifiers to our domain specifier δ. Thus, for example $F(\forall x\ \alpha) = \forall x\ (\delta(x) \rightarrow F(\alpha))$.

We think this notion of interpretation is a natural one and comes close to our every day use of the concept of interpretation. And indeed it is a natural tool in comparing the the proof strength of two theories.

A first guess to say what it means that some theory T is at least as strong as some other theory S could be the following. Whenever S sees the truth of a formula ψ, T should also be able to see the truth of ψ. But, S and T might speak different languages. This is where the idea of a translation comes in.

Of course the translation should preserve some structure. Also it seems unreasonable that T should have the same domain of discourse as S. Taking these considerations into account it comes quite natural to say that T is at least as strong as S whenever T interprets S in the sense of Definition 1.

In the mathematical and metamathematical literature the here defined notion of interpretation turns up time and again. Perhaps the most famous example is in the proof of the consistency of non-euclidean geometry. In this proof (see for example [Gre96]) a model for non-euclidean geometry is built in a uniform way inside a model for euclidean geometry. Of course we somehow "know" that euclidean geometry is consistent. This uniform model construction is really nothing but an interpretation.

Tarski, Mostowski and Robinson first studied interpretations as a (meta) mathematical tool in a systematic way in [TMR53]. They also used interpretations to determine the undecidability of certain theories. It is not hard to convince oneself that some consistent theory T is undecidable whenever T interprets some essentially undecidable theory S. We say that S is essentially undecidable if S is undecidable and every consistent extension of T in the same language is also undecidable.

2 Formalized interpretability

In the previous section we have introduced the mathematical notion of interpretability. We have given some arguments to plea that it is a natural and interesting notion to consider. In this section we will add one more argument to our list. We will see that theories can in a certain way speak about interpretations. This insight will provide us with a simple yet expressive formalism in which large parts of metamathematical practise are expressible.

Amongst these are the Model Existence lemma as used in Gödel's Completeness theorem, Gödel's Second Incompleteness theorem, but also the method of relative consistency using interpretations.

Ever since Gödel we know that in theories of some minimal strength we can code syntax and syntactical notions like provability. We write $\Box_T\varphi$ for the very long statement that codes the fact that the sentence φ is provable in the theory T. As usual we denote $\neg\Box\neg\varphi$ by $\Diamond\varphi$. Once we realize that the notion of provability can be coded in a theory, it does not come as a surprise that we can do the same for interpretability.

For, what does it mean that S is interpretable in T? This means that there is a primitive recursive translation such that all translated theorems of S are provable in T. With some sloppy notation this can be written down as $\exists j\,\forall x\,(\Box_S x \to \Box_T x^j)$. Indeed, it turns out that the notion of interpretability can be expressed by a Σ_3-sentence.[2] We will denote the formalized statement of T interprets S by $T \rhd S$.

In this paper we will, for reasons that will become clear below, be mainly interested in interpretability relations between theories that are both finite extensions of some base theory T. Thus, we are interested in statements of the form $(T+\alpha)\rhd(T+\beta)$ which we will abbreviate with $\alpha \rhd_T \beta$. When the base theory T is clear from the context we will even omit sometimes the T in $\rhd_T$ and in $\Box_T$.

After having introduced this notation we see that many interesting properties can be expressed. For example $(i) : \alpha\rhd\beta \to (\Diamond\alpha \to \Diamond\beta)$. The formula (i) expresses that $T+\beta$ is consistent whenever it is interpretable in a consistent theory $T+\alpha$. We would like to say that (i) actually holds for any choice of α and β. One way of doing so is by working with arithmetical realizations and modal logics.

Definition 2 *By* $\mathsf{Form}_{\mathbf{IL}}$ *we denote the set of formulas in the modal language of interpretability logic. This is the smallest set containing* $\bot$, $\top$, *a countable infinite set of propositional variables and being closed under the boolean connectives, a unary modal operator* $\Box$ *and a binary modal operator* $\rhd$. *The* $\Diamond$ *will be an abbreviation for* $\neg\Box\neg$.

Definition 3 *An* arithmetical realization (relative to a theory T) *is a map* $(\cdot)^*$ *that sends any propositional variable* p *to some arithmetical sentence* p^*. *This map is extended to* $\mathsf{Form}_{\mathbf{IL}}$ *by stipulating that it commutes with the boolean connectives and demanding that* $(\Box A)^* = \Box_T A^*$ *and that* $(A \rhd B)^* = A^* \rhd_T B^*$. *An* interpretability principle *of a theory* T *is a formula in* $\mathsf{Form}_{\mathbf{IL}}$ *that is provable in* T *under any arithmetical realization. By the* interpretability logic of T *we mean the set of all interpretability principles of* T *or some system generating this set. We write* $\mathbf{IL}(\mathrm{T})$.

[2] A Σ_3-sentence is one that starts with a sequence of an existential- then universal- and then again existential quantifier to be followed by some formula only containing bounded quantification.

Note that by $\rhd$ we might now denote either the modal operator or the formalized notion of interpretability. We are confident however that this will not cause any confusion. Also note that the interpretability logic of a theory is the interpretability behaviour of that theory as seen by itself.

The modal language we have just introduced is rather expressive. Gödel's Second Incompleteness theorem can be written down as $\Diamond\top \rightarrow \neg\Box\Diamond\top$. Some reflection learns us that $\Diamond\top \rightarrow \neg(\top \rhd \Diamond\top)$ can be seen as a generalized version of Gödel's Second Incompleteness theorem; Under the assumption of the consistency, the consistency itself is not only just not provable, but not even interpretable!

An interpretation of S in T provides in an obvious way a uniform procedure to define a model of S within any model of T. Thus, the formula $\Diamond A \rhd A$ expresses in a certain sense the Model Existence lemma; whenever A is consistent in T, we can make a model of $T + A$.

Now consider a theory T. What is the modal characterization of its interpretability logic? For two classes of theories the answer to this question is known. If T is finitely axiomatizable **IL**(T) is known to be **IL**P as defined below. If T is essentially reflexive[3] **IL**(T) is also known. It is **IL**M, which is defined below. (See for an overview of these results [Vis97].)

We now present a logic **IL** that generates interpretability formulas that are interpretability principles for any reasonable theory. The logic **IL** is the smallest set of formulas in $\mathsf{Form}_{\mathbf{IL}}$ that is closed under the necessitation rule $A/\Box A$ and under Modus Ponens that contains all propositional tautologies and all instantiations of the following axiom schemata.

$$
\begin{array}{ll}
\mathsf{L}_1: & \Box(C \rightarrow D) \rightarrow (\Box C \rightarrow \Box D) \\
\mathsf{L}_2: & \Box A \rightarrow \Box\Box A \\
\mathsf{L}_3: & \Box(\Box A \rightarrow A) \rightarrow \Box A \\
\mathsf{J}_1: & \Box(C \rightarrow D) \rightarrow C \rhd D \\
\mathsf{J}_2: & (C \rhd D) \wedge (D \rhd E) \rightarrow C \rhd E \\
\mathsf{J}_3: & (C \rhd E) \wedge (D \rhd E) \rightarrow C \vee D \rhd E \\
\mathsf{J}_4: & C \rhd D \rightarrow (\Diamond C \rightarrow \Diamond D) \\
\mathsf{J}_5: & \Diamond A \rhd A
\end{array}
$$

The logic that arises from only the provability schemes L_1-L_3 is often called **GL** after Gödel and Löb. In this logic we have only formulas which are built up using the $\Box$ modality. We call this class of formulas $\mathsf{Form}_{\mathbf{GL}}$.

Two other prominent principles are $\mathsf{M}: A \rhd B \rightarrow A \wedge \Box C \rhd B \wedge \Box C$ and $\mathsf{P}: A \rhd B \rightarrow \Box(A \rhd B)$. The logic that arises by adding more axiom schemes to **IL** is denoted by **IL** with the names of the principles postfixed to it.

[3] A theory is reflexive if it proves the consistence of any finitely axiomatized subtheory. It is essentially reflexive if all its finite extensions are reflexive.

For no theory T that is neither finitely axiomatizable nor essentially reflexive, **IL**(T) is known. PRA is such a theory.

3 What is PRA?

Primitive Recursive Arithmetic, we will write PRA, is a theory that has been studied extensively in the literature. We can think of PRA as the theory with minimal strength that can do basic reasoning about primitive recursion. In a rudimentary form PRA was first introduced by Skolem in 1923. (See for a translation [Sko67].) The emergence of PRA is best understood in the light of Hilbert's programme and finitism (see [Tai81]).

The precise formulation is not very much to our interest in this paper but the reader may think of it as $\mathrm{I}\Delta_0$ (see for example [HP93]) together with the Σ_1 induction rule. The latter allows one to conclude $\forall x\, \sigma(x)$ from $\sigma(0)$ and $\forall x\, (\sigma(x) \to \sigma(x+1))$ whenever σ is a Σ_1-formula.

It is well known that PRA is a reflexive theory but not essentially reflexive. However, any extension of PRA by Σ_2-sentences is reflexive (see [Bek97]). This important feature of PRA is reflected in our treatise of a lowerbound of **IL**(PRA). It is worth noting that we use no specific properties of PRA in providing an upperbound for **IL**(PRA) and indeed our results hold for a large class of theories.

4 A specific research tool: restricting the possible arithmetical realizations.

As we mentioned before, it is unknown what is **IL**(PRA). In this situation lower and upper bounds are already quite informative. This section makes some comments on these bounds. Also we shall reflect a bit on one technique that is used in determining upperbounds.

A lowerbound PRA certainly is a reasonable theory according to [JV00]. From [JV00] we thus get for free that $\mathbf{IL}\mathsf{M}_0\mathsf{P}_0\mathsf{W} \subseteq \mathbf{IL}$(PRA). With these letters we refer to the corresponding schemata:

$$\begin{aligned}
&\mathsf{M}_0 : A \rhd B \to \Diamond A \wedge \Box C \rhd B \wedge \Box C\\
&\mathsf{P}_0 : A \rhd \Diamond B \to \Box(A \rhd B)\\
&\mathsf{W} : A \rhd B \to A \rhd B \wedge \Box\neg A
\end{aligned}$$

In [Joo03] two more interpretability principles of PRA are formulated.

$$\begin{aligned}
&\mathsf{B} : A \rhd B \to A \wedge \Box C \rhd B \wedge \Box C\\
&\mathsf{Z} : (A \rhd B) \wedge (B \rhd A) \to A \rhd A \wedge B
\end{aligned}$$

In B we require that A be an ES_2 (essentially Σ_2) formula. In Z we require that both A and B be ED_2 (essentially Δ_2) formulas. These two classes of formulas are defined as follows.

$$\begin{array}{lcl} ES_2 & := & \Box\mathsf{Form}_{\mathbf{IL}} \mid \neg\Box\mathsf{Form}_{\mathbf{IL}} \mid ES_2 \wedge ES_2 \mid ES_2 \vee ES_2 \mid \neg(ES_2 \rhd \mathsf{Form}_{\mathbf{IL}}) \\ ED_2 & := & \Box\mathsf{Form}_{\mathbf{IL}} \mid \neg ED_2 \mid ED_2 \wedge ED_2 \mid ED_2 \vee ED_2 \end{array}$$

Consequently $\mathbf{IL}\mathsf{BM}_0\mathsf{P}_0\mathsf{WZ}$ is also a lowerbound for $\mathbf{IL}$(PRA).

An upperbound In our Definition 3 we defined $\mathbf{IL}$(T) to be the set of all interpretability principles of T. An interpretability principle of T is a modal formula in $\mathsf{Form}_{\mathbf{IL}}$ that is provable in T under any arithmetical realization.

Let $\mathsf{Sub}(\Gamma)$ be the set of realizations that take their values in Γ. We define the Γ-interpretability logic of T to be set of all formulas in $\mathsf{Form}_{\mathbf{IL}}$ that are provable in T under any realization in $\mathsf{Sub}(\Gamma)$. We denote this logic by $\mathbf{IL}_\Gamma$(T). Clearly we have that $\mathbf{IL}_\Delta(\mathrm{T}) \subseteq \mathbf{IL}_\Gamma(\mathrm{T})$ whenever $\Gamma \subseteq \Delta$. This observation can be used to obtain a rough upperbound for $\mathbf{IL}$(PRA). In order to do so, we first calculate the Γ-*provability* logic of PRA for a specific Γ. This is defined completely analogously to its interpretability variant and is denoted by $\mathbf{PL}_\Gamma$(PRA).

First we define the set $\mathcal{B}$ of arithmetical sentences as follows.

$$\mathcal{B} := \bot \mid \top \mid \Box(\mathcal{B}) \mid \Diamond(\mathcal{B}) \mid \mathcal{B} \to \mathcal{B} \mid \mathcal{B} \vee \mathcal{B} \mid \mathcal{B} \wedge \mathcal{B}$$

Definition 4 *The logic* **RGL** *is obtained by adding the linearity axiom schema* $\Box(\Box A \to B) \vee \Box(\boxdot B \to A)$ *to* **GL**. *Here* $\boxdot B$ *is an abbreviation of* $B \wedge \Box B$.

The logic **RGL** (the **R** stands for restricted) has been considered before in the literature. It is the system J in Chapter 13 of Boolos' book [Boo93]. Ever since Solovay (see [Sol76]) we know that **GL** is the provability logic of any strong enough theory and certainly for PRA.

In the proof below we will make use of the standard modal semantics for **GL**. A **GL**-frame F is a pair $\langle W, R\rangle$ where W is a finite non-empty set of worlds and R is a transitive conversely well-founded relation on it. A **GL**-model is a triple $\langle W, R, \Vdash\rangle$. Here $\Vdash$ is a relation on $W \times \mathsf{Form}_{\mathbf{GL}}$ such that for all $m \in M$ the set $\{A \in \mathsf{Form}_{\mathbf{GL}} \mid m \Vdash A\}$ is a maximal **GL**-consistent one. Moreover we demand $m \Vdash \Box A \Leftrightarrow \forall n\ (mRn \to n \Vdash A)$. We write $M \models A$ and say that A holds on M if for all $m \in M$ we have $m \Vdash A$. For F a frame we write $F \models A$ if A holds on any model that has F as its underlying frame. It is well known that $\mathbf{GL} \vdash A$ if and only if A holds on all finite transitive and conversely well-founded models.

Theorem 5 $\mathbf{PL}_{\mathcal{B}}(\mathrm{PRA}) = \mathbf{RGL}$

Proof of Theorem 5. Let L_n be the linear frame with n elements. For convenience we call the bottom world $n-1$ and the top world 0. It is well known that $\mathbf{RGL} \vdash A \Leftrightarrow \forall n\ (L_n \models A)$. Our proof will thus consist of showing that $\forall * \in \mathsf{Sub}(\mathcal{B})\ \mathrm{PRA} \vdash A^* \Leftrightarrow \forall n\ (L_n \models A)$.

For the $\Leftarrow$ direction we assume that $\exists * \in \mathsf{Sub}(\mathcal{B})\ \mathrm{PRA} \nvdash A^*$ and show that for some $m \in \omega$, $L_m \not\models A$. So, fix a $*$ for which $\mathrm{PRA} \nvdash A^*$. The arithmetical formula A^* can be seen as a formula in the closed fragment of **GL**. By the completeness of **GL** we can find a **GL** model such that $M, x \Vdash \neg A^*$. By $\rho(y)$ we denote the rank of y, that is, the length of the longest R-chain that starts in y. Let $\rho(x) = n$. As the valuation of $\neg A^*$ at x solely depends on the rank of x (see for example [Boo93], Chapter 7, Lemma 3), we see that $L_{n+1}, n \Vdash \neg A^*$ for every possible valuation on L_{n+1} (we also denote this by $L_{n+1}, n \models \neg A^*$). We define $\mathbf{L}_{n+1}, m \Vdash p \Leftrightarrow L_{n+1}, m \models p^*$. It is clear that $\mathbf{L}_{n+1}, n \Vdash \neg A$.

For the $\Rightarrow$ direction we fix some $n \in \omega$ such that $L_n \not\models A$ and construct a $*$ in $\mathsf{Sub}(\mathcal{B})$ such that $\mathrm{PRA} \nvdash A^*$. Let $\mathbf{L}_n$ be a model with domain L_n such that $\mathbf{L}_n, n-1 \Vdash \neg A$. Instead of applying the Solovay construction we can assign to each world m the arithmetical sentence

$$\varphi_m := \Box^{m+1}_{\mathrm{PRA}} \bot \wedge \Diamond^m_{\mathrm{PRA}} \top.$$

(We define $\Box^0_{\mathrm{PRA}} \bot := \bot$ and $\Box^{n+1}_{\mathrm{PRA}} \bot := \Box_{\mathrm{PRA}}(\Box^n_{\mathrm{PRA}} \bot)$. From now on we will omit the subscript PRA.) It is easy to see that

1. $\mathrm{PRA} \vdash \varphi_l \to \neg \varphi_m \quad$ if $l \neq m$,
2. $\mathrm{PRA} \vdash \varphi_l \to \Box(\bigvee_{m<l} \varphi_m)$,
3. $\mathrm{PRA} \vdash \varphi_l \to \bigwedge_{m<l} \Diamond \varphi_m$.

We set $p^* := \bigvee_{\mathbf{L}_n, m \Vdash p} \varphi_m$. Notice that $*$ is in $\mathsf{Sub}(\mathcal{B})$. Using 1, 2 and 3 we can prove a truth lemma, that is, for all m

$$\begin{array}{ll} \mathbf{L}_n, m \Vdash C \Rightarrow \mathrm{PRA} \vdash \varphi_m \to C^* & \text{and} \\ \mathbf{L}_n, m \nVdash C \Rightarrow \mathrm{PRA} \vdash \varphi_m \to \neg C^*. & \end{array}$$

By this truth-lemma, $\mathbf{L}_n, n-1 \Vdash \neg A \Rightarrow \mathrm{PRA} \vdash \varphi_{n-1} \to (\neg A)^*$ and consequently $\mathrm{PRA} \vdash \Diamond \varphi_{n-1} \to \neg \Box A^*$. Thus $\mathbb{N} \models \Diamond \varphi_{n-1} \to \neg \Box A^*$. As φ_{n-1} is consistent with PRA we see that $\mathbb{N} \models \Diamond \varphi_{n-1}$ whence $\mathbb{N} \models \neg \Box A^*$ and thus $\mathrm{PRA} \nvdash A^*$. QED

Definition 6 *The logic* **RIL** *is obtained by adding the linearity axiom schema* $\Box(\Box A \to B) \vee \Box(\boxdot B \to A)$ *to* **ILW**.

Theorem 7 $\mathbf{RIL} = \mathbf{IL}_{\mathcal{B}}(\mathrm{PRA})$

PROOF OF THEOREM 7. We will expose a translation from formulas φ in $\mathsf{Form}_{\mathbf{IL}}$ to formulas φ^{tr} in $\mathsf{Form}_{\mathbf{GL}}$ such that

$$\mathbf{RIL} \vdash \varphi \Leftrightarrow \mathbf{RGL} \vdash \varphi^{\mathsf{tr}} \quad (*)$$
and
$$\mathbf{RIL} \vdash \varphi \leftrightarrow \varphi^{\mathsf{tr}}. \quad (**)$$

If we moreover know $(***)$: $\mathbf{RIL} \vdash \varphi \Rightarrow \forall * \in \mathsf{Sub}(\mathcal{B})\ \mathrm{PRA} \vdash \varphi^*$ we would be done. For then we have by $(**)$ and $(***)$ that

$$\forall * \in \mathsf{Sub}(\mathcal{B})\ \mathrm{PRA} \vdash \varphi^* \leftrightarrow (\varphi^{\mathsf{tr}})^*$$

and consequently

$$\begin{array}{ll} \forall * \in \mathsf{Sub}(\mathcal{B})\ \mathrm{PRA} \vdash \varphi^* & \Leftrightarrow \\ \forall * \in \mathsf{Sub}(\mathcal{B})\ \mathrm{PRA} \vdash (\varphi^{\mathsf{tr}})^* & \Leftrightarrow \\ \mathbf{RGL} \vdash \varphi^{\mathsf{tr}} & \Leftrightarrow \\ \mathbf{RIL} \vdash \varphi. & \end{array}$$

We first see that $(***)$ holds. From our remarks concerning a lowerbound of $\mathbf{IL}(\mathrm{PRA})$ we know that $\mathbf{ILW} \subseteq \mathbf{IL}_{\mathcal{B}}(\mathrm{PRA})$. Thus it remains to show that $\mathrm{PRA} \vdash \Box(\Box A^* \to B^*) \vee \Box(\boxdot B^* \to A^*)$ for any formulas A and B in $\mathsf{Form}_{\mathbf{IL}}$ and any $*\in\mathsf{Sub}(\mathcal{B})$. As any formula in the closed fragment of $\mathbf{ILW}$ is equivalent to a formula in the closed fragment of $\mathbf{GL}$ (see [HŠ91]), Theorem 5 gives us that indeed the linearity axiom holds for the closed fragment of $\mathbf{GL}$.

Our translation will be the identity translation except for $\rhd$. In that case we define

$$(A \rhd B)^{\mathsf{tr}} := \Box(A^{\mathsf{tr}} \to (B^{\mathsf{tr}} \vee \Diamond B^{\mathsf{tr}})).$$

We first see that we have $(**)$. It is sufficient to show that $\mathbf{RIL} \vdash p \rhd q \to \Box(p \to (q \vee \Diamond q))$. We reason in $\mathbf{RIL}$. An instantiation of the linearity axiom gives us $\Box(\Box\neg q \to (\neg p \vee q)) \vee \Box((\neg p \vee q) \wedge \Box(\neg p \vee q) \to \neg q)$. The first disjunct immediately yields $\Box(p \to (q \vee \Diamond q))$.

In case of the second disjunct we get by propositional logic $\Box(q \to \Diamond(p \wedge \neg q))$ and thus also $\Box(q \to \Diamond p)$. Now we assume $p \rhd q$. By W we get $p \rhd q \wedge \Box\neg p$. Together with $\Box(q \to \Diamond p)$, this gives us $p \rhd \bot$, that is $\Box\neg p$. Consequently we have $\Box(p \to (q \vee \Diamond q))$.

We now prove $(*)$. By induction on $\mathbf{RIL} \vdash \varphi$ we see that $\mathbf{RGL} \vdash \varphi^{\mathsf{tr}}$. All the specific interpretability axioms turn out to be provable under our translation in $\mathbf{GL}$. The only axioms where the $\Box A \to \Box\Box A$ axiom scheme is really used is in J_2 and J_4. To prove the translation of W we also need L_3.

If $\mathbf{RGL} \vdash \varphi^{\mathsf{tr}}$ then certainly $\mathbf{RIL} \vdash \varphi^{\mathsf{tr}}$ and by $(**)$, $\mathbf{RIL} \vdash \varphi$.

QED

We thus see that **RIL** is an upperbound for **IL**(PRA). Using the translation from the proof of Theorem 7, it is not hard to see that both the principles P and M are provable in **RIL**. Choosing larger Γ will generally yield a smaller $\mathbf{IL}_\Gamma$(PRA) and thus a sharper upperbound.

Finally we remark that if $\mathbf{RGL} \nvdash \varphi$, then φ is certainly not a provability principle. But in this case we can find a counterexample with a "clear (meta)mathematical" content.

I would like to thank Lev Beklemishev for many enligthening discussions and for pointing out an error in an earlier version of this paper. Also I would like to thank an anonymous reviewer who helped improving the readability of this paper.

References

[Bek97] L.D. Beklemishev. Induction rules, reflection principles, and provably recursive functions. *Annals of Pure and Applied Logic*, 85:193–242, 1997.

[Boo93] G. Boolos. *The Logic of Provability*. Cambridge University Press, Cambridge, 1993.

[Gre96] M.J. Greenberg. *Euclidean and Non-Euclidean Geometries, 3rd edition*. Freeman, 1996.

[HP93] P. Hájek and P. Pudlák. *Metamathematics of First Order Arithmetic*. Springer-Verlag, Berlin, Heidelberg, New York, 1993.

[HŠ91] P. Hájek and V. Švejdar. A note on the normal form of closed formulas of interpretability logic. *Studia Logica*, 50:25–38, 1991.

[Joo03] J.J. Joosten. The closed fragment of the interpretability logic of PRA with a constant for $\mathrm{I}\Sigma_1$. Logic Group Preprint Series 128, University of Utrecht, February 2003.

[JV00] J.J. Joosten and A. Visser. The interpretability logic of *all* reasonable arithmetical theories. *Erkenntnis*, 53(1–2):3–26, 2000.

[Sko67] T. Skolem. The foundations of elementary arithmetic established by means of the recursive mode of thought, without the use of apparent variables ranging over infinite domains. In J. van Heijenoort, editor, *From Frege to Gödel*, pages 302–333. iUniverse, Harvard, 1967.

[Sol76] R.M. Solovay. Provability interpretations of modal logic. *Israel Journal of Mathematics*, 28:33–71, 1976.

[Tai81] W. Tait. Finitism. *Journal of Philosophy*, 78:524–546, 1981.

[TMR53] A. Tarski, A. Mostowski, and R. Robinson. *Undecidable theories*. North–Holland, Amsterdam, 1953.

[Vis97] A. Visser. An overview of interpretability logic. In M. Kracht, M. de Rijke, and H. Wansing, editors, *Advances in modal logic '96*, pages 307–359. CSLI Publications, Stanford, CA, 1997.

A Simple Semantics for Destructive Updates

Ján Kľuka
Institute of Informatics, Comenius University, Mlynská dolina, 842 48 Bratislava, Slovakia
jan.kluka@fmph.uniba.sk

Abstract. Many algorithms intended to destructively update arrays can be implemented in declarative programming languages by programs operating on lists. These programs are easy to understand and prove, but possibly inefficient. We sketch a semantic method ensuring that a suitably annotated program operating on lists can be correctly transformed into an efficient program that destructively updates arrays.

1 Introduction

Many algorithms on arrays can be presented in declarative languages in terms of list constructions and concatenations. Such declarative programs operating on lists are often simpler to understand and easier to prove than naive translations of array algorithms to declarative programs that use array indexing and updates. On the other hand, programs that use indexing are easy to compile to a lower level code that actually indexes and destructively updates arrays.

The note [Vod02] has suggested how to transform a declarative program that operates on lists into one that uses array indexing and updates. In this paper, we describe how a programmer should control the transformation by annotating the program that operates on lists. Moreover, we outline a method for ensuring correctness of the program resulting from the transformation. The method is based on checking certain semantic properties of an intermediate form of the original program. We use the algorithm of quicksort partitioning as an example.

Various approaches to the problem of destructive updates in declarative languages have been proposed. Static analysis enables the compiler to turn some updates into destructive. It is based either on specialized nonstandard semantics [Hud87, Blo94], or on purely syntactic properties of programs [Sha99]. In other approaches, the programmer marks some updates as destructive, and the program is then checked by typing [Wad90], or

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

abstract interpretation [Ode91]. Our approach lets the programmer annotate the program in a way that does not interfere with syntax, and provides semantic conditions in standard semantics necessary for the correctness of desired destructive updates. For most programs, these conditions can be proved automatically by a proof assistant. We define programs and express their properties in the same language—Peano arithmetic—in the style of the programming language and proof assistant *CL* [KV99, CL].

2 Two Declarative Implementations of Quicksort Partitioning

2.1 Preliminaries: Pairs, Lists, and Clausal Definitions. All functions and predicates in this paper are definable in Peano arithmetic as functions and predicates over natural numbers. We use the pairing function written in infix form as x, y to construct both lists and pairs. This function is best thought of as the `(cons x y)` operation of Lisp. The pairing function is defined so that no pair equals the number 0, which thus serves as the *empty list* (`()` in Lisp).

Throughout the paper, we use the *singleton list* function $[x]$ which yields the list $x, 0$ (an equivalent of Lisp's `(list x)`), and the *concatenation* function $x \oplus y$ (analogous to `(append x y)` in Lisp) yielding a list of all elements of x followed by all elements of y. We also use the predicate $x \; \varepsilon \; y$ which holds if x is an element of the list y. The *modified subtraction* $x \dot{-} y$, mentioned in the next paragraph, equals $x - y$ if $x \geq y$, and 0 otherwise.

Functions and predicates are defined either *explicitly* (in which case the definition is preceded by the symbol $\vdash_{\mathrm{PAx}}$) or *clausally*. Clausal definitions resemble the clausal definitions of Prolog. Unlike in Prolog, we define both predicates and functions clausally in our language, which is why we strictly demand the clauses to be mutually exclusive. We allow pattern matching in both clause heads (within formal arguments) and clause bodies (pattern matching may occur only on the right-hand side of an identity).

More information on clausal definitions and the pairing function can be found in [KV99].

2.2 A Naive Declarative Partitioning Operating on an Array. In Peano arithmetic, we can represent arrays as lists. If a is an array, let $(a)_i$ denote its element at index i, and $a[i \leftrightarrow j]$ a new array which is just like a, but with elements at indices i and j swapped. If our compiler can compile these operations into actual indexing and destructive array updates, they take constant time. Then we can define an efficient quicksort partitioning function $Part_n$ as shown in Fig. 1.

Let $a[i \,..\, j)$ denote the *segment* of the array a between indices i (including) and j (excluding). The function takes three arguments—an ar-

$$
\begin{aligned}
&Part_n(a,l,u) = a, l \dot{-} 1 && \leftarrow l = u\\
&Part_n(a,l,u) = Part_n(a,l,u \dot{-} 1) && \leftarrow l < u \wedge (a)_{u \dot{-} 1} \geq (a)_{l \dot{-} 1}\\
&Part_n(a,l,u) = a[l \dot{-} 1 \leftrightarrow l], l && \leftarrow l < u \wedge (a)_{u \dot{-} 1} < (a)_{l \dot{-} 1} \wedge l + 1 = u\\
&Part_n(a,l,u) = Part_n(a_2, l+1, u) && \leftarrow l < u \wedge (a)_{u \dot{-} 1} < (a)_{l \dot{-} 1} \wedge l + 1 < u \wedge\\
& && \quad a[l \dot{-} 1 \leftrightarrow l] = a_1 \wedge a_1[l \dot{-} 1 \leftrightarrow u \dot{-} 1] = a_2
\end{aligned}
$$

Figure 1: A naive declarative implementation of quicksort partitioning.

ray a and indices l, u—, partitions the segment $a[l \dot{-} 1 \,..\, u)$ according to the pivot $(a)_{l \dot{-} 1}$, and returns a pair consisting of the modified array a_1 and the new position l_1 of the pivot. Elements of the segment $a[l \dot{-} 1 \,..\, u)$ that are less than the pivot are placed in $a_1[l \dot{-} 1 \,..\, l_1)$, those greater than or equal to the pivot are in $a_1[l_1 + 1 \,..\, u)$. This is expressed by the following property:

$$
\begin{aligned}
&Part_n(a,l,u) = a_1, l_1 \wedge 0 < l \wedge l \leq u \wedge u \leq L(a) \rightarrow\\
&\quad a_1 \sim a \wedge l \dot{-} 1 \leq l_1 \wedge l_1 < u \wedge\\
&\quad a_1[0 \,..\, l \dot{-} 1) = a[0 \,..\, l \dot{-} 1) \wedge a_1[u \,..\, L(a)) = a[u \,..\, L(a)) \wedge\\
&\quad \forall x(x \,\varepsilon\, a_1[l \dot{-} 1 \,..\, l_1) \rightarrow x < (a)_{l \dot{-} 1}) \wedge (a_1)_{l_1} = (a)_{l \dot{-} 1} \wedge\\
&\quad \forall x(x \,\varepsilon\, a_1[l_1 + 1 \,..\, u) \rightarrow x \geq (a)_{l \dot{-} 1})
\end{aligned}
$$

where $L(a)$ denotes the length of the array a, and $a_1 \sim a$ is a predicate expressing that a_1 is a permutation of a. The property and consequently its proof rely heavily on indexing and arithmetic which makes them rather complicated. Besides, the property must contain explicit statement that elements at indices below $l \dot{-} 1$ and above u remain unchanged.

2.3 An Inefficient Implementation Operating on Lists. If correctness is our primary concern, we can define a partitioning function operating on two lists r and x as shown in Fig. 2 (for the time being, ignore the arrows). The list r represents the array segment containing only the pivot, the list x represents the segment to be partitioned. The function *Part* yields a triple consisting of a list of elements of x less than the pivot, a singleton (atomic) list containing only the pivot, and a list of elements of x greater than or equal to the pivot.

This function is quite inefficient—it performs pattern matching at the end of its second argument and several list concatenations. On the other hand, its main property

$$
\begin{aligned}
&Part([p], x) = l, r, u \rightarrow\\
&\quad [p] \oplus x \sim l \oplus r \oplus u \wedge \forall v(v \,\varepsilon\, l \rightarrow v < p) \wedge r = [p] \wedge \forall v(v \,\varepsilon\, u \rightarrow v \geq p)
\end{aligned}
$$

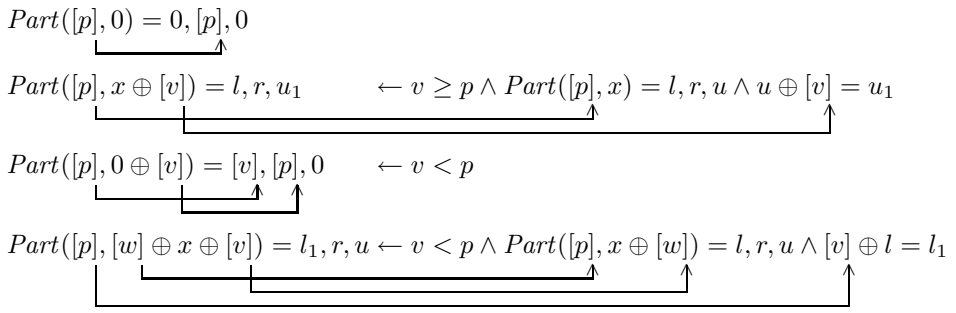


Figure 2: A partitioning function operating on lists.

is expressed in terms of operations on lists, and is considerably easier to prove than the corresponding property of $Part_n$.

In the following paragraphs, we shall sketch how, given some additional information provided by the programmer, we can mechanically transform this function into one that operates destructively on an array similarly to $Part_n$. We also outline how the correctness of the transformed function is ensured.

2.4 Annotation and Additional Requirements. *Annotations* represent a programmer's intention to place certain values at specific indices in an array. We represent such an intention by an arrow leading from a pattern matching of a singleton list to a construction of another singleton list. The value used in the destination singleton is intended to be placed at the index of the source singleton. The annotation of function *Part* displayed in Fig. 2 closely follows the pattern of indexing from the function $Part_n$.

Apart from annotating the reuse of indices, the programmer should state additional requirements for the transformation of the *Part* function:

1. Its two arguments are supposed to be adjacent segments of an input array, and the members of the result triple are supposed to be adjacent segments of an output array.

2. The arguments reside at the same indices of the input array as the indices occupied by the members of the result triple in the output array.

3. The input array is the same as the output array, which is thus updated *destructively*.

$$
\begin{array}{ll}
Lln(0) & \\
Lln((v,\ell),w) \leftarrow Lln(w) & [v]^{\ell} = (v,\ell),0 \\
Labels(0) = 0 & Unl(0) = 0 \\
Labels((v,\ell),w) = \ell, Labels(w) & Unl((v,\ell),w) = v, Unl(w)
\end{array}
$$

$$
\begin{array}{c}
Lln(x \oplus y) \leftrightarrow Lln(x) \wedge Lln(y) \\
Unl([v]^{\ell}) = [v] \\
Lln(x \oplus y) \rightarrow Unl(x \oplus y) = Unl(x) \oplus Unl(y)
\end{array}
$$

Figure 3: Labeled lists.

$$
\begin{array}{ll}
Part_{\ell}([p]^{\ell_1},0) = 0,[p]^{\ell_1},0 & \\
Part_{\ell}([p]^{\ell_1},x \oplus [v]^{\ell_2}) = l,r,u_1 & \leftarrow v \geq p \wedge Part_{\ell}([p]^{\ell_1},x) = l,r,u \wedge \\
 & \quad u \oplus [v]^{\ell_2} = u_1 \\
Part_{\ell}([p]^{\ell_1},0 \oplus [v]^{\ell_2}) = [v]^{\ell_1},[p]^{\ell_2},0 & \leftarrow v < p \\
Part_{\ell}([p]^{\ell_1},[w]^{\ell_3} \oplus x \oplus [v]^{\ell_2}) = l_1,r,u & \leftarrow v < p \wedge Part_{\ell}([p]^{\ell_3},x \oplus [w]^{\ell_2}) = l,r,u \wedge \\
 & \quad [v]^{\ell_1} \oplus l = l_1
\end{array}
$$

Figure 4: The labeled version of the function *Part*.

3 From Lists to Destructively Updated Arrays

3.1 Labeled Lists. The intention to put a value at a certain index can be expressed semantically in terms of labeled lists. A *labeled list* is a list of pairs v, ℓ where v is a value, and ℓ is a label representing an array index where the value resides.

The predicate *Lln* (cf. Fig. 3) is defined to hold for such lists. We shall construct labeled lists by concatenation and the labeled-singleton function $[v]^{\ell}$. We also define two projection functions: *Labels* yielding a list of labels of a labeled list, and *Unl* yielding an unlabeled version of such a list, i.e., its list of values.

3.2 Transforming a Function Operating on Lists to a Function Operating on Labeled Lists. Given the annotated function, we now mechanically construct a function operating on labeled lists such that it manipulates the values of list elements just like the function *Part*, and manipulates their labels as prescribed by the annotation. We call the new function $Part_{\ell}$, or the *labeled version* of the function *Part*. Its clausal definition is shown in Fig. 4.

$$\vdash_{\mathrm{PAx}} x \boxplus y \leftrightarrow \forall p \forall v_1 \forall \ell_1 \forall v_2 \forall \ell_2 \forall q (x = p \oplus [v_1]^{\ell_1} \wedge [v_2]^{\ell_2} \oplus q = y \rightarrow \ell_1 + 1 = \ell_2)$$

$$\vdash_{\mathrm{PAx}} \mathit{Vect}(x) \leftrightarrow \forall y \forall z (y \oplus z = x \rightarrow y \boxplus z)$$

$$\vdash_{\mathrm{PAx}} x \approx y \leftrightarrow \mathit{Lln}(x) \wedge \mathit{Lln}(y) \wedge \mathit{Labels}(x) = \mathit{Labels}(y)$$

$x \approx x$	$x \boxplus 0$
$x \approx y \rightarrow y \approx x$	$0 \boxplus x$
$x \approx y \wedge y \approx z \rightarrow x \approx z$	$x \approx y \rightarrow x \boxplus z \leftrightarrow y \boxplus z$
$x \approx y \leftrightarrow z \oplus x \approx z \oplus y$	$x \approx y \rightarrow z \boxplus x \leftrightarrow z \boxplus y$
$x \approx y \leftrightarrow x \oplus z \approx y \oplus z$	$\mathit{Vect}(x \oplus y) \leftrightarrow \mathit{Vect}(x) \wedge \mathit{Vect}(y) \wedge x \boxplus y$
$[v_1]^{\ell_1} \approx [v_2]^{\ell_2} \leftrightarrow \ell_1 = \ell_2$	$x \approx y \rightarrow \mathit{Vect}(x) \leftrightarrow \mathit{Vect}(y)$

Figure 5: The Adjacency ($\boxplus$), Vector (Vect), and the Same-Sequences-of-Labels ($\approx$) predicates.

3.3 The Basic Correctness Property of the Transformed Function. Before continuing any further, we have to be sure that the results of the function Part_ℓ are equivalent to those of the function Part except for labeling, which is stated formally as follows:

$$\mathit{Lln}(r \oplus x) \rightarrow$$
$$\mathit{Part}_\ell(r, x) = l, r_1, u \leftrightarrow \mathit{Part}(\mathit{Unl}(r), \mathit{Unl}(x)) = \mathit{Unl}(l), \mathit{Unl}(r_1), \mathit{Unl}(u)\,.$$

This can be proved automatically on object level by a proof assistant from obvious properties of unlabeling mentioned in Fig. 3, or we can prove on the meta-level that any function and its labeled version behave the same except for labeling.

3.4 Labeled Lists and Array Segments. Our next step is to show that the function Part_ℓ meets requirements 1 and 2 from Par. 2.4. We need some definitions (see Fig. 5) to do that.

We say that two labeled lists x and y are *adjacent* ($x \boxplus y$) iff the first label in y immediately succeeds the last label in x. We call a labeled list representing an array segment a *vector*, and define the predicate Vect to hold for a list x iff every two lists y and z such that $x = y \oplus z$ are adjacent. Finally, $x \approx y$ means that the labeled lists x and y have *identical lists of labels*.

The first two requirements from Par. 2.4 can now be expressed by the two formulas (proof obligations) below:

$$\mathit{Lln}(r \oplus x) \wedge \mathit{Vect}(r \oplus x) \wedge \mathit{Part}_\ell(r, x) = l, r_1, u \rightarrow \mathit{Vect}(l \oplus r_1 \oplus u) \quad (1)$$
$$\mathit{Lln}(r \oplus x) \wedge \mathit{Vect}(r \oplus x) \wedge \mathit{Part}_\ell(r, x) = l, r_1, u \rightarrow r \oplus x \approx l \oplus r_1 \oplus u \quad (2)$$

Properties of the predicates *Vect* and $\approx$ together with (2) imply (1). It is therefore sufficient to prove the second obligation. A sufficiently sophisticated proof assistant can construct the proof automatically when given the properties from Fig. 5.

Nevertheless, there is another problem to be addressed here. A concatenation of array segments can be performed if and only if they are adjacent. It can then be carried out in constant time, since it involves only calculation of limits of the resulting segment. Note that the result of a concatenation of labeled lists is a vector (i.e., a representation of an array segment) iff the concatenated lists are adjacent vectors (Fig. 5). We refer to a concatenation whose result is a vector as *admissible*.

To outline how to check that all concatenations are admissible, let $\varphi[\vec{a}]$ be a formula representing the requirements imposed on the arguments $\vec{a}$ of a labeled version f_ℓ of a function, and assume a concatenation occurs in the following context in a clause of that version ($\psi[x, y]$ and ω are possibly empty conjunctions of literals):

$$f_\ell(\vec{a}) = \vec{r} \leftarrow \psi[x, y] \wedge x \oplus y = z \wedge \omega\,.$$

Then to ensure that this concatenation is admissible, it is sufficient to prove

$$\varphi[\vec{a}] \wedge \psi[x, y] \rightarrow \mathit{Vect}(x \oplus y)\,.$$

Formulas of this type are mechanically derivable from the function definition, and if the annotation is correct, they are automatically provable using the properties of vectors, the requirement (2), and the properties of the predicate $\approx$.

In the particular case of the function Part_ℓ, we have $\varphi[r, x] \equiv \mathit{Lln}(r \oplus x) \wedge \mathit{Vect}(r \oplus x)$. From the last clause of the function, the following proof obligations are generated:

$$\varphi\big[[p]^{\ell_1}, [w]^{\ell_3} \oplus x \oplus [v]^{\ell_2}\big] \rightarrow \mathit{Vect}(x \oplus [w]^{\ell_2})$$

$$\varphi\big[[p]^{\ell_1}, [w]^{\ell_3} \oplus x \oplus [v]^{\ell_2}\big] \wedge v < p \wedge \mathit{Part}_\ell([p]^{\ell_3}, x \oplus [w]^{\ell_2}) = l, r, u \rightarrow \mathit{Vect}([v]^{\ell_1} \oplus l)\,.$$

3.5 Labeled Lists and Destructive Updates. Recall that in Par. 2.4, an arrow leading from a pattern matching of a singleton list to a construction of another singleton list, say $[v]$, declares a programmer's intention to put the value v at the same index where the pattern matching occurred. In a labeled version of a function, the index is represented by a label.

Suppose we have a clause whose body contains a pattern matching $x = [v]^\ell \oplus y$ followed by a construction $[w]^\ell = z$. The construction is intended to destructively update an array at the index ℓ. If x is implemented as a pair of indices limiting a segment of that array, then x cannot be used further in

$$\vdash_{\mathrm{PAx}} x \# y \leftrightarrow \forall \ell \forall v_1 \forall v_2 (v_1, \ell \notin x \vee v_2, \ell \notin y)$$
$$\vdash_{\mathrm{PAx}} x \between y \leftrightarrow \forall \ell \forall v_1 \forall v_2 (v_1, \ell \varepsilon x \wedge v_2, \ell \varepsilon y \rightarrow v_1 = v_2)$$

$$x \# 0$$
$$Lln(x) \rightarrow x \# x \leftrightarrow x = 0$$
$$x \# y \rightarrow y \# x$$
$$x \# [v]^{\ell} \leftrightarrow \forall v (v, \ell \notin x)$$
$$x \# y \oplus z \leftrightarrow x \# y \wedge x \# z$$
$$x \approx y \rightarrow x \# z \leftrightarrow y \# z$$
$$Lln(x \oplus y) \wedge Vect(x \oplus y) \rightarrow x \# y$$

$$x \between 0$$
$$x \between y \rightarrow y \between x$$
$$x \between y \oplus z \leftrightarrow x \between y \wedge x \between z$$
$$x \# y \rightarrow x \between y$$
$$Lln(x \oplus y) \wedge Vect(x \oplus y) \rightarrow x \oplus y \between x$$
$$Lln(x \oplus y) \wedge Vect(x \oplus y) \rightarrow x \oplus y \between y$$

Figure 6: Definitions and selected properties of the Disjoint-Labels predicate $\#$, and the Correctly-Aliased predicate $\between$.

the clause since the list of values of x may be no longer actually present in the corresponding array segment.

A simple solution to this problem is to allow each list variable to be used only once after it is given a value. Although the function *Part* (and consequently $Part_\ell$) complies with this constraint, it is too restrictive—in the above example, using the variable x between the pattern matching and the construction would do no harm.

To deal with this problem in a more permissive semantic manner, we define the predicates shown in Fig. 6. Formula $x \# y$ expresses the fact that the sets of labels of two labeled lists x and y are disjoint. We shall call two lists x and y *correctly aliased* and write $x \between y$ iff for every label ℓ shared by these lists, the value that occurs at label ℓ in the list x is the same as the value occurring at that label in the list y. If these lists are vectors, and they actually share a label, they represent two overlapping array segments.

Let $\varphi[\vec{a}]$ stand for the requirements imposed on the arguments of a labeled version f_ℓ of a function (as in Par. 3.4). Furthermore, assume that labeled-list variables x and y occur in a clause of f_ℓ in the following context (ξ, ψ, and $\omega[x]$ are possibly empty conjunctions of literals, τ and ρ are terms):

$$f_\ell(\vec{a}) = \vec{r} \leftarrow \xi \wedge \tau = x \wedge \psi \wedge \rho = y \wedge \omega[x] \,.$$

The identity $\tau = x$ gives a value to the variable x, $\rho = y$ gives a value to y. After that, x is used in the rest $\omega[x]$ of the clause.

We claim, that if a property of the form

$$\varphi[\vec{a}] \wedge \xi \wedge \tau = x \wedge \psi \wedge \rho = y \rightarrow y \between x$$

holds, then the contents of the array segment denoted by the variable x will not be affected by a destructive update which implements the identity $\rho = y$ if ρ contains a singleton construction. This holds even if variables x and y denote overlapping array segments. The clause context above shows one of three cases. The other two cases apply when the variable x is one of arguments $\vec{a}$, or a member of the result tuple $\vec{r}$; properties that should hold are similar.

As an example, consider the last clause of $Part_\ell$ with atomized list operations:

$$\begin{aligned} Part_\ell([p]^{\ell_1}, [w]^{\ell_3} \oplus x \oplus [v]^{\ell_2}) = l_1, r_1, u \leftarrow\ & v < p \wedge [p]^{\ell_3} = r \wedge \\ & x \oplus [w]^{\ell_2} = x_1 \wedge \\ & Part_\ell(r, x_1) = l, r_1, u \wedge \\ & [v]^{\ell_1} \oplus l = l_1 \end{aligned}$$

The corresponding proof obligations are:

$$\varphi\big[[p]^{\ell_1}, [w]^{\ell_3} \oplus x \oplus [v]^{\ell_2}\big] \rightarrow [p]^{\ell_3} \between x \wedge x \oplus [w]^{\ell_2} \between [p]^{\ell_3}$$

$$\varphi\big[[p]^{\ell_1}, [w]^{\ell_3} \oplus x \oplus [v]^{\ell_2}\big] \wedge Part_\ell([p]^{\ell_3}, x \oplus [w]^{\ell_2}) = l, r_1, u \rightarrow [v]^{\ell_1} \oplus l \between r_1 \wedge [v]^{\ell_1} \oplus l \between u\,.$$

Again, a suitable proof assistant can automatically construct a proof of these obligations, which is based on the properties from Fig. 6 and the property 3.4(2).

3.6 Mechanical Transformation of the Function Operating on Labeled Lists into One Operating on Arrays. Finally, we sketch the transformation of the function $Part_\ell$ to a function operating destructively on an array. Let $a[i := v]$ denote a destructive update of the array a placing the value v at index i. Every vector from the function $Part_\ell$ is transformed into a pair of indices that limit the corresponding array segment. The singleton construction $[v]^\ell$ is transformed into the destructive update $a[\ell := v]$. Concatenations are replaced by calculations of limits of array segments.

After elimination of unnecessary variables, a function $Part_a(a, r, x, t) = a_1, l, r_1, u, t_1$ is obtained, where a is the array to be partitioned, the pair of indices r, x limits the singleton segment containing the pivot, indices x, t limit the segment to be partitioned, l, r_1 limit the segment containing elements less then the pivot, r_1, u limit of the new pivot segment, and indices u, t_1 are limits of the segment of elements greater than the pivot.

References

[Blo94] A. Bloss. Path analysis and the optimization of nonstrict functional languages. *ACM Transactions on Programming Languages and Systems*, 16(3):328–369, 1994.

[CL] The *CL* homepage. http://www.ii.fmph.uniba.sk/cl .

[Hud87] P. Hudak. A semantic model for reference counting and its abstraction. In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Declarative Languages*. Ellis Horwood Ltd., 1987.

[KV99] J. Komara and P. Voda. Theorems of Péter and Parsons in computer programming. In G. Gottlob, E. Grandjean, and K. Seyr, editors, *Proceedings of CSL'98*, number 1584 in LNCS, pages 204–223. Springer Verlag, 1999.

[Ode91] M. Odersky. How to make destructive updates less destructive. In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages, Orlando, Florida*, pages 25–26. ACM Press, 1991.

[Sha99] N. Shankar. *Efficiently Executing PVS*. Technical report, SRI International, Menlo Park, CA, 1999. Available through WWW from http://www.csl.sri.com/users/shankar/shankar-drafts.html.

[Vod02] P. Voda. On modification of declarative arrays, 2002. Available through WWW from http://www.diku.dk/undervisning/2002e/213/segm.ps.

[Wad90] P. Wadler. Linear types can change the world! In M. Broy and C. Jones, editors, *IFIP TC 2 Working Conference on Programming Concepts and Methods, Sea of Galilee, Israel*, pages 347–359. North Holland, 1990.

A New Proof of Decidability for the Modal Logic of Subset Spaces

GISELA KROMMES
FernUniversität Hagen
krommes@aol.com

ABSTRACT. We present a simplified proof of decidability for the bimodal *logic of subset spaces*, introduced by Moss and Parikh, and discuss some difficulties in determining the complexity of the decision problem. Basically designed for elementary reasoning about points and sets in general topology, this logic gives basis to suitable formalisms for reasoning about various subjects, especially topology or the acquisition of knowledge.

1 Introduction

The *logic of subset spaces* of Moss and Parikh [MP92], let us call it *LSS*, combines the two well-known modal systems *S*4 and *S*5. As the name already indicates, the most natural interpretation of *LSS* is not in usual Kripke structures but in *subset spaces* $(X,\mathcal{O})$, where the *S*5-operator *K* quantifies 'horizontally' across points from a given set *X* and the *S*4-operator $\Box$ quantifies 'vertically' across the elements of a distinguished set $\mathcal{O}$ of non-empty subsets of *X*. The interaction of both modalities is basically described by the *cross axiom* $K\Box\varphi \rightarrow \Box K\varphi$.

This interpretation leads to an interesting application of *LSS* as a *topological logic of knowledge*. McKinsey [McK41] first investigated the close connection between *S*4 and topology. He pointed out that the interior operator on a topological space has *S*4-like properties and although $\mathcal{O}$ is not necessarily closed under finite union and intersection, it gives a good intuition to think about $(X,\mathcal{O})$ as a kind of *topological space*. Moreover, the *S*5 component of *LSS* can be taken as the widely used tool for the logical treatment of an agent's knowledge, cf. [FHMV96], i.e. $K\varphi$ can be read as 'the agent knows φ'. In this way *LSS* connects the notion of knowledge with elementary topological reasoning.

What is the benefit of this connection? *LSS* provides a formalism to describe *knowledge acquisition*. In subset spaces, formulae are evaluated at *neighborhood*

Proceedings of the Eighth ESSLLI Student Session.
Balder ten Cate (editor)

situations $(p,U) \in \{(q,V) \in X \times \mathcal{O} \mid q \in V\}$, consisting of a point (or state) p and a neighborhood U thereof.
The basic *LSS* deals only with the knowledge of one agent: U consists of all states the agent considers alternative to his actual state p and the knowledge operator K quantifies across all the states in U. Hence the agent's knowledge depends crucially upon the actual neighborhood of its state p.

If in the actual situation (p,U) the agent doesn't know whether φ holds or not, he considers both as possibilities, so we have

$$p,U \vDash L\varphi \wedge L\neg\varphi$$

(as usual we use $L\varphi$ as an abbreviation of $\neg K\neg\varphi$ and $\Diamond\varphi$ as an abbreviation of $\neg\Box\neg\varphi$). But the more the agent knows, the smaller number of alternatives he takes into account, thus acquisition of knowledge corresponds to shrinking the set of possible alternatives or, in the notion of topology, to shrinking p's neighborhood U. At this point the $S4$-$\Box$ comes into play: forced by the cross axiom, it quantifies across all neighborhoods of p that are subsets of U.

But since points and sets are general tools for formalization, *LSS* can also be useful for the treatment of quite different subjects as the description of decision or selection processes, where step-by-step additional constraints are satisfied. Moreover the underlying ideas gave rise to the development of special variants, which differ in the class of subset spaces, cf. [DMP96] (here is additionally the case treated where $\mathcal{O} \cup \varnothing$ is in fact a topology on X), [Geo97] (treelike spaces), [Hei98a] (spaces satisfying finite chain conditions), [WP02] (directed spaces, which are related to bases of neighborhood filters of topological spaces) or the modalities in use, cf. [Hei98b] ($S4$ weakened to a partial function to obtain a "next-time" operator), [DG00] (two $S4$ systems). The last paper is motivated by the application domain of *continuous and hybrid dynamic systems* with practical use in several engineering tasks.

Syntax, semantics and a list of axioms for *LSS* are introduced in the next Section. Section 3 is dedicated to the decision problem. For this section, we assume some familiarity with canonical models and filtration methods. (For a very accessible introduction see e.g. [BdRV01]). The proof of decidability given in [DMP96] is based on the finite model property (f.m.p.) of *LSS* w.r.t. a special class of models. The f.m.p. is established via filtration of the canonical model using the minimal filtration for both modalities. But proving that the so obtained quotient structure belongs to the relevant class of models turns out to be hard to do. Pursuing a suggestion by Heinemann, we replace the minimal filtration for the $\Box$ modality by its transitive closure, thus having transitivity by definition and cutting away the main difficulties of the original proof. We also use a bigger filter set with better closure properties and for the resulting quotient structure, we can state a nice property which is as easy to imagine as to prove. With this property at hand, the proof is quickly done.

Up to now we could only establish PSPACE as a lower bound for the complexity of the decision problem. We discuss this result and the difficulties in determining the exact complexity briefly in Section 4. We will finish with some concluding remarks in Section 5.

The work presented in this paper is part of our forthcoming diploma thesis.

2 The Logic *LSS*

We start with the definition of the logical language for *LSS*.

Definition 2.1 (Syntax of *LSS*) *Let PV be a recursive*[1] *and countable set of atomic propositions. The set $\mathcal{L}$ of LSS-formulae is recursively generated by:*

$$\varphi ::= A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Box\varphi \mid K\varphi$$

with $A \in PV$, an atomic proposition.

The other Boolean connectives, logical constants and dual modalities are definable in the standard way. The semantical domains of *LSS* are *subset models* and defined as follows:

Definition 2.2 (Subset Models) *A* subset space *is a pair*

$$(X, \mathcal{O})$$

where X is a non-empty set of points (*or states*) *and $\mathcal{O}$ is a set of non-empty subsets of X, called* opens.
A subset model *is a triple*

$$\mathcal{S} = (X, \mathcal{O}, \sigma)$$

where $(X, \mathcal{O})$ is a subset space and σ is a mapping $\sigma : PV \to \mathcal{P}(X)$, called valuation.

Although formulae are evaluated at neighborhood situations the valuation σ is defined with respect only to points and this fact gives reason to the *persistence axiom* introduced later on. For a given subset model $\mathcal{S}$ we now define the *satisfaction relation* $\models_s$ between neighborhood situations of the underlying model and formulae in $\mathcal{L}$.

Definition 2.3 (Satisfaction relation) *Let $\mathcal{S} = (X, \mathcal{O}, \sigma)$ be a subset model. For $p \in U \in \mathcal{O}$ and $\varphi \in \mathcal{L}$ the satisfaction relation $\models_s$ is defined by recursion on φ:*

$(p,U) \models_s A$ *iff* $p \in \sigma(A)$
$(p,U) \models_s \neg\varphi$ *iff* $(p,U) \not\models_s \varphi$

[1] Since we focus on the decidability of *LSS* we stress that *PV* is a decidable set and therefore the set of valid *LSS*-formulae is decidable as well.

$(p,U) \models_s \varphi \wedge \psi$ *iff* $(p,U) \models_s \varphi$ *and* $p,U \models_s \psi$
$(p,U) \models_s K\varphi$ *iff* $(q,U) \models_s \varphi$ *for all* $q \in U$
$(p,U) \models_s \Box\varphi$ *iff* $(p,V) \models_s \varphi$ *for all* $V \in \mathcal{O}$ *such that* $p \in V \subseteq U$

In other words, we are considering a Kripke structure whose worlds are the pairs (p,U) and with two accessibility relations corresponding to shrinking an open ($\Box$) while maintaining a reference point, or to moving a reference point inside the given open (K).

The following schemata of formulae are suitable for the axiomatization of the set of *LSS*-validities:

(1) All $\mathcal{L}$-instances of propositional tautologies
(2) $K(\varphi \to \psi) \to (K\varphi \to K\psi)$
(3) $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$
(4) $\Box\varphi \to \varphi$ (*reflexivity*) } *S*4-*axioms*
(5) $\Box\varphi \to \Box\Box\varphi$ (*transitivity*) } *S*4-*axioms*
(6) $K\varphi \to \varphi$ (*reflexivity*) } *S*5-*axioms*
(7) $K\varphi \to KK\varphi$ (*transitivity*) } *S*5-*axioms*
(8) $L\varphi \to KL\varphi$ (*Euclidean property*) } *S*5-*axioms*
(9) $(A \to \Box A) \wedge (\neg A \to \Box\neg A)$ for atomic A (*persistence axiom*)
(10) $K\Box\varphi \to \Box K\varphi$ (*cross axiom*)

The deductive system *LSS* is established by adding rules of inference:

$$\frac{\varphi \to \psi, \varphi}{\psi}\,(\text{modus ponens}) \qquad \frac{\varphi}{K\varphi}\,(K\text{-necessitation}) \qquad \frac{\varphi}{\Box\varphi}\,(\Box\text{-necessitation})$$

The cross axiom is the typical one for the logic of subset spaces. In the notion of knowledge, this so-called *cross property* demands that the agent under consideration does not forget. The axioms are often used in their dual form, e.g. the cross axiom $K\Box\varphi \to \Box K\varphi \equiv \Diamond L\varphi \to L\Diamond\varphi$. Note that because of the persistence axiom, *LSS* is not closed under substitution.

We conclude the introduction of *LSS* by stating its soundness and completeness w.r.t. subset frames and turn to the proof of decidability in the next section.

Theorem 2.4 (Soundness and Completeness of *LSS*) *Let* $\varphi \in \mathcal{L}$ *be a formula. Then* φ *is LSS-derivable iff it is valid in every model based on a subset frame.*

The soundness part of theorem 2.4 is obvious, completeness is much harder to show. See [DMP96] for a proof using a rather complicated step-by-step construction.

3 Decidability of *LSS*

It is folklore that the above given axiomatization together with the f.m.p. would immediately yield the decidability of the satisfaction problem. But we have bad news: the logic of subset spaces lacks the f.m.p. To see this, consider the following scheme of infinity axioms:

$$\psi := \Box(\Diamond\varphi \wedge \Diamond\neg\varphi).$$

No sentence of this form can have a finite subset model. For suppose that $\mathcal{S}$ were a finite model containing p and U such that $p,U \vDash \psi$. We may assume that U is a $\subseteq$-minimal open about p with this property. But the minimality implies $(p,U) \vDash \varphi \wedge \neg\varphi$, and this is absurd.

However, substitute φ by $L(B \wedge \Box(KB \vee LC))$ and see [DMP96] for an infinite subset model of the resulting formula ψ'.

The problem that ψ' has only infinite subset models, stems from the fact that in subset spaces the accessibility relation corresponding to the $\Box$-operator is antisymmetric, as according to the $\subseteq$-relation, although this property is not demanded by the axioms. But there is a class of models avoiding this difficulty, the class of *cross axiom models*, to which *LSS* is also sound and complete and satisfies the f.m.p. as shown below.

Definition 3.1 (Cross Axiom Model) *A* cross axiom frame *is a tuple*

$$(J, \xrightarrow{L}, \xrightarrow{\Diamond})$$

such that J is a non-empty set, $\xrightarrow{L}$ is an equivalence relation on J, $\xrightarrow{\Diamond}$ is a preorder on J, and the following property holds: If $i \xrightarrow{\Diamond} j \xrightarrow{L} k$, then there is some l such that $i \xrightarrow{L} l \xrightarrow{\Diamond} k$.
A cross axiom model *is a cross axiom frame together with an interpretation σ of the atomic propositions of $\mathcal{L}$;. σ must satisfy the condition that*

$$\text{if } i \xrightarrow{\Diamond} j, \text{ then } i \in \sigma(A) \text{ iff } j \in \sigma(A).$$

A cross axiom model is simply a bimodal Kripke structure such that the accessibility relations satisfy the corresponding axioms and also the cross axiom holds. In addition, the valuation σ satisfies the persistence axiom. Hence, *LSS* is obviously sound w.r.t. the class of cross axiom *models* (not *frames*, because of the persistence axiom). Moreover, every subset model $\mathcal{S}=(X,\mathcal{O},\sigma)$ gives rise to a cross axiom model

$$\mathcal{M}=(J, \xrightarrow{L}, \xrightarrow{\Diamond}, \sigma')$$

as follows:

$J := \{<p,U> \in X \times \mathcal{O} \mid p \in U\}$ (<p,U> denotes a single point of J)

$<p,U> \xrightarrow{L} <q,V>$ iff $U = V$

$<p,U> \xrightarrow{\Diamond} <q,V>$ iff $p = q$ and $V \subseteq U$

$\sigma'(A) := \{<p,U> \mid p \in \sigma(A) \cap U\}$ for all atomic propositions A

A simple induction shows that

$$(p,U) \vDash_S \varphi \quad \Leftrightarrow \quad <p,U> \vDash_{\mathcal{M}} \varphi$$

for all formulae $\varphi \in \mathcal{L}$, thus we can state:

Lemma 3.2 (Completeness w.r.t. Cross Axiom Models) *LSS is sound and complete w.r.t. the class of cross axiom models.*

Hence, for any formula $\varphi \in \mathcal{L}$ it suffices to look for a finite cross axiom model. Since the *canonical model* $\mathcal{C}$ of *LSS* satisfies every *LSS*-consistent φ and the *filtration method* constructs a finite quotient structure $\tilde{\mathcal{C}}$ out of $\mathcal{C}$, also a model of φ, the remaining problem is to choose a filtration method such that $\tilde{\mathcal{C}}$ is a cross axiom model.

The canonical model

$$\mathcal{C} =: (\mathcal{T}, \xrightarrow{L}, \xrightarrow{\Diamond}, \sigma)$$

is formed in the usual way, i.e. $\mathcal{T}$ consists of all maximal *LSS*-consistent sets of formulae and the accessibility relations induced by the modal operators K and $\Box$ are for all $S, T \in \mathcal{T}$ defined by

$$S \xrightarrow{L} T :\Leftrightarrow \{\psi \in \mathcal{L} \mid K\psi \in S\} \subseteq T \quad \text{and}$$
$$S \xrightarrow{\Diamond} T :\Leftrightarrow \{\psi \in \mathcal{L} \mid \Box\psi \in S\} \subseteq T .$$

Finally, the valuation σ is defined by

$$A \in \sigma(T) :\Leftrightarrow A \in T \text{ for all } A \in PV \text{ and } T \in \mathcal{T}.$$

Let us now fix a formula $\varphi \in \mathcal{L}$ and gather together the notions needed to define $\tilde{\mathcal{C}}$: Corresponding to φ, we build the filter set $\Sigma \subseteq \mathcal{L}$ in several steps. Starting with the set $sf(\varphi)$ of subformulae of φ we define:

$\Sigma^{\neg} := sf(\varphi) \cup \{\neg\psi \in \mathcal{L} \mid \psi \in sf(\varphi)\}$

$\Sigma' := \Sigma^{\neg} \cup \{\psi \mid \psi$ is a conjunction or disjunction of pairwise distinct $x \in \Sigma^{\neg}\}$

$\Sigma'' := \Sigma' \cup \{\psi \mid \psi$ is a conjunction or disjunction of pairwise distinct $x \in \Sigma'\}$

$\Sigma := \Sigma'' \cup \{L\psi \mid \psi \in \Sigma''\} \cup \{K\psi \mid \psi \in \Sigma''\}$.

Note that all sets are finite and subformula closed and that Σ'' is a kind of Boolean closure of $sf(\varphi)$.

For $T \in \mathcal{T}$ let $[T]$ denote the set $\{S \in \mathcal{T} \mid T \cap \Sigma = S \cap \Sigma\}$ which is the *equivalence class* of T w.r.t. Σ. The minimal filtration $\xrightarrow{[L]}$ of $\xrightarrow{L}$ is defined by

$[S] \xrightarrow{[L]} [T] \;:\Leftrightarrow\; \exists S' \in [S], T' \in [T] :\; S \xrightarrow{L} T$ for all $S, T \in \mathcal{T}$

and the minimal filtration of $\xrightarrow{\Diamond}$ is defined accordingly. With these notions available, we can define

$$\tilde{\mathcal{C}} := ([\mathcal{T}], \xrightarrow{[L]}, \xrightarrow{[\Diamond]}, \tilde{\sigma}),$$

where $[\mathcal{T}] := \{[T] \mid T \in \mathcal{T}\}$, $\xrightarrow{[L]}$ is the minimal filtration of $\xrightarrow{L}$, $\xrightarrow{[\Diamond]}$ is the transitive closure of the minimal filtration of $\xrightarrow{\Diamond}$, and for all atomic A we have

$$\tilde{\sigma}(A) := \{ [T] \in [\mathcal{T}] \mid A \in T \cap \Sigma \}.$$

The following proposition guaranties that $\tilde{\mathcal{C}}$ is indeed a model of φ.

Lemma 3.3 *Let* $\mathcal{C} = (\mathcal{T}, \xrightarrow{L}, \xrightarrow{\Diamond}, \sigma)$ *be the canonical model of LSS and* $\tilde{\mathcal{C}} = ([\mathcal{T}], \xrightarrow{[L]}, \xrightarrow{[\Diamond]}, \tilde{\sigma})$ *the quotient structure described above. Then* $\tilde{\mathcal{C}}$ *is a filtration of* $\mathcal{C}$.

Proof. We have to show that $\xrightarrow{[L]}$ and $\xrightarrow{[\Diamond]}$ are filtrations of $\xrightarrow{L}$ and $\xrightarrow{\Diamond}$, respectively. For $\xrightarrow{[L]}$, the common minimal filtration, this is clear. Concerning $\xrightarrow{[\Diamond]}$ this is an easy consequence of the definition of the minimal filtration and its transitive closure; for the complete proof we refer to [CZ97], p. 141 ff.

To prepare the proof that $\tilde{\mathcal{C}}$ is a cross axiom model, we first state a very useful property of the filter set Σ.

Lemma 3.4 *Let* $\gamma_T := \bigwedge_{\gamma \in T \cap \Sigma} \gamma$ *be the identifying formula for the above defined equivalence class* $[T]$ *of* T w.r.t. Σ. *Then*

$$L\gamma_T \in S' \text{ for some } S' \in [S] \;\Rightarrow\; L\gamma_T \in S'' \text{ for all } S'' \in [S].$$

Proof. Let $L\gamma_T \in S' \in [S]$. By definition of $\xrightarrow{L}$, there is some $T' \in \mathcal{T}$ such that $S' \xrightarrow{L} T'$ and $\gamma_T \in T'$. By the same argument, we get $\{L\psi \mid \psi \in T' \cap \Sigma - \Sigma''\} \subseteq S'$. We now make use of the $S5$ property that every formula $O\psi \in \mathcal{L}$ with a prefix $O \in \{K, L\}^2$ is equivalent to some $O'\psi$, where $O' \in \{K, L\}^1$. So with the definition of $\Sigma - \Sigma''$ we also get $\{L\psi \mid \psi \in T' \cap \Sigma - \Sigma''\} \subseteq S''$ for arbitrary $S'' \in [S]$.
Since all formulae in $\Sigma - \Sigma''$ are K or L sentences we obtain with the $S5$ laws

$$K\gamma'_T := K\left(\bigwedge_{\gamma \in T' \cap \Sigma - \Sigma''} \gamma\right) \in S''. \qquad *$$

Clearly $L\gamma_T \in S' \Rightarrow L\gamma''_T \in S'$, where $\gamma''_T := \bigwedge_{\gamma \in T \cap \Sigma''} \gamma$, and since $L\gamma''_T \in \Sigma$ by construction of Σ, we also have

$$L\gamma''_T \in S''. \qquad **$$

Putting * and ** together we get $\{K\gamma'_T, L\gamma''_T\} \subseteq S''$. Hence there exists some $U \in \mathcal{T}$ such that $S'' \xrightarrow{L} U$ and $\{\gamma'_T, \gamma''_T\} \subseteq U$. Since $\gamma'_T \wedge \gamma''_T \equiv \gamma_T \in U$ we finally get $L\gamma_T \in S''$. □

Now let us turn to the question of how to show that $\tilde{\mathcal{C}}$ has the cross property. Whenever $[S] \xrightarrow{[\Diamond]} [T] \xrightarrow{[L]} [U]$, we need some $[V] \in [\mathcal{T}]$ such that $[S] \xrightarrow{[L]} [V] \xrightarrow{[\Diamond]} [U]$.

By definition of $\xrightarrow{[\Diamond]}$ and $\xrightarrow{[L]}$, there are $S' \in [S]$, T' and $T'' \in [T]$, $U' \in [U]$ such that $S' \xrightarrow{\Diamond} T'$ and $T'' \xrightarrow{L} U'$. If $T' = T''$, the cross property of $\mathcal{C}$ supplies the desired $V' \in [V]$ such that $S \xrightarrow{L} V' \xrightarrow{\Diamond} U$ and hence, $[S] \xrightarrow{[L]} [V] \xrightarrow{[\Diamond]} [U]$. But possibly $T' \neq T''$. However, Lemma 3.4 gives a solution: According to T' we can always find some $U'' \in [U]$ such that $T' \xrightarrow{L} U''$, i.e. we have $S' \xrightarrow{\Diamond} T' \xrightarrow{L} U''$ and therefore, the set V' we are looking for must exist. This is the announced nice property of $\tilde{\mathcal{C}}$.

Lemma 3.5 *Let* $\tilde{\mathcal{C}} := ([\mathcal{T}], \xrightarrow{[L]}, \xrightarrow{[\Diamond]}, \tilde{\sigma})$ *be defined as above. Then for all* $[S], [T] \in [\mathcal{T}]$ *the following holds:*

$$[S] \xrightarrow{[L]} [T] \quad \Rightarrow \quad \text{for all } S' \in [S] \text{ there exists some } T' \in [T] \text{ such that } S' \xrightarrow{L} T'.$$

Proof. This is a direct consequence of Lemma 3.3, observing that $[S] \xrightarrow{[L]} [T]$ implies the existence of some $S' \in [S]$ such that $L\gamma_T \in S'$.

We are now well prepared to show that $\tilde{\mathcal{C}}$ is a cross axiom model and in this way prove that *LSS* has the f.m.p. with respect to cross axiom models.

Theorem 3.6 (Finite Model Property) *Every satisfiable* $\varphi \in \mathcal{L}$ *has a finite cross axiom model.*

Proof. Let φ be satisfiable and $\tilde{\mathcal{C}}$ the model described above. Obviously $\xrightarrow{[\Diamond]}$ is reflexive and transitive and $\tilde{\sigma}$ satisfies the required property. Reflexivity and symmetry of $\xrightarrow{[L]}$ are direct consequences of the fact that this is the minimal filtration of $\xrightarrow{L}$. Finally, we get transitivity of $\xrightarrow{[L]}$ and the cross property by iterated application of Lemma 3.4. Hence $\tilde{\mathcal{C}}$ is a finite cross axiom model of φ.

Since satisfiability and validity are dual problems, this leads to the following:

Corollary 3.7 (Decidability) *The set of satisfiable* $\varphi \in \mathcal{L}$ *and the set of LSS-theorems are both decidable.*

4 Complexity

There are few variants of the basic *LSS* with reasonable computational behavior, cf. [Hei98c], where *linear subset spaces* and *binary computation structures* are investigated, both of them NP-complete.

But not surprisingly, the basic *LSS* turned out to be at least *PSPACE-hard*. Using techniques introduced by Ladner [Lad77] and very comprehensively applied in [HM92], we proved this result by reducing the problem of deciding whether an element of the *quantified Boolean formulae* (*QBF*) is true to that of deciding whether a formula is *LSS* satisfiable[2]. The first problem is known to be PSPACE-hard.

However, this result gives us only *one* lower bound and says nothing about the exact complexity of *LSS* since up to now we could not establish PSPACE as an upper bound. Obviously, *LSS* lacks the tree model property, cf. [Grä99a] definition 3.5, so the commonly used tableau method to get a PSPACE decision algorithm may not work.

Possibly, the complexity of the decision problem is worse, thus we tried to prove EXTIME as a lower bound. One may take it as a good sign that we did not get along with it. The methods we know to show EXTIME-hardness are based on tiling problems, cf. [vEm96], or on formulae that force any satisfying model to have a path of exponential length, cf. [HM92]. In any case, these methods require the encoding of an n-bit binary counter, but *LSS* seems to be too weak for this purpose.

There is still another possibility to show EXTIME-hardness: The criterion Edith Hemaspaandra provides in [Spa93]. This criterion looks for a formula capable to simulate full irreflexive, asymmetric, intransitive binary trees of arbitrary but finite height. To this end we would have to translate a formula φ out of some logical language having such a tree as model into a satisfiable formula $\psi \in \mathcal{L}$, simulating this tree. The problem is that in *LSS* both accessibility relations are transitive. To overcome this we could add special atomic propositions $H_0, H_1, ..., H_n$, each one true at exactly one level of the tree, to encode the height of a node. But since we do not know a bound for the height of the tree model for φ we do not know how to choose n. Hence this does not seem to work either.

So until now the facts we can state about the complexity of the decision problem for *LSS* are: PSPACE as a lower bound and NEXTIME as an upper bound, supplied by the original proof of decidability. The filter set we used in our proof leads to N2EXTIME as an upper bound, that is the price we have to pay for the simplification of the proof.

[2] This proof is also part of our diploma thesis.

5 Conclusions and Future Work

We introduced the bimodal logic of subset spaces invented by Moss and Parikh and explained its interpretation as a topological logic of knowledge. For decidability a simplified proof was presented and finally we discussed the difficulties in determining the complexity of the decision problem. To overcome these difficulties and find out what the complexity of the decision problem really is seems to us a task worthy of further research. We didn't yet give up the hope that *LSS* may be PSPACE-complete – and if so, this would be a reason for a party![3]
Furthermore, we think it is a good idea to follow the referee's suggestion and try to find other applications for *LSS* or related systems. This seems indeed another interesting and important task and we appreciate this hint very much.

Acknowledgement

We would like to thank Bernhard Heinemann and the anonymous referees for several valuable suggestions and helpful comments.

References

[BdRV01] P. Blackburn, M. de Rijke, Y. Venema. 2001. *Modal Logic*, volume 53 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 2001.

[CZ97] A. Chagrov, M. Zakharyaschev. 1997. *Modal Logic,* volume 35 of Oxford Logic Guides. Clarendon Press, Oxford, 1997.

[DG00] J.M. Davoren, R.P. Goré. 2000. *Bimodal logics for reasoning about continuous dynamics.* In Proceedings of Advances in Modal Logic. Leipzig, to appear.

[DMP96] A. Dabrowski, L.S. Moss, R. Parikh. 1996. *Topological reasoning and the logic of knowledge*. Ann. Pure Appl. Logic, 78 (1996) 73-110.

[FHMV96] R. Fagin, J.Y. Halpern, Y. Moses, M.Y. Vardi. 1995. *Reasoning about Knowledge*. MIT Press, Cambridge, MA.

[Geo97] K. Georgatos. 1997. *Knowledge on Treelike Spaces.* Studia Logica, 59:271-301.

[Grä99a] E. Grädel. 1999. *On the restraining power of guards*. Journal of Symbolic Logic, 64(4): 1719–1742, 1999.

[3] This remark about logics with a decision problem solvable with a PSPACE-algorithm is said to stem from Vardi, cf. [Grä99b].

[Gra99b] E. Graedel. 1999. *Why are modal logics so robustly decidable?*. Bulletin of the European Association for Theoretical Computer Science, vol 68, pp 90-103.

[Hei98a] B. Heinemann. 1998. *Topological Modal Logics Satisfying Finite Chain Conditions*. Notre Dame Journal of Formal Logic, 39(3):406-421.

[Hei98b] B. Heinemann. 1998. *Topological nexttime logic*. In M. Kracht et al., editor, Advances in Modal Logic, Volume 1, pages 99-113. CSLI.

[Hei98c] B. Heinemann. 1998. *Subset pace logics of knowledge and time.* Habilitation Thesis, Fachbereich Informatik, FernUniversität Hagen.

[Spa93] E. Spaan. 1993. *Complexity of modal logics*. PhD thesis, University of Amsterdam, Institute for Logic, Language and Computation.

[HM92] J.Y. Halpern, Y. Moses. 1992. *A Guide to Completeness and Complexity for Modal Logics of Knowledge and Belief.* Artificial Intelligence 54:319-379.

[Lad77] R.E. Ladner. 1977. *The Computational Complexity of Provability in Systems of Modal Propositional Logic*. SIAM Journal of Computing 6:467-480.

[McK41] J.C.C. McKinsey. 1941. *A solution to the decision problem for the Lewis systems S*2 *and S*4*, with an application to topology.* J. Symbolic Logic 6 (3) 117-141.

[MP92] L.S. Moss, R. Parikh. 1992. *Topological reasoning and the logic of knowledge.* Theoretical Aspects of Reasoning about Knowledge, TARK 1992, ed. Y. Moses, 95-105. Morgan Kaufmann.

[vEm96] P. van Emde Boas. 1996. *The convenience of tilings*. Technical Report CT-96-01, Institute for Logic, Language and Computation, University of Amsterdam

[WP02] A. Weiss, R. Parikh. 2002. *Completeness of Certain Bimodal Logics for Subset Spaces*. Studia Logica, 71:1-30.

An application of Sahlqvist Theory to Bisorted Modal Logic

Wouter Kuijper
ILLC/University of Amsterdam, the Netherlands.
wkuijper@science.uva.nl

Jorge Petrúcio Viana
Institute of Mathematics/Federal Fluminense University and COPPE/Federal University of Rio de Janeiro, Brazil. Visiting ILLC/University of Amsterdam, the Netherlands. Supported by CAPES, Proc. n. BEX0493/02-3.
petrucio@cos.ufrj.br

Abstract. We axiomatize two bisorted modal logics: basic hybrid logic and subset modal logic. The systems are presented as normal multimodal logics allowing us to freely apply Sahlqvist Theory. The proposed axiomatizations are clearer and more concise than existing axiomatizations of the same theories. In particular, the axioms have better motivations because each corresponds in a very direct sense to a frame condition. Proving completeness becomes a straightforward exercise by appealing to canonicity directly avoiding the usual canonical model construction. Moreover the resulting systems are automatically complete to all Sahlqvist extensions, even those containing nominals.

1 Introduction

It is a well established fact that modal logic over *models* can be seen as a fragment of first order logic whereas over *frames* it is a fragment of monadic second order logic . The study of the connections in the expressivity hierarchy of these three languages — or language *families* — is what is commonly called Correspondence Theory [10]. Comparing different systems with regard to expressivity, in some sense, presupposes a common, adequate semantics. Although for a lot of modal logics that interest us relational structures do the job nicely, not every logic is sound and complete with respect to some frame class [5, 9]. A huge amount of research has gone into uncovering which logics *are*. In all generality we could call this field Completeness Theory [3].

At the intersection of both these well explored paths we find a nice set of results generally referred to as Sahlqvist Theory. In essence Sahlqvist Theory

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

seeks to identify large classes of modal formulas that are both *canonical* and that *correspond* to an elementary frame condition which can be effectively computed from the formula. The landmark result in this research is the theorem below. (All terminological and conceptual pre–requisites can be found in [3].)

Definition 1.1 (Sahlqvist Formulas). *Positive* and *negative* formulas are defined as usual. A *Sahlqvist antecedent* is a formula built up from formulas that are either negative or of the form $\Box^n p$, using only $\wedge$, $\vee$ and $\Diamond$. A *Sahlqvist implication* is any implication that has a Sahlqvist antecedent and a positive consequent. A *Sahlqvist formula* is built up from Sahlqvist implications using $\Box$ and $\wedge$ freely, and $\vee$ only to disjuncts that do not share proposition symbols.

Theorem 1.1 (Sahlqvist 1974). For every Sahlqvist formula A it holds

(i) A is canonical,

(ii) A has a local first order frame correspondent A^{FO} that can be effectively computed from A.

It is not decidable in general whether an arbitrary formula has a first order correspondent [4]. The Sahlqvist fragment can be seen as a good compromise between simplicity and generality in picking out a large portion of formulas for which this is decidable. Of course this leaves room for generalization of the result [6, 8]. Here we will use a version that in addition to unary modalities allows nullary modalities (or modal constants) to occur in Sahlqvist formulas [3].

Sorting was proposed by P. Blackburn [2] and V. Goranko [7] as a general strategy to improve the expressive power of modal languages. A basic example of a bisorted logic is basic hybrid logic (BHL), obtained from basic modal logic by introducing *nominals*, a second type of proposition symbols that serve as unique names for states, also added are satisfaction operators to jump to these named states and continue evaluation there [1]. As a first step in establishing a general theory of sorted modal logics, Goranko started to investigate Subset Modal Logic (SML), inspired on BHL, SML is obtained by relieving the restriction of nominals to be interpreted strictly as singletons [7].

In relation to the traditional logical issues, such as expressiveness, axiomatization, decidability, and complexity, a lot of results to BHL have already been established (cf. [2] and the bibliography therein).

In this paper we study versions of these systems. We consider the problem of the transfer of results from the basic (unisorted) modal setting to this more general context. In particular, we investigate Sahlqvist theory and its use to obtain simpler sets of axioms and simpler completeness proofs to our versions of BHL and SML.

This paper contains three main contributions in the line of research sketched above. Firstly, we provide new axiomatizations to two bisorted modal logics: basic hybrid logic and subset modal logic. Secondly, we exemplify how to treat sorted modal logics as normal multimodal logics using infinite axiomatizations. Lastly we show that it is possible, as a consequence of this approach, to easily apply Sahlqvist Theory to sorted modal logics.

The paper is structured as follows. In Section 2 we review the syntax and semantics of the basic hybrid logic in its common form. In Section 3 we present the basic subset modal logic as a normal multimodal logic with nullary modalities. In Section 4 we axiomatize the minimal subset modal logic. Next, we proceed to show, in Section 5, how by extending the minimal subset modal logic with two very simple axioms we obtain the minimal basic hybrid logic. We conclude with some more comments, discussion and suggestions for further research.

2 Basic Hybrid Logic

In this section we review syntax and semantics of basic hybrid logic [1]. BHL is usually presented as a two sorted modal logic, viewing nominals as proposition symbols whose interpretation on models is restricted. In Section 3, we present BHL as a multimodal logic, viewing nominals as special kinds of modal constants, interpreted on frames. As will be shown, this change in approach allows us transfer existing results for orthodox modal logics, in particular the multimodal version of Theorem 1.1 to BHL.

The basic hybrid language has countably many distinct nominals i and associated satisfaction operators $@_i$ to identify, respectively refer to, potentially countably many named states. The idea is that by using these nominals we can identify a *unique* state in our model. Basic modal logic, as it is bisimulation invariant, cannot express this [3]. Hence we have improved the expressive power of BML getting it closer to first–order logic, hopefully without losing its modal flavor [2].

Definition 2.1 (Basic Hybrid Language). The *basic hybrid language* is produced by

$$\phi := \bot \mid p \mid i \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \Diamond\phi \mid @_i\phi, \quad \text{where } i \in \Omega.$$

Basic hybrid logic is interpreted with respect to exactly the same frame class as the basic modal logic. This means that at the level of frames there is no interpretation for nominals nor for their associated satisfaction operators. The following definitions make clear exactly how this information arises at the level of models.

Definition 2.2 (Hybrid Models). A *hybrid model* is a pair $(\mathfrak{F}, V)$ where $\mathfrak{F} = (W, R)$ is a basic modal frame and V is a *hybrid valuation*, i.e., a function

[K$_@$]	$@_i(p \to q) \to (@_i p \to @_i q)$
[self-dual]	$@_i p \leftrightarrow \neg @_i \neg p$
[introduction]	$i \wedge p \to @_i p$
[ref]	$@_i i$
[sym]	$@_i j \leftrightarrow @_j i$
[nom]	$@_i j \wedge @_j p \to @_i p$
[agree]	$@_j @_i p \leftrightarrow @_i p$
[back]	$\Diamond @_i p \to @_i p$

Table 1: The axioms to the minimal Hybrid Logic given in [3].

that maps proposition symbols to subsets of W and nominals to singletons of W.

Definition 2.3 (Hybrid Satisfaction). A hybrid formula ϕ *being satisfied in the hybrid model* $\mathfrak{M}$ *at state* w, notation:

$$\mathfrak{M}, w \Vdash \phi,$$

is inductively defined as follows

$$\begin{array}{rll} \mathfrak{M}, w \Vdash p & \text{iff} & w \in V(p), \\ \mathfrak{M}, w \Vdash i & \text{iff} & \{w\} = V(i), \\ \mathfrak{M}, w \Vdash \Diamond\phi & \text{iff} & \text{there exists } v \in W \text{ such that } Rwv \text{ and } \mathcal{M}, v \Vdash \phi, \\ \mathfrak{M}, w \Vdash @_i\phi & \text{iff} & \mathfrak{M}, v \Vdash \phi \text{ where } \{v\} = V(i). \end{array}$$

with the boolean cases in the obvious way. Validity in a frame is defined as usual.

For some examples of hybrid validities cf. Table 1.

Note that here nominals have same status as proposition symbols. This is in fact inconvenient when we want to define the class of *hybrid* frames, i.e. frames with interpretation for nominals and their associated satisfaction operators.

3 Subset Modal Logic

Subset modal logic was introduced by V. Goranko [7] — in connection with what he calls "vanilla set sorts" — as a step in extending the basic hybrid mechanisms to sorted modal languages. In [7], Goranko advocates the

study of sorted modal formalisms and provides examples from the literature, showing that this kind of strategy can improve the expressive power of modal logics. One of the basic ideas of Goranko's approach, also present in [2], is that sorted modal logics can be seen as a more general type of hybrid logics. Goranko starts the extension of hybrid ideas to sorted modal logics, presenting SML and proposing an axiomatization to it. As a problem he left to prove his axioms are complete.

For some intuition about SML observe that in the basic hybrid case a frame can be seen as a general frame in which the family of admissible subsets, for nominals, is exactly the set generated from all singletons. The hybrid language has the power to denote these unique states. In SML, in the basic case, this restriction to singletons is dropped. Nominals —in SML we call them *subset constants*— can be interpreted on any subset again, just like ordinary proposition symbols. This in turn establishes the duality of our @ operators now written as $@^{\exists}$ and $@^{\forall}$. In terms of satisfaction this will work out in the following way

$$\mathfrak{M}, w \Vdash @_i^{\exists}\phi \quad \text{iff} \quad \text{there exists } v \in W \text{ such that } \mathfrak{M}, v \Vdash i \wedge \phi, \tag{3.1}$$

and dually,

$$\mathfrak{M}, w \Vdash @_i^{\forall}\phi \quad \text{iff} \quad \text{for all } v \in W \text{ it holds } \mathfrak{M}, v \Vdash i \rightarrow \phi. \tag{3.2}$$

Goranko specifies the SML syntax and semantics in an informal way. We choose to formulate SML as a "regular" normal modal logic [3], in this way, trying to avoid the difficulties we pointed out in the last paragraph of Section 2.

Definition 3.1 (SML Similarity Type). The similarity type for SML expands the similarity type for basic modal logic with a countable set Ω of nullary modalities called *subset constants* and an associated countable set $\{@_i^{\forall}\}_{i\in\Omega}$ of unary modalities called *subset accessibility operators*.

In the frame correspondence language the unary predicate associated to the subset constant i is denoted S_i and the binary predicate associated to the the subset accessibility operator $@_i^{\exists}$ is denoted R_i.

The definitions of satisfaction and validity are the same as in modal logic. To ensure condition (3.1) and (3.2) we define the appropriate semantics for SML.

Definition 3.2 (SML Frame). An *SML frame*

$$\mathfrak{F} = \left(W, R, (R_i)_{i\in\Omega}, (S_i)_{i\in\Omega}\right)$$

is a frame for the SML similarity type that satisfies for each $i \in \Omega$ the following defining condition on the interplay between subsets and their associated

accessibility relations:

$$\forall xy(R_i xy \leftrightarrow S_i y). \tag{3.3}$$

A *hybrid* SML frame is an SML frame where all S_i are singleton.

Observation 3.1 (Hybrid SML Frames and Hybrid Models). When we identify the $@^{\forall}$ and $@^{\exists}$ operators in the SML language over the class of all hybrid SML frames, we end up with the same theory as the basic hybrid language over the class of all hybrid models.

4 Axiomatizing the Minimal SML

In this section we present an infinite axiomatization to minimal SML and prove completeness.

Recall that a *normal modal logic* is a set of modal formulas closed under *generalization*, *uniform substitution* and *modus ponens*; containing all propositional tautologies plus special axioms saying how each modality relates to its dual and how it distributes over implication.

We denote by **K** the minimal modal logic, i.e. the normal modal logic that is sound and complete with respect to the class of all frames.

Definition 4.1 (Logic KS). Let S be the countable collection of Sahlqvist axioms given in Table 2. Define **KS** as the normal modal logic for the SML similarity type with the additional axioms in S.

[trans]	$@_j^{\exists} @_i^{\exists} p \rightarrow @_i^{\exists} p$	$\forall xyz(R_j xy \wedge R_i yz \rightarrow R_i xz)$
[trans$_{\Diamond}$]	$\Diamond @_i^{\exists} p \rightarrow @_i^{\exists} p$	$\forall xyz(Rxy \wedge R_i yz \rightarrow R_i xz)$
[eucl]	$@_i^{\exists} p \rightarrow @_j^{\forall} @_i^{\exists} p$	$\forall xyz(R_i xy \wedge R_j xz \rightarrow R_i zy)$
[eucl$_{\Diamond}$]	$@_i^{\exists} p \rightarrow \Box @_i^{\exists} p$	$\forall xyz(R_i xy \wedge Rxz \rightarrow R_i zy)$
[label]	$@_i^{\forall} i$	$\forall xy(R_i xy \rightarrow S_i y)$
[ref]	$i \wedge p \rightarrow @_i^{\exists} p$	$\forall x(S_i x \rightarrow R_i xx)$

Table 2: Sahlqvist axioms to minimal SML with their First–Order correspondents; $i, j \in \Omega$.

The following lemma motivates our choice of axioms. It shows that we have defined a broader class of frames that has the same modal theory as the intended frame class of all SML frames.

First we recall the definition of a *rooted* or *point generated subframe* $\mathfrak{F}_w$ which is the restriction of $\mathfrak{F}$ to all nodes that are reachable from w by a

path consisting of R and R_i steps. Taking generated subframes is a frame construction that preserves truth of modal formulas.

Lemma 4.1. The following holds

(i) S is valid on SML frames.

(ii) If a rooted frame validates S then it is an SML frame.

Proof. Note that by the correspondence part of Sahlqvist's theorem to multimodal logic we can actually confuse our axioms with the first order properties they define.

For (i), it suffices to check that all first–order correspondents in Table 2 are consequences of (3.3). For [label] and [ref] this is immediate. For [trans], observe that by (3.3) R_iyz implies S_iz which warrants R_ixz. The other cases are similar.

For (ii), let $\mathfrak{F}_w$ be a point generated subframe that satisfies S. We show (3.3) holds on $\mathfrak{F}_w$. Take arbitrary $x, y \in \mathfrak{F}_w$. For the left to right direction, assume R_ixy then by [label] we have S_iy. For the right to left direction, assume S_iy then by [ref] we have R_iyy. Because y is in $\mathfrak{F}_w$ there is a path leading from w up to y. By induction on the length of this path, using [trans] and [trans$_\Diamond$], we get R_iwy. Since x is in $\mathfrak{F}_w$, again, there is a path leading from w up to x. By induction on the length of this path, using [eucl] and [eucl$_\Diamond$], we get R_ixy. □

Theorem 4.1. KS is the minimal subset modal logic.

Proof. Let S be the class of all SML frames. With S' we denote the class of frames defined by S. Note that by Sahlqvist's theorem we have automatic soundness and completeness with respect to S'

$$\mathbf{KS} = \mathrm{Th}(\mathsf{S}'),$$

hence it suffices to show that

$$\mathrm{Th}(\mathsf{S}') = \mathrm{Th}(\mathsf{S}).$$

By Lemma 4.1(i), $\mathsf{S} \subseteq \mathsf{S}'$ and we have $\mathrm{Th}(\mathsf{S}') \subseteq \mathrm{Th}(\mathsf{S})$. This corresponds to soundness of our axiom system.

For the other direction $\mathrm{Th}(\mathsf{S}) \subseteq \mathrm{Th}(\mathsf{S}')$, we will show that every non–validity of $\mathrm{Th}(\mathsf{S}')$ has a counterexample in S. This corresponds to completeness.

Let $\phi \notin \mathrm{Th}(\mathsf{S}')$ then there exists a counterexample $\mathfrak{F}, w \nVdash \phi$. Clearly by modal invariance under taking generated subframes $\mathfrak{F}_w \nVdash \phi$. Then by Lemma 4.1(ii), we have $\mathfrak{F}_w \in \mathsf{S}$ is the required counterexample. □

5 Back to Hybrid Logic

We adapt SML to obtain the minimal hybrid logic. To this end we add the two new axiom schemes in Table 3.

The known axiomatizations of the minimal hybrid logic require us to replace the *uniform substitution* rule in the normal modal logic Hilbert–style proof system by a new rule called *sorted substitution* that allows us to uniformly substitute proposition symbols by formulas and nominals by nominals [3]. Our logic is equipped with plain old uniform substitution, but this does not present a problem because nominals are now treated as nullary modalities and not as proposition symbols anymore. To simulate the power of sorted substitution we have axioms to all (permutations of) nominals.

The next proposition shows the strength of the two new axioms [ser] and [func] in relation to the axioms to minimal SML.

Proposition 5.1. [trans], [trans$_\Diamond$], [ser], [func] $\vdash$ [eucl], [eucl$_\Diamond$], [self-dual], [label].

Proof. We prove only [eucl] and [label].

To prove [eucl] assume R_ixy and R_jxz. By [ser] there is a w such that R_izw and S_iw. By [trans] we get R_ixw and this, together with R_ixy, gives, by [func], $y = w$. So, we obtain R_izy.

To prove [label] assume R_ixy. By [ser] there is a z such that R_ixz and S_iz. Hence $y = z$, by [func], and we get S_iy. □

Definition 5.1 (Logic KH). Let H be the countable collection of Sahlqvist axioms given by [trans], [trans$_\Diamond$], [ref], [ser], and [func]. Define **KH** as the normal modal logic for the SML similarity type with the additional axioms in H.

[ser]	$@_i^\exists i$	$\forall x \exists y (R_ixy \wedge S_iy)$
[func]	$@_i^\exists p \to @_i^\forall p$	$\forall xyz (R_ixy \wedge R_ixz \to y = z)$

Table 3: Additional Sahlqvist axioms for minimal Hybrid Logic; $i \in \Omega$.

Lemma 5.1. The following holds

(i) Every axiom in Table 3 is valid in a hybrid SML frame.

(ii) Every point generated subframe satisfying H is a hybrid SML frame.

Proof. The first part is left to the reader.

Let $\mathfrak{F}_w$ be a point generated subframe that satisfies H. By Proposition 5.1 $\mathfrak{F}_w$ is an SML frame.

We show that $\mathfrak{F}_w$ is hybrid, i.e. each S_i is singleton.

Take arbitrary $x, y \in \mathfrak{F}_w$. By [ser] we have $x', y' \in \mathfrak{F}_w$ such that $R_i x x'$ and $R_i y y'$. By [trans] and [trans$_\Diamond$] we have $R_i w x'$ and $R_i w y'$. Finally by [func] $x' = y'$. Hence R_i is a constant total function. By condition (3.3) it follows S_i is singleton. □

Theorem 5.1. KH is the minimal Hybrid Modal Logic.

Proof. Analogous to the proof of Theorem 4.1 now using Lemma 5.1 to carry over the counterexample. □

6 Perspectives

The hybrid logic literature tends to stress the analogies between hybrid logic and first-order logic, and to emphasize the first order aspects of hybrid logic. But, as we showed in this paper, hybrid logic is genuinely modal too. An important aspect of our approach, that brings out the modal aspect of hybrid logic very well, is that our results give automatic completeness for all Sahlqvist formulas. In fact, the following result is immediate.

Corollary 6.1. Let Σ be a set of Sahlqvist formulas in the SML similarity type, possibly containing nominals. Then, the logics obtained by extending, **KS** and **KH** to, respectively **KS**Σ and **KH**Σ are sound and complete for, respectively, the subclass of all SML frames and the subclass of all hybrid SML frames they define.

To see this, note that the proofs of Theorems 4.1 and 5.1 do not depend in any way on the properties we want to impose on the generated subframes. In fact both proofs are completely analogous. For *any* Sahlqvist formula that locally corresponds to a first order canonical property the proof is still valid.

There are some obvious directions for future research. Firstly, in relation to subset modal logic, we have only touched the obvious aspect of completeness. We can ask what can be said about other properties of SML, e.g. expressivity, filtrations, finite model property, decidability, and complexity.

Secondly, one of the main motivations of this work was to exemplify the alternative technique to prove completeness presented. We can ask what are the merits of the new technique and if it really leads to more general results.

For HL it is known that by adding stronger deductive capabilities to the proof system, allowing introduction of new names while reasoning, we can obtain a strong automatic completeness result for all *pure* extensions with respect to the class of all *named models*. Where a named model is simply a hybrid model where every state has at least one name (makes at least one nominal true) and a pure extension is any axiom that doesn't contain proposition symbols, only nominals.

In a way these extensions can be viewed as *hybrid extensions* since they must obviously use the added hybrid expressivity to define meaningful properties. In contrast, Sahlqvist axioms can be viewed as intrinsically *modal*.

As we noted in Corollary 6.1 our systems are trivially complete to all pure extensions. However they are complete with respect to the class of *all* hybrid models which contains among all the names ones also models with unnamed states. This reduces significantly the expressivity of pure axioms since, obviously, a pure axiom can only define meaningful properties on a countable subset of the universe: the named part of the model. Hence the question remains in what way we can obtain automatic correspondence and completeness of bisorted Sahlqvist formulas to the class of all named models.

Acknowledgments

We are much indebted to Patrick Blackburn, Balder ten Cate, Valentin Goranko, Maarten Marx and the two anonymous referees.

References

[1] P. Blackburn. Internalizing labelled deduction. *Journal of Logic and Computation*, 10:137–168, 2000.

[2] P. Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of the IGPL*, 8:339–365, 2000.

[3] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, Cambridge, 2002.

[4] L.A. Chagrova. An undecidable problem in correspondence theory. *Journal of Symbolic Logic*, 56:1261–1272, 1991.

[5] K. Fine. An incomplete logic containing S4. *Theoria*, 40:23–29, 1974.

[6] M. Gehrke, H. Nagahashi, and Y. Venema. A Sahlqvist theorem for distributive modal logic. ILLC Prepublications, PP-2002-9, 2002.

[7] V. Goranko. Sorting and hybrid logics. Unpublished manuscript, 2000.

[8] V. Goranko and D. Vakarelov. Sahlqvist formulas in hybrid polyadic modal logics. *Journal of Logic and Computation*, 11:737–754, 2001.

[9] S.K. Thomason. An incompleteness theorem in modal logic. *Theoria*, 40:150–158, 1974.

[10] J. van Benthem. Correspondence theory. In *Handbook of Philosophical Logic, volume 2*, pages 167–247. Kluwer, Dordrecht, 1974.

Contextual Grammars and Go Through Automata

Florin Manea
Faculty of Mathematics, Bucharest University
flmanea@funinf.cs.unibuc.ro

Abstract. In this paper we apply different variants of go through automata for the recognition of languages generated by internal contextual grammars (with or without choice) and shuffled contextual grammars. Go through automata are generalizations of the push-down automata in the area of context-sensitivity.

1 Introduction

Contextual grammars were introduced in 1969 by Solomon Marcus [3] as an attempt to transform in generative devices some procedures developed within the framework of analytical models (see [4] for a comprehensive discussion on the linguistic motivations of contextual grammars).

Contextual grammars are alternatives to Chomsky-like generative devices. Main aspects that individualize contextual grammars are the integrity of the derivation sequence and the global derivation, at the sentence level.

Classes of languages generated by contextual grammars are usually incomparable with the classical classes of languages from the Chomsky hierarchy. Usual automata, like finite or push-down automata do not apply for any class of contextual grammars. With a notable exception (see [5]), there is a lack of automata for the recognition of contextual languages.

In this paper, we propose the recognition of threes classes of contextual grammars (shuffled, internal without choice and internal with choice) by means of various *go through automata*. Go through automata were introduced in [1], as a generalization of push-down automata.

The following general definitions and notations will be used throughout the paper. Let V be finite alphabet. By V^+ we denote the set of non empty words over V, by λ the empty word, and by $V^* = V^+ \cup \{\lambda\}$. We denote by $|w|$ the length of the word w. For a set A, $\mathcal{P}_f(A)$ is the set of all finite subsets of A. The *shuffle* operation between sentences,denoted ш , is defined recursively by $av \text{ш} bw = a(v \text{ш} bw) \cup b(av \text{ш} w)$ and $w \text{ш} \lambda = w$, $\lambda \text{ш} w = w$

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

where $a, b \in V$ and $v, w \in V^*$ (where singleton sets are denoted by their unique element).

2 Basic Definitions

We will present the definitions of contextual grammars that will be used later in this paper. For further details, the reader is referred to [6].

An **internal contextual grammar without choice** is a construct $G = (V, A, C)$ where V is an alphabet, A is a finite language over V (i.e. $A \subset V^*$ and A is finite), and C is a finite subset of $V^* \times V^*$. The derivation style is defined as $x \Rightarrow_{in} y \iff x = x_1x_2x_3$ and $y = x_1ux_2vx_3$ where $(u, v) \in C$. By $\overset{*}{\Rightarrow}_{in}$ we denote the reflexive and transitive closure of $\Rightarrow_{in}$. Then $L_{in}(G) = \{x \in V^* |\ w \overset{*}{\Rightarrow}_{in} x, w \in A\}$ is the language generated by G.

An **internal contextual grammar with choice** is a construct $G = (V, A, C.\phi)$ where V is an alphabet, A is a finite language over V (i.e. $A \subset V^*$ and A is finite), C is a finite subset of $V^* \times V^*$, and $\phi : V^* \to \mathrm{P}(C)$. The derivation style is defined as $x \Rightarrow_{in} y \iff x = x_1x_2x_3$ and $y = x_1ux_2vx_3$ where $(u, v) \in \phi(x_2)$. By $\overset{*}{\Rightarrow}_{in}$ we denote the reflexive and transitive closure of $\Rightarrow_{in}$. Then $L_{in}(G) = \{x \in V^* |\ w \overset{*}{\Rightarrow}_{in} x, w \in A\}$ is the internal language generated by G. G is called an internal contextual grammar with **finite** choice if $\phi^{-1}(c)$ is a finite set, for any $c \in C$.

A **shuffled contextual grammar** is a construct $G = (V, A, C)$ where V is an alphabet, A is a finite language over V, and C is a finite subset of V^*. The language generated by G is the smallest subset L of V^* such that: $A \subseteq L$ and if $x \in L$ and $c \in C$ then $x \sqcup\!\sqcup c \subseteq L$.

We will present the definitions of go through automata that will be used later in this paper.

A **go through automaton** (gta) is a construct $M = (Q, V, W, q_0, Z, F, \delta)$ where: Q,V and W are non-empty sets (called the set of states, the input alphabet and the stack alphabet, respectively); q_0 is the initial state; Z is the initial symbol of the stack; $F \subseteq Q$ is the set of final states; $\delta : Q \times (V \cup \{\lambda\}) \times W \to \mathcal{P}_f(Q \times W^* \times \{\downarrow, \uparrow\})$ is the transition function, satisfying the condition:

$$[q_2, \eta, \downarrow] \in \delta(q_1, a, b) \iff a = \lambda,$$

$$[q_2, \eta, \uparrow] \in \delta(q_1, a, b) \iff a \neq \lambda.$$

$\uparrow$ and $\downarrow$ are denotations for the movement of the head in the stack.

Remark 2.1 *A gta with the constraint* $[q_2, \eta, x] \in \delta(q_1, a, b) \Rightarrow x = \uparrow$ *is a (real-time) push-down automaton (pda).*

An **instant configuration** of a gta M is a triple $(q, \alpha, \beta.\gamma)$ where $q \in Q$ is the current sate of M, $\alpha \in V^*$ is the input tape and $\beta, \gamma \in W^*$; $\beta\gamma$ is the

stack of the automaton (with the top of the stack to the left), and . indicates the position of the head in the stack. The behavior of a gta is described by the transitions between instant configurations. We extend the transition functions to instant configurations in the following way:

$$(q_1, a\alpha, \beta.b\gamma) \vdash (q_2, \alpha, .\beta\eta\gamma) \iff [q_2, \eta, \uparrow] \in \delta(q_1, a, b),$$
$$(q_1, a\alpha, \beta.b\gamma) \vdash (q_2, a\alpha, \beta\eta.\gamma) \iff [q_2, \eta, \downarrow] \in \delta(q_1, \lambda, b)$$

for any $q_1, q_2 \in Q, a \in V, \alpha \in V^*, b \in W, \beta, \gamma, \eta \in W^*$. An initial configuration of the automaton M is an instant configuration (q_0, α, Z). We denote by $\overset{*}{\vdash}$ the reflexive and transitive closure of $\vdash$. For the rest of this paper, we will consider the empty stack as the acceptance method. Consequently, we will take every time an empty set of final states, $F = \emptyset$. The language recognized by the go through automaton M (with the empty stack) is: $L_\lambda(M) = \{w \in V^* | (q_0, w, Z) \overset{*}{\vdash} (q, \lambda, .)\}$.

A go through automaton M is **non-expansive** iff: $[q_2, \lambda, x] \in \delta(q_1, a, b) \Rightarrow a \neq \lambda$. A go through automaton M is **weak** iff: $[q_2, \eta, \downarrow] \in \delta(q_1, a, b) \Rightarrow \eta = b$. A go through automaton M is **simple** iff: $[q_2, \eta, \downarrow] \in \delta(q_1, a, b) \Rightarrow q_2 = q_1 \ \& \ \eta = b$.

3 Implementations

Theorem 3.1 *For any shuffled contextual grammar G there exists a simple go through automaton M such that $L(M) = L(G)$.*

Proof: Let $G = (V, A, C)$ be a shuffled contextual grammar. Let $M = (Q, V, W, q_0, Z, \emptyset, \delta)$ be a simple gta defined by: $Q = \{q_0, q_1\}$, $W = \{Z\} \cup \{(c, k) \in (C \cup A) \times I\!N \mid k < |c|\}$ where by $[c : k]$ we highlight that we have found the first k symbols of c. The function δ is defined by the following transitions:

1. $(q, yx, \uparrow) \in \delta(q, a, x)$ iff $q \in Q$ is an arbitrary state, $c \in C$ is a context and $a \in V$ is its first symbol, $x \in W$ is an arbitrary stack symbol and $y = [c, 1]$ if $|c| > 1$, otherwise $y = \lambda$

2. $(q_1, y, \uparrow) \in \delta(q_0, a, Z)$ iff $c \in A$ is an axiom and a is its first symbol and $y = [c, 1]$ if $|c| > 1$, otherwise $y = \lambda$

3. $(q, x, \downarrow) \in \delta(q, \lambda, x)$ iff $q \in Q$ is an arbitrary state and $x \in W$ is an arbitrary stack symbol

4. $(q, y, \uparrow) \in \delta(q, a, [c, k])$ iff $q \in Q$ is an arbitrary state, $c \in C \cup A$ is a context or an axiom and a is the $(k+1)$th symbol of c and $y = [c, k+1]$ if $k < |c| - 1$, otherwise $y = \lambda$

The algorithm implemented by M is mainly the following: for every input symbol a we try to match it to a context in the stack(we can go through the stack to find that context) or we put into the stack a new context starting with a. It is necessary to have in the stack a word from A or we will not empty the stack (the state q_1 appears only when a word from A was put into the stack, replacing the initial symbol Z, and also guarantees that we will not put another word from A in the stack). After a context is completed (every symbol of the context was matched) it will be erased from the stack. We have $L(M) = L(G)$. □

Example 3.1 Let $G = (V, A, C)$ be a shuffle contextual grammar, with $V = \{a, b, c\}$, $A = \{abc\}$ and $C = \{abc\}$. Then $L(G) = \{w \in V^+ \mid |w|_a = |w|_b = |w|_c$ and for any x such that $w = xy, |x|_a \geq |x|_b \geq |x|_c\}$. Let $M = (\{q_0, q_1\}, \{a, b, c\}, \{Z, [abc : 0], [abc : 1], [abc : 2]\}, \emptyset, \delta)$ be a simple gta for $L(G)$, where δ is defined by:

$(q, [abc : 1]x, \uparrow) \in \delta(q, a, x)$ iff $q \in Q$, $x \in W$;
$(q, [abc : 1], \uparrow) \in \delta(q, a, Z)$ iff $q \in Q$;
$(q_1, [abc : 1]x, \uparrow) \in \delta(q_0, a, x)$ iff $x \in W$;
$(q_1, [abc : 1], \uparrow) \in \delta(q_0, a, Z)$
$(q, x, \downarrow) \in \delta(q, \lambda, x)$ iff $q \in Q$, $x \in W$;
$(q, [abc : 2], \uparrow) \in \delta(q, b, [abc : 1])$ iff $q \in Q$;
$(q, \lambda, \uparrow) \in \delta(q, c, [abc : 3])$ iff $q \in Q$.

Let $w = aaabbcbcc$. We will show how the automaton works on this example:
$(q_0, aaabbcbcc, .Z) \vdash (q_1, aabbcbcc, .[abc : 1]) \vdash$
$(q_1, abbcbcc, .[abc : 1][abc : 1]) \vdash (q_1, bbcbcc, .[abc : 1][abc : 1][abc : 1]) \vdash$
$(q_1, bcbcc, .[abc : 2][abc : 1][abc : 1]) \vdash (q_1, bcbcc, [abc : 2].[abc : 1][abc : 1]) \vdash$
$(q_1, bcbcc, [abc : 2][abc : 1].[abc : 1]) \vdash (q_1, cbcc, .[abc : 2][abc : 1][abc : 2]) \vdash$
$(q_1, bcc, .[abc : 1][abc : 2]) \vdash (q_1, cc, .[abc : 2][abc : 2]) \vdash$
$(q_1, c, .[abc : 2]) \vdash (q_1, \lambda, .\lambda)$

Note that it does not matter the order of the insertion of contexts in the initial axiom, so it is useless to try to keep a track of this order of insertion. In the next we will need such a technique, and for this we will use the order of insertion in the stack.

In the following we will work with grammars $G = (V, A, C, \phi)$, such that C contains only contexts with their sides not empty, and also $\lambda \notin A$. These restrictions are due to the fact that any insertion and deletion in the stack of the gta should be made without λ transitions.

Theorem 3.2 *For any internal contextual grammar without choice G, having any $(u, v) \in C \Rightarrow u \in V^+, v \in V^+$ and $\lambda \notin A$, there exists a weak go through automaton M such that $L(M) = L(G)$.*

Proof: Let $G = (V, A, C)$ an internal contextual grammar without choice. For this grammar we define a gta $M = (Q, V, W, q, Z, F, \delta)$ where: $Q = \{q\}$,

$W = \{Z\} \cup \{[u : k, v : l] | (u, v) \in C, k, l \in I\!N, k < |u| + 1, l < |v| + 1\} \cup$ $\{[w : k] | w \in A, k \in I\!N, k < |w| + 1\}$. Here by $[u : k, v : l]$ we highlight that we have found the first k symbols of u and the first l from v, where (u, v) is a context. For axioms the same interpretation of $[w : k]$ holds. The transition function δ is defined by the following relations:

1. $(q, [u : 1, v : 0]Z, \uparrow) \in \delta(q, a, Z)$ iff $(u, v) \in C$ and the first symbol of u is a
2. $(q, [w : 1], \uparrow) \in \delta(q, a, Z)$ iff $w \in A$ and the first symbol of w is a
3. $(q, [u : 1, v : 0]x, \uparrow) \in \delta(q, a, x)$ iff $(u, v) \in C$ and the first symbol of u is a and $x \in W$
4. $(q, [u : 0, v : 0], \uparrow) \in \delta(q, a, [w : |w| - 1])$ iff $(u, v) \in C$ and the last symbol of w is a;
5. $(q, [u^* : 0, v^* : 0], \uparrow) \in \delta(q, a, [u : |u|, v : |v| - 1])$ iff $(u^*, v^*) \in C$ and the first symbol of v is a
6. $(q, [u : k + 1, v : 0], \uparrow) \in \delta(q, a, [u : k, v : 0])$ iff $(u, v) \in C,\ k < |u|$ and the $(k + 1)$th symbol of u is a
7. $(q, [w : k + 1], \uparrow) \in \delta(q, a, [w : k])$ iff $w \in A,\ (k + 1) < |w|$ and the $(k + 1)$th symbol of w is a
8. $(q, [u : |u|, v : 0], \downarrow) \in \delta(q, \lambda, [u : |u|, v : 0])$ iff $(u, v) \in C$
9. $(q, [w : 1], \uparrow) \in \delta(q, a, Z)$ iff $w \in A$ and a is the first symbol of w
10. $(q, [u : |u|, v : k + 1], \uparrow) \in \delta(q, a, [u : |u|, v : k])$ iff $(u, v) \in C,\ (k + 1) < |v|$ and the $(k + 1)$th symbol of v is a
11. $(q, \lambda, \uparrow) \in \delta(q, a, [u : |u|, v : |v| - 1])$ iff $(u, v) \in C$ and the last symbol of v is a
12. $(q, \lambda, \uparrow) \in \delta(q, a, [w : |w| - 1])$ iff $w \in A$ and the last symbol of w is a

We will prove that $L(M) = L(G)$.

First we will prove that $L(M) \subseteq L(G)$. For this let w be a word from $L(M)$. Let C_1 be the set of all the contexts (u, v) such that $[u : k, v : l]$ was inserted in the stack during the computation for w.If the context (u, v) was inserted more than once we will consider each apparition as a distinct one. In this set we will consider the relation $>$ defined by $(u, v) > (x, y)$ for some $(u, v), (x, y) \in C_1$ iff $[u : k, v : l]$ appeared above $[x : s, y : t]$. Let $\succ$ be the transitive closure of $>$. We will say that (x, y) is in relation with (u, v) iff any of $(u, v) \succ (x, y)$ or $(x, y) \succ (u, v)$ holds. We will define inductively $(A_k)_{k \in I\!N}$ by:

$A_0 = \{(u,v) \in C_1 | (x,y)$ in relation with $(u,v) \Rightarrow (u,v) \succ (x,y)\}$
$A_k = \{(u,v) \in C_1 | (x,y)$ in relation with $(u,v) \Rightarrow (u,v) \succ (x,y)$ if $(x,y) \notin A_i$ for $i < k\}$ for $k > 0$.
Let $max = argmax(A_k | A_k \neq \emptyset)$. A_0 is the set of contexts that have been on the top of the stack and no other contexts have been above them until they were completed. A_k is the set of contexts that have had above them in the stack only contexts from $A_i, i < k$. A_k can be seen as the set of contexts that could have been on the top only if the contexts from $A_i, i < k$ were not present. From the condition that we can go over one context in the stack only if it has its left side completed it results that the contexts from A_i were inserted in the word after the contexts A_{i+1}. Let α be the word from A found on the stack. If we insert into α the contexts from $A_k, k > 0$ starting from A_{max} and we continue descending, according to the indexes, it is easy to see that we obtain w.

We will prove now that $L(G) \subseteq L(M)$. For this let w be a word from $L(G)$ such that it was obtained from $\alpha \in A$ by inserting contexts $(c_i)_{i>0}$ in increasing order, according to the indexes. The acceptance of w follows the next algorithm. A context c_i is inserted (in the stack) such that every context inserted above has bigger index and every context below him (in the stack) has lower index. This can be done because when c_i is inserted, if we have the context $c_k = (u_k, v_k)$, where $k > i$, present in the stack then we have read u_k entirely from the input tape, or we have read the whole c_k from the input tape. In this way we will have on the top of the stack the most recent inserted context. It is easy to see that an acceptance can be done only when we have deleted the initial symbol of the stack Z, i.e. when a word from A was inserted in the stack. So, we have proven that $L(M) = L(G)$. □

Remark 3.1 *The restriction that we cannot go over contexts in stack if they do not have their left side completed, and their right side is not in the process of completion, is natural if we remember that in this grammars the contexts are not shuffled in the word, but they are inserted as a right word and a left word, with the property that inside this two words we will not find symbols inserted before them.*

Remark 3.2 *The way gtas are used in acceptance of internal contextual grammars without choice can be now summarized: for every symbol of the input tape we will try to match it into an existing context or axiom (and this is done by going through the stack following the rules stated in remark 3.1) or by inserting in the stack a new context having its first symbol the symbol of the input tape that was just read. The transitions of the automaton from Theorem 3.2 implement this strategy, taking into account all the different cases that could occur.*

Remark 3.3 *Note that the sets A_k produced during the acceptance of some word w can be used as a description of the strucutre of a derivation of w. If we associate with every context c from A_k the string that was read from the input tape before c was inserted in the stack, then c will be inserted in the sentential form after every context from A_{k-1} was inserted, and right after (as position) the string associated with it. We will start the derivation with the axiom found in the stack.*

Theorem 3.3 *For any internal contextual grammar with finite choice G, having any $(u,v) \in C \Rightarrow u \in V^+, v \in V^+$ and $\lambda \notin A$, there exists a non-expansive go through automaton M such that $L(M) = L(G)$.*

Proof: Let $G = (V, A, C, \phi)$ be an internal contextual grammar with finite choice. We define a non-expansive gta $M = (Q, V, W, q, Z, F, \delta)$ by the following: $Q = \{q\} \cup \{q^l | l \in V\}$, $W = \{[u : k, x : p, v : t] | (u,v) \in \phi(x), k, p, t \in I\!N, k < |u| + 1, p < |x| + 1, t < |v| + 1\} \cup \{[w : k] | w \in A, k \in I\!N, k < |w| + 1\}$ (where we will interpret $[u : k, x : p, v : t]$ and $[w : k]$ as we have done in Theorem 3.2) and δ is defined using the same principles as in the precedent theorem:

1. $(q, [w : 1], \uparrow) \in \delta(q, a, Z)$ iff a is the first symbol of $w \in A$
2. $(q, [u : 1, x : 0, v : 0]Z, \uparrow) \in \delta(q, a, Z)$ iff a is the first symbol of u where $(u,v) \in \phi(x), x \in V^*$
3. $(q, [u : 0, 0 : 0, v : 0], \uparrow) \in \delta(q, a, [w : |w| - 1])$ iff a is the last symbol of w and $(u,v) \in \phi(\lambda)$
4. $(q, [u^* : 0, 0 : 0, v^* : 0], \uparrow) \in \delta(q, a, [u : |u|, x : |x|, v : |v| - 1])$ iff a is the last symbol of v and $(u^*, v^*) \in \phi(\lambda)$
5. $(q, [u : 1, x : 0, v : 0]M, \uparrow) \in \delta(q, a, M)$ iff $(u,v) \in \phi(x)$, the first symbol of u is a and $M \in W$
6. $(q, [u : k + 1, x : 0, v : 0], \uparrow) \in \delta(q, a, [u : k, x : 0, v : 0])$ iff $(u,v) \in \phi(x)$, $k < |u|$ and the $(k+1)$th symbol of u is a
7. $(q, [w : k + 1], \uparrow) \in \delta(q, a, [w : k])$ iff $w \in A$, $(k + 1) < |w|$ and the $(k+1)$th symbol of w is a
8. $(q, [u : k + 1, x : 0, v : 0], \uparrow) \in \delta(q^l, l, [u : k, x : 0, v : 0])$ iff $(u,v) \in \phi(x)$, $k < |u|$ and the $(k+1)$th symbol of u is l
9. $(q, [w : k + 1], \uparrow) \in \delta(q^l, l, [w : k])$ iff $w \in A$, $(k + 1) < |w|$ and the $(k+1)$th symbol of w is l
10. $(q, [u : |u|, x : |x|, v : k + 1], \uparrow) \in \delta(q^l, l, [u : |u|, x : |x|, v : k])$ iff $(u,v) \in \phi(x)$, $(k + 1) < |v|$ and the $(k+1)$th symbol of v is l

11. $(q, [u : 1, x : 0, v : 0]M, \uparrow) \in \delta(q^l, l, M)$ iff $(u, v) \in \phi(x)$, the first symbol of u is l and $M \in W$

12. $(q, [u : 0, 0 : 0, v : 0], \uparrow) \in \delta(q^l, a, [w : |w| - 1])$ iff a is the last symbol of w and $(u, v) \in \phi(\lambda)$

13. $(q, [u^* : 0, 0 : 0, v^* : 0], \uparrow) \in \delta(q^l, a, [u : |u|, x : |x|, v : |v| - 1])$ iff a is the last symbol of v and $(u^*, v^*) \in \phi(\lambda)$

14. $(q, \lambda, \uparrow) \in \delta(q^l, l, [w : |w| - 1])$ iff $w \in A$ and the last symbol of w is l

15. $(q^l, [u : |u|, x : k + 1, v : 0], \downarrow) \in \delta(q, \lambda, [u : |u|, x : k, v : 0])$ iff $(u, v) \in \phi(x)$ and l is the $k + 1$th symbol of x

16. $(q^l, [u : |u|, x : k + 1, v : 0], \downarrow) \in \delta(q^l, \lambda, [u : |u|, x : k, v : 0])$ iff $(u, v) \in \phi(x)$ and l is the $k + 1$th symbol of x

17. $(q, [u : |u|, x : |x|, v : k + 1], \uparrow) \in \delta(q, a, [u : |u|, x : |x|, v : k])$ iff $(u, v) \in \phi(x)$, $(k + 1) < |v|$ and the $(k + 1)$th symbol of v is a

18. $(q, \lambda, \uparrow) \in \delta(q, a, [u : |u|, x : |x|, v : |v| - 1])$ iff $(u, v) \in \phi(x)$ and the last symbol of v is a

19. $(q, \lambda, \uparrow) \in \delta(q, a, [w : |w| - 1])$ iff $w \in A$ and the last symbol of w is a

To prove that $L(G) = L(M)$, we can take the same strategy used in Theorem 3.2

First we prove that $L(M) \subseteq L(G)$. Let w be a word from $L(M)$. We can construct, as we have already seen, an ordered set C_1 as being the set of all triples (u, v, p) such that $(u, v) \in \phi(p)$ and $[|u| : k, |p| : l, |v| : t]$ was present during the computation in the stack. In this set we will consider the relation $>$ defined by $(u, v, p) > (x, y, r)$ for some $(u, v, p), (x, y, r) \in C_1$ iff $[|u| : k, |p| : t, |v| : l]$ appeared above $[|x| : s, |r| : j, |y| : t]$.Let $\succ$ be the transitive closure of $>$. We will say that (x, y, r) is in relation with (u, v, p) iff any of $(u, v, p) \succ (x, y, r)$ or $(x, y, r) \succ (u, v, p)$ holds. It is easy to see that the relation is not contradictory. We will define inductively $(A_k)_{k \in I\!N}$ by:
$A_0 = \{(u, v, p) \in C_1 | (x, y, r)$ in relation with $(u, v, p) \Rightarrow (u, v, p) \succ (x, y, r)\}$
$A_k = \{(u, v, p) \in C_1 | (x, y, r)$ in relation with $(u, v, p) \Rightarrow (u, v, p) \succ (x, y, r)$ if $(x, y, r) \notin A_i$ for $i < k\}$ for $k > 0$.
Let $max = argmax(A_k | A_k \neq \emptyset)$. We observe that between any two symbols a, b of w such that a, b are symbols of the left side of a context from A_i there are no symbols from any context in $A_k, k > i$. The same affirmation holds for right sides. A_0 is the set of contexts that have been on the top of the stack and no other contexts have been above them until they were completed. A_k is the set of contexts that have had above them in the stack only contexts from $A_i, i < k$. A_k can be seen as the set of contexts that

could have been on the top of the stack only if the contexts from $A_i, i < k$ were not present. From the condition that we can go over one context in the stack only if it has its left side completed it results that the contexts from A_i were inserted in the word after the contexts A_{i+1}. Let α be the word from A found on the stack. We insert into α the contexts from $A_k, k > 0$ starting from A_{max}, and every context (u, v) is inserted in α around x (if there is more than one we could try all possiblities, or we can keep, together with the contexts, the words that were read from the input tape before certain context was inserted using the same strategy stated in Remark 3.3), where $(u, v, x) \in A_k$ (x is in α, otherwise the computation will not be completed). It is easy to see that, in the end, we will obtain w.

Now we will prove the reverse implication $L(G) \subseteq L(M)$. Let w be a word from $L(G)$ obtained from $\alpha \in A$ by inserting contexts $(c_i)_{i>0}$ in increasing order. The acceptance of w will follow the next algorithm (also simillar with the one in theorem 3.2). An context c_i is inserted such that every context inserted above it in the stack has bigger index and every context below him (in the stack) has lower index. This can be done because when c_i is inserted, any $c_k = (u_k, v_k)$, where $k > i$, which is present in the stack will have its left side u_k already found on the input tape, or the whole c_k was already found; anyways the selector x_k of c_k will be necessarily found when we try to complete v_k. In this way we will have on the top of the stack the most recent inserted context (the last context inserted in the sentential form), and, when we will descend in the stack, for every context that will be passed by, we will complete its selector with one symbol (this symbol is memorized in the state of the automaton). This algorithm will lead to the acceptance of w. Note that an acceptance can be done only when we are in state q, i.e. when a word from A was inserted in the stack, and consequently Z was deleted from the stack.

We now have that $L(M) = L(G)$. □

Remark 3.4 *The way gtas are used in acceptance of internal contextual grammars with finite choice is quite simillar with what we have done in the case of internal contextual grammars without choice. We go through the stack to find the right position of a symbol (in what context it fits), keeping in mind the conditions from Remark 3.1. We will point out the main differences that occur in the design of an automaton for internal contextual grammars with finite choice. An important condition is raised by the usage of selectors: now when we descend in the stack we must change every context that we meet, by highlighting that a symbol from its selector was found (this is done by increasing the number associated with the selector). We can only go past contexts that have uncompleted selectors. Since gta can descend in the stack only using λ transitions we suppose before we start descending that we will find the symbol a on the input tape, we find its place in the stack following the restrictions already stated, and then we check if the first not-consummed*

symbol of the input tape is really a. If it is so we continue the computation, otherwise the gta is blocked.

Remark 3.5 *The sets A_k produced during the acceptance of some word w can be used as a description of the strucutre of a derivation of w in the same way they were used in derivation in internal grammars without choice.*

4 Conclusions

The results presented above point out the main idea in using go through automata in recognition of different languages: for every input symbol we go through the stack trying to match it exactly where is its place. Of course a tool that can permit the search of that place unrestricted is too powerful. So the constraint that for each symbol we can go only once through the stack is meant to restrict the power of the device. Further, we intend to apply go through automata for the recognition of other non-context free languages.

References

[1] R. Gramatovici, Bounded Deterministic Go-Through Automata, *Third International Conference on Discrete Mathematics and Theoretical Computer Science - DMTCS'01*, Constanţa, România, 2001, to appear, 2003.

[2] R. Gramatovici, On the Recognizing Power of Go Through Automata, to appear, 2002

[3] S. Marcus, Contextual grammars, *Rev. Roum. Math. Pures Appl.*, 14 (10), 1969, 69-74.

[4] S. Marcus, Contextual grammars and natural languages, in *The Handbook of Formal Languages*, G. Rozenberg, A. Salomaa, eds., Springer-Verlag, Berlin, Heidelberg, New-York, vol. 2, 215-235, 1997.

[5] F. Mráz, M. Plátek, M. Procházcha, Restarting automata, deleting and Marcus grammars, in *Recent Topics in Mathematical and Computational Linguistics*, C. Martin-Vide, Gh. Păun, eds., Romanian Academy Publishing House, 2000, 218-233.

[6] Gh. Păun, *Marcus Contextual Grammars*, Kluwer, Dordrecht, Boston, London, 1997.

Coreferential Definite and Demonstrative Descriptions in French: A Corpus Study for Text Generation

Hélène Manuélian
LORIA, Nancy, France
helene.manuelian@loria.fr

Abstract. This paper presents a new classification for the use of definite and demonstrative descriptions, its application in a corpus analysis and the results of this analysis. The proposed classification is based on existing literature and extended to support the generation of definite and demonstrative NPs. The corpus analysis shows in particular, that subsequent mentions of a referent can perform two functions (repeating given information and/or introducing new information). The comparison between definite and demonstrative determiners leads to preliminary data for generation algorithms.

1 Introduction

Algorithms for the generation of referring expressions [Dale et Reiter, 1995, Van Deemter, 2001, Van Deemter, 2000, Krahmer et al., 2001] essentially generate definite descriptions. The purpose of these algorithms is to produce given a referent that is already present in the context a referring expression that is informative enough for the identification of the intended referent by the listener. In a different perspective, [Gardent, Striegnitz, 2000] present an algorithm for the generation of bridging descriptions based on a structured model of the context and a tight interleaving between inference and generation. All these algorithms generate definite descriptions referring to objects already present in the context and sometimes handle the definite / pronoun opposition. However, they never handle the distinction between definite and demonstrative descriptions. A good way to extend these algorithms could be to introduce the distinction between the definite and the demonstrative. Both determine anaphoric noun phrases, so we need to know how to choose between them [Kleiber, 1986, Kleiber, 1988, Corblin, 1987].

This paper is a first step in this direction: we propose to explore this distinction through a corpus study of the anaphoric uses of definite and demonstrative. We show that the literature about French definite and demonstra-

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

tive is not precise enough to handle the generation of coreferential expressions. We further present a corpus study based on this literature from which we derived a new classification of determiner use, taking into account important data for generation which is not mentioned in the literature namely, the global content of the noun phrase. We give in conclusion some directions for the generation of referring expressions.

2 A First Corpus Study

The corpus studied is a part of the PAROLE corpus[1] which contains 65 000 words, 8777 definite noun phrases and 555 demonstrative noun phrases. The corpus is composed of articles from the French newspaper *Le Monde* which are taken from every section (national and international politics, economics, culture, sports, and leisure). It is annotated at the morphosyntactic level according to the Multext / Multitag annotation scheme [Lecomte, 1997, Beaumont et al. 1998].

Our goal was to maximally automatize the corpus processing, so we applied much preprocessing before performing the annotation described in this section. We used the G-search tool [Corley et al., 2001] to identify the noun phrases in the corpus and we wrote several filters to adapt the resulting output to the format used by our annotation tool, MMAX [Muller, Strube, 2001]. The annotation with MMAX is completely manual.

2.1 Annotation Scheme

The first distinction between demonstrative and definite is linked with the process of referent identification. The definite determines an expression whose referent is unique in the context with respect to the description contained in the noun phrase. The demonstrative NP denotes a referent which is highly focused, and the semantic content of the description is not used to identify the intended referent [Kleiber, 1986, Kleiber, 1988, Corblin, 1987]. The literature identifies three main uses for both determiners: first mention, anaphoric mention or bridging. Our annotation identifies all these uses for both determiners.

First mention This is the case when the referent has not been mentioned before in the context and cannot be inferred from any antecedent. With the definite, the referent must be uniquely identifiable in the context. The demonstrative must be used with a gesture ("deictic use" of the demonstrative).

[1]This corpus is shared with the ATILF research unit (Analyse et Traitement Informatique de la Langue Française) in the context of the regional collaboration "CPER Intelligence Logicielle".

Anaphoric Noun Phrases Both determiners can be used in coreference. The difference is explained by the process of referent identification. The demonstrative requires the referent to be highly focused whereas it is not the case with the definite. Three types of anaphoric uses are found and annotated in our corpus. For each anaphoric use, an antecedent has to be identified in the previous text.
Direct Coreference Both determiners can be used in direct coreference situations. This is the case when the phrase head noun is the same in the antecedent and in the anaphor.
Indirect Coreference This is the case when the head noun of the anaphoric phrase is different from the head noun of the antecedent. The indirect coreference is also found with both determiners. Several ways of realising coreference are found and subtyped in the annotation, but this is out of the scope of this paper (for more details, see [Manuélian, 2003]).
Bridging This category of referring expression is essentially used with the definite [Clark, 1977]. This is the case when the antecedent and the anaphor do not corefer, but the anaphor is interpreted as a part of the antecedent or as an object linked to the antecedent by the world knowledge. We can find cases of demonstrative in bridging descriptions but these are rather rare.

Attributes and appositions We classified in a separated category definite noun phrases which corefer to another noun phrase in apposition, or attributes coreferring with the subject of the verb to be, considering that it was a particular case of coreference, expressed explicitly, and using different mechanism from classical coreference.

2.2 Results and Analysis

The table (1) shows the annotation results for both determiners. They confirm the results of previous theoretic and empirical studies [Corblin, 1987, Kleiber, 1986, Kleiber, 1988, Poesio, Vieira 1998, Vieira et al. 2002].

We can see clearly that the definite can be used in first mention cases, which is less the case for demonstrative. The proportion of first mention for the definite is very high compared with other empirical studies [Poesio, Vieira 1998], because it also contains the "containing inferrable" which are usually distinguished from other first mention uses (cases as *the light of the Sun*, that are uniquely identifiable in their first mention). Indeed, we can see that the most important use for demonstrative is the anaphoric use, which is not the case for definite (about 75% of the uses of the demonstrative are - directly or indirectly - coreferential against 16% for the definite).
We can also find an illustration of the reclassification power of the demonstrative if we look at only the coreferential uses: 312 cases of coreferential

uses of the demonstrative are indirect, which represent 77% of its coreferential uses (direct and indirect). 819 cases of indirect coreferential uses of the definite are found, which represents 57% of its coreferential use.
The bridging phenomena is not so important for both categories and we can confirm that it is more frequent with the definite than with the demonstrative (For more details see [Gardent et al. 2003]). This annotation nonetheless confirmed the existing results but also permitted the extraction of coreferential noun phrases (1400 definite and 400 demonstrative), and their more detailed study presented in the next section of this paper.

Relation	demonstrative number	demonstrative proportion	definite number	definite proportion
First Mention	113	20,36%	6893	78,53%
Association	9	1,62%	417	4,75%
Direct Coreference	94	16,94%	607	6,92%
Indirect Coreference	312	58,02%	819	9,33%
Appositions / attributes	17	3,06%	41	0,47%
TOTAL	555	100	8777	100

Figure 1: First Annotation Results

3 A New Classification of Determiner Use

3.1 Motivations

The classification used in our annotation is helpful if we want to study the linguistic uses of coreferential descriptions. The problem for generation is that this classification does not handle certain elements which are essential for the generation of anaphoric expressions. In particular, modifiers are not taken into account, and one of the problems for generation is to decide the semantic content of a complete description, not only the semantic content of the head noun. Moreover, studying our data, we found that the information contained in the anaphoric noun phrases is not necessarily given information (i.e. explicitly or implicitly entailed by the context). Indeed, in some cases, we found that the anaphoric noun phrase introduces new information about the referent. Given these observations, questions that need to be adressed to support better generation are:

- What is the communicative function of the noun phrase to be generated: does it conveys given information or does it introduce new information?
- If the information contained in the anaphoric noun phrase is given, does it always come from the antecedent, or from the context?
- If the information contained in the anaphoric noun phrase is new, which linguistic means are used to express it?

3.2 Classification and Examples

We first distinguished anaphoric expressions repeating information (Information Repeating Anaphors) from anaphoric expressions which add information (Information Adding Anaphors). The first category is divided into five subcategories, according to the source from which the information is taken (explicitly or inferred). The second category is divided into four classes according to the linguistic mean used to carry the new information. All the examples here are taken from the corpus.

Information Repeating Anaphors (IRA)

The information is given by the antecedent only (AO)

Example: *Celle-ci, (...) aurait tissé un réseau de* ***liens ambigus dans la gendarmerie, la sûreté de l'Etat, les clubs de tir****. Le procès (...) avait permis de mettre* ***ces liens*** *en relief.*
Translation: She would have established **some ambiguous links in the police, and clubs**. The trial brought **these links** further.

The information is given by the antecedent and the context (A+C)

Example: *Le patronat avait très sensiblement modifié* ***son comportement****. (...) La clé de* ***ce nouveau comportement*** *tient en deux chiffres.*
Translation: The employers modified strongly **their behaviour**. The reason for **this new behaviour (...).**

The given information is inferred from a lexical relation with the antecedent (LR)

Example: *L'Inde paie un tribut sans cesse plus lourd à* ***la sécheresse****,(...).* ***Ce phénomène*** *a été accentué par des choix économiques erronés.*
Translation: India suffers more and more of **the drought**. **This phenomenon** has been provoked by wrong economical choices.

The given information is inferred from a lexical relation with the antecedent and from the context (LR + C)

Example: *La municipalité s'est dotée récemment d'****un somptueux Palais des concerts****. C'est dans* ***ce bâtiment confortable et flambant neuf*** *qu'a eu lieu l'inauguration.*
Translation: The city council build recently **a beautiful Palace of concert**. The inauguration took place in **this comfortable and very new building**.

The given information is inferred by world knowledge from the antecedent and from the context (WKL)

Example: *Les journalistes ne feront pas de reportage sur* ***la visite de M. Honecker au cimetière de Neunkirchen, dans la Sarre, où sont enterrés ses parents****. (...) après que le chef d'Etat eut requis la "tranquillité"*

pour ***cette partie "privée" de son voyage en République fédérale.***
Translation: The journalist won't make reports about **the visit of M. Honecker in the Neunkirchen graveyard where his parents are buried**. (...) after the head of state asked for quietness during **this private part of his travel in the Federal Republic**.

Information Adding Anaphors

The new information is introduced by a specifying lexical relation (SLR)

Example: ***Ce document*** *souligne (...) les conséquences médicales de la consommation de tabac, (...). Les auteurs de* ***ce rapport*** *(...).*
Translation: **This document** stresses the consequences of consuming tobacco. The authors of **this report** (...).

The new information comes from a specifying lexical relation and from modifiers: (SLR + mod)

Example: *Mais à Roubaix (...),* ***le personnel*** *a l'impression de compter les points. (...) Pour* ***ces ouvrières du bassin houillier dont quelques-unes ont déjà trois heures de transport par jour****, la nouvelle (...).*
Translation: But in Roubaix **the staff** has the feeling to count points. For **these workers from the coalfield who travel three hours a day** the news (...).

The new information comes from modifiers: (mod)

Example: *L' aviation israélienne a effectué (...) un raid sur* ***le camp de refugiés palestiniens d'Ain-el-Heloue, dans les faubourgs de Saida, chef-lieu du Liban-sud****. Les chasseurs-bombardiers israéliens ont effectué (...) plusieurs attaques sur* ***ce camp qui compte soixante-mille habitants****, (...).*
Translation: The Israelian air force attacked **the palestinian refugee camp of Ain El Heloue in Saida suburb**. The israelians bombers made several dive attacks on **this camp which counts sixty thousand of inhabitants**.

The new information is in the whole phrase and no lexical relation is used: (No LR)

Example: *(...) je tombe sur un article intitulé " Pourquoi* ***les maris*** *prennent le large". Je me dis : cherche pas, ils se débinent pendant que tu t'échines à faire des pompes et des flexions,* ***ces salauds-là****.*
Translation: I find a paper which title is "Why **husbands** escape?". I say "They escape when you are practising sports, **these bastards.**"

3.3 Comparison between Definite Use and Demonstrative Use According to the New Classification

The two top tables of figure 2 show the source of the information used in IRA according to the categories defined in section (3.2), in relation to categories defined in the annotation scheme (section 2). For each category of annotation we distinguished in the table the modified anaphors by "+ mod". The bottom tables in 2 synthetize the results by showing the number of demonstrative and definite IRA in regard to the necessity of relating them to their antecedent with inferences. We assume that it is not necessary to make inferences if the information contained in the anaphor is explicitly given in the antecedent. The tables (3) shows the linguistic means for adding information for IAA, according to the classification defined in section (3.2) in relation with the annotation categories. As for tables in (2), we indicate when the anaphoric noun phrase is modified.

We removed from these results the NPs which have non-nominal antecedents. From these two tables which classify 1412 coreferential definite NPs and 352 coreferential demonstrative NPs, we can say as a first result that for both determiners, the most frequent use is the information repeating anaphoric one (about 75% of the noun phrases belong to the category of IRA). However, this leaves at least one fourth of definite and demonstrative descriptions which contain new information, a fact which should be taken into account in text generation. We will now compare the definite and the demonstrative inside each category of use.

Information Repeating Anaphors From the tables (2) we can observe the following facts: First, the information comes from the antecedent (directly or not - results in AO and LR column of the tables) in 70% of the cases for definite and in 57% of the cases with the demonstrative. This means that the demonstrative is probably more able to allow inferences from the context. Second, the content of IRA must be inferred with 69% of the demonstrative NPs and with 51% of the definite NPs.

These data are important because the existing generation algorithms only generate anaphoric noun phrases from the explicitly given information in the antecedent. These results show that if we introduce the possibility of generating demonstrative anaphoric noun phrases we will have to allow inference from other sources than the antecedent.

Information Adding Anaphors For noun phrases which add information, we found (3) that the most IAA demonstrative fall in the "modifier" category and most of the definite description fall into the No Lexical Relation one. We should approach this result with caution because the high proportion of proper nouns, combined with the fact a proper noun never have a lexical relation with a common noun, strongly biases the result,

Demonstrative	AO	A+C	LR	LR+C	WKL
Dir+mod	7	4	0	0	0
Dir	72	0	0	0	0
Ind+mod	1	0	10	15	17
Ind	0	22	57	10	44
total	80	26	67	25	61
proportion	21%	10%	26%	10%	23%

Definite	AO	A+C	LR	LR+C	WKL
Dir+mod	90	59	0	0	4
Dir	410	0	0	0	0
Ind+mod	4	6	72	31	53
Ind	0	9	136	22	126
total	511	73	208	53	183
proportion	50%	7%	20%	5%	18%

Demonstrative	Without inferences	With inferences
Direct	72	0
Direct + mod	7	4
Indirect	0	133
Ind + mod	1	42
total	80 (21%)	179 (69%)

Definite	Without inferences	With inferences
Direct	410	0
Direct + mod	90	63
Indirect	0	299
Ind + mod	4	162
total	504 (49%)	524 (51%)

Figure 2: Results for Information Repeating Anaphors

thus hiding the possibility that if the content of the anaphor has no lexical relation with a common noun antecedent, the demonstrative is used very frequently. Moreover, 43% of the demonstrative add information by this mean and most of them have a common noun as antecedent which is not the case for definite. So, the reclassifying power of demonstrative is illustrated by this data, and is obviously a mean to add information about a referent with the demonstrative.

Demonstrative	SLR	Mod	SLR +mod	No LR
Direct mod	0	11	0	0
Indirect mod	2	33	8	22
Indirect	4	0	0	13
total	6	40	8	41
	6,3%	42%	8,4%	43%

Definite	SLR	Mod	SLR +mod	No LR
Direct mod	0	43	0	0
Ind. mod	1	7	0	108
Indirect	12	38	14	61
total	13	81	21	169
	4,6%	28,5%	7,4%	59,5%

Figure 3: Results for Information Adding Anaphors

Synthesis We have now to observe the distribution of the different phenomena among all the anaphoric noun phrases. We present it in the table (4). The top table presents the proportion of the different anaphoric noun phrases within the category of IRA, and the bottom table the proportion of each category for IAA (each category is abreviated as before and preceded by the type of determiner). Because of the bias mentioned in the previous section we divided each table into two parts, one for anaphoric noun phrases with proper nouns as antecedent, the second for common nouns as antecedent.

We propose the following basis for a generation algorithm. It is based on the salience criteria proposed in the literature and on the frequency of occurences in the corpus which is the only parameter available at this mo-

IRA	dem AO	dem A+C	dem LR	dem LR+C	dem WKL	def AO	def A+C	def LR	def LR+C	def WKL
proper N	0	0	0	0	25	3	1	0	0	91
proportion	0%	0%	0%	0%	20,8%	2,5%	0,8%	0%	0%	75,8%
common N	80	26	67	25	36	508	72	208	53	92
proportion	6,8%	2,2%	5,7%	2,1%	3,1%	43,5%	6,2%	17,8%	4,5%	7,9%

IAA	dem SLR	dem mod	dem SLR+mod	dem NoLR	def SLR	def mod	def SLR+mod	def NoLR
proper N	0	10	0	11	0	15	0	160
proportion	0%	5,1%	0%	5,6%	0%	7,6%	0%	81,6%
common N	6	30	8	30	13	66	21	9
proportion	3,3%	16,4%	4,4%	16,4%	7,1%	36%	11,5%	4,9%

Figure 4: Functions of anaphoric noun phrases in the whole corpus

ment:

If the function of the description is IRA

If antecedent = proper noun

If the referent is focused, use a demonstrative and infer the content of the anaphor from the world knowledge

If the referent is unique in the context, use the definite and infer the content from:

- world knowledge
- the antecedent (give the type of the antecedent)
- the antecedent and the context

If antecedent = common noun:

If the referent is focused, use the demonstrative and infer the content from:

- the antecedent
- a lexical relation
- world knowledge
- antecedent and context
- lexical relation and context

If the referent is unique, use the definite and infer the content from:

- antecedent
- lexical relation
- world knowledge
- antecedent and context
- lexical relation and context

If the function of the description is IAA

If antecedent = proper noun

If the referent is unique in the context, use the definite and

- a NP without lexical relation
- a noun describing the type of the referent and modifiers

If the referent is focused use the demonstrative and :

- a NP without lexical relation
- a noun describing the type of the referent and modifiers

If antecedent = common noun
    If the referent is unique in the context, use the definite and :
    - modifiers
    - specifying lexical relation and modifiers,
    - specifying lexical relation
    - a NP without lexical relation
If antecedent = common noun and if focused, use the demonstrative and
    - equally use modifiers or a NP without lexical relation
    - a specifying lexical relation and modifiers
    - lexical relation

4 Conclusion and Future Work

In this paper we laid the ground for an extension of the existing generation algorithms for referring expressions which would encompass not only anaphoric definite descriptions but also anaphoric demonstrative descriptions. Based on the classification proposed in the literature, we described the results of a first corpus analysis. We then proposed a more detailed classification whose classes are arguably needed for a better specification of the different uses of definite and demonstrative. We applied this classification to the corpus thereby extracting from the resulting analysis interesting facts about both definite and demonstrative. We ended by sketching the basis of an algorithm supporting the choice between the two determiners. This study must be completed by adding parameters to lead to a real algorithm. These parameters could be discourse-linked restrictions (position in the anaphoric chain for example) or syntactic restrictions (distance from the antecedent, grammatical functions of the antecedent...).

5 Acknowledgements

I would like to thank Claire Gardent for her precious help and support, and Eric Kow for all the scripts and programmes he wrote for the corpus exploitation.

References

[Beaumont et al. 1998] Beaumont C., Lecomte J., et Hatout N., (1998) *Etiquetage morpho-syntaxique du corpus "Le Monde" pour les besoins du projet PAROLE*, Technical Report, INALF, Nancy.

[Clark, 1977] Clark H.H., Bridging *Thinking: Readings in Cognitive Science*, Johnson-Laird P.N., Wason P.C. (eds), Cambridge, Cambridge University Press.

[Corblin, 1987] Corblin F. (1987), *Indéfini, Défini et Démonstratif*, Genève, Paris, Droz.

[Corblin, 1999] Corblin F. (1999), Les références mentionnelles : le premier, le dernier, celui-ci. In *La référence (2) Statut et processus*, Mettouch A. and Quinyin H. (eds.), Travaux linguistiques du CERLICO, Rennes, PUF.

[Corley et al., 2001] Corley S., Corley M., Keller F., Crocker M.W., et Trewin S., (2001) Finding Syntactic Structure in Unparsed Corpora : The Gsearch corpus query system, *Computer and Humanities*, 35(2), pp81-94.

[Dale et Reiter, 1995] Dale R. Reiter E., (1995), Computational Interprerations of the Gricean Maxims in the Generation of Referring Expressions *Cognitive Sciences* 19(2), pp233-263.

[Gardent, Striegnitz, 2000] Gardent C., Striegnitz K., (2000), Generating Indirect Anaphora, proceedings of *IWCS'00 (International Workshop on Computational Semantics)*.

[Gardent et al. 2003] Gardent C., Manuélian H., Kow E., (2003), Which Bridges for Bridging Descriptions, in *EACL Workshop on Linguistically Interpreted Corpora* proceedings.

[Kleiber, 1986] Kleiber G., (1986), Pour une explication du paradoxe de la reprise immédiate un N - le N / un N - Ce N *Langue Française*, 72, pp 54-79.

[Kleiber, 1988] Kleiber G., (1988), Reprise immédiate et théorie des contrastes, Studia Romanica Posnaniensa, 13, pp 67-83.

[Krahmer et al., 2001] Krahmer E., van Erk S., Verleg A., (2001), A Meta-Algorithm for the Generation of Referring Expressions, proceedings of *8th European Workshop on Natural Language Generation* pp 29-39.

[Lecomte, 1997] Lecomte J., (1997) *Codage Multext - GRACE pour l'action GRACE / Multitag*, Technical Report, INALF, Nancy.

[Manuélian, 2002] Manuélian H., (2002), Annotation des descriptions définies : le cas des reprises par les rôles thématiques, proceedings of *RECITAL 2002, Nancy, France*, pp455-467.

[Manuélian, 2003] Manuélian H., (2003), Une analyse du démonstratif en corpus, proceedings of *TALN 2003, Batz sur Mer, France*.

[Muller, Strube, 2001] Muller C., Strube M., (2001) Annotating Anaphoric and Bridging Relations with MMAX, proceedings of *2nd SIGDial Workshop on Discourse and Dialogue*, pp90-95.

[Poesio, Vieira 1998] Poesio M., Vieira R., (1998), A Corpus Based Investigation of Definite Description Use, *Computational Linguistics*, 24-2 pp183-216.

[Vieira et al. 2002] Vieira R., Salmon-Alt S., Gasperin C., Schang E., Othero G., (2002), Coreference and Anaphoric Relations of Demonstrative Noun Phrases in a Multilingual Corpus, proceedings of *DAARC*.

[Van Deemter, 2000] Van Deemter K., (2000), Generating Vague Descriptions, proceedings of *First International Conference on Natural Language Generation*, pp 179-186.

[Van Deemter, 2001] Van Deemter K., (2001), Logical Form Equivalence : the Case of Referring Expressions Generation, proceedings of *8th European Workshop on Natural Language Generation* 21-29.

Formalizing a Constituency Based Dynamic Grammar

ALESSANDRO MAZZEI
Dip. Informatica, Università di Torino
mazzei@di.unito.it

ABSTRACT. Dynamic grammars are a grammatical formalism that describes the derivation and recognition processes via *dynamics*: a string is derived and parsed as a consequence of a transition between states. This paper formalizes tree adjoining based dynamic grammar (DV-TAG). In particular, we show that DV-TAG is adequate to describe some linguistic phenomena, and that we can also generate languages beyond the context-free power.

1 Introduction

Incrementality is a property of natural language understanding: humans build up partial analyses of the sentences as soon as the words are heard or seen ([MW73]). This property has important effects at syntactic and semantic analysis level: the language processor interprets syntactic structures, word by word, and incrementally produces semantic representation.

In two ways a syntactic theory can satisfy incrementality: working at *performance* level, it can respect the constraints in parsing and derivation strategies. Or working at *competence* level, it can modify the grammatical formalism, thus the grammar derives only structures that respect incremental constraints. In the latter case we fulfill the *transparency hypothesis* ([BW84]): a syntactic theory must reflect the behaviour of the human syntactic processor in its formal operations.
Stabler (cf. [Sta94]) formalized a stronger property (that we call *strong incrementality*): people incorporate each word into a single, totally connected syntactic structure before any further words follow. Incremental interpretation is achieved interpreting this single connected syntactic structure.

Dynamic grammars[1] can satisfy the strong incrementality and transparency hypothesis both: the time evolution of a single fully connected syntactic structure is governed by the appearance of lexical items. The syntactic process in a dynamic grammar is a sequence of transitions between adjacent states S_{i-1} and S_i moving from left to right in the input string.

[1] cf. [Gel98] for an overview of the dynamic paradigm in cognitive science.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

In the schema proposed by Milward ([Mil94]) to define a dynamic grammar, we have to define the *states* of the dynamic process, a set of *axioms* that relate a lexical item with a transition between states, some *deduction rules* that specify the time evolution of the process, and the sets of *initial* and *final* states. In this schema a grammar derives a sentence if and only if starting from an initial state and following the transitions associated with the words in the sentence, the decision process will be in a final state after reaching the last word of the sentence. We can also specify a dynamic grammar starting from a lexicalized grammatical formalism, and redefining the combination operations in a dynamic fashion, as the axioms of the dynamic grammar. In such a way Milward defined the dynamic version of dependency grammars.

In this paper, we formalize a TAG-related dynamic grammar, called *Dynamic Version of TAG* (DV-TAG), that satisfies strong incrementality. Following the informal work presented in [LS02], this dynamic grammar builds upon the lexicon and the attachment operations of Lexicalized TAG (L–TAG [Sch90]) in order to define the states and the transitions. This paper is an ongoing work about viability of the formalism in term of basic mechanisms of syntactic construction. States S_i are partially derived trees that span the input string from the beginning to the i-th word; transitions extend the left contexts S_{i-1} by attaching some elementary tree anchored by the word w_i through dynamic versions of Substitution and Adjoining. To our knowledge, DV-TAG is the first attempt to define a constituency based dynamic grammar. The constituency nature of DV-TAG also allows to state easy relationship with the mainstream frameworks of computational linguistics. For example, we can introduce some statistical parameters and define a stochastic version of our formalism to yield a wide coverage parser.

The following purely syntactic question remains open: where are DV-TAG grammars in the Chomsky hierarchy? We will show, as a starting point, that DV–TAG is powerful enough to derive a linguistically relevant context–sensitive language. Many other linguistic phenomena, including extractions and nested dependencies, can be accounted by DV-TAG. As an example we will show a derivation in DV-TAG for cross-serial dependencies.

2 Dynamic Version of Tree Adjoining Grammar (DV-TAG)

The standard version L-TAG does not respect the strong incrementality: we cannot always build a fully connected partially derived tree composing the elementary trees and following the word sequential order.

The intuitive idea behind DV-TAG is to convert the derivation process of Lexicalized TAG into a dynamic system. A DV-TAG grammar, as L-TAG grammar ([Sch90]), is a set of elementary trees, divided into initial trees and auxiliary trees, and attachment operations for combining them. In DV-TAG attachment operations always occur between the left context and an elementary tree, and some combinations are not permitted. In particular, the derivation tree, which illustrates the

derivation process in L-TAG (a context-free process), becomes a derivation sequence. In the rest of this section, we sketch a formal definition of DV-TAG. This definition is necessary for the study of the expressive power of the formalism, that we will show in future works. We will not use linguistic examples, only formal symbols (a, b, c, etc.) without semantic role, since we want to stress the discussion about syntactic issues. Therefore the paper does not present a formal definition of semantic interpretation.

First of all, we provide some general terminology. Let Σ be an alphabet of terminal symbols, and V an alphabet of non–terminal symbols. $a, b, c, d, ... \in \Sigma$ indicate terminal symbols, $A, B, C, D, ... \in V$ for non–terminal symbols, $x, y, w, z \in \Sigma^*$ are strings of terminals, $\rho, \sigma, \tau \in (\Sigma \cup V)^*$ are mixed strings; $|\rho|$ is the length of the string ρ. We denote initial trees with α, auxiliary trees with β, derived and generic trees with γ, δ, ζ; with $\mathcal{N}$ we denote a node belonging to a tree. $yield(\gamma)$ is the string of terminal and non–terminal symbols on the frontier of a tree γ; $yield_i(\gamma)$ is the i–th element of the $yield(\gamma)$.

We introduce two useful notions that are borrowed from parsing[2] *dotted tree* and *fringe*.

Definition 2.1. *A **dotted tree** is a couple $\langle \gamma, i \rangle$ where γ is a tree and $i \in 0...|yield(\gamma)|$.*

Definition 2.2. *The **left fringe** of dotted tree $\langle \gamma, i \rangle$ is a path from $yield_i(\gamma)$ to the root, minus the path from $yield_{i+1}(\gamma)$ to the root. The **right fringe** of a dotted tree $\langle \gamma, i \rangle$ is the path from $yield_{i+1}(\gamma)$ to root, minus the path from $yield_i(\gamma)$ to root. We will use dotted lines to denote fringes in the trees (fig 1).*

Dotted tree is the basic data structure in DV–TAG: the dot denotes a point in the yield of the tree and the fringes point where the next attachment operation can be applied.

Now we can define the attachment operations. Since an elementary tree brings new lexical material in the string, we need to be careful about the linear positioning of such material during the derivation process, in order to avoid holes in the left fragment of the sentence. Constraints involve the shape of the elementary trees (in particular, we keep distinct three types of auxiliary trees) and the definition of operations. Following the definitions of [SW95], we define **left auxiliary trees** $\mathcal{A}_L$ as auxiliary trees that have (non–empty) terminal symbols only on the left of the foot node, **right auxiliary trees** $\mathcal{A}_R$ as auxiliary trees that have (non–empty) terminal symbols only on the right of the foot node, **wrapping auxiliary trees** $\mathcal{A}_W$ as auxiliary trees that have (non–empty) terminal symbols on both the left and the right of the foot node. Elementary trees where the leftmost symbol of the yield is a terminal symbol are called **left anchored trees**. For left anchored auxiliary trees, it can be the case that the foot node is at the left of the leftmost anchor.
We define six attachment operations on a dotted tree: two substitutions (similar

[2] In dynamic grammars parsing and derivation are achieved by the same mechanisms, then it is very spontaneously to use some parser concepts.

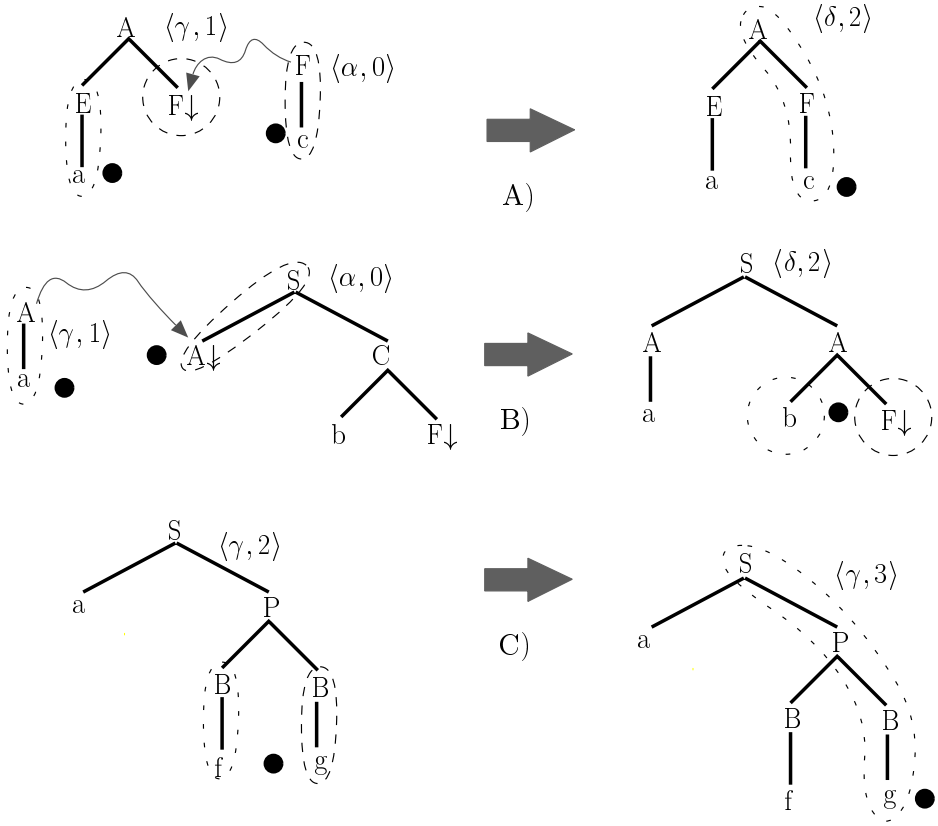


Figure 1: Operations in DV–TAG.

to L–TAG substitution), three adjunctions (similar to L–TAG adjunction), and one shift.

- **Substitution** $\langle \delta, i+1 \rangle = Sub^{\rightarrow}(\langle \alpha, 0 \rangle, \langle \gamma, i \rangle)$
  Let α be a left anchored initial tree that has the same root label as the substitution node N$\downarrow$ on the right fringe of $\langle \alpha, i \rangle$. The root of α is merged into the node N, and the dot in the new dotted tree is immediately to the right of the leftmost anchor of α (figure 1-A).

- **Inverse Substitution** $\langle \delta, i+1 \rangle = Sub^{\leftarrow}(\langle \alpha, 0 \rangle, \langle \gamma, i \rangle)$
  Let α be an initial tree, such that $yield_1(\alpha)$ is a substitution node N$\downarrow$ labelled like the root of γ, and $yield_2(\alpha)$ is the leftmost anchor of α: the root of γ is merged into N$\downarrow$, and the dot in the new dotted tree is on the right of the left anchor of α (figure 1-B).

- **Shifting** $\langle \gamma, i+1 \rangle = Shi(\langle \gamma, i \rangle)$
  Let $\langle \gamma, i \rangle$ be a dotted tree. The dot is shifted on the right to the next position

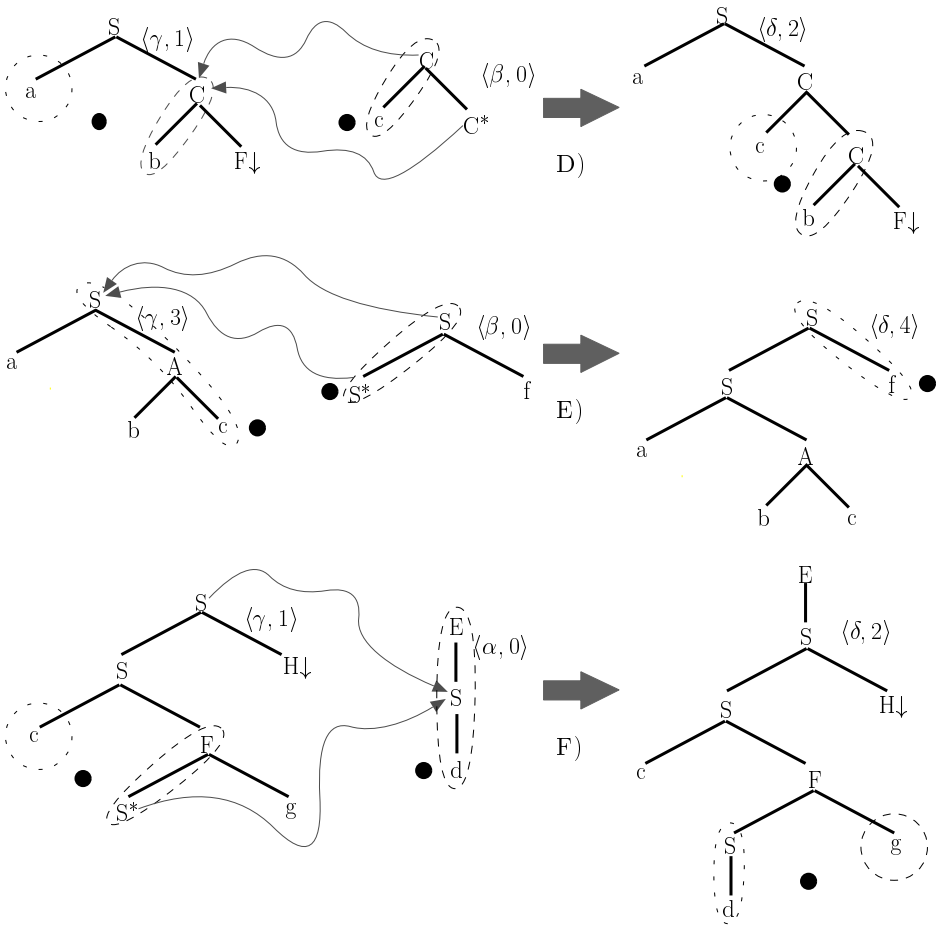


Figure 2: Operations in DV–TAG.

of the frontier (figure 1-C).

- **Adjoining from the left** $\langle \delta, i+1 \rangle = \nabla_L^{\rightarrow}(\langle \beta, 0 \rangle, \langle \gamma, i \rangle, \mathcal{N})$
  Let $\beta \in (\mathcal{A}_L \cup \mathcal{A}_W)$ be a left anchored left or wrapping auxiliary tree: β is grafted into a non–terminal node $\mathcal{N}$ that belongs to the right fringe of the dotted tree $\langle \gamma, i \rangle$ and has the same label as the root of β. In the new dotted tree, the dot is at the position $i+1$ of the frontier (figure 2-D).

- **Adjoining from the right** $\langle \delta, i+1 \rangle = \nabla_R^{\rightarrow}(\langle \beta, 0 \rangle, \langle \gamma, i \rangle, \mathcal{N})$
  Let $\beta \in \mathcal{A}_R$ be a left anchored right auxiliary tree: β is grafted into a non–terminal node $\mathcal{N}$ that belongs to the left fringe of the dotted tree $\langle \gamma, i \rangle$ and has the same label as the root of β. In the new dotted tree the dot is in the position $i+1$ of the frontier (figure 2-E).

- **Inverse Adjoining from the Left** $\langle \delta, i+1 \rangle = \nabla_L^{\leftarrow}(\langle \zeta, 0 \rangle, \langle \gamma, i \rangle, \mathcal{N})$

Let γ be a left or wrapping auxiliary tree: γ is grafted into a non–terminal node $\mathcal{N}$ that belongs to the right fringe of the dotted tree $\langle\zeta, 0\rangle$. In the new dotted tree the dot is to the right of the leftmost anchor of ζ (figure 2-F).

Finally, we can give a formal definition of a **DV–TAG**. We adapt the schema defined in [Mil94] for the dynamic dependency grammars. Like in L–TAG, we have two sets of trees: a set of initial trees $\mathcal{I}$, and a set of auxiliary trees $\mathcal{A} = \mathcal{A}_L \cup \mathcal{A}_R \cup \mathcal{A}_W$.

Definition 2.3. *Given a lexicon $L = (\mathcal{I}, \mathcal{A})$, the DV–TAG G is a quadruple consisting of the set of states (including initial and final states), the set of axioms, the deduction rule, and the specification of the legal strings and trees.*

- ***Set of states**, where a **state** is a dotted tree $\langle\gamma, i\rangle$;*
  - ***Initial states**:* $\{\langle S\downarrow, 0\rangle\}$
  - ***Final states**:*
    $\{\langle\gamma, n\rangle : Rfringe(\langle\gamma, n\rangle) = \emptyset\}$.
- ***Set of axioms**:* $\langle\gamma, i\rangle \xrightarrow{a} \langle\delta, i+1\rangle$
  1. $a \in \Sigma, a \in Rfringe(\langle\gamma, i\rangle)$
     $$\langle\delta, i+1\rangle = Shi(\langle\gamma, i\rangle) = \langle\gamma, i+1\rangle$$
  2. $a \in \Sigma$, *leftmost anchor of* ζ, *with*
     $\zeta = \alpha \in \mathcal{I}$, *or*
     $\zeta = \beta_L \in \mathcal{A}_L \cup \mathcal{A}_W$, *or*
     $\zeta = \beta_R \in \mathcal{A}_R$
     $$\langle\delta, i+1\rangle = \begin{cases} Sub^{\rightarrow}(\langle\alpha, 0\rangle, \langle\gamma, i\rangle) \\ Sub^{\leftarrow}(\langle\alpha, 0\rangle, \langle\gamma, i\rangle) \\ \nabla_L^{\rightarrow}(\langle\beta_L, 0\rangle, \langle\gamma, i\rangle, \mathcal{N}) \\ \nabla_R^{\rightarrow}(\langle\beta_R, 0\rangle, \langle\gamma, i\rangle, \mathcal{N}) \\ \nabla_L^{\leftarrow}(\langle\zeta, 0\rangle, \langle\gamma, i\rangle, \mathcal{N}) \end{cases}$$
- ***Deduction rule** (a, b terminal symbols):*
  $$\frac{\langle\gamma, i\rangle \xrightarrow{a} \langle\delta, i+1\rangle \qquad \langle\delta, i+1\rangle \xrightarrow{b} \langle\zeta, i+2\rangle}{\langle\gamma, i\rangle \xrightarrow{ab} \langle\zeta, i+2\rangle}$$
- ***Specification of the legal strings and trees**:*
  - *A **string** w is generated by the grammar if and only if*
    $$\langle S\downarrow, 0\rangle \xrightarrow{w} \langle\gamma, n\rangle$$
    with $\langle\gamma, n\rangle$ final state.

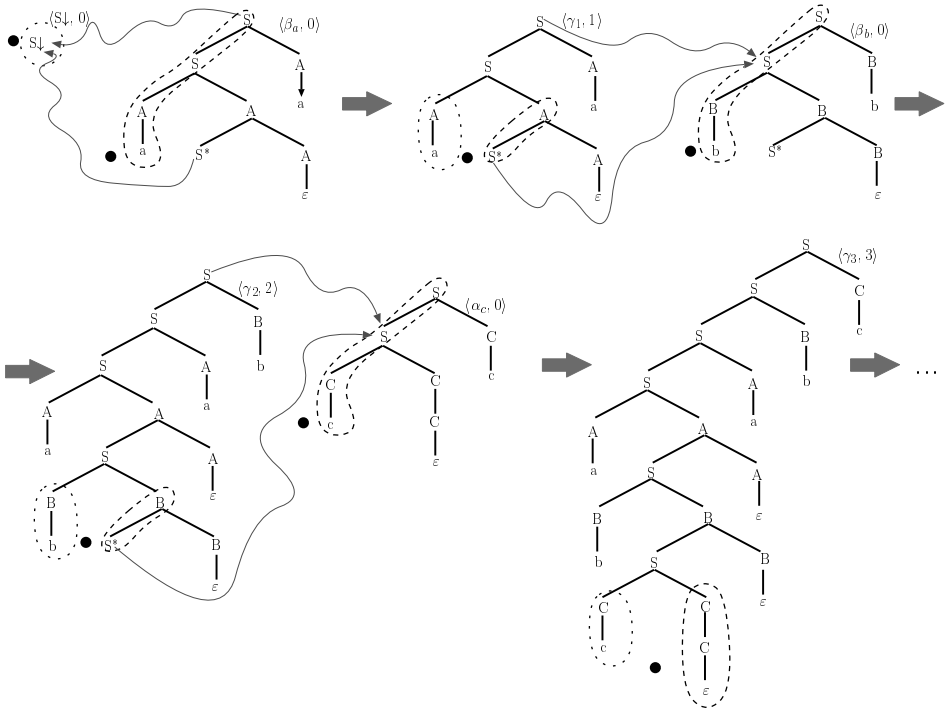


Figure 3: A fragment of DV–TAG derivation of the string “abcabc”.

– *A **tree** γ is derived by the grammar if and only if* $\exists w$ *such that*

$$\langle S\downarrow, 0\rangle \xrightarrow{w} \langle \gamma, n\rangle$$

with $\langle \gamma, n\rangle$ *final state.*

Left–context can be defined as a dotted tree $\langle \gamma, i\rangle$ such that $yield(\gamma) = w\rho$, with $w \in \Sigma^*$, $\rho \in (V \cup \Sigma)^*$ and $|w| = i$. **DV–TAG derivation** is the sequence of left contexts.

Cross–serial dependencies

Formally speaking, the *dependencies* in a string are equivalence relations between elements of the words. As usual we represent these relations with lines that link the elements. A grammatical system generates dependencies in a string, if in the derivation of the string, elements in the dependency relation are inserted in the same step of the derivation ([BRN92]). The derivational generative capacity of a grammatical formalism is the set of the dependencies that the formalism can generate.

Cross-serial dependencies, expressed by the string a b c a b c, are well-known test-bed for grammatical formalism attempting to describe natural languages. There

are real linguistic constructions, as subordinate phrase construction in Dutch, that show these dependencies. It is easy to show that context-free grammars cannot generate this type of dependencies, therefore we need a more powerful formalism to take this phenomenon into account. Indeed the cross-serial dependencies play a key role in the definition of mildly *context-sensitive languages* ([VSJW90]).
We show that DV-TAG is powerful enough to derive cross-serial dependencies, and to produce a derived tree equal to the verb-raising analysis of Dutch subordinate phrase construction given in [KS91], using a DV-TAG with the lexicon $\mathcal{I} = \{\alpha_c\}, \mathcal{A} = \{\beta_a, \beta_b\}$ (figure 3). The complete DV–TAG derivation of the string "abcabc" is:

$$\langle S\downarrow, 0\rangle \xrightarrow{a} \langle\gamma_1, 1\rangle \xrightarrow{b} \langle\gamma_2, 2\rangle \xrightarrow{c} \langle\gamma_3, 3\rangle \xrightarrow{\varepsilon} \langle\gamma_4, 4\rangle \xrightarrow{\varepsilon} \langle\gamma_5, 5\rangle \xrightarrow{\varepsilon} \langle\gamma_6, 6\rangle \xrightarrow{a} \langle\gamma_7, 7\rangle \xrightarrow{b} \langle\gamma_8, 8\rangle \xrightarrow{c} \langle\gamma_9, 9\rangle,$$

where the first operation is an adjoining from the left operation, the second and third operations are inverse adjoining from the left operations (figure 3 depicts these operations), and the last six steps are shifting operations.

Beyond the CF–power

Weakly generative capacity of DV-TAG is greater than weakly generative capacity of CF-grammars. This expressive power is a direct consequence of L–TAG generative capacity ([VSJW90]). DV–TAG with the lexicon $\mathcal{I} = \{\alpha_e\}, \mathcal{A} = \{\beta_a, \beta_b\}$ (figure 4) derives the language

$$\begin{aligned} L = \{wew' \mid\ & w, w' \in \{a,b\}^*, |w| = |w'|, \\ & number\ of\ a\ in\ w = number\ of\ a\ in\ w', \\ & number\ of\ b\ in\ w = number\ of\ b\ in\ w'\ \}. \end{aligned}$$

This language is the context-sensitive language[3] related to example discussed by Shieber ([Shi85]) against the context-freeness of natural languages. There is the initial fragment of the DV-TAG derivation for the sentence "$abeab$" in figure 4.

3 Open questions

We gave a formal definition for a dynamic grammar based on tree adjoining grammars, and we showed that several linguistic related formal languages can be derived by these grammars.

In this paper we do not define any semantics for DV-TAG (we are stile working on this part of the formalism) even if there are two important issues that could be pointed out about semantics:

- In standard L-TAG we can couple each elementary tree with a *semantic unit*. Thus obtaining the semantics of the derived tree via the composition of these

[3]The intersection of L with the regular language $a^*b^*ea^*b^*$ gives the language $a^nb^mea^nb^m$: the CF-languages are closed under intersection with regular languages, then L is not context-free.

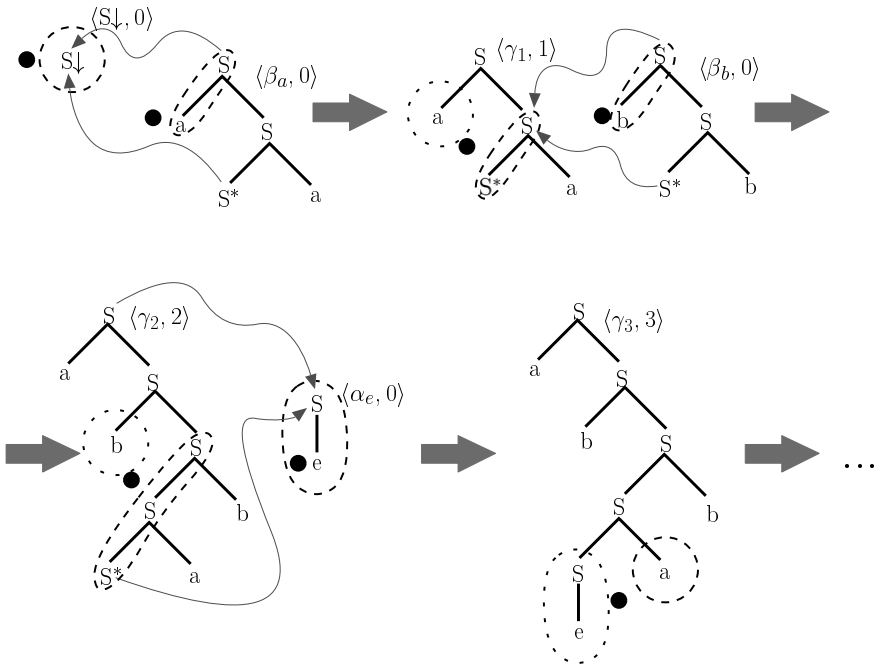


Figure 4: A DV–TAG deriving a context–sensitive language

units. We have to distinguish adjoining and substitution through different rules of semantic composition. We can say that "the derivation trees represent predicate-argument structures" ([AR00]).

In DV-TAG, in order to satisfy the strong incrementality, and then to allow incremental construction of a derivation tree, we have to "augment" the elementary structures with nodes that are not projected by the lexical anchor of the tree[4]. To respect the semantic compositionality we have to define an "unification-substitution" operation to check which nodes of the elementary tree are yet in the partially derived tree ([LS02]). In this paper, we did not formalize this unification operation, because it does not change the expressive power of the formalism, even if it is necessary for linguistic reasons.

- In the standard TAG, the derivation tree plays a key role in the definition of semantic interpretation but, the way the derivation tree is built is irrelevant. In DV-TAG we have a more powerful structure, called *derivation sequence*, describing the derivation tree and also how it is built. In our opinion, semantics for DV-TAG could be better defined using this structure.

In the future work we will first draw the exact position of DV–TAG in Chomsky hierarchy, answering the question whether DV–TAGs are mildly context-sensitive

[4]This procedure resembles the *type-raising* procedure of CCG formalism ([Ste00])

grammars ([VSJW90]). In this case, we have to show that DV–TAG is polynomially parsable and this will allow us to develop a wide coverage parser.

References

[AR00] Abeillé Anne and Owen Rambow, editors. *Tree Adjoining Grammars: Formalisms, Linguistic Analysis and Processing*. (CSLI-LN) Center for the Study of Language and Information-Lecture Notes, 2000.

[BRN92] T. Becker, O. Rambow, and M. Niv. The Derivational Power of Formal System or Scrambling is beyond LCFRS. Technical report, Institute for Research in Cognitive Science, University of Pennsylvania, 1992.

[BW84] R. Berwick and A. Weinberg. *The grammatical basis of linguistic performance: language use and acquisition*. MIT Press, 1984.

[Gel98] Tim Van Gelder. The dynamical hypothesis in cognitive science. *Behavioral and Brain Sciences*, 21:1–14, 1998.

[KS91] A. Kroch and B. Santorini. The derived constituent structure of the West Germanic verb-raising construction. In R. Freidin, editor, *Principles and Parameters in comparative grammar*, pages 269–338. MIT Press, Cambridge: MA, 1991.

[LS02] Vincenzo Lombardo and Patrick Sturt. Towards a dynamic version of tag. In *Proceedings of 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG +6)*, pages 101–110, Venice (Italy), 20-23 May 2002.

[Mil94] D. Milward. Dynamic dependency grammar. *Linguistics and Philosophy*, 17(6), 1994.

[MW73] W. Marslen-Wilson. Linguistic structure and speech shadowing at very short latencies. *Nature*, 244:522–533, 1973.

[Sch90] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1990.

[Shi85] S. M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343, 1985.

[Sta94] E. P. Stabler. The finite connectivity of linguistic structure. In C. Clifton, L. Frazier, and K. Reyner, editors, *Perspectives on Sentence Processing*, pages 303–336. Lawrence Erlbaum Associates, 1994.

[Ste00] M. J. Steedman. *The syntactic process*. A Bradford Book, The MIT Press, 2000.

[SW95] Yves Schabes and Richard C. Waters. Tree insertion grammar: A cubic-time, parsable formalism that lexicalizes Context-free grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–513, 1995.

[VSJW90] K. Vijay-Shanker, A. Joshi, and D. Weir. The convergence of mildly context-sensitive grammatical formalisms. In Peter Sells, Stuart Shieber, and Tom Wasow, editors, *Foundational Issues in Natual Language Processing*. MIT Press, Cambridge MA, 1990.

Exploiting Sequent Structure in Membership Algorithms for the Lambek Calculus

Ryan T. McDonald
Department of Computer and Information Science
University of Pennsylvania
ryantm@cis.upenn.edu

Abstract. This paper will examine the open problem of whether or not a sequent is derivable in the Lambek Calculus (**L**) in polynomial-time. This will be done through an investigation of *Lambek Calculus Graphs* (LC-Graphs), which were introduced by Penn[7] to represent the well-formedness constraints of a sequent's derivation in **L**. Presented here is a simplified version of LC-Graphs and their integrity criteria. We also show that storing a small amount of structural information about a sequent during parsing can reduce the number of integrity criteria for LC-Graphs from four to two. To this effect, a polynomial-time membership algorithm is presented that recognizes all derivable sequents and falsely recognizes an identifiable class of underivable sequents.

1 Introduction

Substructural logics are a group of logics whose proof systems only use a subset of the structural rules of classical proof systems. The most well known substructural logics include *Relevance Logic*, which does not employ weakening, and Girard's[3] *Linear Logic*, which uses neither weakening nor contraction. The complexity of membership algorithms for these logics is well studied with results often proving their intractability [4, 5].

Also in the class of substructural logics is the Lambek Calculus (**L**), which has the following structural rules:

$$(/L)\ \frac{\Gamma' \vdash B \quad \Gamma, A, \Gamma'' \vdash S}{\Gamma, A/B, \Gamma', \Gamma'' \vdash S} \qquad (/R)\ \frac{\Gamma, B \vdash A}{\Gamma \vdash A/B}\,(*) \qquad (\backslash L)\ \frac{\Gamma' \vdash B \quad \Gamma, A, \Gamma'' \vdash S}{\Gamma, \Gamma', B\backslash A, \Gamma'' \vdash S} \qquad (\backslash R)\ \frac{B, \Gamma \vdash A}{\Gamma \vdash B\backslash A}\,(*)$$

$$(\bullet L)\ \frac{\Gamma, A, B, \Gamma' \vdash S}{\Gamma, A \bullet B, \Gamma' \vdash S} \qquad (\bullet R)\ \frac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma, \Gamma' \vdash A \bullet B}$$

* if Γ is non-empty.

These operators are known as left-implication ($\backslash$), right-implication ($/$) and product ($\bullet$), where A/B means looking for premise B on the right to imply A

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

and A\B means looking for premise A on the left to imply B. For simplicity this paper will focus on the product-free fragment of **L** and sequents with non-empty sequences of premises.

The Lambek Calculus is of interest to both computer scientists and linguists because it is a basis for a deductive system for *Categorial Grammars* (CG). A CG consists of a set of base categories (N, NP, S, etc.), a lexicon and a distinguished category, s. Given a grammar, a string of words, $w = (w_1, \ldots, w_n)$ and their corresponding categories, $C = (C_1, \ldots, C_n)$[1], the principle question is whether or not w is generated by the grammar. If we view each category C_i as a premise and the distinguished category, s, as the consequent, then this question is the same as asking "is the sequent $C_1\ C_2\ \ldots\ C_n\ \vdash\ s$, derivable in **L**?". It is, however, unknown whether or not a sequent can be derived in **L** in polynomial-time as a function of the input size.

There are many reasons to believe that sequent derivability in **L** is intractable, including results showing that membership in **LP** (**L** with permutation) and Semidirectional Lambek Calculus [2] is NP-complete. However, since both these logics either partially or completely allow for permutation there is still a reasonable basis for believing that the Lambek Calculus can achieve membership recognition in polynomial-time. This paper essentially focuses on finding such a polynomial solution, but it may be the case that observations made here lead to an NP-complete reduction. During this process we will exclusively work with Lambek Calculus Graphs (LC-Graphs)[7]. Section 2 describes the basic framework that will be used in this investigation as well as presents a simplified version of LC-Graphs and their correctness criteria. In section 3 a complete but unsound polynomial-time chart parsing algorithm is presented which satisfies all but two of the four correctness criteria of LC-Graphs.

2 The Framework

Much of this and the next section have been drawn from Penn[7]. Both it and Roorda[8] have a more complete account. Below is only meant to provide enough background information to proceed.

Before the definition of LC-Graphs is presented we must first define two central constructs, **axiomatic formula** and **axiomatic linkage**.[2] Their definitions follow from how they are constructed from some given sequent. To illustrate, we will consider the sequent S:

$$(A/(A\backslash A))/A\ A\ A\backslash A\ A\backslash A\ \vdash\ A$$

[1] Here we assume a one-to-one correspondence of words and categories.

[2] Axiomatic formulae and linkages also serve as the basis for Lambek proof-nets [8], from which the correctness criteria of LC-Graphs is based [7].

2.1 Axiomatic Formulae

Polarize S so that each premise category becomes negatively polarized and the consequent becomes positively polarized. Then label each of these categories with a unique variable (see below).

For each polarized category, unfold it to obtain a sequence of axiomatic formulae using the following *lexical unfolding rules*:

$$
\begin{aligned}
(A\backslash B)^{-}{:}\,t &\longrightarrow A^{+}{:}\,u\ B^{-}{:}\,tu \\
(A\backslash B)^{+}{:}\,v &\longrightarrow B^{+}{:}\,v'\ A^{-}{:}\,u\ [v := \lambda u.v'] \\
(A/B)^{-}{:}\,t &\longrightarrow A^{-}{:}\,tu\ B^{+}{:}\,u \\
(A/B)^{+}{:}\,v &\longrightarrow B^{-}{:}\,u\ A^{+}{:}\,v'\ [v := \lambda u.v']
\end{aligned}
$$

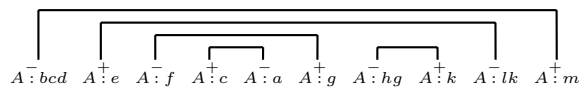
An *axiomatic formula* is an unfolded base category that was part of some base/complex category in either the sequent's premise set or it's conclusion. For example, the following sequent is labeled and unfolded to a produce a sequence of axiomatic formulae as follows:

$$
\begin{aligned}
&(A/(A\backslash A))/A\ A\ A\backslash A\ A\backslash A \vdash A \longrightarrow \\
&((A/(A\backslash A))/A)^{-}{:}\,b\ A^{-}{:}\,a\ (A\backslash A)^{-}{:}\,h\ (A\backslash A)^{-}{:}\,l\ A^{+}{:}\,m \longrightarrow \\
&(A/(A\backslash A))^{-}{:}\,bc\ A^{+}{:}\,c\ A^{-}{:}\,a\ (A\backslash A)^{-}{:}\,h\ (A\backslash A)^{-}{:}\,l\ A^{+}{:}\,m \longrightarrow \\
&A^{-}{:}\,bcd\ (A\backslash A)^{+}{:}\,d\ A^{+}{:}\,c\ A^{-}{:}\,a\ (A\backslash A)^{-}{:}\,h\ (A\backslash A)^{-}{:}\,l\ A^{+}{:}\,m \longrightarrow \\
&A^{-}{:}\,bcd\ A^{+}{:}\,e\ A^{-}{:}\,f\ A^{+}{:}\,c\ A^{-}{:}\,a\ (A\backslash A)^{-}{:}\,h\ (A\backslash A)^{-}{:}\,l\ A^{+}{:}\,m \longrightarrow \\
&A^{-}{:}\,bcd\ A^{+}{:}\,e\ A^{-}{:}\,f\ A^{+}{:}\,c\ A^{-}{:}\,a\ A^{+}{:}\,g\ A^{-}{:}\,hg\ (A\backslash A)^{-}{:}\,l\ A^{+}{:}\,m \longrightarrow \\
&A^{-}{:}\,bcd\ A^{+}{:}\,e\ A^{-}{:}\,f\ A^{+}{:}\,c\ A^{-}{:}\,a\ A^{+}{:}\,g\ A^{-}{:}\,hg\ A^{+}{:}\,k\ A^{-}{:}\,lk\ A^{+}{:}\,m \qquad \text{where } d = \lambda f.e
\end{aligned}
$$

Creating a sequence of axiomatic formulae from some sequent S is completely deterministic and takes a linear amount of time, with respect to the length of the sequent.

2.2 Axiomatic Linkages

To create an axiomatic linkage, match up pairs of positively and negatively polarized axiomatic formulae, X^{+} and X^{-}, i.e. with the same base category. Below is an example of a spanning linkage (spanning the whole sequent) for the above sequence of axiomatic formulae:



There can be many such linkages for a given sequence of axiomatic formulae. Any non-spanning linkage is defined as a sub-linkage and a linkage in which no two links cross is called a planar linkage (i.e. the above linkage is planar).

2.3 LC-Graphs

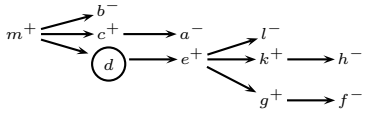
An LC-Graph for a spanning (or sub) linkage is a directed graph, $G = (V, E)$ such that V is the set of unique labels created at any point during lexical

unfolding[3] and $(v, u) \in E$ iff either,

1. v labels a category that was positively unfolded and u labels the positive category that resulted from the unfolding.

2. There is a link between categories $X \overset{+}{:} v$ and $X \overset{-}{:} u$ in the linkage that G represents. Where u may be of the form $u = u_1 u_2 \dots u_n$, in which case $(v, u_1), (v, u_2), \dots, (v, u_n) \in E$.

For example, the spanning linkage above has the following LC-Graph:



Nodes labeled with $^+$ or $^-$ are referred to as **plus-nodes** and **minus-nodes** respectively. These nodes either label a positive axiomatic formula or a negative axiomatic formula after unfolding. We define a **lambda-node** as any node that labels a positive category which was unfolded during the creation of the axiomatic formulae. For instance, the node d, in the above graph, is a lambda-node since it labels the positive category (A\A) that was unfolded. We represent these nodes by enclosing them in circles. We may also define a **lambda-minus-daughter** as any node that labels a negative category which was the direct result of a positive category being unfolded. In the above graph, f is a lambda-minus-daughter. Similarly we can define a **lambda-plus-daughter** as any node that labels a positive category which was the direct result of another positive category being unfolded. The node e is a lambda-plus-daughter in the above graph. The concept of lambda-nodes and their daughters is central to a sequent's derivability in **L**.

A planar linkage and its corresponding LC-Graph, G, are **Integral** iff G satisfies:

- **I(1)** there is a unique node in G with in-degree 0 (a unique root), from which all other nodes are path-accessible,
- **I(2)** G is acyclic,
- **I(P)** for every lambda-node $v \in V$, there is a path from its plus-daughter, u, to its minus-daughter, w, and
- **I(CT)** for every lambda-node $v \in V$, there is a path in G, $v \rightarrow ... \rightarrow u \rightarrow x$, where x is a terminal node and u is not a lambda-plus-daughter (of any lambda-node).

Penn[7] showed that a sequent is valid if and only if there exists a planar spanning linkage whose corresponding LC-Graph is integral[4].

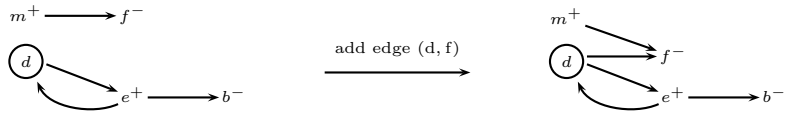
[3] V={a,b,c,d,e,f,g,h,k,l,m} for the above axiomatic formulae.

[4] The definition of an LC-Graph presented here is not exactly the same as that provided by [7]. However, it can be easily shown that, in terms of a sequent's derivability, they are equivalent.

2.4 Definitions

Definition 2.1 *An LC-Graph, G, is said to be* ***lambda-fragile*** *iff by adding edges from every lambda-node to their corresponding minus-daughters causes G to become connected (in the category of undirected graphs).*

Example:



It should be noted that all connected LC-Graphs are already lambda-fragile.

Definition 2.2 *A* ***lexical-unfolding*** *is a contiguous sequence of axiomatic formulae that can be derived from some polarized category using the lexical-unfolding rules from §2.1.*

Example: The category $((A/(A\backslash A))/A)^-\!:b$ unfolds to the following lexical-unfolding,

$$(A/(A\backslash A))^-\!:bc\ A^+\!:c \Rightarrow A^-\!:bcd\ (A\backslash A)^+\!:d\ A^+\!:c \Rightarrow \mathbf{A^-\!:bcd\ A^+\!:e\ A^-\!:f\ A^+\!:c}$$

Observe that all the axioms of a lexical-unfolding will be contiguous, and all axioms belong to only one lexical-unfolding.

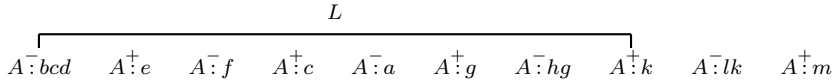
Definition 2.3 *An axiomatic formula is considered* ***left-peripheral*** *(right-peripheral) iff it exists at the left (right) endpoint of a lexical-unfolding.*

Example: $A^+\!:g$ and $A^-\!:bcd$ are left-peripheral and $A^-\!:gh$ and $A^+\!:c$ are right-peripheral in the following two lexical-unfoldings that result from unfolding the categories, $(A\backslash A)^-\!:h$ and $((A/(A\backslash A))/A)^-\!:b$

$$A^+\!:g\ A^-\!:gh\ A^-\!:bcd\ A^+\!:e\ A^-\!:f\ A^+\!:c$$

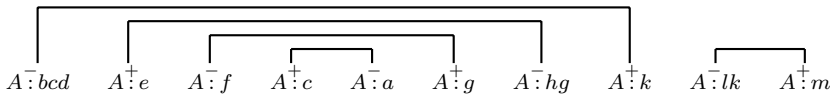
Definition 2.4 *A linkage L (sub or spanning) is said to* ***span*** *an axiomatic formula A iff A is completely enclosed by the two axioms that the L connects.*

Example: The linkage L spans $A^+\!:e$, $A^-\!:f$, $A^+\!:c$, $A^-\!:a$, $A^+\!:g$ and $A^-\!:hg$.



Definition 2.5 *An axiomatic formula X in a linkage L is considered* ***exposed*** *iff in L, there is no linkage that spans X.*

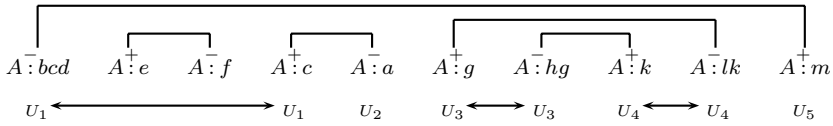
Example: In the following linkage $A^{-}\!:\!bcd$, $A^{+}\!:\!k$, $A^{-}\!:\!lk$ and $A^{+}\!:\!m$ are exposed:



We will further define $A^{-}\!:\!bcd$ and $A^{-}\!:\!lk$ as **left-exposed** and $A^{+}\!:\!k$ and $A^{+}\!:\!m$ as **right-exposed**.

Definition 2.6 *A **closed-linkage** of some set of lexical-unfoldings, U, is a linkage that uses only and all the axioms in U. A sublinkage may contain a closed-linkage.*

Example: Consider the sequent below with the following lexical-unfoldings, U_1, U_2, U_3, U_4 and U_5. The linkage below contains two closed-linkages, $U' = \{U_1, U_2, U_5\}$ and $U'' = \{U_3, U_4\}$



2.5 Simplifying LC-Graphs

At this point we consider three new integrity criteria for LC-Graphs:

- **I(C)** G is connected
- **I(LF)** G is lambda-fragile
- **I(R)** G has a unique root node (node with in-degree of 0)[5]

Proposition 2.1 *If an LC-Graph satisfies I(R), then I(1), I(2) and I(C) are equivalent.*

Proof. I(C) ⇒ I(2): Follows from the fact that all nodes in an LC-Graph have a maximum in-degree of 1. I(1) ⇒ I(C): Follows directly from the definition of I(1). I(2) ⇒ I(1): Given by [7] proposition 5.1. ■

Proposition 2.2 *I(P) and I(LF) imply I(C).*

Proof. Consider some lambda-node d, and it's corresponding plus and minus-daughters e and f. By I(P), there is a path from e to f. Furthermore, by the definition of an LC-Graph, there is a path from d to e. Therefore there is a path from d to f. So adding another edge (d, f) to the graph has no bearing on that graphs connectivity. Therefore the graph must already be connected. ■

[5]Note we don't require that the root node have a path to all other nodes as is the case with I(1)

Proposition 2.3 *A sequent S is derivable in L iff there exists a planar spanning linkage, whose corresponding LC-Graph satisfies I(R), I(LF), I(P) and I(CT).*

Proof. Consequence of propositions 2.1 and 2.2 ■

Where I(R) can be easily checked during lexical unfolding by ensuring the existence of exactly one consequent in the sequent, leaving only I(LF), I(P) and I(CT) needing to be enforced.

3 Enforcing Lambda-Fragility

Penn[7] displayed an algorithm that used a chart parser to incrementally create all the possible planar spanning linkages for a given sequent. Chart parsers were designed for CFGs, in which the use of non-terminals on RHSs is invariant over their specific LHS derivations. Therefore each edge in the chart could correspond to possibly many different sublinkages. Penn also showed how to store LC-Graphs on each edge so that each graph corresponds to some linkage represented by that edge. A sequent's derivability may then be determined by ensuring that at least one of the LC-Graphs on the spanning edge is integral. However, because an edge may represent an exponential number of linkages, it must then also store an exponential number of LC-Graphs.[6]

Here an extension to this algorithm is presented that enforces lambda-fragility. In other words it forces each LC-Graph associated with the spanning edge in the chart to be lambda-fragile.

Proposition 3.1 *All nodes $a_1, \ldots, a_n$ that represent labels in the same lexical unfolding will exist in the same connected component (in the undirected sense) for any spanning lambda-fragile LC-Graph.*

Proof. By induction on the number of unfoldings to create the lexical-unfolding (see lexical unfolding rules in §2.1). A single unfolding will either result in two axiomatic formula with a common label (negative unfolding) or two axiomatic formula that are the positive and negative daughters of some lambda node (positive unfolding). In the former case, the common label will ensure that all the labels are in the same connected component and in the latter case, the edges from the lambda-node to its plus and minus daughters will force all labels to be in the same component.

Inductively assume after n unfolding steps a lexical unfolding consits of m different unfolded categories (possibly not axiomatic). The $n + 1^{st}$ unfolding step must be on only one of these categories. By a similar analysis

[6] This approach is similar to the parsing algorithm presented by Morrill[6]. Where Penn stores possible LC-Graphs over spans, Morrill stores possible unifications over spans.

to the base case all the labels of the two new categories created will be in the same connected component. Hence by induction, all of the labels for the lexical-unfolding will be in the same component. ■

Proposition 3.2 *A spanning LC-Graph, G, for spanning linkage L is not lambda-fragile iff L contains a closed-linkage.*

Proof. Assume some LC-Graph G is not lambda-fragile. By proposition 3.1 all the labels in a single lexical-unfolding are in the same connected component. So if G is not lambda-fragile, then there are at least two connected components and each will consist of all and only a set of labels for some subset of lexical unfoldings. This component can only have been created by a closed-linkage (since links correspond to edges in LC-Graphs).

Conversely assume that L contains a closed-linkage over some subset of lexical-unfoldings U and that G is lambda-fragile. By proposition 3.1 we know that all the labels of some unfolding in U will be in the same connected component. Also, since each link creates an edge in the LC-Graph, it must be the case that all the labels of all the axioms in U will be in the same connected component. And, since there are no other links to lexical-unfoldings outside of U, then it also must be the case that there are no edges from any node in the component of the labels of U to nodes outside of it. Hence, this component is disconnected from the rest of the graph and G cannot be lambda-fragile since adding an edge from a lambda-node to a minus-daughter can only connect two nodes in the same lexical-unfolding. ■

With proposition 3.2 in mind we would like each edge to store some kind of periphery information such that it forces the parser to never add an edge to the chart that can only be created using a sub-linkage that contains a closed-linkage.

To do this the chart parser will use the following six rules[7]:

1) $L \rightarrow B\ L$
2) $B \rightarrow X^-\ L\ X^+$, for every basic category X
3) $B \rightarrow X^+\ L\ X^-$, for every basic category X
4) $B \rightarrow X^-\ X^+$, for every basic category X
5) $B \rightarrow X^+\ X^-$, for every basic category X
6) $L \rightarrow B$

Using these six rules and by following the procedures outlined by Penn[7, sec. 7] it is possible to store LC-Graphs on each edge in the chart that correspond to the different sublinkages that that edge represents. However, as stated earlier, this may result in an exponential number of LC-Graphs. In order to keep the amount of information polynomial in size, we will drop

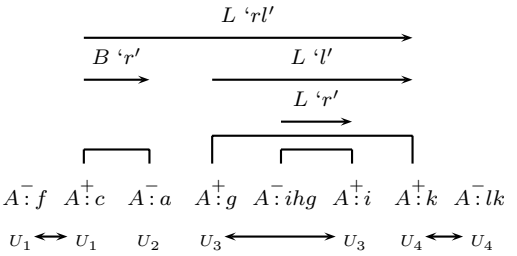
[7]These are the rules Penn[7] uses in his algorithm.

the idea of keeping LC-Graphs on the edges[8]. Instead we will store a small amount of information on each edge that will allow us to claim that if a spanning edge is created, then there exists at least one spanning linkage whose corresponding LC-Graph is lambda-fragile. The idea here is to take the first step towards storing just enough information so that one can answer the yes/no derivability question after parsing.

The information that will be stored on each edge are the following strings:

- **'l'**: there is a left-peripheral axiom left-exposed in the sublinkage this edge represents.
- **'r'**: there is a right-peripheral axiom right-exposed in the sublinkage this edge represents.
- **'rl'**: a combination of the two cases above, with all right-peripheries existing to the left of all left-peripheries.
- **'nil'**: there are no exposed peripheries in the sublinkage this edge represents.

Example: Consider the partial sequent with lexical-unfoldings, U_1, U_2, U_3 and U_4. Below is an example of a sub-linkage with the final edge having both left and right-peripheral exposed axioms, $A^{-}\!:a$ and $A^{+}\!:g$:



In order to use this new information some additional steps must be taken as the chart-parser uses each rule.

Rules 2 & 4: If axiomatic formula X^{-} is left-peripheral or axiomatic formula X^{+} is right peripheral then B will store 'l' or 'r' accordingly. However, if X^{-} is left-peripheral and X^{+} is right-peripheral then do not add B to the chart unless B will be a spanning edge. B stores nothing otherwise. There is never a case when B will store an 'rl'. Note: Rule 2 pays no attention to what L is storing.

Rules 3 & 5: are symmetric to rules 2 & 4.

Rule 6: L stores whatever B is storing.

Rule 1: The following table outlines what L_{left} (the left-hand-side non-terminal) stores for all cases:

[8]The number of edges stored for chart parsing is known to be polynomial in size for a CFG.

B	$\mathbf{L}_{right}$	$\mathbf{L}_{left}$
'nil'	'nil','r','l' or 'rl'	'nil','r','l' or 'rl' respectively
'l'	'nil','l'	'l'
'l'	'r','rl'	do not add to chart
'r'	'nil','r'	'r'
'r'	'l','rl'	'rl'
'rl'	*	cannot happen (B never stores 'rl')

Observe that we do not add any edge to the chart that adjoins a left-exposed left-periphery to a right-exposed right-periphery. Also, we stipulate that if L_{left} is the spanning edge, then it is always added to the chart.

In order to make the number of edges polynomial in size we will union edge periphery information. If two edges of the same type (B or L) for the same span are created, E_1 and E_2, we will replace them with a third edge E_3 that stores the union of the periphery information stored in E_1 and E_2. For example, if E_1 is storing {'l','rl'} and E_2 is storing {'nil','l'} then E_3 will store {'l','rl','nil'}.

Edge union causes no change in the behaviour of parsing rules 2-6 due to the uniqueness of B-edges for a given span. Parsing rule 1 does need to be updated so that the resulting edge L_{left} stores all the possible periphery outcomes when adjoining edges B and L_{right}. This can easily be done in constant time by considering the bounded number of finite possibilities.

Proposition 3.3 *An edge E that is stored on the chart has a right-exposed right-peripheral axiom iff it is storing either 'r' or 'rl' (similarly for left-exposed left-peripheral axioms).*

Proof. Assume edge E has a right-exposed right-peripheral axiom. If the edge is a B-edge, then it can only have a right-exposed right-peripheral axiom at its rightmost axiom. By inspection of the augmented parsing rules 2-5 it is can be seen that E will store an 'r'. Say that E is an L-edge. If it was created by rule 6, then its structure will be identical to some B-edge and must store an 'r'. If it was created by rule 1, then it is the adjunction of some B-edge and some L-edge. Inductively we can assume that if one of these edges contain a right-exposed right-peripheral axiom then they will be storing 'r' or 'rl'. Then, by inspecting the augmented parsing rule 1, it can be seen that E will store an 'r' or an 'rl' depending on the structure of the B and L-edge it is being created from.

Assume E is storing either an 'r' or an 'rl'. If the edge is a B-edge, then by inspection of the augmented parsing rules 2-5 it can be seen that E will store an 'r' iff its rightmost axiom is a right-periphery (and hence right-exposed). Say that E is an L-edge. If it was created by rule 6, then its structure will be identical to some B-edge and so will store an 'r' only if its rightmost axiom is a right-periphery. If it was created by rule 1, then it is the adjunction of some B-edge and some L-edge. By inspection of the

parsing rules, E will store an 'r' or an 'rl' iff at least one of the two adjoining edges are storing an 'r' or 'rl'. Inductively we can assume that since one of these edges is storing an 'r' or an 'rl' then it contains a right-exposed right-peripheral axiom and therefore so will E. ■

Proposition 3.4 *An edge E, for a sub-linkage SL is added to the chart iff it is not the case that SL contains a closed-linkage for some set of lexical-unfoldings U.*

Proof. Assume an edge E for a sub-linkage SL is added to the chart. By an examintion of the augmented parsing rules above, the only means by which an edge cannot be added to the chart is if it can only be created over a sub-linkage that has a left-exposed left-periphery axiom to the left of some right-exposed right-periphery axiom. This is precisely when a closed-linkage occurs and therefore SL does not contain a closed-linkage.

Conversely, assume that SL does not contain a closed-linkage. Again, there cannot be a left-exposed left-periphery axiom to left of a right-exposed right-periphery axiom (otherwise there would be a closed-linkage). So again, E will be added to the chart.

Proposition 3.5 *A spanning edge is added to the chart iff at least one of its representative LC-Graphs is lambda-fragile.*

Proof. A consequence of 3.2 and 3.4. ■

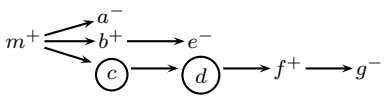
After the parsing algorithm is run, it can be said whether or not there exists a lambda-fragile LC-Graph for some spanning linkage. Since all integral LC-Graphs are lambda-fragile, then this algorithm recognizes all valid sequents. Furthermore, if we stored LC-Graphs on chart edges in conjunction with periphery information, then all lambda-fragile graphs will be stored on the spanning edge and all non-lambda-fragile graphs discarded. Hence, the algorithm does not throw away any valid parses that may become useful in methods to efficiently enforce I(P) and I(CT) - if such methods exist.

We can also identify the precise class of invalid sequents that are being recognized by this algorithm - those that have a planar linkage/LC-Graph pair that satisfy I(R) and I(LF), but do not satisfy at least one of I(P) or I(CT). For example, the following underivable sequent falls into this class:

$$(A/(A\backslash(A\backslash A))) : a \;\vdash\; A : m \;\Rightarrow\; A^{-} : abc\; A^{+} : f\; A^{-} : g\; A^{-} : e\; A^{+} : b\; A^{+} : m$$

$$\text{where } c = \lambda e.d \text{ and } d = \lambda g.f$$

has a planar spanning linkage with the following LC-Graph satisfying I(R) and I(LF):



4 Discussion

By storing a small amount of bounded information on each edge in a chart parser, it was shown that we could force certain properties of the LC-Graphs that are associated with those edges - namely that they are all lambda-fragile. Of course the only problem is that not all lambda-fragile LC-Graphs are integral, which means the algorithm presented here is not sound.

Future studies should focus on finding a similar method that uses sequent structure to force all LC-Graphs on the spanning edge to satisfy I(P). If this can be done in conjunction with the above algorithm, then it would be likely that I(CT) could be satisfied as well, since both integrity criteria involve ensuring the existence of particular paths.

References

[1] Carpenter, Bob. *Type-Logical Semantics*. Cambridge, MA: The MIT Press (1997).

[2] Dörre, Jochen. *Parsing for semidirectional lambek calculus is NP-Complete*, Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics (1996), pp. 95-100.

[3] Girard, J.-Y. *Linear Logic*, Theoretical Computer Science **56** (1987), pp. 1-102.

[4] Kanovich, M. *The multiplicative fragment of linear logic is NP-complete*, Technical Report X-91-13, Institute for Language, Logic and Information (1991).

[5] Lincoln, P., Mitchell, A., Scedrov, A. and Shankar, N. *Decision problems for propositional linear logic*, In Proceedings 31st Annual IEEE Symposium on Foundations of Computer Science (1990).

[6] Morrill G. *Memoisation of categorial proof nets: parallelism in categorial processing*, Technical Report LSI-96-24-R, Dept. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Datlunya (1996).

[7] Penn, Gerald. *A Graph theoretic approach to sequent derivability in the lambek calculus*, Electronic Notes in Theoretical Computer Science **53** (2001).

[8] Roorda, Dirk. "Resource Logics: Proof Theoretic Investigations" Ph.D. Thesis, Universiteit van Amsterdam (1991).

[9] Urquhart, A. *The undecidability of entailment and relevant implication*, Journal of Symbolic Logic, **49** (1990), pp. 1059-1073.

Acknowledgments

The author would like to thank Gerald Penn for donating much of his valuable time to provide many useful discussions and insights on the presented work. This work was supported by a grant from the Natural Sciences and Engineering Reasearch Council of Canada.

A Category Theoretic Method for Comparing Evolutionary Computation Techniques via Their Representation

Boris Mitavskiy
Department of Mathematics
University of Michigan
Ann Arbor, MI, 48109
bmitavsk@umich.edu

Abstract. In the current paper we use category theory to build a rigorous foundation for comparing various evolutionary computation techniques via their representation. Moreover, the best possible coarse graining of a given search algorithm to make it embeddable into a binary semi-genetic and/or a binary genetic algorithm is constructed in terms of the left adjoint to the corresponding forgetful functor.

1 Introduction

Over the past 25 years evolutionary algorithms have been widely exploited to solve various optimization problems. The aim of this paper is to develop a rigorous mathematical foundation to compare the representations of various evolutionary computation techniques. The importance of this problem is emphasized in the introduction to chapter 17 of [9] in [10] and in [8]. To accomplish this task we exploit the language of Category Theory (see [3] for a detailed exposition). In the current section we describe the notation and introduce the actual notion of what a representation is. In the next section we introduce a few special evolutionary algorithm which play an important role in this paper. In section 3 we introduce rigorous definitions and list a number of results (the proofs are omitted due to space limitations). Many of these results (proposition 3.1, Theorem 3.2, corollary 3.3 and corollary 3.4 as well as a very special case of theorem 3.7) are available with proofs in [5]. The use of universal constructions (or, equivalently, adjunctions) in connection with coarse graining issues (see definition 3.8, theorem 3.6, definition 3.10 and theorem 3.7) appears for the first time in the current paper. The proofs are available upon request from the author. Theorem 3.7 (a special case

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

of this theorem, namely theorem 20 of [5] is sufficient) implies that every Genetic Programming algorithm which uses either one-point or the uniform homologous crossover operators can be embedded into the classical binary genetic algorithm.

In order to apply an evolutionary algorithm to attack a specific optimization problem, one needs to model the problem in a suitable manner. That is, one needs to construct a search space Ω (the set whose elements are all possible solutions to the problem) together with a computable positive valued fitness function $f : \Omega \to (0, \infty)$ and an appropriate family of "mating" and "mutation" transformations. One can say, therefore, that a representation of a given problem by an evolutionary algorithm is an ordered 4-tuple $(\Omega, \mathcal{F}, \mathcal{M}, f)$ where Ω is the search space, $\mathcal{F}$ is a family of binary operations on Ω and $\mathcal{M}$ is the family of unary transformations on Ω, that is, $\mathcal{M}$ is just a family of functions from Ω to itself. Intuitively $\mathcal{F}$ is the family of mating transformations: every element of $\mathcal{F}$ takes two elements of Ω (the parents) and produces one element of Ω (the child).[1] while $\mathcal{M}$ is the family of mutations (or asexual reproductions) on Ω. For theoretical purposes it is usually assumed that $\mathcal{M}$ is ergodic in the sense that the only invariant subsets under $\mathcal{M}$ are the $\emptyset$ and the entire search space Ω. (The ergodicity assumption ensures that the Markov process modelling the algorithm is irreducible (see, for instance, [1]). A typical evolutionary algorithm works as follows: A population $P = (x_1, x_2, \ldots, x_{2m})^T$ where $x_i \in \Omega$ is selected randomly [2]. The algorithm cycles through the following stages:

Evaluation:

Individuals of P are evaluated:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{2m} \end{pmatrix} \begin{matrix} \to \\ \to \\ \vdots \\ \to \end{matrix} \begin{matrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{2m}) \end{matrix}$$

Selection:

A new population $P' = (y_1, y_2, \ldots, y_{2m})^T$ is obtained where $y_i = x_j$ with probability $\frac{f(x_j)}{\Sigma_{l=1}^{2m} f(x_l)}$.

In other words, all of the individuals of P' are these of P, and the expectation of the number of occurrences of any individual of P in P' is proportional to the number of occurrences of that individual in P times the individual's fitness value. In particular, the fitter the individual is, the more copies of that individual are likely to be present in P'. On the other hand,

[1] In general there is no reason to assume that a child has exactly two parents. All of the results in this paper are valid for the families of m-ary operations on Ω. The only reason $\mathcal{F}$ is assumed to be the family of binary transformations is to simplify the notation.

[2] Here we exploit the transpose notation to convert a column vector into a row vector to save space

the individuals having relatively small fitness value are not likely to enter into P' at all. This is designed to imitate the natural survival of the fittest principle.

Partition:

The individuals of P' are partitioned into m pairwise disjoint couples for mating according to some probabilistic rule: For instance the couples could be

$$Q_1 = (y_{i_1^1}, y_{i_2^1})^T,\ Q_2 = (y_{i_1^2}, y_{i_2^2})^T,\ \ldots,\ Q_j = (y_{i_1^j},\ y_{i_2^j})^T,\ \ldots,\ Q_m = (y_{i_1^m},\ y_{i_2^m})^T$$

Reproduction:

Replace every one of the selected couples $Q_j = (y_{i_1^j},\ y_{i_2^j})^T$ with the couples $Q' = (T_1(y_{i_1^j}, y_{i_2^j}),\ T_2(y_{i_1^j}, y_{i_2^j}))^T$ for some couple of transformations $(T_1,\ T_2) \in \mathcal{F}^2$. The couple $(T_1,\ T_2)$ is selected according to a fixed probability distribution on $\mathcal{F}^2$. This gives us a new population $P'' = (z_1,\ z_2, \ldots, z_{2m})^T$

Mutation:

Finally, with small probability we replace z_i with $F(z_i)$ for some randomly chosen $F \in \mathcal{M}$. This, once again, gives us a new population

$$P''' = (w_1,\ w_2, \ldots, w_{2m})^T$$

Upon completion of mutation start all over with the initial population P'''. The cycle is repeated a certain number of times depending on the problem. A more general and extensive description is given in [9]. A few special evolutionary algorithms are introduced in the next section.

2 Special Evolutionary Algorithms

Classical Genetic Algorithm with Masked Crossover:

Let $\Omega = \prod_{i=1}^{n} A_i$. For every subset $M \subseteq \{1,\ 2, \ldots, n\}$, define a binary operation L_M on Ω as follows:

$$L_M(\mathbf{a}, \mathbf{b}) = (x_1, x_2, \ldots, x_i, \ldots, x_n)$$

where $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_n) \in \Omega$ and $x_i = \begin{cases} a_i & \text{if } i \in M \\ b_i & \text{otherwise} \end{cases}$.

The reader will recognize L_M as a masked crossover operator with mask M.

Let $\mathcal{F} = \{L_M \mid M \subseteq \{1, 2, \ldots, n\}\}$. That is, $\mathcal{F}$ in this example is simply the family of masked crossover transformations. The probability distribution on the set $\mathcal{F}^2$ is concentrated on the pairs of the form $(L_M,\ L_{\bar{M}})$ where $\bar{M}$ denotes the complement of the set M in $\{1, 2, \ldots, n\}$. Most often the pairs are equally likely to be chosen from that diagonal-like subset.

The family of mutation transformations, $\mathcal{M}$ in this (and in all of the following examples) consists of the transformations $M_{\vec{a}} : \Omega \to \Omega$ where

$\vec{a} \in \bigcup_{S \subseteq \{1,\, 2, \ldots, n\}} \prod_{i \in S} A_i$ so that $\vec{a} = (a_{i_1},\, a_{i_2}, \ldots, a_{i_k})$ for $i_1 \leq i_2 \leq \ldots \leq i_k \in S_{\vec{a}} \subseteq \{1,\, 2, \ldots, n\}$ defined as follows: $\forall\, \mathbf{x} = (x_1,\, x_2,\, \ldots, x_n) \in \Omega$ we have $M_{\vec{a}}(\mathbf{x}) = \mathbf{y} = (y_1,\, y_2 \ldots, y_n)$ where $y_q = \begin{cases} a_q & \text{if } q = i_j \text{ for some } j \\ x_q & \text{otherwise} \end{cases}$

In other words, $M_{\vec{a}}$ simply replaces the q^{th} coordinate of its argument with $a_q \in A_i$ whenever $q \in S_{\vec{a}}$.

Binary Genetic Algorithm with Masked Crossover:

When every $A_i = \{0,\, 1\}$ (which means that $\Omega = \{0,\, 1\}^n$) in the example above, one obtains the classical binary genetic algorithm.

Binary Random Respectful Recombination

The search space Ω and the family of mating transformations $\mathcal{F}$ and the family of mutations $\mathcal{M}$ are exactly the same as these for the binary genetic algorithm with masked crossover described above. The only difference is that the probability distribution on $\mathcal{F}^2$ is now completely uniform (rather than being concentrated on the diagonal-like subset described in the classical genetic algorithm example). This particular family of transformations was introduced in [6]. For instance, if $n = 5$, $M_1 = \{2,\, 3,\, 4\}$, $M_2 = \{1,\, 3,\, 5\}$ and the pair (T_{M_1}, T_{M_2}) is selected for mating, we have, for instance,

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \longmapsto \begin{pmatrix} T_{M_1}((1,\, 0,\, 0,\, 1,\, 1),\, (1,\, 1,\, 0,\, 0,\, 1)) \\ T_{M_2}((1,\, 0,\, 0,\, 1,\, 1),\, (1,\, 1,\, 0,\, 0,\, 1)) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

This type of a binary search algorithm can be classified by the following property: If both parents have a 1 in the i^{th} position then the offspring also has a 1 in the i^{th} position. Likewise, if both parents have a 0 in the i^{th} position then the offspring also has a 0 in the i^{th} position. If, on the other hand, the alleles of the i^{th} gene don't coincide, then the i^{th} allele could be either a 0 or a 1.

The following type of algorithm may seem useless at first. Its importance will become clear in the next section when we present the binary embedding theorem which shows that the binary semi-genetic algorithm (described below) possesses an interesting universal property.

Binary Semi-Genetic Algorithm:

The search space $\Omega = \{0,\, 1\}^n$, just as in the case of the binary genetic algorithm. The family of mating transformations $\mathcal{F}$ is defined as follows: Fix $\mathbf{u} = (u_1, u_2, \ldots, u_n) \in \Omega$. Define a semi-crossover transformation $F_{\mathbf{u}} : \Omega^2 \to \Omega$ as follows: For any given pair $(\mathbf{x},\, \mathbf{y}) \in \Omega^2$ with $\mathbf{x} = (x_1,\, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1,\, y_2, \ldots, y_n)$ we have $F_{\mathbf{u}}(\mathbf{x},\, \mathbf{y}) = \mathbf{z} = (z_1, z_2, \ldots z_n) \in \Omega$ where

$$z_i = \begin{cases} 1 & \text{if } x_i = y_i = 1 \\ u_i & \text{otherwise} \end{cases}$$

In other words, $F_{\mathbf{u}}\, m$ preserves the i^{th} gene if it is equal to 1 in all of the rows of P, and replaces it with u_i otherwise. Let $\mathcal{F} = \{F_{\mathbf{u}} \,|\, \mathbf{u} \in \Omega\}$ be

the family of all semi-crossover transformations. The family of mutation transformations $\mathcal{M}$ is exactly the same as in the examples above.

Example:

With $n = 5$ and $\mathbf{u}_1 = (0, 1, 1, 0, 1)$, $\mathbf{u}_2 = (0, 1, 0, 0, 1)$ we have

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \longmapsto \begin{pmatrix} F_{\mathbf{u_1}\,2}((1,\,0,\,0,\,1,\,1),\,(1,\,1,\,0,\,0,\,1)) \\ F_{\mathbf{u_2}\,2}((1,\,0,\,0,\,1,\,1),\,(1,\,1,\,0,\,0,\,1)) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Notice, that if 1 is present in the i^{th} position of both parents, then it remains in the i^{th} position of both offsprings. There are absolutely no other restrictions, though.

In practice the choice of the search space Ω is primarily determined by the specific problem and related circumstances. The general methodology for the construction of the search spaces first appeared in the work of Radcliffe (see, for instance, [6]). In the current paper we introduce the category of evolutionary heuristic search tuples which allows us to compare various evolutionary computation techniques via their representation.

3 The Category of Heuristic 3-tuples

As we have seen in the introduction, a given evolutionary heuristic search algorithm is determined primarily by the ordered 4-tuple $(\Omega, \mathcal{F}, \mathcal{M}, f)$. In the current paper we shall only be concerned with the search space Ω, the family of mating transformations $\mathcal{F}$ and the family of mutations $\mathcal{M}$. As mentioned in the introduction, the family of mutation transformations is ergodic, meaning that the only invariant subsets under $\mathcal{M}$ is the $\emptyset$ and the entire search space Ω. This motivates the following definitions:

Definition 3.1. For a given family of m-ary operations $\mathit{\Gamma}$ on a set Ω (that is, functions from Ω^m into Ω) a subset $S \subseteq \Omega$ is invariant under $\mathit{\Gamma}$ if and only if $\forall\, T \in \mathit{\Gamma}$ we have $T(S^m) \subseteq S$. We shall denote by $\Lambda_{\mathit{\Gamma}}$ the family of all invariant subsets of Ω under $\mathit{\Gamma}$. In other words, $\Lambda_{\mathit{\Gamma}} = \{S \mid S \subseteq \Omega,\ T(S^m) \subseteq S\ \forall\, T \in \mathit{\Gamma}\}$.

Definition 3.2. A heuristic 3-tuple $\mathit{\Omega} = (\Omega, \mathcal{F}, \mathcal{M})$ is a 3-tuple where Ω denotes an arbitrary set, $\mathcal{F}$ is a family of binary operations on Ω (in other words, a family of functions from Ω^2 to Ω) and $\mathcal{M}$ is a family of unary transformations on Ω (in other words, a family of functions from Ω to itself) such that $\Lambda_{\mathcal{M}} = \{\emptyset,\ \Omega\}$.

It is easy to verify (see Proposition A1 of [4]) that the family $\Lambda_{\mathit{\Gamma}}$ is closed under arbitrary intersections and contains Ω. It then follows that for every element $x \in \Omega$ there is a unique element of $\Lambda_{\mathit{\Gamma}}$ containing x (namely the intersection of all the members of $\Lambda_{\mathit{\Gamma}}$ containing x.)

Definition 3.3. Given a heuristic 3-tuple $\Omega = (\Omega, \mathcal{F}, \mathcal{M})$, denote by S_x^{Ω} the smallest element of $\Lambda_{\mathcal{F}}$ containing x.

The following definition is a natural extension of the notion of a genetic representation function.

Definition 3.4. Given two heuristic 3-tuples $\Omega_1 = (\Omega_1, \mathcal{F}_1, \mathcal{M}_1)$ and $\Omega_2 = (\Omega_2, \mathcal{F}_2, \mathcal{M}_2)$, a morphism $\delta : \Omega_1 \to \Omega_2$ is just a function $\delta : \Omega_1 \to \Omega_2$ which respects the reproduction transformations in the following sense: $\forall\, T \in \mathcal{F}_1$ and $\forall\, \mathbf{x} = (x_1, x_2) \in \Omega^2\ \exists\, F_{\mathbf{x}} \in \mathcal{F}_2$ such that $\delta(T(x_1, x_2)) = F_{(x_1, x_2)}(\delta(x_1), \delta(x_2))$. Analogously, we must have $\forall\, M \in \mathcal{M}_1$ and $\forall\, x \in \Omega\ \exists\, H_x \in \mathcal{M}_2$ such that $\delta(M(x)) = H_x(\delta(x))$. We shall denote by $Mor(\Omega_1, \Omega_2)$ the collection of all morphisms from Ω_1 into Ω_2.

Example 3.1. In [7] many examples where a group L acts on the search space Ω so that the action commutes with the crossover scheme (see the beginning of section 3.4 of [7] up to and including theorem 5) where given. Every element of L induces a permutation π_g of Ω associated with the group action. From theorem 5 of [7] it follows that commutativity of the group action with $\mathcal{F}$ implies that every π_g is a morphism from Ω onto itself. In fact, $\pi_{g^{-1}}$ is also a morphism from Ω onto itself, so that every π_g is an isomorphism from Ω onto itself.

It is easy to see that heuristic 3-tuples and morphisms between them form a well-defined concrete category. A morphism $\delta : \Omega_1 \to \Omega_2$ provides the means for encoding the heuristic 3-tuple Ω_1 by the heuristic 3-tuple Ω_2. Unless the underlying function δ is one to one, there is some nontrivial coarse graining involved. We, therefore have a special name for these morphisms whose underlying functions are injective.

Definition 3.5. We say that a morphism $\delta : \Omega_1 \hookrightarrow \Omega_2$ is an embedding if the underlying function $\delta : \Omega_1 \to \Omega_2$ is one-to-one[3].

It can be shown that under certain technical conditions on the family of mating transformations $\mathcal{F}_2$ a given function $\delta : \Omega_1 \to \Omega_2$ is a morphism of evolutionary heuristic 3-tuples $\Longleftrightarrow$ the preimage of an invariant set under δ is invariant:

Proposition 3.1. *Let $\Omega_1 = (\Omega_1, \mathcal{F}_1, \mathcal{M}_1)$ and $\Omega_2 = (\Omega_2, \mathcal{F}_2, \mathcal{M}_2)$ denote heuristic 3-tuples with $\mathcal{F}_2$ and $\mathcal{M}_2$ being composition closed (see definition A.6 of [4]). Let $\delta : \Omega_1 \to \Omega_2$ be a function. Then δ is a morphism of heuristic 3-tuples if and only if $\forall\, S \in \Lambda_{\mathcal{F}_2}$ we have $\delta^{-1}(S) \in \Lambda_{\mathcal{F}_1}$.*

[3]these are precisely all the monomorphisms in the category of heuristic 3-tuples. In fact, one can show that the category of heuristic 3-tuples is complete (has all limits) and that forgetful functor into the category of sets preserves limits. The dual statements hold as well

It is fairly straightforward to verify that every one of the families of mating transformations involved in the examples of section 2 is composition closed.

These facts allow us to characterize all possible morphisms from various heuristic 3-tuples to the standard types described in section 2 in terms of n-tuples of invariant subsets.

Definition 3.6. Fix any heuristic 3-tuple $\Omega = (\Omega,\ \mathcal{F},\ \mathcal{M})$. We say that collection

$$\Upsilon_n = \{\mathcal{I} \mid \mathcal{I} = (I_1, I_2, \ldots, I_n)\ I_j \in \Lambda_{\mathcal{F}},\ \forall\ x,\ y \in \Omega \text{ with } x \neq y\ \exists\ 1 \leq j \leq n$$

such that either ($x \in I_j$ and $y \notin I_j$) or vise versa: ($y \in I_j$ and $x \notin I_j$)$\}$

is a family of separating n-tuples. Notice that $\Upsilon_n \subseteq (\Lambda_{\mathcal{F}})^n$.

Theorem 3.2. *Fix a heuristic 3-tuple $\Omega = (\Omega,\ \mathcal{F},\ \mathcal{M})$. We now have the following bijection $\phi : (\Lambda_{\mathcal{F}})^n \to Mor(\Omega, \mathcal{S}_n)$ (here $\mathcal{S}_n$ denotes the binary semi-genetic heuristic 3-tuple with the search space $\{0,\ 1\}^n$, see also definition 3.4 for the meaning of $Mor(\Omega, \mathcal{S}_n)$) which is defined explicitly as follows: Given an ordered n-tuple of sets from $\Lambda_{\mathcal{F}}$, call it $\mathcal{I} = (I_1, I_2, \ldots, I_n) \in (\Lambda_{\mathcal{F}})^n$ let $\phi(\mathcal{I}) = \delta_{\mathcal{I}}$ where $\delta_{\mathcal{I}}(x) = (x_1, x_2, \ldots, x_n) \in S = \{0,1\}^n$ with $x_j = \begin{cases} 1 & \text{if } x \in I_j \\ 0 & \text{otherwise} \end{cases}$ $\forall x \in \Omega$. Moreover, $\delta_{\mathcal{I}}$ is an embedding (injective) if and only if $\mathcal{I} \in \Upsilon_n$ (see definition 3.6). In other words, the restriction of ϕ to Υ_n is a bijection onto the collection of all embeddings of Ω into $\mathcal{S}_n$.*

It turns out that the conditions under which a given heuristic 3-tuple can be embedded into a binary semi-genetic heuristic 3-tuple are rather mild and naturally occurring as the following two corollaries demonstrate:

Corollary 3.3. *Given a heuristic 3-tuple $\Omega = (\Omega, \mathcal{F}, \mathcal{M})$, the following are equivalent:*

1. *Ω can be embedded into an n-dimensional semi-genetic heuristic k-tuple for some n.*
2. *$\forall\, x, y \in \Omega$ with $x \neq y$ we have either $x \notin S_y^{\Omega}$ (see definition 3.3) or vise versa: $y \notin S_x^{\Omega}$.*
3. *$\forall\, x, y \in \Omega$ with $x \neq y$ we have $S_x^{\Omega} \neq S_y^{\Omega}$. (Another way to say this, is that the map sending x to S_x^{Ω} is one-to-one.)*

Moreover, if an embedding exists for some n, then there exists one for $n = |\Omega|$. We also must have $n \geq \lceil \log_2 |\Omega| \rceil$.

Corollary 3.4. *Given a heuristic* 3*-tuple* $\Omega = (\Omega, \mathcal{F}, \mathcal{M})$*, if for every* $T \in \mathcal{F}$*,* T *is pure in the sense of [6] (in other words,* $\forall\, x \in \Omega\; T(x, x) = x$ *) then* Ω *can be embedded into a binary semi-genetic heuristic* 3*-tuple of dimension less than or equal to* $|\Omega|$*.*

Definition 3.7. We say that a heuristic 3-tuple $\Omega = (\Omega,\ \mathcal{F},\ \mathcal{M})$ is embeddable if it can be embedded into a semi-genetic heuristic 3-tuple (see example in section 2).

Remark 3.1. According to corollary 3.3, a given heuristic 3-tuple $\Omega = (\Omega,\ \mathcal{F},\ \mathcal{M})$ is embeddable if and only if $\forall\, x \neq y$ we have $S_x \neq S_y$.

Proposition 3.1, Theorem 3.2 as well as corollaries 3.3 and 3.4 appear with proofs in [5]. In what follows we use the language of Category Theory (see [3] for a detailed exposition) to extend the results of [5]. We construct "the least possible coarse graining" of a given evolutionary search algorithm to make it embeddable into a binary semi-genetic, or, better yet, to a binary genetic algorithm. The notion of the 'the least possible coarse graining" is captured precisely in terms of a universal object in an appropriate category.

Denote by $\mathcal{C}$ and by $\mathcal{C}_e$ the category of all heuristic 3-tuples and its full subcategory of the embeddable heuristic 3-tuples respectively.

The following definition gives an explicit construction of the universal embeddable binary heuristic 3-tuple $F\Omega$ for a given heuristic 3-tuple Ω:

Definition 3.8. Given a heuristic 3-tuple $\Omega = (\Omega,\ \mathcal{F}, \mathcal{M})$, let $F\Omega = (F\Omega,\ F\mathcal{F},\ F\mathcal{M})$ denote the heuristic 3-tuple obtained from Γ as follows: $F\Omega = \{S_x \mid x \in \Omega\}$ and $F\Gamma_j$ is obtained from Γ_j in the following manner:

$$F\Gamma_j = \{T \mid T : F\Omega^j \to F\Omega\ \forall\ \mathbf{u} = (u_1,\, u_2, \ldots, u_j) \in F\Omega^j$$

$$\exists\ H \in \Gamma_j \text{ and } \mathbf{x} = (x_1,\, x_2, \ldots, x_j) \text{ with } x_i \in u_i \text{ such that } T(\mathbf{u}) = S_{H(\mathbf{x})}\}$$

where $\Gamma_j = \begin{cases} \mathcal{F} & \text{if } j = 2 \\ \mathcal{M} & \text{if } j = 1 \end{cases}$. Let $\phi_\Omega : \Omega \to F(\Omega)$ be the function sending $x \in \Omega$ to S_x.

There are a number of little things needed to prove that $F(\Omega)$ is, indeed, a universal embeddable heuristic 3-tuple.

Proposition 3.5. *The function* $\phi_\Omega : \Omega \to F\Omega$*, sending* $x \in \Omega$ *to* S_x*, in the definition 3.8 is a morphism of heuristic* 3*-tuples in the sense of definition 3.2.*

It still remains to show that the morphism $\phi_\Omega : \Omega \to F\Omega$ is universal in the following sense:

Theorem 3.6. *Given any morphism of heuristic* 3*-tuples* $\alpha : \Omega \to \Phi$ *where* $\Phi = (\Phi,\ \Theta,\ \Gamma)$*, and* Φ *is embeddable,* $\exists!$ *morphism* $\bar{\alpha} : F\Omega \to \Phi$ *such that we have* $\bar{\alpha} \circ \phi_\Omega = \alpha$*.*

Notice that propositions 3.5 and 3.6 together say that the functor $F : \mathcal{C} \to \mathcal{C}_e$ is the left adjoint to the forgetful functor from $\mathcal{C}_e$ into $\mathcal{C}$. A somewhat analogous construction can be carried out for the full subcategory of the heuristic tuples which can be embedded into a binary genetic algorithm:

Definition 3.9. We say that a heuristic 3-tuple $\Omega = (\Omega, \mathcal{F}, \mathcal{M})$ is genetic if it can be embedded into a binary genetic algorithm (see example in section 2). Denote by $\mathcal{C}_g$ the full subcategory of all genetic 3-tuples.

Definition 3.10. Given a heuristic 3-tuple $\Omega = (\Omega, \mathcal{F}, \mathcal{M})$ let $T\Omega = (T\Omega, T\mathcal{F}, T\mathcal{M})$ be the heuristic 3-tuple obtained from Ω as follows: Introduce the following equivalence relation $\sim$ on Ω: $x \sim y$ if and only if x and y can not be separated by a complementary pair of invariant subsets. In other words, $\forall\, A,\, B \in \Lambda_{\mathcal{F}}$ with $A \cup B = \Omega$ and $A \cap B = \emptyset$ we have either x and $y \in A$ or x and $y \in B$. ($\sim$ is an intersection of equivalence relations, so it is also an equivalence relation.) Let $T\Omega$ be the set of equivalence classes under $\sim$. $T\mathcal{F}$ and $T\mathcal{M}$ are defined in the same way as in definition 3.8: just replace F with T everywhere in the second part of definition 3.8.

Theorem 3.7. *The functor $T : \mathcal{C} \to \mathcal{C}_g$ constructed in definition 3.10 is the left adjoint to the forgetful functor $\mathcal{C}_g \to \mathcal{C}$. The map $\eta_\Omega : \Omega \to T\Omega$ sending an element in Ω to its equivalence class under $\sim$ in $T\Omega$ is the unit of adjunction.*

Theorem 3.7 implies, in particular, that a heuristic 3-tuple $\Omega = (\Omega, \mathcal{F}, \mathcal{M})$ can be embedded into a binary genetic algorithm if and only if any $x,\ y \in \Omega$ with $x \neq y$ can be separated by a disjoint pair of invariant subsets. (According to definition 3.10 this is what it means to say that $x \nsim y\ \forall\ x \neq y$ so that every $S_x^\Omega = \{x\}$ and η_Ω is an isomorphism) It is easy to show (but the details are omitted due to space limitations) that the above scenario is satisfied by the heuristic 3-tuples corresponding to various genetic programming techniques so that they can be regarded as "sub-algorithms" of the classical binary genetic algorithm introduced by John Holland.

4 conclusions

In the current paper we have developed a category theoretic framework for comparing various evolutionary computation techniques via their representation. In particular, we have constructed the "most suitable" coarse graining of a given heuristic search algorithm to make it imbeddable into a binary semi-genetic and/or by a binary genetic algorithm in terms of universal properties (see theorems 3.6 and 3.7).

5 Acknowledgements

I want to thank my thesis advisor, Professor Andreas Blass for the numerous helpful advisor meetings which have stimulated many ideas for this and for my future work. I also want to thank Professors John Holland and Rick Riolo and the entire University of Michigan Complex Systems Group for helpful discussions. Finally I want to thank the unanimous referees for the useful comments and suggestions.

References

[1] Coffey, S. (1999) An Applied Probabilist's Guide to Genetic Algorithms. *A Thesis Submitted to The University of Dublin for the degree of Master in Science.*

[2] Liepins, G. and Vose, M. (1992). Characterizing Crossover in Genetic Algorithms. *Annals of Mathematics and Artificial Intelligence*, 5: 27 - 34.

[3] Mac Lane, S. (1971) Categories for the working mathematician. *Graduate Texts in Mathematics 5*, Springer-Verlag.

[4] Mitavskiy B. (Resubmitted). Crossover Invariant Subsets of the Search Space for Evolutionary Algorithms. *Evolutionary Computation.* http://www.math.lsa.umich.edu/∽bmitavsk/

[5] Mitavskiy B. (2003). Comparing Evolutionary Computation Techniques via Their Representation. *Proceedings of the Genetic and Evolutionary Computation (GECCO) Conference*, The Conference will be held July 12 - 16.

[6] Radcliffe, N. (1992). The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10:339-384.

[7] Rowe, J., Vose, M., and Wright, A. (2002). Group properties of crossover and mutation. *Evolutionary Computation*, 10(2): 151-184.

[8] Stephens, C. (2001). Some exact results from a coarse grained formulation of genetic dynamics. *Proceedings of the Genetic and Evolutionary Computation (GECCO) conference*, pages 631-638, Morgan Kaufmann.

[9] Vose, M. (1999). The simple genetic algorithm: foundations and theory. *MIT Press, Cambridge, Massachusetts.*

[10] Vose, M. and Wright, A. (2001). Form invariance and implicit parallelism. *Evolutionary Computation*, 9(3): 355-370.

[11] Wright, A., Rowe, J., Poli, R., and Stephens C. (2002). A fixed point analysis of a gene pool GA with mutation. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* Morgan Kaufmann. http://www.cs.umt.edu/u/wright/.

Properties of Translations for Logic Programs

Juan Antonio Navarro Pérez
Universidad de las Américas - Puebla
ma108907@mail.udlap.mx

Abstract. We present a study of some properties of program translations in the context of logic programming. In particular we provide, under the answer set semantics, a translation for arbitrary propositional theories into the simple class of disjunctive programs. We also show how syntactic and semantic properties of translations can be related. The work follows a line of research that applies mathematical logic to study notions and concepts in logic programming.

1 Introduction

Program translations are functions that map logic programs in a given class of programs to another class. Translations for logic programs can be very interesting for several different reasons: they can allow to simplify the structure of programs (Osorio et al. 2001; Pearce et al. 2002; Sakama and Inoue 1998), to derive correct programs from specifications (Pettorossi and Proietti 1998), and even to perform program updates and belief revision for agent systems (Alferes et al. 2002; Eiter et al. 2000).

We are interested in the study of the properties that translations should have in order to preserve the semantical meaning of programs. In particular we want to investigate this sort of properties in the context of Answer Set Programming (ASP). This semantic, originally introduced by Gelfond and Lifschitz (1988) and generalized later to include broader classes of programs (Lifschitz et al. 1999; Osorio et al. 2003), rapidly became a popular logic programming paradigm.

The great power of this semantic to express a wide variety of problems, and the existence of very efficient software to compute answer sets, allowed the development of several real life applications based on ASP. The possibilities range from solving combinatorial problems, modeling logic agents, planning (Dimopoulos et al. 1997), knowledge representation (Baral 2003) and querying deductive databases (Eiter et al. 1997); just to mention a few.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

Previous work from Janhunen (2001) studies several syntactic and semantic properties of translations for arbitrary semantic operators. This kind of properties are relevant for several theoretical and practical reasons, as we will discus along this presentation. Pearce et al. (2002) shows, in particular, a translation for programs containing nested expressions to the class of disjunctive programs that preserves the original semantics of programs.

In this paper we follow an approach, motivated from (Pearce 1999; Osorio et al. 2003), that tries to explain answer set programming in terms of intermediate logics. We provide a translation, with nice syntactical and semantical properties, that can reduce propositional theories to the class of nested programs considered by Pearce et al. (2002).

We also show that, under certain simple assumptions, the syntactic properties of a translation that preserves the answer set semantics can be enough to be sure that the translation can also preserve the semantics for partially translated programs. This sort of results show how the use of intuitionistic and other intermediate logics is an approach that can help us to better understand the notion and concepts of answer sets.

Because of the lack of space, the proofs of some minor results were left out from the main discussion of the paper. A complete version of the paper, including an appendix with all the omitted proofs, will be made available at `http://www.udlap.mx/~ma108907/papers.html`.

2 Background

We review in this section the language of propositional logic used to describe logic programs. We briefly introduce intuitionistic and some multivalued-logics, state some notation and basic results. A more detailed presentation can be found at (Mendelson 1987; Zakharyaschev et al. 2001). We also define several concepts for logic programming: classes of logic programs, semantic operators and answer sets. Original definitions and related results are available in (Lloyd 1987; Lifschitz et al. 1999; Osorio et al. 2003).

Propositional Logic We consider a formal language built from an alphabet containing: a denumerable set $\mathcal{L}$ of elements called atoms, the 2-place connectives $\wedge$, $\vee$, $\rightarrow$, the 0-place connective $\bot$, and auxiliary symbols (,). Formulas are constructed as usual in logic. The formula $\top$ is introduced as an abbreviation of $\bot \rightarrow \bot$, $\neg A$ as an abbreviation of $A \rightarrow \bot$, and $A \leftrightarrow B$ to abbreviate of $(A \rightarrow B) \wedge (B \rightarrow A)$. The notation $A \leftarrow B$ is just another way of writing the formula $B \rightarrow A$.

A *theory* is a set of formulas, we will restrict our attention, however, to finite theories. We use Prp to denote the set of all finite propositional theories. For a given theory T its *signature* is the set $\mathcal{L}_T$ of atoms occurring in the theory T. A *literal* is either an atom a or a negated atom $\neg a$. Given a theory T we also define the negated theory $\neg T = \{\neg A \mid A \in T\}$ and, for

a set of atoms M, the *closure* of M (w.r.t. T) as $\overline{M} = M \cup \neg(\mathcal{L}_T \setminus M)$.

Intermediate Logics Intuitionistic logic, denoted I, was developed as an alternative to classical logic. It explains the meaning of the connectives in terms of *knowledge* or *provability*, instead of absolute *truths*. We consider a formulation of intuitionistic logic where a *juantheorem* is a formula that can be proved using the ten axioms that define I and *modus ponens* as inference rule (van Dalen 1980).

Gödel also defined the multivalued logics G_i, with i truth values, where a *model* of a formula is a truth assignment to the atoms that, when propagated over its connectives, evaluates to some designated *true* value. A formula is a *tautology* if it is true for every possible truth assignment. The two valued logic G_2 corresponds to the classical logic, denoted C. (Mendelson 1987)

It was shown that the set of intuitionistic theorems is a proper subset of the set of classical tautologies, and that the logics G_i lie in between. So we call *intermediate logics* to those logics whose set of provable formulas is between intuitionistic and classical logics (inclusive). A *proper intermediate logic* is an intermediate logic that is not the classical one.

Notation and Basic Results We use the notation $\vdash_{\mathrm{X}} F$ to denote that the formula F is provable (a theorem or tautology) in logic X. If T is a theory we use the symbol $T \vdash_{\mathrm{X}} F$ to denote $\vdash_{\mathrm{X}} (F_1 \wedge \cdots \wedge F_n) \rightarrow F$ for some formulas $F_i \in T$. We say that a theory T is *consistent* if it is not the case that $T \vdash_{\mathrm{C}} \bot$. It is easy to prove that the definition of a theory of being consistent does not depend on the logic chosen. The proof is given in the appendix at the end of the full version of this document.

We also introduce, if T and U are two theories, the symbol $T \vdash_{\mathrm{X}} U$ to denote that $T \vdash_{\mathrm{X}} F$ for all formulas $F \in U$. We will write $T \Vdash_{\mathrm{X}} U$ to denote the fact that (i) T is consistent and (ii) $T \vdash_{\mathrm{X}} U$. Finally we say that two theories T_1 and T_2 are equivalent (w.r.t. logic X), denoted $T_1 \equiv_{\mathrm{X}} T_2$ if it holds that both $T_1 \vdash_{\mathrm{X}} T_2$ and $T_2 \vdash_{\mathrm{X}} T_1$.

Logic Programs In order to introduce the terminology of logic programs, using the propositional logic we have just presented, we define a *clause* as a formula $H \leftarrow B$ where principal connective is implication. The formulas H and B are known as the *head* and the *body* of the clause respectively.

The special case of a clause with the form $\bot \leftarrow B$ is known as a *constraint*, in this case the head is said to be *empty*. Each formula H that does not have implication as the principal connective can be associated with the clause $H \leftarrow \top$, this kind of clauses are known as *facts*. Then we can define a *logic program* as a finite set of clauses and a *class of logic programs* as some set of logic programs.

There are several kind of clauses defined in literature. The head and the body of *augmented clauses* are constructed from the connectives $\wedge$, $\vee$ and $\neg$ arbitrarily nested. Note that the general case of implication is not allowed. The clause $\neg(a \wedge \neg b) \leftarrow p \vee (\neg q \wedge r)$ is, for example, augmented while the

clause $a \leftarrow (b \rightarrow c)$ is not. An *augmented program* is a set of augmented clauses and Aug denotes the class of all augmented programs.

Another, more restricted, class is the disjunctive one. A *disjunctive clause* allows only a (non-empty) disjunction of positive literals in the head and a conjunction of literals in the body, for example $a \vee b \leftarrow p \wedge q \wedge \neg r$. Similarly, a *disjunctive program* is a set of disjunctive clauses and Dis is the class of all disjunctive programs. Observe that we have $\mathsf{Dis} \subset \mathsf{Aug} \subset \mathsf{Prp}$.

Answer Sets Given a class of programs C, a *semantic operator* Sem is a function that assigns to each program $P \in C$ a set of sets of atoms $M \subseteq \mathcal{L}_P$. These sets of atoms are usually some "preferred" models of the program P. One popular semantic operator is the answer sets AS operator. We consider the definition provided in (Osorio, Navarro, and Arrazola 2002) that extends the notion of an answer set to the whole set of propositional formulas.

Definition 1. If $P \in \mathsf{Prp}$ then $\mathrm{AS}(P) = \left\{M \subseteq \mathcal{L}_P \mid P \cup \neg\neg\overline{M} \Vdash_{\mathrm{I}} M\right\}$.[1]

3 Conservative Transformations

Suppose that we have a logic program P and we want to compute $\mathrm{AS}(P)$. We could simplify this task if we are able to construct some other simpler program P', such that the answer sets of P and P' are somehow related, compute $\mathrm{AS}(P')$ and recover the answer sets of the original P. It will be important that this "recovery" of the answer sets of P, knowing those of P', can be done trough a simple and efficient method. A conservative transformation is a relation between logic programs with this kind of properties.

Definition 2. Let Sem be a semantic operator defined for a class of programs C and let $P, P' \in C$. We say P' is a *conservative transformation* of P, denoted $P \xrightarrow{\mathrm{Sem}} P'$, if $\mathcal{L}_P \subseteq \mathcal{L}_{P'}$ and

$$\mathrm{Sem}(P) = \left\{M \cap \mathcal{L}_P \mid M \in \mathrm{Sem}(P')\right\} .$$

A conservative extension, as presented in (Baral 2003), is a conservative transformation where the programs also satisfy $P \subseteq P'$. Our definition is more general since it allows the program to be modified or "transformed" and not only extended. As far as we know this definition is new, and so the results presented in this section are all original.

A conservative transformation of a logic program P can, possibly, introduce new atoms (those in $\mathcal{L}_{P'} \setminus \mathcal{L}_P$) in order to achieve simplifications. But, if we ignore this newly introduced atoms, we obtain exactly the answer sets of P. We refer to this new atoms in $\mathcal{L}_{P'} \setminus \mathcal{L}_P$ as the *reserved* atoms of the transformation.

[1] The original definition uses the notation $\neg M \cup \neg\neg\widetilde{M}$, where $\widetilde{M} = \mathcal{L}_P \setminus M$, instead of the set $\neg\neg\overline{M}$, where $\overline{M} = M \cup \neg(\mathcal{L}_P \setminus M)$. It is easy to verify that the two conditions are equivalent since $\vdash_{\mathrm{I}} \neg a \leftrightarrow \neg\neg\neg a$.

It is easy to verify that conservative transformations define a transitive relation, a formal proof is included in the appendix. Much more surprising is that arrows in the notation of conservative extensions can sometimes be traveled backwards. If a logic program Q is a conservative extension of two programs P and R then, under the premise that $\mathcal{L}_P \subseteq \mathcal{L}_R$, we can also relate the answer sets of P and R by a conservative transformation.

Proposition 1. *If* $P \xrightarrow{\text{Sem}} Q$, $R \xrightarrow{\text{Sem}} Q$, *and* $\mathcal{L}_P \subseteq \mathcal{L}_R$ *then* $P \xrightarrow{\text{Sem}} R$.

Proof. Take $M \in \text{Sem}(R)$, since $R \xrightarrow{\text{Sem}} Q$ there must be $N \in \text{Sem}(Q)$ such that $M = N \cap \mathcal{L}_R$. But, since $P \xrightarrow{\text{Sem}} Q$, we have that $N \cap \mathcal{L}_P \in \text{Sem}(P)$. Observe, since $\mathcal{L}_P \subseteq \mathcal{L}_R$, that $M \cap \mathcal{L}_P = (N \cap \mathcal{L}_R) \cap \mathcal{L}_P = N \cap \mathcal{L}_P$. Thus we obtain $M \cap \mathcal{L}_P \in \text{Sem}(P)$.

Now take $M \in \text{Sem}(P)$, since $P \xrightarrow{\text{Sem}} Q$ there must be $N \in \text{Sem}(Q)$ such that $M = N \cap \mathcal{L}_P$. But, since $R \xrightarrow{\text{Sem}} Q$, we have that $N \cap \mathcal{L}_R \in \text{Sem}(R)$. Observe, since $\mathcal{L}_P \subseteq \mathcal{L}_R$, that $(N \cap \mathcal{L}_R) \cap \mathcal{L}_P = N \cap \mathcal{L}_P = M$. Then $N \cap \mathcal{L}_R \in \text{Sem}(R)$ is the model such that $(N \cap \mathcal{L}_R) \cap \mathcal{L}_P \in \text{Sem}(P)$. □

The following proposition allows us to construct a very simple conservative translation for the semantic of answer sets. It can be used to extend the language of a given program without modifying its answer sets. This result will be used later in the proof of Theorem 3.

Proposition 2. *Given a set of atoms* A, *let* $L = \{a \leftarrow a \mid a \in A\}$. *For any program* $P \in \mathsf{Prp}$, $P \xrightarrow{\text{AS}} P \cup L$.

Another particular case of a conservative transformation, which is presented in the following proposition, stands that is possible to introduce new atoms as a definition of a formula that can be expressed using the atoms already in the program.

Proposition 3. *Let* $P \in \mathsf{Prp}$. *Given a formula* F *such that* $\mathcal{L}_F \subseteq \mathcal{L}_P$ *and an atom* $x \notin \mathcal{L}_P$, $P \xrightarrow{\text{AS}} P \cup \{x \leftrightarrow F\}$.

Conservative transformations may seem very effective in the context of logic programming. But they dot not satisfy, in general, an important property for concrete programming applications. We would expect that making a conservative transformation of one piece of a program would also result, "globally", in a conservative transformation for the whole program.

This is not true for simple conservative transformations as just defined. Consider the two programs $P_1 = \{a \leftarrow \neg b\}$ and $P_2 = \{a,\ b \leftarrow b\}$. According to Definition 2, P_2 is a conservative translation of P_1 in the answer set semantics since they both have $\text{AS}(P_1) = \text{AS}(P_2) = \{\{a\}\}$. However, replacing P_1 with P_2 in the larger program $P = \{a \leftarrow \neg b,\ b\}$, to obtain the

program $P' = \{a,\ b \leftarrow b,\ b\}$, will break this relation. Now P has only one answer set $\{b\}$, while P' has the answer set $\{a, b\}$.

In order to ensure that making local transformations of code inside logic programs will preserve global equivalence, we introduce the notion of a strong conservative transformation.

Definition 3. Let Sem be a semantic operator for a class of programs C. Given two logic programs $P, P' \in C$, such that $\mathcal{L}_P \subseteq \mathcal{L}_{P'}$, we say that P' is a *strong conservative translation* of P, denoted $P \xrightarrow{\text{Sem}^*} P'$, if for every program Q, such that $\mathcal{L}_Q \cap (\mathcal{L}_{P'} \setminus \mathcal{L}_P) = \emptyset$, $P \cup Q \xrightarrow{\text{Sem}} P' \cup Q$.

The condition $\mathcal{L}_Q \cap (\mathcal{L}_{P'} \setminus \mathcal{L}_P) = \emptyset$ states that the programs Q, used to extend P, may not contain any of this reserved atoms of the transformation. In an actual implementation we could ensure this condition by defining a special set of atoms reserved for internal translations and not available to the user for writing programs. As we may expect, strong conservative translations also define a transitive relation.

In the particular case when $\mathcal{L}'_P = \mathcal{L}_P$, when there are no reserved atoms, this relation is known as a strong equivalence between programs. This notion was originally introduced in (Lifschitz, Pearce, and Valverde 2001) where the authors provide a characterization of strong equivalence for augmented programs, using the answer set semantics, in terms of the logic HT equivalent to the 3 valued logic G_3. These relations between answer sets and intermediate logics are also studied in (Navarro 2002) where the characterization of strong equivalence is revised and extended to arbitrary propositional theories.

Theorem 1. *(Lifschitz et al. 2001; Navarro 2002) Let $P, P' \in \mathsf{Prp}$, such that $\mathcal{L}_P = \mathcal{L}_{P'}$. $P \xrightarrow{\text{AS}^*} P'$ if and only if $P \equiv_{\mathrm{G}_3} P'$.*

4 Program Translations

A *translation* is a function $\mathrm{Tr}\colon C \to C'$, where C and C' are two classes of logic programs. Janhunen (2001) discusses important properties of program translations, relevant to logic programming, for arbitrary semantic operators. Particular applications for the answer set semantics are also given in (Pearce et al. 2002).

Definition 4. (Janhunen 2001; Pearce et al. 2002) Let Sem be a semantic operator for a class of programs D, a translation $\mathrm{Tr}\colon C \to C'$, where the classes $C, C' \subseteq D$ are closed under unions[2], is said to be:

polynomial if the time required to compute $\mathrm{Tr}(P)$ is polynomial with respect to the number of symbols in P.

[2] A class of programs C is *closed under unions* if $P_1, P_2 \in C$ implies that $P_1 \cup P_2 \in C$.

faithful if, for all programs $P \in C$, $P \xrightarrow{\text{Sem}} \text{Tr}(P)$.

strongly faithful if, for all programs $P \in C$, $P \xrightarrow{\text{Sem}^*} \text{Tr}(P)$.

modular if, for all programs $P_1, P_2 \in C$, $\text{Tr}(P_1 \cup P_2) = \text{Tr}(P_1) \cup \text{Tr}(P_2)$.

reductive if $C' \subseteq C$ and $\text{Tr}(P') = P'$ for all programs $P' \in C'$.[3]

The property of a translation being polynomial (P) is related with the order of complexity that an actual computer implementation of the translation should have. A faithful (F) translation can be applied to a program preserving the semantics, while a strongly faithful (S) translation can also be applied *locally* to some section of the program without altering the semantics.

The last two properties deal with the form of the translation, not with its particular semantics. If a translation is modular (M) we could split a program into several pieces and then perform the translation *piece by piece*. A reductive (R) translation maps one class of programs into some given subclass, and the programs that are already in the subclass will not been modified by the translation.

As a form of notation we say that a translation is PFM if it is simultaneously polynomial, faithful and modular. Analogously, a PSMR translation is polynomial, strongly faithful, modular and reductive. We can drop any of the letters from the notation if we are just interested in some properties.

Proposition 4. *(Pearce et al. 2002) There is a PSM translation, for the the semantic of answer sets,* $\text{AugDis}\colon \mathsf{Aug} \to \mathsf{Dis}$.

Using the machinery of logic, based on Definition 1 and results like Theorem 1, it is also possible to provide a translation of logic programs, containing arbitrary propositional formulas, into augmented programs.

Definition 5. If a formula F contains a proper subformula $A \to B$, where A and B contain no more implications, we say that $A \to B$ is a *simple embedded implication* of the formula F. We define recursively the translation $\text{PrpAug}\colon \mathsf{Prp} \to \mathsf{Aug}$ for every $P \in \mathsf{Prp}$, as follows:

(i) P contains no clause with embedded implications. Then P is already an augmented program and $\text{PrpAug}(P) = P$.

(ii) P contains a clause F with some embedded implications. Take one simple embedded implication, $A \to B$, from the formula F; and take a new atom $x \in \mathcal{L} \setminus \mathcal{L}_P$ not already in P. Let F' be the formula obtained

[3] The definition of a *modular* translation we introduce here corresponds to the one given in (Pearce et al. 2002). Janhunen (2001) presents a different definition which corresponds to *modular + reductive* (when $C' \subseteq C$ is satisfied).

by replacing the occurrence of $A \to B$ in F with the new atom x, and let P' be the program obtained by replacing F with F' in P. Also let $\Delta = \{x \wedge A \to B,\ \neg A \vee B \to x,\ x \vee A \vee \neg B\}$. We finally define $\mathrm{PrpAug}(P) = \mathrm{PrpAug}(P' \cup \Delta)$.

The recursive definition of PrpAug is well-founded since, on each recursion step, the program $P' \cup \Delta$ has one implication less than P.

Proposition 5. *The translation* PrpAug: $\mathsf{Prp} \to \mathsf{Aug}$ *is* PSMR.

Proof. The number of recursion steps required to complete the translation is exactly the number of embedded implications in the program, therefore the translation is polynomial. It is also clear, since PrpAug acts on one clause at a time and does not modify programs already in the augmented class, that the translation is modular and reductive.

To justify that the recursion step is a strong conservative transformation observe that, from Proposition 3, $P \xrightarrow{\text{AS}} P \cup \{x \leftrightarrow (A \to B)\}$. The key point is that $\{x \leftrightarrow (A \to B)\} \equiv_{\mathrm{G_3}} \Delta$ and, therefore, we also have the equivalence $P \cup \{x \leftrightarrow (A \to B)\} \equiv_{\mathrm{G_3}} P' \cup \Delta$. Using Theorem 1, and transitivity, we end up with $P \xrightarrow{\text{AS}} P' \cup \Delta$. □

Current implementations of the answer set programming paradigm restrict the syntax of formulas to the class of disjunctive programs where, in particular, implication in the body is not allowed. A common work around to this limitation was to use the intuitive equivalence $(A \to B) \leftrightarrow (\neg A \vee B)$. This practice, however, caused sometimes the appearance of unexpected models (or the miss of expected ones) when computing answer sets.

The first rule $x \wedge A \to B$ in our equivalence is used to model the behavior of the implication symbol in the head. The second rule $\neg A \vee B \to x$ comes from the classical intuitive meaning of the implication connective. This two rules, however, are not enough to provide the required equivalence in the logic $\mathrm{G_3}$. This could possibly explain the unexpected results obtained from the erroneous work around. The less intuitive third rule $x \vee A \vee \neg B$ required was discovered, in fact, by an examination of the $\mathrm{G_3}$ models of the original formula $x \leftrightarrow (A \to B)$. This points out the importance of results like Theorem 1 that allows us to better understand the notion of answer sets, proposing the logic $\mathrm{G_3}$ as a more correct guide for our intuition.

5 Properties of Translations

In the previous section we provided a translation for the class of propositional theories into the class of augmented programs and, together with other translations (Pearce et al. 2002), it is possible to reduce them to the class of disjunctive programs. Using any of the popular answer set finders for

disjunctive programs, `dlv`[4] or `smodels`[5], it is possible to provide an efficient method to compute answer sets for propositional theories. The following theorem is a direct consequence from Propositions 4 and 5.

Theorem 2. *There is a translation* PrpDis: $\mathsf{Prp} \to \mathsf{Dis}$*, which is PSM for the semantic of answer sets.*

Observe that the properties of strongly faithful and modular are somehow related. Both notions can be interpreted in terms of a program that has been split into several pieces. A strongly faithful translation can be applied to one of these pieces preserving the semantics of the program. On the other hand, a modular translation can also be applied "piece by piece" to the program but we do have to complete the translation for each one of the pieces. It is possible, indeed, to construct a FM translation which is not strongly faithful.

Example 1. Let C be the class of disjunctive programs that have exactly one atom in the head and zero or one atoms in the body. Also let C' be the class of disjunctive programs that have exactly two atoms in the head and zero or two atoms in the body. Clearly both C and C' are closed under unions. For each atom a in the user language let a' be a new reserved atom. The translation Hide: $C \to C'$ is defined mapping each clause, a or $a \leftarrow b$, as follows:

$$\begin{array}{lcl} \mathrm{Hide}(\{a\}) & = & \{a' \vee a',\ a \vee a \leftarrow a' \wedge a'\} \\ \mathrm{Hide}(\{a \leftarrow b\}) & = & \{a' \vee a' \leftarrow b' \wedge b',\ a \vee a \leftarrow a' \wedge a',\ b \vee b \leftarrow b' \wedge b'\} \end{array}$$

It is clear, by construction, that the translation is modular. The translation is also faithful since it rewrites the original program, with new reserved atoms, and appending a set of rules, equivalent to $a \leftarrow a'$, in order to recover answer sets in the original signature.

Consider the program $P = \{a \leftarrow b,\ b\} \in C$. Both P and $\mathrm{Hide}(P)$ have the same answer sets. If we apply the translation, however, just to the first clause the program $\mathrm{Hide}(\{a \leftarrow b\}) \cup \{b\}$ will have different semantics. The rule $a \leftarrow b$ is now "hidden" and the existence of the fact b can not be used anymore to infer a. The translation is not strongly faithful.

We will see, however, that there is a wide class of interesting and useful translations where the syntactic properties of being modular and reductive are enough to be strongly faithful. The following theorem shows how, in the context of the answer sets semantics, this can be possible.

Theorem 3. *Given two classes of logic programs C and C', closed under unions and such that* $\mathsf{Dis} \subseteq C' \subseteq C \subseteq \mathsf{Prp}$*. If the translation* Tr: $C \to C'$ *is FMR in the answer set semantics then it is also strongly faithful.*

[4] `http://www.dbai.tuwien.ac.at/proj/dlv/`
[5] `http://www.tcs.hut.fi/Software/smodels/`

Proof. Let $Q \in \mathsf{Prp}$ be a program containing no atoms from $\mathcal{L}_{\mathrm{Tr}(P)} \setminus \mathcal{L}_P$ and let $L = \{a \leftarrow a \mid a \in \mathcal{L}_{\mathrm{Tr}(P)}\}$, so that the signatures are $\mathcal{L}_P \subseteq \mathcal{L}_{\mathrm{Tr}(P)} = \mathcal{L}_L$. Construct then the disjunctive program $D = \mathrm{PrpDis}(Q \cup L)$ and, since Theorem 2 states that the translation is strongly faithful, $Q \cup L \xrightarrow{\mathrm{AS}^*} D$.

Neither P nor $\mathrm{Tr}(P)$ contain reserved atoms from $\mathcal{L}_D \setminus \mathcal{L}_{Q \cup L}$. Using the result from Proposition 2 and from the definition of strongly faithful we obtain that $P \cup Q \xrightarrow{\mathrm{AS}} P \cup (Q \cup L) \xrightarrow{\mathrm{AS}} P \cup D$ and, similarly for the translated program, $\mathrm{Tr}(P) \cup Q \xrightarrow{\mathrm{AS}} \mathrm{Tr}(P) \cup (Q \cup L) \xrightarrow{\mathrm{AS}} \mathrm{Tr}(P) \cup D$.

Now, since the translation Tr is faithful, we have $P \cup D \xrightarrow{\mathrm{AS}} \mathrm{Tr}(P \cup D)$. Also, from the modular and reductive properties, we obtain $\mathrm{Tr}(P \cup D) = \mathrm{Tr}(P) \cup \mathrm{Tr}(D) = \mathrm{Tr}(P) \cup D$. Thus we get $P \cup Q \xrightarrow{\mathrm{AS}} P \cup D \xrightarrow{\mathrm{AS}} \mathrm{Tr}(P) \cup D$ and, by Proposition 1, we can finally conclude $P \cup Q \xrightarrow{\mathrm{AS}} \mathrm{Tr}(P) \cup Q$. □

6 Conclusions

We have presented in this paper a translation that can be used to map arbitrary propositional theories into the class of simple disjunctive programs. Moreover, the translation is strongly faithful and has good syntactical properties. Previous work, from Pearce et al. (2002), presented a similar translation but only for programs in the augmented class. Using results from Lifschitz et al. (2001) and Osorio et al. (2003) we could show how to extend this translation for any propositional theory.

The existence of such translation has some theoretical significance. It shows, in particular, that the class of disjunctive programs is as expressive as the class of propositional theories. Also an important practical consequence of the result is that it allows the development of software tools to compute answer sets for logic programs containing arbitrary propositional formulas.

We also exhibit how, for a wide range of program translations, the conditions of being faithful, modular and reductive are sufficient to state that the translation is also strongly faithful. The main assumption required for this result to hold is that the translation takes programs from a class of logic programs to some, more simple in principle, subclass.

It is important to stress out the fact that this results were obtained through the "Logic Programming via Logic" approach initiated by Pearce (1999) and developed later by Osorio et al. (2003). Thus showing how the use of intermediate logics in the study of answer sets can be useful to develop the theory and applications of answer sets. This could open a new line of research and provide a lot of feedback between this two areas.

Acknowledgments This research is sponsored by the Mexican National Council of Science and Technology, CONACyT (project 37837-A).

References

Alferes, J. J., L. M. Pereira, H. Pryzmusinska, and T. Prymusinski (2002). LUPS—a language for updating logic programs. *Artificial Intelligence 138*, 87–116.

Baral, C. (2003). *Knowledge Representation, reasoning and declarative problem solving with Answer Sets*. Cambridge: Cambridge University Press.

Dimopoulos, Y., B. Nebel, and J. Koehler (1997). Encoding planning problems in non-monotonic logic programs. In *Proceedings of the Fourth European Conference on Planning*, pp. 169–181. Springer-Verlag.

Eiter, T., M. Fink, G. Sabbatini, and H. Tompits (2000). On updates of logic programs: Semantics and properties. Research Report 1843-00-08, INFSYS. A shortened version of this paper appears in *Theory and Practice of Logic Programming*, 2002.

Eiter, T., N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello (1997, June). The architecture of a disjunctive deductive database system. In M. Falaschi, M. Navarro, and A. Policriti (Eds.), *Proceedings Joint Conference on Declarative Programming (APPIA-GULP-PRODE '97)*, pp. 141–151.

Gelfond, M. and V. Lifschitz (1988). The stable model semantics for logic programming. In R. Kowalski and K. Bowen (Eds.), *5th Conference on Logic Programming*, pp. 1070–1080. MIT Press.

Janhunen, T. (2001, September). On the effect of default negation on the expressiveness of disjunctive rules. In T. Eiter, W. Faber, and M. Truszczynski (Eds.), *Logic Programming and Nonmonotonic Reasoning, 6th International Conference*, Number 2173 in Lecture Notes in Computer Science, Vienna, Austria, pp. 93–106. Springer.

Lifschitz, V., D. Pearce, and A. Valverde (2001). Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2*, 526–541.

Lifschitz, V., L. R. Tang, and H. Turner (1999). Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25*, 369–389.

Lloyd, J. W. (1987). *Foundations of Logic Programming* (Second ed.). Berlin: Springer.

Mendelson, E. (1987). *Introduction to Mathematical Logic* (Third ed.). Belmont, CA: Wadsworth.

Navarro, J. A. (2002, August). Answer set programming through G_3 logic. In M. Nissim (Ed.), *Seventh ESSLLI Student Session, European Summer School in Logic, Language and Information*, Trento, Italy.

Osorio, M., J. A. Navarro, and J. Arrazola (2001, November). Equivalence in answer set programming. In A. Pettorossi (Ed.), *Logic Based Program Synthesis and Transformation. 11th International Workshop, LOPSTR 2001*, Number 2372 in LNCS, Paphos, Cyprus, pp. 57–75. Springer.

Osorio, M., J. A. Navarro, and J. Arrazola (2002). A logical approach for A-Prolog. In R. de Queiroz, L. C. Pereira, and E. H. Haeusler (Eds.), *9th Workshop on Logic, Language, Information and Computation (WoLLIC)*, Volume 67 of *Electronic Notes in Theoretical Computer Science*, Rio de Janeiro, Brazil, pp. 265–275. Elsevier Science Publishers.

Osorio, M., J. A. Navarro, and J. Arrazola (2003). Applications of intuitionistic logic in answer set programming. Accepted to appear at the TPLP journal.

Pearce, D. (1999). Stable inference as intuitionistic validity. *Logic Programming 38*, 79–91.

Pearce, D., V. Sarsakov, T. Schaub, H. Tompits, and S. Woltran (2002, August). A polynomial translation of logic programs with nested expressions into disjunctive logic programs: Preliminary report. In P. J. Stuckey (Ed.), *Logic Programming. 18th International Conference, ICLP 2002*, Number 2401 in LNCS, Copenhagen, Denmark, pp. 405–420. Springer.

Pettorossi, A. and M. Proietti (1998). Transformation of logic programs. In C. J. H. D. M. Gabbay and J. A. Robinson (Eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming*, Volume 5, Chapter D, pp. 697–787. Oxford University Press.

Sakama, C. and K. Inoue (1998). Negation as failure in the head. *Journal of Logic Programming 35(1)*, 39–78.

van Dalen, D. (1980). *Logic and Structure* (Second ed.). Berlin: Springer.

Zakharyaschev, M., F. Wolter, and A. Chagrov (2001, December). Advanced modal logic. In D. M. Gabbay and F. Guenthner (Eds.), *Handbook of Philosophical Logic* (Second ed.), Volume 3, pp. 83–266. Dordrecht: Kluwer Academic Publishers.

Worst-case upper bounds for SAT: automated proof

Sergey Nikolenko
St.-Petersburg State University
sergey@logic.pdmi.ras.ru

Alexander Sirotkin
St.-Petersburg State University
sirotkin@hotbox.ru

Abstract. Propositional satisfiability (SAT), an NP-hard problem, can certainly be solved in less than 2^K steps, where K denotes the number of clauses in the input formula, for instance, with the help of DPLL-type SAT solvers. In this paper, describing a work in progress, we deal with automated proofs of upper bounds on the running time of DPLL-type algorithms for SAT.

For a class of algorithms working with the pure literal elimination, we prove a previously unknown upper bound w.r.t. the number of clauses in the input formula. We also note that the employed line of reasoning cannot lead to better results.

1 Introduction

The problem of propositional satisfiability is known to be NP-complete, and, therefore, it is highly unlikely that a polynomial-time algorithm for SAT will be discovered. Therefore, the need in "good" exponential-time algorithms arose. The naive algorithm would in the general case take at least $poly(N)2^N$ steps to find a satisfying assignment, where N is a complexity measure such as number of variables, clauses or length of the input formula. Therefore, it has for quite a long time been a challenging problem to prove better and better worst-case upper bounds for various SAT solvers. The current record holder is due to Edward A. Hirsch. He proved in [6] that his (rather complex) algorithm works for the time $2^{0.30897\ldots K}$, where K is the number of clauses in the input formula. We prove a weaker bound, but for a much simpler algorithm.

One of the two main approaches to obtaining less-than-2^N bounds is the local search approach, when an algorithm starts from a random assignment

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

and modifies it to get closer to a satisfying assignment (for references see, for example, [1]).

Algorithms of the other are called DPLL-type, due to the works of Davis, Putnam, Logemann and Loveland ([3], [2]). The DPLL-type algorithms build a tree, the so-called *branching tree*, whose nodes are partial assignments to the variables of the input formula. On each step, a DPLL-type algorithm first tries to simplify the input formula, employing a number of *simplification rules*. Then, if the result is not obvious yet, it makes a *splitting*, that is, it chooses a literal l in the input formula, constructs two formulae, one corresponding to $l := true$ and the other to $l := false$, and recursively calls itself for these two formulas. If any of the calls returns the answer "Satisfiable", the algorithm also returns this answer. Otherwise, it returns "Unsatisfiable".

During the past decade the proofs of worst-case upper bounds for DPLL-type SAT solvers have become quite similar. The point of such proofs is to construct the branching tree, described in Section 2, and prove that in any possible case of the literals' mutual arrangement in the input formula the algorithm in question will be able to pick a literal, the splitting by which would provide the necessary branching vector with respect to the necessary complexity measure.

The proofs have become less elegant and more technical. Simple combinatorial properties of all the possible cases usually comprise the whole proof (probably [6] is the best example of such a proof). The only "creative" element of these proofs are the elimination rules.

In the sequel we present a successful attempt to automate the proofs of worst-case upper bounds for DPLL-type algorithms. Attempts to prove such bounds usually employ the Kullmann-Luckhardt's lemma, to which Section 2 is devoted. Sections 3 describes the ideology of automating these proofs, Section 4 provides a hard instance for this line of reasoning that matches with the upper bound presented, and Section 5 presents a new bound proven with the help of a computer, the complete output of whose work is presented in the appendix.

2 Basic definitions and Kullmann-Luckhardt's lemma

We denote by X a set of boolean variables. The negation of a variable x is denoted by $\overline{x}$. If $U \subseteq X$, then $\overline{U} = \{\overline{x} \mid x \in U\}$. *Literals* are members of the set $X \cup \overline{X}$. A *clause* is a set of literals that does not contain simultaneously any variable together with its negation. A *formula in CNF* is a finite set of clauses. A clause is called *unit* if it consists of one literal.

We count the number of appearances of a chosen literal in the formula. We call l an (i, j)-*literal* with respect to a formula F, if F contains i appearances of l and j appearances of $\overline{l}$. We call l an $(i, j+)$-*literal* with respect to

a formula F, if F contains i appearances of l and at least j appearances of $\overline{l}$. A literal is called *pure* with respect to a formula if the formula contains only the literal, but does not contain its negation.

An *assignment* is a finite subset $I \subseteq X \cup \overline{X}$ that does not contain any variable together with its negation. We denote by $F[I]$ a formula that results from F and an assignment $I = \{x_1, x_2, \ldots, x_n\}$ after removing all clauses containing the literals x_i and deleting all occurrences of the literals $\overline{x_i}$ from the other clauses. An assignment I is said to *satisfy* the formula F, if $F[I]$ is the empty formula (that is, $F[I]$ contains no clauses).

Basically, all one needs to prove worst-case upper bounds on such algorithms is to solve recurrent equations. However, the job was significantly simplified (and made available for automatization) by O. Kullmann and H. Luckhardt. The notion of a branching tree was introduced by them in [7]. The branching tree in our case is not binary, though the number of sons of a node is limited to eight. We attach to each formula (each node of the tree) a non-negative integer $\mu(F)$, which denotes the *complexity* of F. In the sequel we prove an upper bound with respect to the number of clauses in F: $\mu(F) = \mathcal{K}(F)$.

Let us now consider a node of the branching tree (labeled with a formula F) and denote its sons by $F_1, \ldots, F_m$. We call a *branching vector of a node* an m-tuple $(t_1, \ldots, t_m)$, where $t_i = \mu(F) - \mu(F_i)$. Note that the following makes sense only when each of the resulting formulae is "simpler" that its predecessor, that is, when $\mu(F) \geq \mu(F_i)$.

The *characteristic polynomial* of a branching vector $t = (t_1, \ldots, t_m)$ is defined by

$$h_t(x) = 1 - \sum_{i=1}^{m} x^{-t_i}.$$

This polymonial has exactly one positive root (since it is a monotone function on $(0, +\infty)$). Following [6], we denote this root by $\tau(t) = \tau(t_1, t_2, \ldots, t_m)$ and call it the *branching number*.

The branching numbers have quite a few remarkable properties. For our purposes two of them, presented in [7], are necessary.

Lemma 1 (Kullmann and Luckhardt [7]). *Let T be a branching tree, and let its root be labeled with a formula F. Then the number of leaves in T does not exceed $(\tau_{max,T})^{\mu(F)}$, where $\tau_{max,T}$ is the largest of all the branching numbers of the nodes of T.*

Lemma 2 (Kullmann and Luckhardt [7]). *Let $m, k \in \mathbb{N}$, $x_1, x_2, \ldots, x_m$, $y_1, y_2, \ldots, y_k \in \mathbb{Q}_+$. The problem whether $\tau(x_1, \ldots, x_m)$ is not greater than $\tau(y_1, \ldots, y_k)$ is solvable in time polynomial of* $\max(x_1, \ldots, x_m, y_1, \ldots, y_k)$.

These lemmata allow us to estimate the running time of a DPLL-type algorithm. Indeed, if we know the branching number of its splitting tree

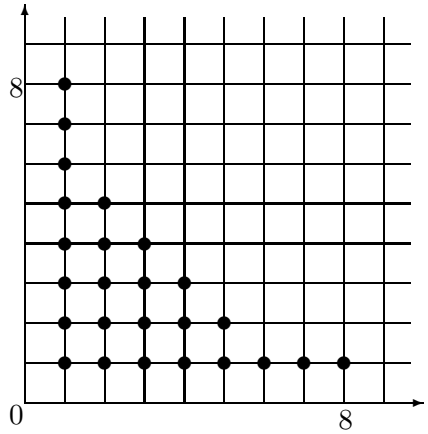


Figure 1: Branching numbers $(i, j) \in \mathbb{N} \times \mathbb{N}$ such that $\tau(i, j) \geq \tau(3, 4)$

($\tau_{max,T}$) and provided it processes each of its leaves in polynomial time, Lemma 1 gives a worst-case upper bound on its running time as $poly(\mu(F))\tau_{max,T}^{\mu(F)}$, where F is its input formula and μ an arbitrary complexity measure, strictly decreasing with each splitting.

Besides, Lemma 2 allows the algorithm to estimate the branching numbers in polynomial time and choose the next literal according to the suitable branching number.

3 Programming issues

The main idea of the program was to introduce in it the notion of an *unknown literal*. The program actually looks at a rather small piece of formula, just big enough to hold all the occurrences of several literals. The rest of the literals is supposed unknown; they are not subject to elimination rules.

The program receives as input the target branching number τ. First of all, let us note that if a formula contains an (i, j)-literal such that $\tau(i, j) \leq \tau$, making a split on this literal would achieve the purpose of the current step. Therefore, it is possible to reduce the possibilities for positive and negative occurrences of the literals to a certain finite subset of $\mathbb{Z} \times \mathbb{Z}$, depending on the target branching number (see an example on Fig. **??**).

Then the program initializes a literal and marks all other literals as unknown (if the first considered literal is an (i, j)-literal, then there would be $i + j$ clauses in the formula after its initialization – the rest are unknown and therefore irrelevant for reducing). After initializing, it tries to reduce the resulting formula using the current rules. If succeeded, it means that

this formula cannot be obtained as an input for the splitting and, therefore, we may proceed to other cases.

Then the program tries to find an acceptable split, that is, literals $l_1, \ldots, l_k$ such that the corresponding branching number is less than τ. If succeeded, it goes on through all the remaining possibilities of the literals' mutual arrangement. If it does not succeed, it initializes another literal and tries to split or reduce the resulting formula, looking for the target branching number.

4 Lower bounds for this type of reasoning

Of course, this kind of reasoning has its limitations. We present here an example of a formula that is very simple for any solver by itself. However, it constitutes an unavoidable obstacle for the line of reasoning based on Kullmann-Luckhardt's lemma.

Consider the following formula:

$$G(x_1, x_2, x_3) = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}).$$

Note that in the terms of Kullmann-Luckhardt's lemma, whichever literal we split on first, we produce a $(1, 3)$-split (that is, one clause is eliminated when x_i is set to $false$ and all three clauses disappear when $x_i := true$). The second splitting (we have to consider it only in one case, since in the other the whole formula disappears) would inevitably reduce the formula to an empty one.

Now let us consider the formula

$$F(x_1, \ldots, x_n) = G(x_1, x_2, x_3) \wedge G(x_4, x_5, x_6) \wedge \ldots \wedge G(x_{n-2}, x_{n-1}, x_n).$$

Due to the symmetry of the formula and the independence of subformulas with three variables, we can rearrange the splittings made by the pure literals elimination algorithm in any suitable order (provided we preserve the order of every pair of splittings made in any independent subformula). Let us rearrange them in such a way that splittings made by variables occuring in each subformula appear one after another. Then it is easy to see that every sequence of two splittings result in a branching vector with a branching number of $\tau(3, 3, 3) = 1.442250\ldots$.

Note that the formula itself is remarkably simple, because the majority of all the assignments are satisfiable, and it may not be considered a hard instance for any SAT solver. However, the line of reasoning along which the automated search is going would inevitably fail at this point.

5 Proof of the $1.442250\ldots^K = 2^{0.528321\ldots K}$ upper bound

We present here the formal proof of a previously unknown upper bound for the same solver, working with pure literals elimination only. We begin with several lemmata. Throughout this section we denote by F the formula in the current node of the branching tree and fix the number of clauses as the complexity measure. The bound in question is obtained as $\tau(3,3,3) = 1.442250\ldots$.

Lemma 3. *If F contains a $(1+,4+)$-literal or a $(2+,2+)$-literal, then its branching number is greater than the necessary bound.*

Proof. Straightforward calculations. □

Note that the unit clause rule is an intrinsic property of nearly all DPLL-type algorithms.

Lemma 4. *If the input formula F contains a unit clause $C = \{x_i\}$, then splitting by the literal x_i produces a branching vector of at least $(1, K)$, where K is the number of clauses in F.*

Proof. $F[x_i]$ contains an empty clause and is obviously unsatisfiable. Therefore, one of the branches is a leaf of the branching tree. □

Therefore, for every formula F with the number of clauses $K \geq 4$, a unit clause immediately provides us with the necessary bound, since $\tau(3,3,3) > \tau(1,4)$.

Lemma 5. *If F does not contain unit clauses, does not contain pure literals and contains a $(1,3)$-literal, then it is possible to pick a single literal l such that $\tau(\mathcal{K}(F) - \mathcal{K}(F[\overline{l}]_{red}), \mathcal{K}(F) - \mathcal{K}(F[l]_{red}) \geq \tau(3,3,3)$, where F_{red} denotes the result of applying the pure literals rule to F.*

Proof. We denote by a the $(1,3)$-literal and by b the second literal appearing in the clause with a. By assumption, b is a $(1+,1+)$-literal. Consider splitting by b. $\mathcal{K}(F) - \mathcal{K}(F[\overline{b}]_{red}) \geq 1$ and $\mathcal{K}(F) - \mathcal{K}(F[b]_{red}) \geq 4$, because in $F[b]_{red}$ all clauses containing a disappear, since $\overline{a}$ becomes a pure literal. Therefore, the necessary bound is obtained as $\tau(1,4)$. □

Lemma 6. *If F does not contain unit clauses, does not contain pure literals and does not contain $(1+,3+)$-literals or $(2+,2+)$-literals, then it is possible to pick two literals l_1, l_2 in F such that $\tau(\mathcal{K}(F) - \mathcal{K}(F[\overline{l_1}]_{red}), \mathcal{K}(F) - \mathcal{K}(F[l_1, \overline{l_2}])_{red}, \mathcal{K}(F) - \mathcal{K}(F[l_1, l_2])_{red}) \geq \tau(3,3,3)$ or a single literal l such that $\tau(\mathcal{K}(F) - \mathcal{K}(F[\overline{l}]_{red}), \mathcal{K}(F) - \mathcal{K}(F[l]_{red}) \geq \tau(3,3,3)$, where F_{red} denotes the result of applying the pure literals rule to F.*

Proof. The automated proof of this statement is too long to be published here. It is available from [11]. The notations and output format are explained there as well. □

All of the above, after applying Lemmata 1 and 2, proves the following.

Theorem 1. *The described DPLL-type algorithm with pure literals elimination as the only simplification rule gives the correct answer in the time not exceeding* $poly(K)2^{0.64743...K}$*, where* K *is the number of clauses in the input formula.*

6 Further work

One direction for further work in this field is obvious: we will implement other elimination rules and obtain better bounds. Other known elimination rules include elimination by resolution, unit clause elimination, blocked clauses elimination and the black-and-white literals rule (for an example where all of them are used, see [6]). It would be also interesting to try to prove bounds w.r.t. other complexity measures, the most interesting being the number of variables. We are currently working on it, and on this way we expect some new results and, possibly, records.

Another direction is to extend these methods to other NP-complete problems. There exist algorithms for other problems that employ similar splitting methods and bounds on which also have been achieved with the help of Kullmann-Luckhardt's lemma. An example of such algorithms for MAX-2-SAT and MAX-CUT can be found in [5], and the current record holder for MAX-CUT ([4]) also uses such techniques.

And, last but not least, it has been an equally challenging problem to obtain exponential lower bounds for such algorithms. Previous attempts usually employed graph-theoretical techniques ([9],[10],[8]). The automated search for upper bounds may yield explicit formulae that are hard for DPLL-type algorithms: after all, it does not succeed every time, and analysis of its output when a bound cannot be proven might turn out to be very useful. For example, this output may provide the so-called "bottleneck cases", i.e. explicit formulae where the algorithm performs with complexity close to its proven worst-case upper bound.

Acknowledgements

The authors would like to express gratitude to Edward A. Hirsch, who posed the problem and supervised our work.

References

[1] E.Y. Dantsin, E.A. Hirsch, S.V. Ivanov, M.A. Vsemirnov. Algorithms for SAT and upper bounds on their complexity (in Russian: Algoritmy dlja propozitsional'noj vypolnimosti i verhnie otsenki ih slozhnosti). Zapiski nauchnyh seminarov POMI, 277:14-46, 2001. English version appears in Elec-

tronic Colloquium on Computational Complexity, Technical Report 01-012, ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/2001/TR01-012/Paper.ps

[2] M. Davis, G. Logemann, D. Loveland. A machine program for theorem-proving. *Communications of the ACM* 5(7) (1962), 394-397.

[3] M. Davis, H. Putnam. A computing procedure for quantification theory. *Journal of the ACM* 7(3) (1960), 201-215.

[4] S. Fedin, A. Kulikov. A $2^{|E|/4}$-time algorithm for MAX-CUT. Zapiski nauchnyh seminarov POMI, 293:129-138, 2002.

[5] J. Gramm, E.A. Hirsch, R. Niedermeier, P. Rossmanith. New worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. *Electronic Colloquium on Computational Complexity*, Report No. 37(2000).

[6] E.A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24:397-420, 2000.

[7] O. Kullmann, H. Luckhardt. Algorithms for SAT/TAUT decision based on various measures. *Informatics and Computation*, 1998.

[8] S.I. Nikolenko. Hard satisfiable instances for DPLL-type algorithms. Zapiski nauchnyh seminarov POMI, 293:139-148, 2002.

[9] G.S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Matematical Logic*, Part II. Consultants Bureau, New York – London, 1968, 115-125.

[10] A. Urquhart. Hard Examples for Resolution. *The Bulletin of Symbolic Logic*, vol. 1, num. 4, Dec. 1995.

[11] http://logic.pdmi.ras.ru/~sergey/

A Logic Approach to Supporting Collaboration in Learning Environments

M. MAGDALENA ORTIZ DE LA FUENTE
Universidad de las Américas – Puebla. Mexico
is103378@mail.udlap.mx

ABSTRACT. We introduce a new application of extended disjunctive logic programs in the area of CSCL environments. A logic representation of the learner model is proposed as a set of beliefs that an agent holds about the interests and capabilities of its user. Based on these beliefs, the agent assures awareness and assists the collaboration among the learners in the community. It is an application of ASP, a formalism for non-monotonic reasoning, which has proved to be suitable since it allows simple and clear modeling and the behavior of the system is appropriate. Here we present the main aspects of the formalized model and some general properties of it.

1 Introduction

Due to the need of adding certain features to traditional logic programming (two types of negation, disjunction in the head of rules) many researchers have focused on the development of new semantics that support these extensions [12, 16, 29, 31]. Result of this effort is the Stable Models Semantics introduced in the late 80s by Gelfond and Lifschitz [17], also known as Answer Sets Programming (ASP). It is an alternative logic programming paradigm for disjunctive deductive databases (function free logic programs that allow the use of negation in the body of rules and disjunctions in the heads). Unlike most of the other semantics for extended logic programs (Well Founded Semantics [31], Perfect Models [29], etc.) the stable models semantics does not restrict logic programs to a single intended model, but they allow the existence of sets of various intended models (stable models or answer sets) [22]. ASP is now widely accepted as one of the best semantics for non-monotonic reasoning and disjunctive deductive databases.

With all these new semantics, we are able to develop more adequate reasoning models for applications in many fields. In the areas of computer assisted learning and tutoring systems, logic programming has been widely used, usually for knowledge representation [7, 9]. However, the development of logic-based applications for collaboration in learning communities is very scarce. In this paper, we address the problem from the perspective of ASP. There is some work on learner modeling for collaborative learning environments based on

Proceedings of the Eighth ESSLLI Student Session.
Balder ten Cate (editor)

probabilistic approaches, which provided good results [26]. In our previous work [23], we proposed ASP as a suitable basis for learner modeling in Computer Supported Collaborative Learning (CSCL) environments. Now we use ASP to model how the agents in this kind of learning environments can effectively support collaboration in the community according to their learner models.

In agent based CSCL environments [4, 5, 6, 30], the agents hold a set of beliefs about the learners, which they use to infer the best learning and collaboration opportunities for them within the community. These beliefs are usually incomplete or insufficient, and thus sometimes, when there is a lack of evidence, we need to reach non-monotonic conclusions. That's why we find that Answer Set Programming (ASP) is an appropriate logical framework [16]. Another reason to prefer ASP above other programming paradigms, is that in this context, there is usually not such thing as a single "right answer". We only want to suggest the learner possible learning tasks and workgroups. The multiple models given by the ASP semantics fit perfectly this need, since they allow us to have more than one adequate proposal for the learner [22].

Since we are using extended programs with two types of negation (classical negation and default negation), we can deal with different levels of reliability without needing a wider truth value lattice. Having both negations proved very useful for our purpose, since it makes modeling natural and direct, allowing us to weaken rules by the *normally* construct [16]. For more on negation see [18, 21]. There are also extensions of ASP that give us useful additional features like strong and weak constraints and the possibility of dealing with preferences [8, 12]. As we will discuss later, this can also be enriching since it allows more natural and accurate modeling.

The organization of the paper is as follows: First, in section 2, we present some preliminary concepts, focusing on CSCL. Sections 3 and 4 give an overview of the system, presenting the formalization of the domain and the learner model. In Section 5 we explain how the agents can promote collaboration based on a reasoning model formalized in ASP. Finally our conclusions and perspectives for future work are given in section 6.

2 Preliminaries

Computer Supported Collaborative Learning (CSCL) environments consist on the establishment of learner groups where the users learn as they collaborate in common tasks [6]. In a CSCL environment, learning is a collaborative process that is made possible with the active participation and the interaction among its members, who share and construct knowledge [5]. For the learning process in this kind of environments, collaboration is the most relevant issue [4]. In agent-based CSCL environments, each learner interacts with an agent that holds a set of beliefs about him/her known as *Learner Model* (LM) [30]. The agent uses its

own learner model and the beliefs of the other agents in the community to reach decisions that will give its learner the best collaboration and learning opportunities within the community. Our work consists of a logic formalization of this learner model and the way it can be used to promote better collaboration.

In our approach, we consider that the knowledge repository is possessed by the learners, instead of stored in the system. However, it is necessary that the agents are able to identify certain relations between the domain elements in order to understand its learner's development. For this reason, our model includes a representation schema for the knowledge domain. We also present a formalization of the learner model (agent's beliefs), based on the approach presented in [23]. According to these domain and learner models, we are going to propose a way in which the agents can help the learners establish an adequate learning plan and configure workgroups.

We suppose that the reader is familiar with the main aspects of logic programming as well as with the essential features of ASP. For a deeper reading or unexplained concepts see [16, 17, 21, 24]. Here we will refer more to the features of ASP that make it a suitable formalism for the application that we are modeling. For most of the formal details, we will simply refer to the literature on the subject. Throughout this paper, we are going to use the standard A-Prolog [16] notation, with slight differences that will be pointed out.

3 Domain

In a CSCL environment, the learning process implies the learners' participation in solving common tasks. In our approach, the *tasks* are related to one or more *areas* of interest (topics). To every task, there is a set of *situations*. Note that theses situations[1] are understood as small parts of a task where a particular *knowledge element* can be applied. We also relate support material to different situations and tasks which we call *learning resources*. Based on the approach of Goldstein [19], we are going to represent the relations between the knowledge elements in a *genetic graph*. This is a way of representing knowledge which consists of individual knowledge elements connected by *genetic* relations, which are specification/generalization, refinement/simplification and analogy. Based on this definition of the domain, we formalize it as follows:

D = < **DM, GG** > is the pair representing the domain, where **DM** is the *domain map* and **GG** the *genetic graph.*

[1] The term *situation* is taken from the theory of *situated cognition* [10], and is not related with situation calculus or action languages in the context of ASP.

The *domain map* **DM** is represented as a 5-tuple: **DM** = < **A, T, S, R, K, Rel** >, being **A, T, S, R** and **K** the sets of *areas*, *tasks*, *situations*, *resources* and *knowledge elements* respectively. In the set **Rel** there are three types of relations:

- **similarity** relations between areas:

$$similar \subseteq \mathbf{A} \times \mathbf{A}$$

- **association** relations between tasks and areas, situations and tasks, resources and situations or resources and tasks:

$associated_1 \subseteq \mathbf{T} \times \mathbf{A}$ $associated_3 \subseteq \mathbf{S} \times \mathbf{T}$ $associated_2 \subseteq \mathbf{R} \times \mathbf{S}$ $associated_4 \subseteq \mathbf{R} \times \mathbf{T}$

When we use the term *associated* we refer to any of these four.

- **applicability** relations between knowledge elements and situations:

$$applicable \subseteq \mathbf{K} \times \mathbf{S}$$

The *genetic graph* **GG** is the pair **GG** = < **K**, **Rk** >; where **K** is the set of knowledge elements and **Rk** $\subseteq$ **K** x **K** is the set of genetic relations between them.

For example, in our German grammar domain, we represent different tasks where the users have to write a conversation applying different grammar rules (knowledge elements). Here we present a part of this domain, including only one area and two tasks. For the moment, we leave out the associated learning resources. The **DM** would be as follows:

A = { a_1:[Services] } **T** = { t_1:[Eating in a restaurant], t_2: [Shopping in the market] }

S = { s_1:[Request an article], s_2:[Ask about the availability of an article], s_3:[Ask the price of an article], s_4:[Request the bill] }

K = { k_1:[request_obj+*bitte*], k_2:[request_*möchte*/*hätte* (+*gern*) + obj], k_3:[request_(bill)+*bitte*], k_4:[question_verb], k_5:[question_QW], …}

Both task are related to the only area we are dealing with. Situations s_1, s_2 and s_4 are related to task t_1, and situations s_1, s_2 and s_3 are related to t_2. The knowledge elements k_1 and k_2 can be applied in situations s_1 and s_4, while k_3 can only be applied in situation s_4, and so on. This way the applicability relations are defined for all elements in **K** x **S.** In the genetic graph, k_1 holds a *refinement* relation with k_2, as k_3 does with k_4. At the same time, k_2 holds an *specification* relation with k_4. The inverse relation from refinement is *simplification*, and from specification is *generalization*. There are also *analogy* relations, which are symmetric, as between k_4 and k_5 for example.

To represent our domain as an ASP program, we use the following predicates.

area/1	*resource/1*	*similar/2*
task/1	*knowledgeElement/1*	*associated/2*
situation/1		*applicable/2*

analogy/2	*refinement/2*	*specification/2*
generalization/2	*simplification/2*	*geneticRelation/2*

As an example, here we present a part of our German grammar example already coded in DLV[2], an implementation of ASP [20]. The complete program, as well as a full description of the example, is available online[3].

```
task(restaurant).
situation(req_art).
knowledgeElement(request_bitte).
knowledgeElement(request_moechtehaette).
refinement(request_bitte, request_moechtehaette).
associated(req_art, restaurant).
applicable(request_bitte, req_art).
analogy(X, Y) :- analogy(Y, X).
simplification(X, Y) :- refinement(Y, X).
generalization(X, Y) :- specification(Y, X).
geneticRelation(X , Y) :- analogy(X, Y).
geneticRelation(X , Y) :- refinement(X, Y).
```

4 Learner model

A *Learner Model (LM)* is a particular case of an user model consisting of a set of beliefs about the learner's interests and capabilities [6, 30]. The *LM* is what the agent believes about its learner. All the decisions made by the agent concerning the learner's learning proposal, configuration of groups, etc. are based on the beliefs contained in the *LM*.

The beliefs in the *LM* are represented by instances of the predicates *interested*/2 and *capable*/2. For example, the LM of *learner* would be defined by facts like the following:

interested (learner, a). *learner* is interested in the (area/task/situation/resource) *a.*
capable (learner b). *learner* is capable of applying correctly knowledge element *b.*

This type of facts are called *basic beliefs*. However, they are often not enough to allow the agent to find the best learning opportunities for its learner. On the basis of its basic beliefs, the agent can derive more beliefs about him/her.

First of all, there are two derived capability relations that can be expressed through the following rules:

- a learner is capable of handling a situation correctly, when he/she is capable of apply a suitable knowledge element.

  capable(Learner, Situation) ←
  applicable(*KnowElem, Situation*), *capable*(*Learner, KnowElem*)
- a learner is capable of fulfilling a task, when he/she is capable of handling all the situations associated to the task.

  capable(Learner, Task) ←
  (∀*Situation*: *associated*(*Situation, Task*)→ *capable*(*Learner, Situation*))

[2] http://www.dbai.tuwien.ac.at/proj/dlv/
[3] http://mailweb.udlap.mx/~is103378/tesis

Note that we use universal quantifiers for abbreviation reasons, as well as implication the body of rules. They do not belong to the strict ASP syntax, however rules of the form $r \leftarrow \forall x$: $(p(x) \rightarrow q(x))$ are used due to their usefulness in the modeling process. Certain intuitions are clearly and naturally expressed through this construction. Theses rules can be expressed in ASP through simple transformations. The quantifier can expressed in terms of conjunctions of elements (the Herbrand universe is finite), and the implication can be re-written in terms of negation and disjunction, as we would do in classical logic. This alternative can be questioned in ASP, but it is possible under certain considerations. For a deeper reading on the subject, go to [24, 25, 28].

4.1 Belief derivation rules

There are many ways in which an agent can derive beliefs about its learner. We propose a set of rules that allow the agent to do this derivation on the basis of its basic beliefs, the structure of the domain and the beliefs of other agents. We call them *Belief Derivation Rules.* The extension and improvement of these rules is an open area of research, as we mentioned in [23], where we also suggest some approaches that we believe could suitable for this purpose. Here we present some of the rules that we have been working with. They are believed to de adequate in most cases, however, they can be adapted to suit the needs of different learning communities.

4.1.1 Structural derivation of interests

According to the structural relations between the domain elements, the agent can infer new beliefs about the learner. For example, when the agent believes that a learner is interested in an area, it will also believe that he/she is *normally* interested in similar areas. Associated domain elements will be treated in a similar way.

$$interested(Learner, Area_1) \leftarrow similar(Area_1, Area_2), interested(Learner, Area_1), not \sim interested(Learner, Area_2)$$

$$interested(Learner, Element_2) \leftarrow associated(Element_1, Element_2), interested(Learner, Element_1), not \sim interested(Learner, Element_2)$$

4.1.2 Social derivation of interests

The agent can also suppose that the user could or should be interested in certain resources because they are very popular in the community, or because

they appear frequently related to other users with similar interests. The agent will believe, for example, that an user is normally interested in the domain elements (areas, tasks, situations, resources) that are a part of the interests of all the people in his workgroup.

$$
\begin{aligned}
&interested(Learner, Element) \leftarrow \\
&\quad (\forall\ Learner_1:\ (\ Learner_1 \neq Learner, \\
&\quad taskgroup(Learner, Task), \\
&\quad taskgroup(Learner_1, Task)) \rightarrow \\
&\quad interested(Learner_1, Element)\ ), \\
&\quad not\ {\sim}interested(Learner, Element)
\end{aligned}
$$

4.1.3 Derivation of capabilities

Finally, the following rules are an example of the ones we have used for deriving beliefs about the capabilities of the learner.

$$
\begin{aligned}
&capable(Learner, KnowElem_2) \leftarrow \\
&\quad simplification(KnowElem_1, KnowElem_2), \\
&\quad capable(Learner, KnowElem_1), \\
&\quad not\ {\sim}capable(Learner, KnowElem_2)
\end{aligned}
$$

$$
\begin{aligned}
&capable(Learner, Ke_1) \leftarrow \\
&\quad (\forall Ke: specialization(Ke_1, Ke) \rightarrow capable(Learner, Ke)), \\
&\quad not\ {\sim}capable(Learner, Ke_1)
\end{aligned}
$$

The set of all basic beliefs and all derived beliefs are the agent's *learner model.* We will use the term $\boldsymbol{Interests_{Learner}}$ to name the set of all the domain elements (areas / tasks / situations / resources) that the agent believes that *Learner* is interested in. On the other hand, $\boldsymbol{Interests\tilde{}_{Learner}}$ will contain all elements that the agent believes *Learner* is not interested in. Analogously, we define $\boldsymbol{Capabilities_{Learner}}$ and $\boldsymbol{Capabilities\tilde{}_{Learner}}$ as the corresponding sets of knowledge elements.

For example, according to the domain example presented in section 3, the basic beliefs of the of a learner named John, could be coded as follows:

```
interested(john, market).
interested(john, request_article).
capable (john, request_bitte).
capable(john, request_moechtehaette).
```

In this case, since John is capable of applying the knowledge element k_4:[request_moechtehaette], which is applicable in situation s_4:[Request the bill], the agent will believe that John is also capable of handling s_4. In this way, the agent will derive John's learner model, which will be the answer sets of the logic program that contains the basic beliefs of the agent as well as the belief derivation rules.

5 Supporting collaboration

As mentioned above, collaboration is the key issue in CSCL environments. The agents are going to support it by assuring the social and concept awareness [4] of all the learners in the community, carrying out two main tasks:

a) Proposing the learner an appropriate set of tasks in order to help him/her to establish a learning plan.

b) Assisting group configuration according to the interests and capabilities of the members of the community.

5.1 Learning proposal

In this section, we explain briefly how the agent can propose the learner the tasks that it considers appropriate for him/her, so that the learner can maintain his/her own learning plan. We are going to do it based on the proposal of Ayala and Yano [6], which is inspired on the theory of social learning [32] and consists on the creation of zones of proximal development. A more detailed description of this approach and of our formalization of it are available in [6, 23]. We will call the rules presented in this section *Learning Proposal Rules.*

5.1.1 Knowledge frontier (KF)

We define the Knowledge Frontier of a learner as the set of knowledge elements that the agent does not believe to be in the learner's capabilities, and which are related via genetic relation to those knowledge elements that the agent believes that he/she is already capable of applying. Under ASP, we can define the knowledge frontier through the following rule:

$$\begin{aligned} &memberOfKF(Learner, KnowElem_2) \leftarrow \\ &\quad geneticRelation\ (KnowElem_1, KnowElem_2), \\ &\quad capable(Learner, KnowElem_1)\ , \\ &\quad not\ capable(Learner, KnowElem_2) \end{aligned}$$

5.1.2 Zone of proximal development (ZPD)

The ZPD is the set of knowledge elements that are in the learners KF and that have already been internalized by other mebers of the community. They are the knowledge elements that are believed to have the best opportunities of being internalized by collaborating in the community. We can derive the learner's ZPD in ASP with the following rule:

$$\begin{aligned} &memeberOfZPD(KnowElem) \leftarrow \\ &\quad memberOfKF(Learner, KnowElem), \\ &\quad capable(Learner_1, KnowElem), \\ &\quad Learner_1 \neq Learner \end{aligned}$$

5.1.3 Learning Proposal (LP)

The LP will be the set of tasks where the knowledge elements in the learner's ZPD are applied and that are related to the learners' interests. These are the tasks that the agent will propose to the learner. This can be expressed in ASP through the following rule:

memberOfLP(Learner, Task) ←
memberOfZPD(Learner, KnowElem),
applicable(KnowElem, Situation),
associated(Situation, Task),
interested(Learner, Task)

5.1.4 Establishment of a learning plan

We will call ***P*** the logic program which contains the learner model (basic beliefs and belief derivation rules) as well as the learning proposal rules. The semantics of ***P*** under ASP are a set of stable models (answer sets). We will refer to this set as $\boldsymbol{S(P)}=\{s_1\boldsymbol{(P)}, s_2\boldsymbol{(P)}, \ldots, s_n\boldsymbol{(P)}\}$. All the answer sets $s_1\boldsymbol{(P)}$, $s_2\boldsymbol{(P)}$, … give us valid learning proposals for the learner, which the agent will propose to him/her. In the presence of multiple models, we can sort the possible proposals in meta-language. Further considerations about this issue were presented in [23]. For the time being, we present only one simple criteria that allows us to select those tasks that the agent believes are more likely to be appropriate for the leaner when we have multiple learning proposals,

For this purpose, we define the *first choice* and *second choice* of ***LP***. During the interaction with the learner, the agent proposes those tasks that are part of the learning proposal in every answer set (*cautious entailment* [16]) as a *first choice* for the students learning plan. In case that the learner request more tasks, or does not accept the firsts one proposed, the agent proposes the second choice, which includes those tasks that are part of the learning proposal in only some of the answer sets (*brave entailment* [16]).

5.2 Group proposal

Assisting the configuration of groups is the next relevant task for the agent. All the groups are formed for an specific task on which the group members will work together. To define a group, we need the following conditions:

1. There must be more than one member.
2. The task must be a part of the learning plan of all members

With this two conditions, we can make a task group. However, it would also be desirable to fulfill some additional conditions:

3. All the learners in the community should have a group.
4. Every member can make an effective contribution in the task (apply a knowledge element of which he/she is already capable).

5. For every situation in the task, there is a member of the group that is capable of applying a suitable knowledge element.

Conditions 1 and 2 can be expressed in ASP through the following rule:

groupProposal(Learner,Task) ←
memberOfLp(Learner,Task),
memberOfLp(Learner1,Task),
Learner ≠ *Learner1*

However, it would be good if conditions 3, 4 and 5 could also be satisfied. This can be easily solved using an extension of Disjunctive Datalog enhanced with weak constraints [12, 13]. The traditional (integrity) constraints :-α must always be satisfied (α must be false). Weak constraints, on the other side, are only satisfied if possible. They are usually written :~α and the semantics of programs with this type of constraints give the *best models* of a program, i.e., the answer sets where the true instances of α are minimized. Weak constraints are a very useful extension for representing common sense reasoning. They allow us to enhance our model by rules similar to the following one:

:~ *groupProposal (Learner, Task), not capableOfAssociatedSituation(Learner, Task).*

capableOfAssociatedSituation(Learner, Task) ←
associated(Situation, Task),capable(Learner, Situation).

In the full model more of these constraints are given. They are implemented using different priority levels and weight, an useful feature of DLV.

6 Conclusions

Applying and implementing the theoretical results obtained has always been a challenge for the Logic Programming community. Here we have tried to do it in a different type of application. ASP turned out to be well suitable, since it allows easy and natural modeling, and model behaves as expected. Collaborative learning is a suitable field for applications of disjunctive programming, since it deals naturally with changing beliefs and incomplete information. Some examples have already been implemented in DLV with satisfactory results.

We are currently working on dynamic issues of the model using well known semantics for updating knowledge bases under the ASP framework, like Dynamic Logic Programming [2, 3] and Update Programs [14, 15]. There are still many research projects in the framework of ASP that can lead to interesting extensions of the model [1, 8, 11, 27].

***Aknowledgements*:** This project was partially supported by CONACYT, Mexico. Projects: 37837-A and W8056.

References

[1] Acosta, J. C., Arrazola, J. and Osorio M.. Making belief revision with LUPS. In Juan Humberto Sossa Azuela and Gustavo Arroyo Figueroa, editors, *XI International Conference on Computing*, PO Box 75-476 Col. Nueva Industrial Vallejo 07738, México, D.F., November 2002. CICIPN.

[2] Alferes, J. J.; Pereira, L. M.; Przymusinska, H. and Przymusinski, T. C. LUPS - a language for updating logic programs. *Artificial Intelligence* 138(1-2), 2002.

[3] Alferes J. J.; Leite J. A.; Pereira L. M.; Przymusinska H., and Przymusinski, T. Dynamic logic programming. In A. Cohn and L. Schubert, editors, *KR'*98. Morgan Kaufmann, 1998. http://citeseer.nj.nec.com/article/alferes98dynamic.html

[4] Ayala, G. Towards Lifelong Learning Environments: agents supporting the collaborative construction of knowledge in virtual communities, *Computer Support for Collaborative Learning CSCL2003*, Bergen, Norway, June (aceptado).

[5] Ayala, G. Intelligent Agents Supporting the Social Construction of Knowledge in a Lifelong Learning Enviroment, Proceedings of the *International Workshop on New Technologies for Collaborative Learning* NTCL 2000, November, Hyogo, Japan, 79-88, 2000.

[6] Ayala, G. and Yano, Y. Learner Models for Supporting Awareness and Collaboration in a CSCL Environment. *Intelligent Tutoring Systems*, Claude Frasson, Gilles Gauthier and Alan Lesgold (Eds.), *Lecture Notes in Computer Science* 1086, Springer Verlag 1996, pp: 158-167.

[7] Baral, C. and Gelfond, M. Logic Programming and Knowledge Representation. *Journal of Logic Programming* 19, 20: 1994. pp. 73-148.

[8] Brewka, G. and Eiter, T. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297-356, 1999.

[9] Brna, P. Logic Programming in Education: a Perspective on the State of the Art. 1994. http://citeseer.nj.nec.com/brna94logic.html

[10] Brown, J. S., Collins, A. and Duguid, P. Situated cognition and the culture of learning. *Education Researcher* 18 (1) 32 - 42. 1989.

[11] Buccafurri, F.; Faber, W. and Leone, N. Disjunctive logic programs with inheritance. In D. De Schreye, editor, ICLP'99. MIT Press, 1999. http://citeseer.nj.nec.com/buccafurri99disjunctive.html

[12] Buccafurri, F.; Leone, N. and Rullo, P. Adding Weak Constraints to Disjunctive Datalog. In *Proceedings of the 1997 Joint Conference on Declarative Programming APPIA-GULP-PRODE'97*, Grado, Italy, June 1997.

[13] Dix, J. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In Andre Fuhrmann and Hans Rott, editors, *Logic, Action and Information* -- Essays on Logic in Philosophy and Artificial Intelligence, pages 241--327. DeGruyter, 1995

[14] Eiter, T.; Fink, M.; Sabbatini, G. and Tompits, H. *On Updates of Logic Programs: Semantics and Properties*, 2002

[15] Eiter, T.; Fink, M.; Sabbatini, G. and Tompits, Hans. An Update Front-End for Extended Logic Programs. *Logic Programming and Non.Monotonic Reasoning*. LPNMR 2001: 397-401

[16] Gelfond, M. Representing Knowledge in A-Prolog, volume 2408 of *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, pages 413-451. Springer-Verlag, Berlin, 2002.

[17] Gelfond, M. and Lifschitz, V.. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Logic Programming*: Proc. of the Fifth Int'l Conf. and Symp., pages 1070-1080, 1988.

[18] Gelfond, M. and Lifschitz, V., Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing* 9(3,4):365--385 (1991). http://citeseer.nj.nec.com/gelfond91classical.html

[19] Goldstein, I. P. The genetic graph: A representation for the evolution of procedural knowledge. *International Journal of Man-Machine Studies*, 11, pp. 51-77. 1979.

[20] Leone, N.; Pfeifer, G.; Faber, W.; Calimeri, F.; Dell'Armi, T.; Eiter, T.; Gottlob, G.; Ianni, G.; Ielpa, G.; Koch, C.; Perri, S. and Polleres, A. The DLV System. In Giovambattista Ianni and Sergio Flesca, editors, *Proceedings of the 8th European Conference on Artificial Intelligence* (JELIA), number 2424 in Lecture Notes in Computer Science, September 2002.

[21] Lifschitz, V. Foundations of logic programming. in *Principles of Knowledge Representation*, CSLI Publications, 1996, pp. 69-127.

[22] Marek, W. and Truszczynski, M. Stable models and an alternative logic programming paradigm. *In The Logic Programming Paradigm: a 25-Year Perspective,* pages 375-398. Springer-Verlag, 1999. http://citeseer.nj.nec.com/article/marek99stable.html

[23] Ortiz M., Ayala G. and Osorio M. Formalizing the Learner model for CSCL Enviroments, *Fourth Mexican International Conference in Computer Science.* (ENC' 03) Tlaxcala, Mexico, September 2003 (accepted).

[24] Osorio, M.; Navarro, J. A. ; Arrazola, J. Applications of Intuitionistic Logic in Answer Set Programming. *Journal of Theory and Practice of Logic Programming* (Accepted in 2003).

[25] Osorio, M. Navarro, J. A., Arrazola J. A logical approach to A-Prolog . *Electronic notes in theoretical computer science*, 67. Elsevier Science. 2002. http://www.elsevier.nl/locate/entcs/volume67.html

[26] Paredes, R. G. and Ayala, G. An user model server for the personalization of digital services and collections, proceedings of the *XII Congreso Internacional de Ingeniería Electrónica, Comunicaciones y Computadoras*, IEEE Puebla y UDLA, Acapulco, Mexico, pp.84- 88. 2002

[27] Pearce, D., Sarsakov V., Schaub, T.; Tompits, H. and Woltran, S.: A Polynomial Translation of Logic Programs with Nested Expressions into Disjunctive Logic Programs: Preliminary Report. *ICLP* 2002: 405-420

[28] Pearce, D. From Here to There: Stable Negation in Logic Programming. In D. Gabbay and H. Wansing, editors, *What is Negation?* Kluwer, 1999.

[29] Przymusinski, T. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193-216. Morgan Kaufmann, San Mateo, CA., 1988.

[30] Self, J. The role of learner models in learning environments. *Transactions of the Institute of Electronics, Information and Communication Engineers*, E77-D(1), 1994. pp. http://citeseer.nj.nec.com/self94role.html

[31] Van Gelder, A.; Ross, K. and Schlipf, J. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620-650, 1991.

[32] Vygotsky, L. Mind in Society: The development of higher psychological processes. M.Cole, V. John-Steiner, S. Scribner & E. Souberman, Eds. Cambridge MA: Harvard University Press. 1978

LTL Hierarchies and Model Checking

Radek Pelánek
Department of Computer Science, Faculty of Informatics
Masaryk University Brno, Czech Republic
xpelanek@fi.muni.cz

Abstract. We propose a new hierarchy of LTL formulas based on alternations of Until and Release operators and show that it is more relevant to model checking then previously studied hierarchies. Moreover, we study practically used formulas and conclude that in most cases it is possible to use specialized algorithms which are more efficient then general algorithm for LTL model checking.

1 Introduction

Linear Temporal Logic (LTL) is a popular formalism used in formal verification, particularly in model checking. Despite the high theoretical complexity bound, LTL model checking is well-known to be PSPACE-complete, LTL is successfully used in practice.

This contrast led Demri and Schonoebelen [6] to ask: *"What makes LTL model checking feasible in practice?"*. They tried to answer this question by the study of natural hierarchies of LTL formulas and complexity of model checking of their restricted classes. Unfortunately, they found that the PSPACE lower bound hold even for classes at the bottom of the hierarchies.

Notwithstanding this negative result, LTL hierarchies are a very active research area [27, 15, 25, 11, 21]. However, the recent works address mainly expressivity problems. In this work we overview most hierarchies studied in the literature with the following question in mind: *"Have these hierarchies any relation to model checking?"* We find out that most of the previously studied hierarchies do not have much connection to model checking.

Our main contribution is the introduction of a new hierarchy of LTL formulas based on the alternation depth of Until and Release operators. We provide the relation of this hierarchy with previously studied safety-progress hierarchy [22, 4] and show that it is possible to use more efficient algorithms for verification of formulas from lower classes of the Until/Release hierarchy.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

Thus we claim that this hierarchy is more relevant to model checking than the previously studied hierarchies.

Finally, we study practically used formulas from "Specification Patterns System" [7]. We find out that many practically used formulas lie in the lower classes of the Until/Release hierarchy.

Due to the space limitations we do not go into technical details and provide only general overview of our research.

2 Preliminaries

Linear Temporal Logic. The set of LTL formulas is defined inductively starting from a countable set AP of atomic propositions, Boolean operators, and the temporal operators $\mathtt{X}$ (Next) and $\mathtt{U}$ (Until):

$$\Psi := a \mid \neg\Psi \mid \Psi \vee \Psi \mid \Psi \wedge \Psi \mid \mathtt{X}\,\Psi \mid \Psi\,\mathtt{U}\,\Psi$$

LTL formulas are interpreted in the standard way [9] on infinite words over the alphabet 2^{AP}. We adopt standard abbreviations $\mathtt{R}, \mathtt{F}, \mathtt{G}$ for temporal operators Release ($\alpha\,\mathtt{R}\,\beta \equiv \neg(\neg\alpha\,\mathtt{U}\,\neg\beta)$), Future ($\mathtt{F}\,\alpha \equiv \mathit{true}\mathtt{U}\alpha$), and Globally ($\mathtt{G}\,\alpha \equiv \mathit{false}\,\mathtt{R}\,\alpha$) respectively.

Automata. A Büchi automaton is a tuple $A = \langle \Sigma, Q, q_0, \delta, F \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to 2^Q$ is a nondeterministic transition function, and $F \subseteq Q$ is a set of accepting states. We differentiate *general*, *weak*, and *terminal* automata according to the following restrictions posed on their transition functions:

- *general*: none restriction
- *weak*: there exists a partition of the set Q into components Q_i and an ordering $\leq$ on these sets, such that for each $q \in Q_i, p \in Q_j$, if $\exists a \in \Sigma : q \in \delta(p, a)$ then $Q_i \leq Q_j$. Moreover for each Q_i, $Q_i \cap F = \emptyset$ or $Q_i \subseteq F$.
- *terminal*: for each $q \in F, a \in \Sigma$ it holds $\delta(q, a) \neq \emptyset$ and $\delta(q, a) \subseteq F$.

Terminal and weak automata are jointly called *specialized* automata.

Model Checking. The LTL model checking problem is to decide for a given system S and an LTL formula φ whether $S \models \varphi$, i.e. whether each run of S satisfy φ. The standard approach to LTL model checking is to construct an automaton $A_{\neg\varphi}$ which accepts exactly words which are *not* models of φ and then test the product automaton $S \times A_{\neg\varphi}$ for non-emptiness.

3 LTL Hierarchies

The most classical classes of LTL formulas are obtained by restricting the use of temporal operators. Following the usual notation ([9, 24]) we let

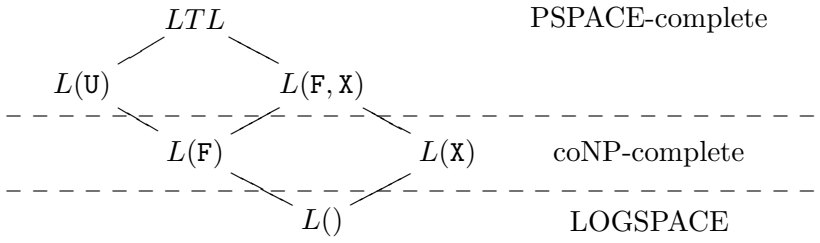


Figure 1: Inclusions which relate basic classes of LTL formulas into a hierarchy and complexity of their model checking.

$L(H_1, H_2, \ldots)$ denote the class of LTL for which only the temporal operators $H_1, H_2, \ldots$ are allowed. In this way we obtain six basic classes (see Fig. 1) – the classes obtained by combinations of other operators (such as $\mathtt{R}, \mathtt{G}$) coincide with one of these classes. Unfortunately, the complexity of model checking problem is very high even for restricted classes of the logic [24].

Practically important is the class $L(\mathtt{U})$. This class is closed under stuttering (i.e. it is insensitive to the number of successive occurrences of a letter in a word). Stutter-invariance enables to use efficient state space reduction techniques such as the partial order reduction or transition compression [17].

The hierarchy in Fig. 1 can be made more finer by constraining the maximum nesting depth of one or more operators. For example, we use the notation $L(\mathtt{U}^2, \mathtt{X})$ to denote the class of LTL with the use of modality $\mathtt{U}$ restricted to maximum nesting depth 2 and with the unrestricted use of modality $\mathtt{X}$. Researchers have studied several hierarchies based on the notion of the nesting depth, mainly $L(\mathtt{X}, \mathtt{F}, \mathtt{U}^m)$, $L(\mathtt{X}, \mathtt{U}^m)$, $L(\mathtt{X}^n, \mathtt{U})$, $L(\mathtt{X}^n, \mathtt{U}^m)$ [15, 11].

The work concerning these hierarchies is interested mainly with expressivity results: the strictness of hierarchies, characterization of languages in particular classes, decidability of membership in a given class, and connections to other formalisms (automata, semigroups, first order logic). The relations with model checking are sparse. Demri and Schnoebelen [6] have shown that the model checking problem is PSPACE-complete even for very restricted class such as $L(\mathtt{U}^2)$.

LTL can be extended with past operators like Since and Previous. The use of past operators do not add expressive power – classes that use past operators coincide with some of the pure-future classes. The advantage of using past operators is that the formulas can be exponentially more succinct than their pure-future counterparts [20] while the complexity of the model checking remains the same [23].

Another class is the consequence of a dispute concerning relative merits

of linear and branching time logics (Vardi [26] gives a good overview). This dispute led to the study of relations among these logics [16] and to the search for properties expressible in both linear and branching time formalisms. The important class of this type is is the LTL^{det} class [21] – properties expressible in both LTL and ACTL. Model checking for this class can be solved in polynomial time.

Several other (more exotic) classes have been proposed. Dams suggested class based on a flat Until [5] – flat Until $\varphi_1 \mathtt{U}^f \varphi_2$ allows only propositional formula to appear in φ_1. However, the complexity of model checking remains the same for the flat class [6]. Another classes are for example these restricting the use of future operators in the scope of past operators [25, 23], or the number of atomic proposition [6]. None of these classes enable more efficient model checking.

Finally, we mention the membership problem. The problem is to decide, whether for a given formula φ and an LTL class C there is a formula φ' such that $\varphi \equiv \varphi'$ and $\varphi' \in C$. This problem is known to be decidable for most classes [27, 11, 15], but the complexity of the problem is usually very high (typically PSPACE-complete).

4 Until/Release Alternating Hierarchy

In this section we propose a new hierarchy of LTL formulas. This hierarchy is based on the *alternation* rather than the *nesting* of operators. We show the connection of the new hierarchy with previously studied safety-progress hierarchy and with model checking. Due to the space restrictions we state only main results without proofs.

Let us define hierarchies Σ_i^{LTL} and Π_i^{LTL} which reflect alternations of Until and Release operators in formulas. We use the Σ/Π notation since the way the hierarchy is defined strongly resembles the quantifier alternation hierarchy of first-order logic formulas or fixpoints alternation hierarchy of μ-calculus formulas [10].

Definition 1

The class $\Sigma_0^{LTL} = \Pi_0^{LTL}$ is the least set containing all atomic propositions and closed under the application of boolean and Next operators.

The class Σ_{i+1}^{LTL} is the least set containing Π_i^{LTL} and closed under the application of conjunction, disjunction, Next and Until operators.

The class Π_{i+1}^{LTL} is the least set containing Σ_i^{LTL} and closed under the application of conjunction, disjunction, Next and Release operators.

The class $\mathcal{B}_i^{LTL}$ is the boolean closure of Σ_i^{LTL} and Π_i^{LTL},

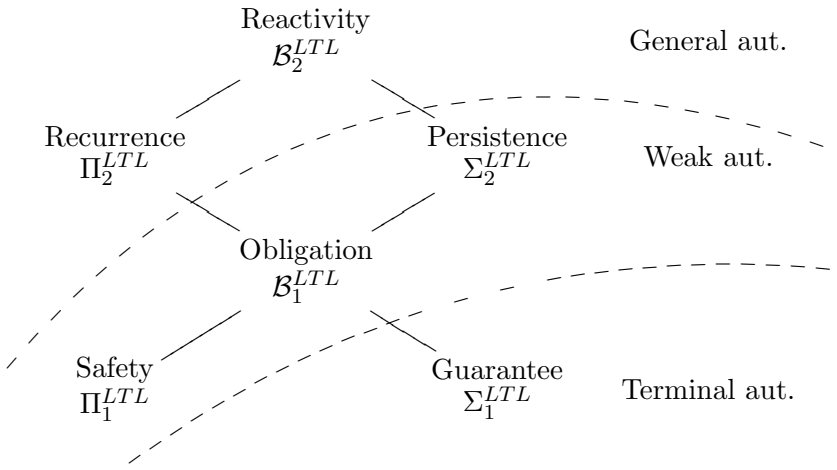


Figure 2: Safety-progress hierarchy, Until/Release hierarchy, and the corresponding types of Büchi automata

It shows up that our new hierarchy strongly relates to the safety-progress hierarchy defined by Manna and Pnueli [22, 4]. This is a classification of properties into a hierarchy consisting of six classes: *guarantee, safety, obligation, persistence, recurrence,* and *reactivity.* Inclusions, which relate the classes into a hierarchy, are depicted in Fig. 2. This hierarchy extends the safety-liveness characterization of ω-languages [18] and the Landweber hierarchy [19]. The classes of the hierarchy can be characterized through four views: a language-theoretic view, a topological view, a temporal logic view, and an automata view [22]. The fact that the hierarchy can be defined in many different ways shows the robustness of this hierarchy. Chang, Manna, and Pnueli have shown that the safety-progress hierarchy can be exploited for more efficient theorem proving [4]. We show that it can be used for better model checking as well.

The safety-progress hierarchy is practically orthogonal to the hierarchy from Fig. 1. The only remarkable relation to classes mentioned in the previous section is that $LTL^{det} \subseteq$ Recurrence. The relation with the Until/Release hierarchy is given by the following theorem:

Theorem 1 *A language that is specifiable by LTL is a guarantee (safety, obligation, persistence, recurrence, reactivity respectively) language if and only if it is specifiable by a formula from the class* Σ_1^{LTL} *(*$\Pi_1^{LTL}, \mathcal{B}_1^{LTL}$, $\Sigma_2^{LTL}, \Pi_2^{LTL}, \mathcal{B}_2^{LTL}$ *respectively) (see Fig. 2).*

Proof. [Sketch] The proof is based on the syntactic characterization of safety-progress classes by Chang, Manna, and Pnueli [4]. Using syntactic identi-

ties we are able to transform their characterization into the corresponding Σ^{LTL}/Π^{LTL} characterization.

As a consequence of the relation with safety-progress hierarchy we obtain the following, quite surprising fact (note that the previously mentioned hierarchies of first-order and μ-calculus formulas are infinite):

Theorem 2 *Both Σ_i^{LTL} and Π_i^{LTL} hierarchies semantically collapse – every LTL specifiable formula is specifiable by a $\mathcal{B}_2^{LTL}$ formula.*

The complexity of model checking is still PSPACE-complete even for lower classes of the hierarchy. Nevertheless, it is possible to employ this hierarchy for model checking. Formulas from lower classes of the hierarchy can be translated into specialized automata and the non-emptiness check for the product automaton can be performed more efficiently.

Theorem 3 *For every Σ_1^{LTL} (Σ_2^{LTL}) formula φ one can construct a terminal (weak) automaton accepting the language defined by φ.*

Proof. [Sketch] The basic idea of the construction is the same as for classical algorithm for transformation of LTL formula into automaton [13]. States of the automaton are sets of subformulas of the formula φ. The transition function is constructed in such a way that the following invariant is valid: if the automaton is in a state S then the remaining suffix of the word should satisfy all formulas in S. The main difference is in the way the acceptance condition is defined. For Σ_1^{LTL} and Σ_2^{LTL} formulas the acceptance condition can be simplified thanks to the special structure of alternation of Until and Release operators in the formula.

The non-emptiness of general automata is usually checked by nested depth-first search with explicit representation of the state space [14] or by nested fixpoint computation with quadratic number of symbolic steps with symbolic representation [12]. The non-emptiness of weak automata can be solved by single depth-first search [8] or by simple fixpoint computation with linear number of steps [2] respectively. The non-emptiness of terminal automata can be solved by classical reachability.

With the symbolic representation there is even asymptotic difference between the non-emptiness algorithms for general and specialized automata. All explicit algorithms have the same complexity, but the use of specialized algorithms still brings several benefits. Time and space optimization, "Guided search" heuristics [8], and the partial order reduction [14] can be employed more directly for specialized algorithms. Algorithms for specialized automata can be performed in distributed environment more easily [3]. Many of these benefits have been already demonstrated experimentally [2, 8, 3].

Since for Σ_1^{LTL} and Σ_2^{LTL} formulas we can use more efficient algorithms then for general formulas, the membership problem for these classes is of particular interest. This problem is decidable. Unfortunately, as for previously considered classes, the complexity of the problem is very high (the direct decision procedure goes via deterministic Streett automata and is doubly exponential). However, as we will see in the next section, practically used formulas are very short (thus it is feasible to perform expensive algorithms for them) and many of them even lie in Σ_1^{LTL} and Σ_2^{LTL} classes.

5 Practically Used Formulas

In order to find in which LTL classes practically used formulas lie, we have studied properties from the "Specification Patterns System" [7]. This system is a collection of the most often used model checking properties. The system provides properties in several different formalisms, LTL being one of them, together with their frequency (i.e. how often they are used) of each of them. We have observed following characteristics of these properties:

- The next operator is not used in these patterns, i.e. practically used properties are stutter-invariant.
- The most often used formulas are very short (three or less temporal operators).
- The nesting depth of until operator is in most cases one or two. The nesting depth is greater than three only in very rare cases.
- Most properties are either from safety class (41%) or from recurrence class (54%). This means that in most cases the resulting automaton is specialized[1].

The last observation is supported by Kupferman and Vardi [16] who claim that: "... most of the LTL formulas ψ used in practice are such that $A_{\neg\psi}$ is a 1-weak automaton" (1-weak automata are strict subclass of weak automata).

6 Conclusions and Future Work

The worst case complexity is not much useful with respect to the practical model checking. In practice, the computational complexity caused by LTL formula is "shadowed" by the size of a verified system. Thus it is important to use algorithms efficient with respect to a system. We conclude

[1] Please remember that in the verification process the negation of the formula is translated into an automaton. Therefore recurrence and safety formulas are translated into weak and terminal automata respectively.

that important classes of LTL are these which enable the use of such efficient algorithms — stutter-invariant class $L(\mathtt{U})$, which enable partial order reduction, and newly identified classes Σ_1^{LTL} (safety properties) and Σ_2^{LTL} (recurrence properties), which enable the use of specialized algorithms for non-emptiness check.

Our intended future works is to try to find out some new connections among other hierarchies and model checking. Particularly, we would like to study the connections with bounded model checking [1].

Acknowledgment

I thank Ivana Černá, my supervisor, for many valuable discussions and Jan Strejček for reading a draft of the paper.
The work on this paper has been partially supported by GA ČR grant number 201/03/0509.

References

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In W. R. Cleaveland, editor, *Proc. Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.

[2] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Proc. Computer Aided Verification*, volume 1633 of *LNCS*, pages 222–235. Springer, 1999.

[3] I. Černá and R. Pelánek. Distributed explicit fair cycle detection. In *Proc. SPIN workshop*, number 2648 in LNCS. Springer, 2003.

[4] E. Y. Chang, Z. Manna, and A. Pnueli. Characterization of temporal property classes. In *Proc. Automata, Languages and Programming*, volume 623 of *LNCS*, pages 474–486. Springer, 1992.

[5] D.R. Dams. Flat fragments of CTL and CTL*: Separating the expressive and distinguishing powers. *Logic Journal of the IGPL*, 7(1):55–78, 1999.

[6] S. Demri and Ph. Schnoebelen. The complexity of propositional linear temporal logics in simple cases (extended abstract). In *Proc. 15th Ann. Symp. Theoretical Aspects of Computer Science*, volume 1373, pages 61–72. Springer, 1998.

[7] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In *Proc. Workshop on Formal Methods in Software Practice*, pages 7–15. ACM Press, 1998.

[8] S. Edelkamp, A. L. Lafuente, and S. Leue. Directed explicit model checking with HSF-SPIN. In *Proc. SPIN workshop*, volume 2057 of *LNCS*, pages 57–79. Springer, 2001.

[9] E. A. Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B, Formal Models and Semantics*, pages 995–1072. North-Holland Publishing Company, 1990.

[10] E. A. Emerson and C. L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proc. IEEE Symposium on Logic in Comuter Science*, pages 267 – 278. Computer Society Press, 1986.

[11] K. Etessami and T. Wilke. An Until hierarchy for temporal logic. In *Proc. IEEE Symposium on Logic in Computer Science*, pages 108–117. Computer Society Press, 1996.

[12] K. Fisler, R. Fraer, G. Kamhi Y. Vardi, and Z. Yang. Is there a best symbolic cycle-detection algorithm? In *Proc. Tools and Algorithms for Construction and Analysis of Systems*, volume 2031 of *LNCS*, pages 420–434. Springer, 2001.

[13] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. Protocol Specification Testing and Verification*, pages 3–18. Chapman & Hall, 1995.

[14] G. J. Holzmann, D. Peled, and M. Yannakakis. On nested depth first search. In *Proc. SPIN Workshop*, pages 23–32. American Mathematical Society, 1996.

[15] A. Kučera and J. Strejček. The stuttering principle revisited: On the expressiveness of nested X and U operators in the logic LTL. In *Proc. Computer Science Logic*, volume 2471 of *LNCS*, pages 276–291. Springer, 2002.

[16] O. Kupferman and M. Y. Vardi. Relating linear and branching model checking. In *Proc. Programming Concepts and Method*, pages 304–326. Chapman & Hall, 1998.

[17] R. P. Kurshan, V. Levin, and H. Yenigun. Compressing transitions for model checking. In *Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 569–581, 2002.

[18] L. Lamport. Proving correctness of multiprocess programs. *IEEE Transactions Software Engineering*, (3):125–143, 1977.

[19] L. H. Landweber. Decision problems for omega-automata. *Mathematical Systems Theory*, 3(4):376–384, 1969.

[20] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *Proc. Logic in Computer Science (LICS)*. IEEE Computer Society, 2002.

[21] M. Maidl. The common fragment of CTL and LTL. In *Proc. 41th Annual Symposium on Foundations of Computer Science*, pages 643–652, 2000.

[22] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proc. ACM Symposium on Principles of Distributed Computing*, pages 377–410. ACM Press, 1990.

[23] N. Markey. Past is for free: on the complexity of verifying linear temporal properties with past. In *Proc. EXPRESS*, volume 68. Elsevier, 2002.

[24] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM (JACM)*, 32(3):733–749, 1985.

[25] D. Therien and T. Wilke. Nesting Until and Since in linear temporal logic. In *Proc. Symposium on Theoretical Aspects of Computer Science*, volume 2285 of *LNCS*, pages 455–464. Springer, 2002.

[26] M. Y. Vardi. Branching vs. linear time: Final showdown. In *Proc. Tools and Algorithms for Construction and Analysis of Systems*, volume 2031 of *LNCS*, pages 1–22. Springer, 2001.

[27] T. Wilke. Classifying discrete temporal properties. In *Symposium on Theoretical Aspects of Computer Science*, pages 32–46, 1999.

Model Checking Epistemic Properties of Interpreted Systems

Franco Raimondi
Department of Computer Science - King's College
Strand, London WC2R 2LS
franco@dcs.kcl.ac.uk

Abstract. Multi-agent systems are often taken as a paradigm in the specification of complex systems because of their ability to abstract away from implementation details. Different kinds of modal logics have been used to model agents' knowledge/beliefs/desires and their evolution with time. In this paper we investigate how model checking techniques can be applied to some problems of verification in multi-agent systems. We present results achieved in the verification of static epistemic properties of two examples encoded in the formalism of Interpreted Systems: the bit transmission problem and the protocol of the dining cryptographers.

1 Introduction

Rational agents in computer science [28] can be seen as the abstraction of any piece of hardware or software enjoying autonomy, social ability, reactivity, and pro-activity. The use of *multi-agent systems* (MAS) in the design of complex systems, where implementations details are unnecessary or unknown [17], has been proven an appealing and successful paradigm: examples include information retrieval using software agents, online auctions, protocols for communication, etc.

In the past twenty years, various logical theories have been developed to formalise *knowledge*, *beliefs*, *desires*, *intentions* and other *intentional attitudes* (in the sense of [9]) of MAS. Such logics include Cohen and Levesque's theory of intention [8], Rao and Georgeff BDI architecture [21], Wooldridge's Logic for Rational Agents (LORA, [26]), and Interpreted Systems by Fagin, Halpern, Moses and Vardi [10].

Comparatively, less effort has been put in the *verification* of MAS. As suggested in [11], model checking techniques can be applied in the verification of MAS. Various results along these lines have been achieved recently by Benerecetti, Wooldrige, Bordini, van der Meyden and others [1, 14, 23, 27, 2, 22, 13, 19]. In this paper we employ Interpreted Systems [10] to specify MAS because they provide a *computationally grounded* theory of agency.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

The notion of *computationally grounded* theory of agency was introduced by M. Wooldridge in [25] to denote a theory that can be interpreted in terms of some concrete computational model.

By means of two well-known examples from the AI arena (the bit transmission problem [10] and the protocol of the dining cryptographers [5]), we present different methodologies that can be employed in the verification of epistemic, temporal and deontic operators in Interpreted Systems.

The rest of the paper is organised as follows: in the next section we introduce the formalism of Interpreted Systems and model checking. Section 3 presents the main results in model checking "static" formulae. We conclude in Section 4 by suggesting future development and by comparing our work to the state of the art in this field.

2 Preliminaries

Due to space considerations, we refer to [10, 15] for details on Interpreted Systems extended with deontic operators. Model checking is explored fully in [7].

2.1 Interpreted Systems

Consider n agents in a system and n non-empty sets $L_1, \ldots, L_n$ of local states, one for every agent of the system. Elements of L_i will be denoted by $l_1, l_1', l_2, l_2', \ldots$. For every agent of the system there is a set Act_i of actions that the agent can perform. Actions are not executed randomly, but following particular specifications called protocols. A protocol P_i for agent i is a function from the set L_i of local states to a non-empty set of actions Act_i

$$P_i : L_i \rightarrow 2^{Act_i}.$$

A *system of global states for n agents* S is a non-empty subset of the cartesian product $L_1 \times \ldots \times L_n$. A global state g of a system S is a tuple of the form $g = (l_1, \ldots, l_n)$. $l_i(g)$ denotes the local state of agent i in global state g.

The evolution of the system can be modelled by means of a transition function π from global states and joint actions to global states:

$$\pi : S \times Act \rightarrow S$$

where $Act = Act_1 \times \ldots \times Act_n$ is the set of joint actions for the system. This defines temporal flows on the set of global states. A *run* r is a function from time to global states: $r : N \rightarrow S$. Hence, a run is a sequence of global states obtained by applying the function π to global states and joint actions. Notice that, in the definition of π, all the local states are updated synchronously at each temporal step.

Given a set of propositional variables P, an *Interpreted* System of global states is a pair $IS = (S, h)$ where S is a system of global states and

$h : S \to 2^P$ is an interpretation function; intuitively, h returns the propositional variables true in a global state. Interpreted systems can then be used to model time and knowledge. The logic CTLK (Computation Tree Logic of Knowledge [13]) is an extensions of CTL to include epistemic operators. For instance, epistemic modalities K_i (one for each agent) are interpreted as follows:

$$(IS, g) \models K_i \varphi \quad \text{if for all } g' \; l_i(g) = l_i(g') \text{ implies } (IS, g') \models \varphi.$$

(intuitively, $K_i \varphi$ captures the idea that agent i knows φ, i.e. φ holds in every state that i considers possible).

The notion of interpreted systems can be extended to incorporate the idea of correct functioning behaviour of some or all of the components [16]. Given n agents and n non-empty sets $G_1, \ldots, G_n$, a *deontic system of global states* is any system of global states defined on $L_1 \supseteq G_1, \ldots, L_n \supseteq G_n$. G_i is called the *set of green states for agent i*. The complement of G_i with respect to L_i, denoted with R_i, is called the *set of red states for agent i*. Deontic systems of global states are used to interpret other modalities on top of CTLK, such as the following one:

$$(IS, g) \models O_i \varphi \quad \text{if for all } g' \text{ we have that } l_i(g') \in G_i \text{ implies } (IS, g') \models \varphi.$$

$O_i \varphi$ is used to represent that φ holds in all (global) states in which agent i is functioning correctly. Another concept of particular interest is the knowledge that an agent i has *on the assumption that the system (the environment, agent j, group of agents X) is functioning correctly*. We employ the (doubly relativised) modal operator $\widehat{K}_i^j$ for this notion, which is interpreted as follows:

$$(IS, g) \models \widehat{K}_i^j \varphi \quad \text{if for all } g' \text{ such that } l_i(g) = l_i(g') \text{ and } l_j(g') \in G_j \text{ we have that } (IS, g') \models \varphi.$$

The resulting logic for modalities K_i is $S5_n$; this models agents with complete introspection capabilities and veridical knowledge. Completeness results can be shown for the O_i as well (see [15]).

2.2 Model Checking and SMV

Given a program P and a property that can be represented as a logical formula φ in some (temporal) logic, model checking techniques allow for the automatic verification of whether or not a model M_P, representing the program P, satisfies the formula φ.

In the last two decades there have been great advances in the effectiveness of this approach thanks to sophisticated data manipulation techniques. Techniques based on Binary Decision Diagrams (BDDs, [3]) have been used

to develop model checkers that are able to check large number of states [4]. Alternative approaches using automata have also been developed [24].

Software tools originated from these lines of research. SPIN (see [12]) exploits automata theory and related algorithms, while SMV [18] uses BDDs to represent states and transitions. In this paper we will use NuSMV, a novel implementation of SMV [6].

By means of these softwares a large number of systems ranging from communication protocols to hardware components have been verified. These are not agent systems, but standard distributed processes.

3 Static model checking of Interpreted Systems

In this section we present a methodology to check *epistemic formulae* expressing properties of an Interpreted System. We are interested in the verification of formulae involving epistemic modalities only, and we do not consider temporal operators in the formulae we want to check. We call this methodology *static model checking*.

Given an Interpreted System IS, it is possible to build an SMV program P_{IS} such that the set of states in the temporal model generated by P_{IS} has a one-to-one correspondence with global states of the Interpreted System. The SMV program is constructed as follows[1]:

1. We declare a variable for each agent, where the values of local states are stored.
2. We declare an array of actions for each agent. Intuitively, this array contains the actions "enabled" in a local state, with respect to the protocol for each agent.
3. Actions are synchronised with local states, for each agent and in each state.
4. The conditions for the evolution of the SMV model are obtained from the evolution function of the Interpreted System.
5. Propositions are defined using the function h of the Interpreted System.

The translation into SMV is manual when deontic operators have to be checked, as in Section 3.1. In Section 3.2 the SMV code is generated automatically. In both cases, there is a one-to-one correspondence between global states of the Interpreted System and SMV states.

NuSMV provides an operator to compute the set of "reachable states"; this set is needed to build epistemic and deontic relations between states to evaluate formulae involving epistemic and deontic operators. Specifically, we have built a parser that takes the set of reachable states as input and produces a model with epistemic and deontic relations in the format of Akka, a Kripke model editor[2]. We chose to use Akka because the Kripke

[1] The methodology presented here is a revised version of the one in [14].

[2] `http://turing.wins.uva.nl/~lhendrik/`

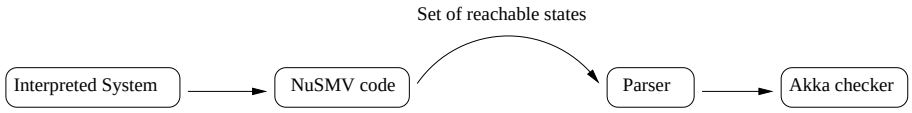


Figure 1: The methodology for static model checking.

model can be provided via a simple text file. Moreover, Akka allows for the verification of multiple modalities. Unfortunately, the representation of states in Akka is not symbolic, but in most cases the set of reachable states is orders of magnitude smaller than the full cartesian product of local states, and reachable states are computed symbolically by NuSMV.

The overall methodology is summarised in Fig. 1. In our examples, the translation into SMV code, the computation of reachable states with NuSMV, and the parsing of NuSMV output required less than one second on a 1GHz PC with 256MBytes of RAM.

3.1 Example: the bit transmission problem with faults

The bit-transmission problem [10] involves two agents, a *sender* S, and a *receiver* R, communicating over a faulty communication channel. The channel may drop messages but will not flip the value of a bit being sent. S wants to communicate some information—the value of a bit for the sake of the example—to R. One protocol for achieving this is as follows. S immediately starts sending the bit to R, and continues to do so until it receives an acknowledgement from R. R does nothing until it receives the bit; from then on it sends acknowledgements of receipt to S. S stops sending the bit to R when it receives an acknowledgement. Note that R will continue sending acknowledgements even after S has received its acknowledgement. Intuitively S will know for sure that the bit has been received by R when it gets an acknowledgement from R. R, on the other hand, will never be able to know whether its acknowledgement has been received since S does not answer the acknowledgement. Using the methodology suggested we can check mechanically that the protocol above guarantees that when sender receives the acknowledgement it then knows (in the information-theoretic sense defined in Section 2) that the receiver knows the value of the bit (see [14]). We exemplify the methodology with a slightly more complicated scenario: we assume that the receiver R may send acknowledgements even when it is not supposed to. We define the following local states for S and R:

$$L_S = G_S = \{0, 1, (0, ack), (1, ack)\}, \quad R_S = \emptyset.$$
$$G_R = \{0, 1, \epsilon\},\ R_R = \{(0, f), (1, f), (\epsilon, f)\}, L_R = G_R \cup R_R.$$

The local states for S represent the value of the bit S is attempting to transmit (0 or 1), and whether or not S has received an acknowledgement from R ($(0, ack)$ or $(1, ack)$). For R, the local states 0 and 1 represent the

value of the received bit; ϵ is a circumstance under which no bit has been received yet; the last three faulty states correspond the fact that R sent a "faulty" acknowledgement. The faulty line can be modelled using another agent. We report here the protocol of the receiver as an example:

$$P_R(\epsilon) = \lambda,$$
$$P_R(0) = P_R(1) = sendack$$
$$P_R((0, f)) = P_R((1, f)) = P_R((\epsilon, f)) = \{sendack, \lambda\}$$

Due to space constraints we cannot report here the full definition of all the parameters (actions, protocols and evolution function), but these can be found in [14] and in the NuSMV code available online at
`http://www.dcs.kcl.ac.uk/pg/franco/is/btpfaults2.smv`.

After translating the Interpreted System into SMV language, we can obtain the set of reachable states running NuSMV; then, with Akka, it is possible to check that none of the epistemic formulae that were true for the basic case (no faults) hold in this version of the protocol. However, a particular form of knowledge still holds. Intuitively if S could make the assumption of R's correct functioning behaviour, then, upon receipt of an acknowledgement, it would make sense for it to assume that R does know the value of the bit; this is exactly the meaning of $\widehat{K}_S^R$. And indeed, using Akka we are able to check the validity of the following formulae in the model, formalising the ideas expressed above[3].

$$IS \models \mathbf{recack} \rightarrow \widehat{K}_S^R(K_R(\mathbf{bit} = \mathbf{0}) \vee K_R(\mathbf{bit} = \mathbf{1}))$$
$$IS \models \mathbf{recack} \wedge (\mathbf{bit} = \mathbf{0}) \rightarrow \widehat{K}_S^R K_R(\mathbf{bit} = \mathbf{0})$$

3.2 Example: the protocol of the dining cryptographers

The methodology presented in the previous section can be improved: we can specify an Interpreted System using XML and generate the SMV code mechanically using a Java translator [19]. This improved methodology is illustrated in Fig. 2. We did not include deontic operators in the Java translator and in the XML specification, but nevertheless we have been able to check a well-known example, the protocol of the dining cryptographers. The general aim of the protocol is to allow an untraceable broadcasting of messages in multi-agent systems, and is originally introduced with the following example:

"*Three cryptographers are sitting down to dinner at their favourite three-star restaurant. Their waiter informs them that arrangements have been made with the maitre d'hotel for the bill to be paid anonymously. One of the cryptographers might be paying for the dinner, or it might have been NSA (U.S. National Security Agency). The three cryptographers respect each other's right to make an anonymous payment, but they wonder if NSA*

[3]The interpretation for the atoms can be found in [14].

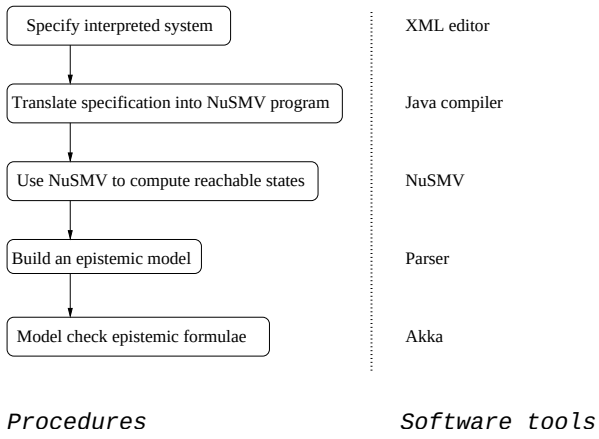


Figure 2: An improved methodology for static model checking.

is paying. They resolve their uncertainty fairly by carrying out the following protocol: Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer on his right, so that only the two of them can see the outcome. Each cryptographer then states aloud whether the two coins he can see – the one he flipped and the one his left-hand neighbour flipped – fell on the same side or on different sides. If one of the cryptographers is the payer, he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number indicates that NSA is paying (assuming that the dinner was paid for only once). Yet if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is."[5]

As with the example of the bit transmission problem, we can model the protocol with Interpreted Systems. We introduce three agents C_i, $i = 1, 2, 3$, to model the three cryptographers, and one agent E for the Environment. By running the Java translator, NuSMV, and Akka, we can formally check that the following propositions hold in the Interpreted System of the dining cryptographers (details can be found in [19]):

$$IS \models \mathbf{odd} \rightarrow (\neg \mathbf{paid_1} \rightarrow (K_{C_1}(\mathbf{paid_2} \vee \mathbf{paid_3}) \wedge \neg K_{C_1}(\mathbf{paid_2}) \wedge \neg K_{C_1}(\mathbf{paid_2})))$$

$$IS \models \mathbf{even} \rightarrow K_{C_1}(\neg \mathbf{paid_1} \wedge \neg \mathbf{paid_2} \wedge \neg \mathbf{paid_3})$$

These two formulae confirm the correctness of the statement: if the first cryptographer did not pay for the dinner and there is an odd number of differences in the utterances, then the first cryptographer knows that either the second or the third cryptographer paid for the dinner; moreover, in this case, the first cryptographer does not know which one of the remaining cryptographers is the payer. Conversely, if the number of differences in the

utterances is odd, then the first cryptographer knows that nobody paid for the dinner.

4 Conclusions

In this paper we have presented results on model checking knowledge in multi-agent systems with two examples. Though the methodology applies to "static" formulae only, we think that these results are particularly interesting because we have been able to model check knowledge in a system that has a clear correspondence with computational states of agents, and is logically complete.

We have explored the issue of verification in MAS. Recently, different works have tackled this subject. In [27], M. Wooldridge et al. present the MABLE language for the specification of MAS. Modalities are modelled as nested data structures, in the spirit of [1]. Bordini et al [2] use a modified version of the AgentSpeak(L) language [20] to specify agents and to exploit existing model checkers.

The works of van der Meyden and Shilov [22], and van der Meyden and Su [23], are probably more related to this paper. They consider the verification of a particular class of Interpreted Systems, namely the class of synchronous distributed systems with perfect recall. An algorithm for model checking is introduced in the first paper, but the algorithm is not implemented and specification is not considered. In [23] verification is limited to a specific class of temporal specifications.

We have obtained preliminary results in the verification of a richer language involving temporal operators and epistemic operators, along different extensions of the methodology presented here. In parallel, we are currently investigating theoretical aspects of the $\widehat{K}_i^j$ operator, extending the work of [15].

References

[1] M. Benerecetti, F. Giunchiglia, and L. Serafini. Model checking multiagent systems. *Journal of Logic and Computation*, 8(3):401–423, June 1998.

[2] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking agentspeak. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, July 2003.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, pages 677–691, Aug. 1986.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[5] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.

[6] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model verifier. In N. Halbwachs and D. Peled, editors, *Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 495–??, 1999.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[8] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence, AI*, 42(2–3):213–261, Mar. 1990.

[9] D. C. Dennett. *The intentional stance*. The MIT Press, Massachusetts, 1987. 388 pages, 1987.

[10] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. The MIT Press, Cambridge, Massachusetts, 1995.

[11] J. Halpern and M. Vardi. *Model checking vs. theorem proving: a manifesto*, pages 151–176. Artificial Intelligence and Mathematical Theory of Computation. Academic Press, Inc, 1991.

[12] G. J. Holzmann. The model checker spin. *IEEE transaction on software engineering*, 23(5), May 1997.

[13] A. Lomuscio and W. Penczek. Bounded model checking for interpreted systems. Technical report, Institute of Computer Science of the Polish Academy of Sciences, 2002.

[14] A. Lomuscio, F. Raimondi, and M. Sergot. Towards model checking interpreted systems. In *Proceedings of MoChArt*, Lyon, France, August 2002.

[15] A. Lomuscio and M. Sergot. Investigations in grounded semantics for multi-agent systems specifications via deontic logic. Technical report, Imperial College, London, UK, 2000.

[16] A. Lomuscio and M. Sergot. On multi-agent systems specification via deontic logic. In J.-J. Meyer, editor, *Proceedings of ATAL 2001*. Springer Verlag, July 2001. To Appear.

[17] J. McCarthy. Ascribing mental qualities to machines. In M. Ringle, editor, *Philosophical Perspectives in Artificial Intelligence*. Harvester Press, 1979.

[18] K. McMillan. *Symbolic model checking: An approach to the state explosion problem*. Kluwer Academic Publishers, 1993.

[19] F. Raimondi and A. Lomuscio. A tool for specification and verification of epistemic and temporal properties of multi-agent system. Submitted, 2003.

[20] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. *Lecture Notes in Computer Science*, 1038:42–??, 1996.

[21] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann Publishers, Apr. 1991.

[22] R. van der Meyden and N. V. Shilov. Model checking knowledge and time in systems with perfect recall. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 19, 1999.

[23] R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. Submitted, 2002.

[24] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logic in Computer Science (LICS'86)*, pages 332–345, Washington, D.C., USA, June 1986. IEEE Computer Society Press.

[25] M. Wooldridge. Computationally grounded theories of agency. In E. Durfee, editor, *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS 2000)*. IEEE Press, July 2000.

[26] M. Wooldridge. *Reasoning about rational agents*. MIT Press, July 2000.

[27] M. Wooldridge, M. Fisher, M.-P. Huget, and S. Parsons. Model checking multi-agent systems with MABLE. In M. Gini, T. Ishida, C. Castelfranchi, and W. L. Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 952–959. ACM Press, July 2002.

[28] M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995.

A Formal Representation of Korean Temporal Marker *dongan*

Hyunjung Son
EHESS Paris,
hyunjung_son@hotmail.com

Abstract. This paper has two objectives. The first is to account for the semantic functions of Korean temporal marker *dongan* in comparison with English marker *for*. The second objective is to represent these functions in terms of type theory, which allows us to establish semantic representations in a coherent and explicit way.

1 Introduction

Comparing cross-lingual semantics contributes to the understanding of the nature of linguistic meaning in two different ways; first, it discovers and verifies the principles representing the universal phenomena across natural languages, and second, it provides evidence to support the claim of language specificity. In this paper, we attempt to describe the semantic characteristics of Korean temporal marker *dongan*, in comparison to English temporal marker *for*. The formal representation of the observed temporal properties is based on type theory of Andrews (1986), Hindley et al.(1986) and Renaud (1996).

2 Semantic Representation

Our semantic representation of temporal marker *dongan* is based on the following assumptions. First, we regard the event described by a sentence as a primitive entity like individuals and denote it by a variable 'e' in line with Davidson (1967). Second, we assume that the linguistic time is ordered, discrete, infinite and consisting of instants corresponding to the natural numbers. Thus, the linguistic time can be expressed with one of these three notions: instant, extended interval and duration. Instants are unitary constituents of linguistic time and noted by a quintuplet of natural numbers [x1,x2,x3,x4,x5] of which x1 stands for year, x2 for month, x3 for day, x4 for hour and x5 for minute.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

(ex) at 3 o'clock on April 5th, 2003: instant [2003,4,5,3,0]

An extended interval is a set of consecutive instants determined by a beginning instant and an ending instant.

(ex) on April 5th, 2003: interval [[2003,4,5,0,0], [2003,4,5,23,59]]

Duration refers to a temporal distance between two distinct instants.

(ex) for 5 years: duration [5,_,_,_,_]

For the purpose of temporal description of a sentential event, we defined the following types and functional terms on the basis of type theory. The symbol λ stands for abstraction and $\bullet$ stands for application.

2.1 Definitions of Types

- i : type symbol denoting the type of individuals
- p : type symbol denoting the type of propositions
- e : type symbol denoting the type of events
- *ent*: type symbol denoting the type of natural numbers
- *inst* : type symbol denoting the type of instants
- *inter* : type symbol denoting the type of extended intervals
- *dur*: type symbol denoting the type of durations

Type symbols may be omitted when no ambiguity is introduced.

2.2 Definitions of Functional Terms

- (λe. moment$\bullet$e): e→inst

  Applying this function to any argument of type *e*, we obtain the moment of e of type *inst*.

- (λe. interv$\bullet$e): e→inter

  Applying this function to any argument of type *e*, we obtain the interval of e of type *inter*.

- (λx. beginning$\bullet$x): e→inst / (λx. ending$\bullet$x): e→inst

  Applying these functions to any argument x of type *e*, we obtain the beginning/ending instant of x of type *inst*.

- (λx. duration•x): e→dur

  Applying this function to any argument x of type *e*, we obtain the duration of x of type *dur*.

- (λx. beg•x): inter→inst / (λx. end•x): inter→inst

  Applying this function to any argument x of type *inter*, we obtain the beginning/ending instant of x of type *inst*. By definition, beg•[A,B] = A and end•[A,B] = B

- (λx. length•x): inter→dur

  Applying this function to any argument x of type *inter*, we obtain the length of x of type *dur*. By definition, length•[A,B]= |B-A|

- (λxλy. x $<_{\ll t \gg}$ y): inst→inst→p

  It denotes that x of type *inst* is anterior to y of the same type. When no ambiguity is introduced, $\ll$ t $\gg$ will be omitted.

- (λxλy. x $=_{\ll t \gg}$ y): inst→inst→p

  It denotes that x and y of type *inst* are simultaneous.

- (λxλy. x $\leq_{\ll t \gg}$ y): inst→inst→p

  It denotes that λxλy. (x $<_{\ll t \gg}$ y $\vee$ x $=_{\ll t \gg}$ y).

- (λxλy. x $\in_{\ll t \gg}$ y): inst→inter→p

  It denotes that x of type *inst* is a member of y of type *inter*. By definition, λxλy. (beg•y $\leq_{\ll t \gg}$ x $\leq_{\ll t \gg}$ end•y)

- (λxλy. x $\subset_{\ll t \gg}$ y): inter→inter→p

  It denotes that x of type *inter* is included by y of the same type. By definition, λxλy. (beg•y < beg•x $\wedge$ end•x < end•y).

- (λxλy. x $=_{\ll t \gg}$ y): inter→inter→p

  x and y of type *inter* are simultaneous. By definition, λxλy. (beg•x=beg•y $\wedge$ end•x=end•y).

- (λxλy. x $\subseteq_{\ll t \gg}$ y): inter→inter→p

  It denotes that λxλy. (beg•y $\leq_{\ll t \gg}$ beg•x $\wedge$ end•x $\leq_{\ll t \gg}$ end•y).

3 Semantic Description of *dongan*-Adverbials

It is generally assumed that Korean tense or aspect is signaled by the infix placed between the infinitive and the terminal suffix in VP.

(1) ***hanshigan dongan*** *kwanghoe bihaenggiga* ***naratta***[1]
an hour/***dongan***/of *Kwangho*/airplane-NOM/**fly-PA-DEC**
Kwangho's airplane flew for an hour.

The VP of this example '*naratta*' comprises infinitive '*na-*', tense/aspect marker '*-at-*' and declarative suffix '*-ta*'. As for the infix '*-at-*', there has been controversy over the semantic roles that it plays, since it brings about several time interpretation possibilities such as simple past, completion, resultant state and progressiveness[2]. This multiple ambiguity can be remedied by time adverbials which contribute to specifying the sentential event in time.

The temporal marker *dongan* has been considered as equivalent to the English marker *for*, as it constructs a durative time adverbial[3] with a NP. For example, (1) is represented as:

(sr 1) $\exists e \exists x$ airplane$\bullet$x $\wedge$ of$\bullet$x$\bullet$kwangho $\wedge$ fly$\bullet$e$\bullet$x $\wedge$
ending$\bullet$e$<$pt_speech[4] $\wedge$ **duration$\bullet$e=(1 hour)**

Thus, the time adverbials with *dongan* have been used for aspectual classifications of Korean verbs on the basis of the ideas of Vendler (1967) and Dowty (1979)[5]. However, the analysis of three thousand sentences extended by a *dongan*-adverbial[6] shows a different result. First, *dongan* allows five distinct types of syntactic structures other than durative NP to construct time adverbials. Moreover, not all these adverbials give rise to the durative meaning, as illustrated by the following examples:

◦ Interval Noun + *dongan*:

(2) ***kyôul banghak dongan*** *nanun shine daehae saenggak'agi chijak'aetta*
winter vacation/***dongan***/I-TOP/about God/think/begin-PA-DEC
During the winter vacation, I began to think about God.

which meaning is:

[1]We used the McCune-Reischauer system to transcribe the Korean data. The following abbreviations are used for glossing grammatical morphemes:
ACC: accusative, AS: attributive suffix, CIRCUM: circumstantial, CL: classifier, CON: connective, DEC: declarative, DUR: durative, INT: interrogative, LOC: locative, NOM: nominative, NS: nominal suffix, PA: past, TOP:topic.

[2]See Lee, H. (1993), Lee, Ch. (1987) and Lee, J.-R. (1982) for more detailed discussion.

[3]Kim (1981), Lee, Ch (1982).

[4]Reichenbach (1966).

[5]Jo (2000), Jung (1984) and Lee, J.-R. (1982).

[6]The sentences are from Yonsei malmunchi corpus built by Yonsei Center for Linguistic Information.

(sr 2) ∃I∃e winter_vacation•I ∧ interval•I ∧ begin•e•(λe1λy. think_about•e1•god•y)•speaker ∧ moment•e<pt_speech ∧ **moment•e∈I**

That is, the moment of the punctual event is identified as an instant belonging to the extended interval indicated by the *dongan*-adverbial.

∘ Interval NP + durative NP + *dongan*:

(3) ***shiwol han dal dongan*** *nalssiga choatta*
October/one month/*dongan*/the weather-NOM/be_nice-PA-DEC
For the month of October, the weather was nice.

(sr 3) ∃e∃I nice•e•weather ∧ I=(October) ∧ length•I=(1 month) ∧ beginning•e<pt_speech ∧ **I⊆interv•e**

It states that the event described by the nuclear sentence[7] can be extended beyond the interval denoted by the *dongan*-adverbial. Thus, it is difficult, in this case, to assign the maximal length to the event.

∘ Deictic/anaphoric determiner + durative NP + *dongan*:

(4) *chônun* ***i han dal dongan*** *haengbok'aessôyô*
I-TOP/**this/one month/*dongan*** /be_happy-PA-DEC
(For the interval of) Last month, I was happy.

(sr 4) ∃e∃I happy•e•speaker ∧ I=rech_interv•pt_speech•year•1[8] ∧ length•I=[0,1,0,0,0] ∧ beginning•e<pt_speech ∧ **I⊆interv•e**

Here again, the interval denoted by the time adverbial is included by the interval of the event.

∘ Attributive clause + *dongan*:

[7] We call the semantically and syntactically independent sentences without modifiers such as time adverbials 'nuclear sentence'.

[8] This function returns a relevant interval consulting the point of speech. It is defined in Prolog as follows:

```
rech_interv(I,J,K,R) :-
    I==pt_speech, !, rech_int(J,K,R).
rech_interv(I,J,K,R) :-
    I==pt_ref, rech_int_ref(J,K,R).
rech_int(year,[X,_,_,_,_],R):-
    time([A,M,_,_,_]),
    A1 is A-X,
    R = [[A1,M,_,_,_],[A,M,_,_,_]].
```

(5) ***kunyôga bôsu chôngrujangul hanghae kônnun dongan*** *nuni naerigi shijakhaetta*
she-NOM/to the bus station/walk-AS/*dongan*/snow-NOM/fall-NS/begin-PA-DEC
While she walked to the bus station, the snow began to fall.

(sr 5) $\exists x \exists e1 \exists e2$ bus_station•x $\wedge$ walk•e1•she $\wedge$ to•e1•x $\wedge$ beginning•e1<pt_speech $\wedge$ begin•e2•(λe. fall•e•snow) $\wedge$ moment•e2<pt_speech $\wedge$ **moment•e2** $\in$ **interv•e1**

The *dongan*-adverbial of (5) correlates two different events; the time of the event described in the main clause (e2) is determined in reference to that of the event indicated by the attributive clause (e1); the moment of (e2) is identified as an instant belonging to the interval of (e1).

$\circ$ Attributive clause + durative NP + *dongan*:

(6) ***naega chibul biun du chu dongan*** *madange haebarakiga mani charatta*
I-NOM/home-ACC/ be out-AS/ two week/ *dongan*/ front yard-LOC/ sunflower-NOM/a lot/grow_up-AP-DEC
For the two weeks when I was out home, the sunflower in the garden has grown up a lot.

(sr 6) $\exists e1 \exists x \exists e2$ be_out•e1•home•speaker $\wedge$ duration•e1=(2 weeks) $\wedge$ ending•e1<pt_speech $\wedge$ sunflower•x $\wedge$ of•front_yard•x $\wedge$ grow_up•e2•x $\wedge$ beginning•e2<pt_speech $\wedge$ **interv•e1$\subseteq$interv•e2**

Similarly, the event time of the main clause (e2) and that of the attributive clause (e1) are temporally related. The only difference with the former case consists in the fact that in this case, (e2) is always expressed by an extended interval and therefore can not be a punctual event[9]

In conclusion, the temporal marker *dongan* gives rise not only to the durative meaning but also the inclusive meaning, whereas *for* triggers only the durative meaning.

The second difference between *dongan*-adverbials and English *for*-adverbials is that only the *dongan*-adverbials are compatible with quantified sentences

[9]Moens and Steedman (1988) introduced the notion of contingency in addition to the temporality in order to account for the infelicity of the following sentence linked by *when*.

*When my car broke down, the sun set.

On the other hand, *dongan* seems to be less influenced by contingency as it is hard to establish any causality or enablement relation between the events linked by *dongan* in (5) and (6). The only relevant extra-linguistic knowledge is whether the related events can happen together in reality.

and the *for*-adverbials are not. This property of the English adverbials has been formalized by Krifka (1989)[10].

(7) *i kongjangun ilnyôn dongan kyôu han dae**ui** **ch'a**rul mandurôtta*
this/factory-TOP/a year/*dongan*/only/**one**/**CL-of**/**car**-ACC/make-PA-DEC
This factory made only one car in a year.

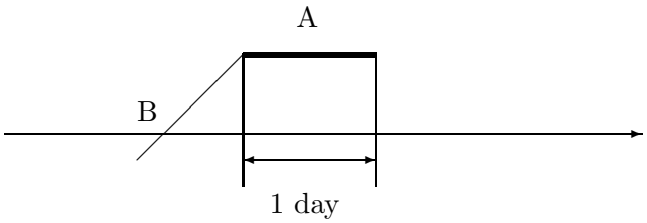
(sr 7) ∃I interval•I ∧ duration•I=[1,0,0,0,0] ∧ | λx. car•x ∧ ∃e construct•e•x•this_factory ∧ ending•e<pt_speech ∧ interv•e⊆I| = 1

The third difference is that *dongan*-adverbials denote not merely the duration of the event but also the duration of the state resulting from the event.

(8) ***haru dongan*** *shiwidaega shichôngul dulrôssatta*
one day/*dongan*/demonstrators-NOM/the city hall-ACC/surround-PA-DEC
The demonstrators surrounded the city hall for one day.

(sr 8) ∃x∃e demonstrator•x ∧ surround•e•city_hall•x ∧ ending•e<pt_speech ∧ ending•(resultant_state•e)<pt_speech ∧ duration•(resultant_state•e)=(1 day)

Note that in this case, it is the state of the city hall's being surrounded by the demonstrators (A), and not the process of surrounding the city hall (B) that lasted one day.



Fourth, unlike the *for*-adverbials, *dongan*-adverbials are compatible with some telic events.

(9) ***han shigan dongan*** *nariga kangul gônnôtta*[11]
one hour/*dongan*/Nari-NOM/the river-ATT/cross-PA-DEC
For an hour, *Nari* has been crossing the river .

[10]The cases that the event is bounded by a quantification modifier are excluded by the wellformedness condition denoted by '/'.
for one hour
λP ∀e [P(e) ∧ h′(e)=1/QMOD$_E$(P,λPλe[P(e)∧ h′(e)=1)]

[11]This example is taken from Kim (1981).

Crossing the river is an action having a terminative point. When associated to such an action, the *dongan*-adverbial indicates its duration, without telling whether its terminative point is reached of not. In (9), we do know that *Nari* has been crossing the river for one hour, but we don't know whether she has arrived at the other side of the river at last, which would be verified by the following possibilities of entailment:

(9′) *han shigan dongan nariga kangul gônnôtta. kuraesô mach'imnae kang kônnôp'yône daatta*
one hour/*dongan*/Nari-NOM/the river-ATT/cross-PA-DEC//and /at last/the river/the other side-LOC/reach-PA-DEC
For an hour, Nari bas been crossing the river. And at last she reached the other side of the river[12].

(9″) *han shigan dongan nariga kangul gônnôtta. kuraedo kangi nômu nôlbôsô ajik kang kônnôp'yône datji mohaetta*
one hour/*dongan*/Nari-NOM/the river-ATT/cross-PA-DEC//however/the river-NOM/too/ wide-CON/yet/the river/the other side-LOC/reach/can not-PA-DEC
For an hour, *Nari* has been crossing the river. However the river was too wide, so she has not yet reached the other side of the river.

Thus, the interpretation of (9) is represented as follows:

(sr 9) $\exists e \exists x$ the_river•x $\wedge$ cross•e•x•nari $\wedge$
Fcpt=λt. distance•t•x $\wedge$ Fcpt•(beginning•e)=ϕ $\wedge$
Fcpt•(ending•e)=<width•x $\wedge$ ending•e<pt_speech $\wedge$
duration•e=(1 hour)

Fcpt is a function measuring the distance of the river that *Nari* has crossed. At the beginning of the event, the distance is zero (Fcpt•(beginning•e)=ϕ) whereas at the end, the distance can be either equal or less than the width of the river (Fcpt•(ending•e)=<width•x). In fact, this function serves to represent the telicity of the event[13].

Fifth, applied to the sentence denoting a set of different events, called multi-occurrent sentence[14], the adverbial with *dongan* determines the distributional nature of the element events.

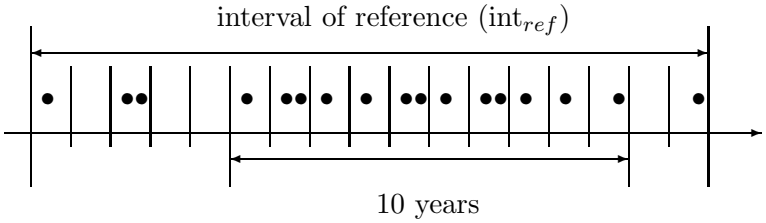
[12]The Korean verb *gônnôda* encompasses the meanings of 'to cross', and 'to be crossing' in English. That's why the sentences containing this verb are compatible with durative adverbials as well as telic adverbials. In fact, there are many other Korean verbs that denote both the telic aspect and the durative aspect of the action.

[13]See Renaud (2002b) for a more formal definition of this measuring function.

[14]Renaud (2002a).

(10) ***shimnyôn dongan*** *yunsôkun mon badaesô kogijabirul haetta*
10 years/*dongan*/Yunsôk-TOP/far ocean-LOC/fishing-ACC/do-PA-DEC
For ten years, *Yunsôk* fished far in the ocean.

That is, the ten years is the maximal length of the interval in which *Yunsôk*'s fishing has been repeated in a consistent way[15].



This interpretation is represented as follows [16] :

(sr 10) λD. $\exists P$ equi-partition•D•P•int_{ref} $\wedge$
$\exists$I I=(λJ. P•J $\wedge$ $\exists$e fish•e•yunsôk $\wedge$ in•e•the_far_ocean $\wedge$
ending•e<pt_speech $\wedge$ interv•e$\subseteq$J) $\wedge$
$\exists$M (max•(λN. N$\subseteq$($\cup$•I) $\wedge$ [inferior•($\cup$•N), superior•($\cup$•N)] $\cap$
int_{ref} = $\cup$•N)•M $\wedge$ duration•M = (10 years))

where the function equi-partition•D•P•L denotes that the element events occurred regularly during the period.

But when the sentence is modified by quantification, the time adverbial no longer imposes the distributional constraint and merely signals the temporal scope of the quantification.

(10′) *shimnyôn dongan yunsôkun mon badaesô* ***se bôn*** *kogijabirul haetta*
10 years/*dongan*/Yunsôk-TOP/far ocean-LOC/**three times**/fishing-ACC/do-PA-DEC
For ten years, Yunsôk fished three times far in the ocean.

(sr 10′) $\exists$I interv•I $\wedge$ duration•I=[10,0,0,0,0] $\wedge$ | λe. fish•e•yunsôk
$\wedge$ in•e•the_far_ocean $\wedge$ ending•e<pt_speech $\wedge$ interval•e$\subseteq$I|=3

Finally, the facts that we observed concerning the temporal adverbials with *dongan* can be summarized as follows:

[15]The occurrences of *Yunsôk*'s fishing are noted by '•' in the figure.

[16]Renaud (2002a) defines the functions used in this formula as :
equi-partition•D•P•N $\equiv$ (N=($\cup$ • P) $\wedge$ | P |>2 $\wedge$
$\forall$K1 K2 ((P•K1 $\wedge$ P•K2 $\wedge$ K1$\neq$K2)$\rightarrow$ (duration•K1=duration•K2=D $\wedge$ K1$\cap$K2=ϕ)))
$\cup$ •R $\equiv$ λx. $\exists$R (R•P $\wedge$ P•x) [A, B] $\equiv$ an interval with the limits A and B
max•E•M $\equiv$ (E•M $\wedge$ $\neg\exists$N(M$\subset$N $\wedge$ E•N))

1. The semantic and syntactic properties of the phrase accompanying the temporal marker play an important role to locate the sentential event in time.

2. To establish the correct temporal interpretation of a sentence, it is crucial to distinguish the mono-occurrent reading from the multi-occurrent reading. The multi-occurrent nature is very often signaled by bare plurals in nominal phrases, adverb like *ch'arero* meaning 'in turn', and quantification modifiers.

3. *dongan*-adverbials don't exclusively belong to one of the categories of time adverbials proposed by Vlach (1993)[17]. They raise durative meaning (ex.1,3,4,8), inclusive meaning (ex. 2,5,6) as well as durative and frequency meaning (ex.10).

4. The quantification negates the frequency meaning brought by the *dongan*-adverbials and gets them to indicate the temporal scope of this semantic operation (ex.10′).

5. The information relevant to the time interpretation is scattered over the whole sentence; not only in VP but also in other sentence constituents such as time adverbials, quantification modifiers and determiners in NP. Therefore, the temporal interpretation of a sentence should be constructed in a compositional way[18].

6. In the same reason, the aspectual value should be attributed to the sentence and not to the VP.

4 Conclusion

In this paper, we formalized the semantic properties of Korean time adverbials with *dongan*, in comparison with English adverbials with *for*. In doing so, we showed that the temporal meanings triggered by *dongan* are not always the same as those by *for*. We also presented some semantic properties that could be relevant to the description of temporal markers of other natural languages as well. We will, in the future, investigate the semantic aspects of Korean telic marker *mane* using the same method.

References

[1] Andrews, P.B., *An Introduction to Mathematical Logic and Type Theory*, Orlando: Academic Press Inc, 1986.

[17] Vlach (1993) proposed four categories of time adverbials: punctual, durative, inclusive and frequency.

[18] On the basis of G Grammar proposed by Renaud (1996), we built a Korean parser in Prolog, which compositionally constructs the semantic representations of Korean sentences extended by *dongan*-adverbials or *oe*-adverbials. See Son (2002, 2003) for more details.

[2] Davidson, D., The logical Form of Action Sentences, in *Essays on Actions and Events*, Oxford: Clarendon Press, 1967 [1980].

[3] Dowty, D.R., *Word Meaning and Montague Grammar*, Dordrecht: Reidel, 1979.

[4] Hindley, J. and Seldin, J.P., *Introduction to Combinators and λ-Calculus*, Cambridge: Cambridge Univ. Press, 1986.

[5] Jo, M.-J., *A Study on the Aspect of Korean* (in Korean), Yonsei Univ. Dissertation, 2000.

[6] Jung, M., "Classification of Korean Verbs with Aspectual Properties" (in Korean), *Study of Grammar* 5 (1984), Seoul: Society of Korean Grammar.

[7] Kim, S.-D., "Aspect of Korean"(in Korean), *Aesan Hakbo* 1 (1981): 25-70, Seoul: *Aesan Hakhoe*.

[8] Krifka, M., "Nominal Reference, Temporal Constitution and Quantification in Event", in Bartsch, R. et al.(eds.), Dordrecht: Foris Publications, 1989.

[9] Moens, M. and Steedman, M., "Temporal Ontology and Temporal Reference", *Computational Linguistics* 14 (1988): 15-28.

[10] Lee, Ch., "Aspects of Aspect in Korean", *Language* 7 (1982): 570-582, Seoul: Korean Linguistic Society.

[11] Lee, H. S., "Tense or aspect : The speaker's communicative goals and concerns as determinant, with reference to the Anterior *-ôss-* in Korean", *Journal of Pragmatics* 20 (1993): 327-358.

[12] Lee, J.-R., "A Study of Aspectual Forms of Modern Korean"(in Korean), *Korean Linguistic Research* 51 (1982), Seoul: Society of Korean Studies.

[13] Reichenbach, H., *The Elements of Symbolic Logic*, New York: The Free Press, 1966.

[14] Renaud, F., *Sémantique du temps et lambda-calculus*, Paris: puf, 1996.

[15] Renaud, F., "Durativité et negation", ms, 2002a.

[16] Renaud, F., "Télicité et quantification", ms, 2002b.

[17] Son, H., "Formal Description of $\ll$ NP+*OE* $\gg$ with Lambda-Calculus and Unification Mechanism"(in Korean), *Annual Meeting of Korean Society for Language and Information* (2002), 77-86.

[18] Son, H., "A Computational Treatment of Korean Temporal Markers, *OE* and *DONGAN*", submitted to *Student Research Workshop of ACL* (2003).

[19] Vendler, Z., *Linguistics in Philosophy*, Ithaca: Cornell Univ. Press, 1967.

[20] Vlach, F., 1993, "Temporal Adverbials, Tense and the Perfect", *Linguistics and Philosophy* 16 (1989), 231-283.

Scalar Implicatures: Exhaustivity and Gricean Reasoning

BENJAMIN SPECTOR
Laboratoire de Linguistique Formelle - Paris VII / Institut Jean-Nicod
benjamin.spector@normalesup.org

ABSTRACT. This paper shows that both scalar implicatures and exhaustification of answers can be understood as the outcome of a pragmatic reasoning based on gricean maxims. I offer a formalization of the gricean reasoning that solves the problems (cf. Chierchia 2001) faced by standard neo-gricean accounts. I further show that positive and non-positive answers pattern very differently, in a way that can be predicted by stating carefully, for a given question-answer pair, what counts as an "alternative answer" - this notion plays the same role as that of "scalar alternative" in previous approaches.

I. Imperfections of Standard Neo-gricean Accounts

According to neo-gricean accounts, scalar implicatures are computed as follows: given a sentence S containing a scalar term t, S is to be compared to all sentences which can be obtained from S by replacing t with a term belonging to t's *scale*. For any such *scalar alternative* S' such that S' asymmetrically entails S, the hearer infers that S' is not part of the speaker's beliefs. (hereafter, rule R1; this derives the so-called *epistemic* implicatures). The underlying principle motivating this inference is Grice's first maxim of *Quantity*. Assuming further that the speaker is maximally informed, the hearer infers that S' is in fact false according to the speaker (hereafter, rule R2).

(1) A or B (2) A and B

Suppose the speaker utters a sentence of the form of (1). Its unique scalar alternative is (2). Since (2) is logically stronger than (1), (2) is not part of the speaker's belief. Moreover, if the speaker is maximally informed, (2) is false, so that *or* in (1) is interpreted as exclusive, even though its literal linguistic meaning is that of inclusive disjunction.

Whatever the merits of this approach (in particular, the fact that it predicts that the exclusive reading of *or* should disappear in monotone decreasing contexts, due to the reversal of entailment patterns), it has been shown to be inaccurate in many cases, especially when a scalar term is interpreted under the scope of some operators. For instance, Chierchia (2001) points out that the neo-gricean procedure yields too weak results for a sentence like (3):

(3) Each of the students read *Othello* or *King Lear*

Proceedings of the Eighth ESSLLI Student Session.
Balder ten Cate (editor)

(3) (sometimes) implicates (4)[1]:

(4) Each of the students read *Othello* or *King Lear* and not both.

The neo-gricean account predicts a much weaker implicature, namely (5):

(5) It is not the case that each of the students read *Othello* and *King Lear*.

Another problem is that the neo-gricean account can also lead to too strong predictions. Take a sentence of the following form: (6) (A or B) or C

Salar alternatives of (6):

a. (A and B) or C b. (A or B) and C c. (A and B) and C

All these alternatives are stronger than (6), so that (6) should implicate that they are all false (by rule R2). In particular, a. should be false, in which case C is. But (6) certainly does not implicate that C is false. Let me call this problem, which is actually very general, that of *unwanted negations*. If, on the other hand, we find a way of blocking this inference, we remain unable to predict that (6) normally implicates that only one of the three disjuncts is true.

I. 1 Chierchia's localist solution

Chierchia (2001) presents a solution based on a recursive interpretation function which computes "strengthened meanings" in tandem with the interpretation function that computes "literal meanings". For him, scalar implicatures are simply an additional dimension of meaning, and the link between scalar implicatures and general principles of conversational rationality becomes less clear, even though some basic aspects of the neo-gricean approaches are retained.

Hereafter, I will defend a "globalist" approach to scalar implicatures, in the sense that it relies on the natural hypothesis that pragmatic processes operate at least at the sentential level.

I. 2. Sauerland (2001)

Sauerland (2001) modifies the neo-gricean procedure by changing what counts as a scalar alternative for a given sentence. If a certain sentence S is of the form "P or Q", then P and Q count as alternative sentences, and so do P and Q's own scalar alternatives. But this move, as such, can only lead to a strengthening of what the 'standard' account derives, i.e. cannot remove the unwanted negations. Sauerland shows that this problem can be solved if one adopts more plausible inference rules. His proposal is quite similar to mine in this respect[2].

I. 3. Is exhaustification the solution ?

Van Rooy (2002) proposes to derive scalar implicatures from the fact that, if a certain question Q is under discussion and a certain sentence S is given as an answer to Q, S is generally interpreted as "exhaustive".

---

[1] In section **III**.**3**.3, I account for the fact that this inference is not systematic.

[2] I am grateful to an anonymous reviewer for pointing out a version of Sauerland's paper that is more recent than the one I had read and bears more similarity to my own proposal.

The exhaustivity operator (Groenendijk & Stokhof 1984) operates on answers of the form 'GQ P', where GQ stands for a generalized quantifier and P for a predicate. The question under discussion is understood as "for which objects is P true of these objects ?".

The exhaustivity (*exh*) operator works as follows[3]:

⟦exh (GQ P)⟧ = 1 iff ⟦P⟧ ∈ (Min ⟦GQ⟧), where (Min ⟦GQ⟧) is the set that includes only the minimal members of ⟦GQ⟧, i.e:

Min ⟦GQ⟧= {x | x ∈ ⟦GQ⟧ and there is no x' in ⟦GQ⟧ such that x' ⊂ x}

(⊂ = "is a proper subset of")

Example:

(7) a. Among John, Mary and Peter, who came?

b. John or Mary came

⟦John or Mary⟧= {{J, M, P}, {J, M}, {J, P}, {J}, {M, P}, {M}}

(Min ⟦John or Mary⟧) = {{J}, {M}}

⟦exh (John or Mary came)⟧ = 1 iff ⟦came⟧ ∈ {{J}, {M}} i.e. iff only John came or only Mary came.

Van Rooy shows that when exhaustification is applied to monotone increasing contexts, it can solve some of Chierchia's puzzles.

I. 4. When exactly do we exhaustify answers ?

However, if exhaustification is applied to a sentence 'GQ P' where GQ is decreasing, exhaustification as defined above leads to unrealistic implicatures: "less than two chemists came" should implicate that nobody came! So Van Rooy uses a second exhaustivity operator (*exh'*) in these cases, following Stechow & Zimmermann (1984):

⟦exh' (GQ P)⟧ = 1 iff ⟦P⟧ ∈ (Max ⟦GQ⟧), where (Max ⟦GQ⟧) is the set that includes only the maximal members of ⟦GQ⟧

There are several problems with this account.

First, the second rule of exhaustification makes wrong predictions:

(8) a. Among the chemists and the philosophers, who came?

b. Less than two of the chemists

Exhaustification leads to b':

b' . Exactly one chemist and all the philosophers came.

But b. does not seem to implicate b'; b. actually *does* indeed suggest that some chemist came, but does not implicate anything regarding non-chemists. It rather suggests that the speaker does not know much about them.

Second, these two rules are unable to account for cases where the speaker combines increasing and decreasing quantifiers, thus creating a non-monotone GQ, as in (9)b:

(9) a. Among the chemists, the philosophers and the linguists, who came?

[3] I reformulate Groenendijk & Stockhof's exhaustivity operator in more simple terms, but the difference is immaterial.

b. Less than two chemists but one philosopher came

If we apply the first exhaustivity operator, what we get is that b. implicates that no chemist and no linguist came, while exactly one philosopher came.

If we apply the second exhaustivity operator, what we get is that exactly one chemist, all the philosophers and all the linguists came.

None of these predictions is in fact borne out. Rather, it seems that (9) implicates that at least one chemist came, exactly one philosopher came, and that the speaker does not know much about linguists.

I. 5. Goal of this paper: deriving exhaustivity

In the next sections, I show that both scalar implicatures and exhaustification of answers can be understood as the outcome of a pragmatic reasoning that is based on the gricean maxims. I will first offer a precise formalization of the gricean reasoning, meant to replace the two rules R1 and R2. I will then show that it is possible to predict the facts reviewed above by defining carefully what counts as an "alternative answer" for a given answer to a certain question under discussion.

II. Formalizing the gricean reasoning

I now assume that a certain sentence A is uttered as an answer to a (maybe implicit) question Q, and I adopt a partition semantics for questions (Groenendijk & Stockhof 1984): Q induces an equivalence relation R_Q, over the set of worlds.

Notation:

- $w\ R_Q\ w'$ w and w' belongs to the same « cell»
- $R_Q\ (v) = \{w \mid w\ R_Q\ v\}$ (= the set of worlds equivalent to v, or v's cell)
- $\alpha\ (w)$ α is true in w (alternatively: $w \in \alpha$)
- $\alpha \subseteq \beta$ α is a subset of β; α entails β
- $\alpha \subset \beta$ α is a proper subset of β; α asymmetrically entails β

The proposition α expressed by A is supposed to meet the condition of *strong relevance*.

Def 1 (strong relevance): A proposition α (= set of worlds) is strongly relevant with respect to a question Q if

a) $\exists w, (R_Q(w) \cap \alpha) = \emptyset$ *(i.e.: α excludes at least one cell)*

and

b) $\forall w, (\alpha(w) \leftrightarrow (R_Q(w) \subseteq á))$ *(α does not distinguish between two worlds that belong to the same cell, i.e. provides no irrelevant information)*

The speaker's information state is modeled as a set of worlds, i.e. a proposition. As an agent believes a lot of things that are irrelevant in the context of a given question, it is useful to define what counts as the *relevant information* contained in a certain information state:

Def 2 (relevant information): Let i be an information state and Q a question. Then we define i relativized to Q, written as i/Q, as follows:

$i/Q = \{w \mid \exists w', (w' R_Q w \text{ and } w' \in i)\}$ $(= \cup_{w \in i} R_Q(w))$.

The gricean reasoning is based on the idea that α (the proposition given as an answer) must be compared to a certain set of *alternative propositions*[4] which the speaker could have chosen instead of α. This **alternative set**, call it S, must contain α itself, and be such that all its members are relevant[5]. The hearer's task when interpreting the speaker's utterance is to address the following question: *given that the speaker has preferred α to all the other members of S, what does this entail regarding his information state i_0?* First, the speaker must believe α to be true (Grice's maxim of quality), i.e. i_0 must entail α. Second, α must be optimal in the sense that there must be no more informative proposition in S entailed by the speaker's beliefs (Grice's maxim of quantity), i.e. there must be no proposition α' such that i_0 entails α' and α' asymmetrically entails α. Put differently, i_0 must belong to the following set I(S,α,Q):

Def 3: $I(S,\alpha,Q) = \{i \mid i \subseteq \alpha \text{ and } \forall \alpha' (\alpha' \in S \text{ and } i \subseteq \alpha') \rightarrow \neg (\alpha' \subset \alpha)\}$

So if a certain proposition β is entailed by no member of I(S,α,Q), the hearer can conclude that β is not part of the speaker's belief. This reasoning plays the role of rule R1. It is immediately predicted that if the speaker utters a sentence P of the form "A or B" and if the propositions expressed by A and by B belong to the alternative set S, as I will assume (so does Sauerland (2001)), then the speaker cannot know A to be either true or false: if A were true, then A would have been a better answer than P; if A were false, B would be true (since P is), and B would have been a better answer than P[6]. Now, let the hearer assume that the speaker is *as informed as possible given the answer he made.* This means that his information state i_0 is maximal in I(S,α,Q) in the following sense: there is no i' in I(S,α,Q) such that i' (relativized to Q) asymmetrically entails i_0 (relativized to Q). In other words, i_0 belongs to Max(S,α,Q), defined as follows:

Def 4: $Max(S,\alpha,Q)=\{i \mid i \in I(S,\alpha,Q) \text{ and } \forall i' (i' \in I(S,\alpha,Q)) \rightarrow \neg (i'/Q \subset i/Q)\}$

From this the hearer can conclude that if a proposition β is entailed by all the members of Max(S,α,Q), then β is believed by the speaker. This reasoning plays the role of R2, but is not equivalent to it: there is no way of deriving an "unwanted negation". In the case of a disjunctive statement in which the disjuncts are logically independent, the disjuncts and their negations are entailed by no member of I(S,α,Q), as shown above, so that they cannot be entailed by any member of Max(S,α,Q) either, since Max(S,α,Q) is included in I(S,α,Q).

[4] As my formulation makes clear, I am now adopting the simplifying view that what the hearer compares are *propositions*.

[5] The exact definition of alternative sets is the topic of section III.

[6] Assuming that A and B are logically independent.

From now on, whenever it clear what the question under discussion is, and considering that the content of an alternative set only depends on the question under discussion and the sentence uttered, I will simply write I(α) and Max(α) instead of I(S,α,Q) and Max(S,α,Q). S(α) will denote the alternative set of α.

III. Alternative sets and Exhaustification

III. 1. An example

Let P be of the form '(A or B) or C', where A, B and C are logically independent. Assume that (1) is uttered in a context in which A, B and C's truth-values are what is relevant i.e. as an answer to a question Q amounting to "Which sentence(s) are true among A, B, and C?"

For any information state i, the relevant part of i in this context (i.e i/Q) belongs to the boolean closure of {A,B,C}. So we will loose nothing if we view information states as *sets of valuations of* {A, B, C}, i.e. as propositions of the propositional language based on {A,B,C}, where any such proposition actually stands for a class of propositions that are all equivalent when relativized to Q. Let S(P) (the alternative set of P) be *the closure under union and intersection of* {A,B,C} [7] . Intuitively, S(P) is the *set of positive answers to Q*: S(P)={A,B,C,A∨B,A∧B,A∨C,A∧C,B∨C,B∧C,(A∨B)∨C,(A∧B)∨C,A∨(B∧C)...}
Assume i_0 = ((A∨B)∨C))∧(¬(A∧B) ∧ (¬(A∧C)∧¬(B∧C))). Then $i_0 \in$ I(P), since P is the only – and therefore best - proposition in S(P) entailed by i_0[8]; i_0 can also be described as the set of the three following valuations:

	A	B	C
W1	T	F	F
W2	F	T	F
W3	F	F	T

I now show that Max (P) = {i_0}, e.g. that i_0 entails all the members of I(P). Suppose i_1 is an information state that is not entailed by i_0 and that belongs to I(P). There is then an element of i_0 that does not belong to i_1. Suppose W_1 does not belong to i_1. Then i_1 entails P': P' = ¬(A∧(¬B∧¬C)) = ¬A∨(B∨C)

But i_1 belongs to I(P), and therefore entails P. Hence i_1 also entails P'':

P'' = ((A∨B)∨C))∧(¬A∨(B∨C)) = (B∨C)

But P'', which belongs to S(P), would have been a better answer than P in information state i_1, so that i_1 does not belong to I(P), contrary to the hypothesis.

[7] As the reader will have noticed, I treat sentences both as sentences of the object-language and as names (in the meta-language) of propositions, i.e. names of sets of worlds, in which case conjunction and disjunction are understood as intersection and union.

[8] I do not give the proof here, due to lack of space.

Things are similar if W2 or W3 does not belong to i_1 (by symmetry). Hence Max(P) = $\{i_0\}$, and P implicates i_0.

This proof can be generalized to all formulas whose only logical operators are disjunctions.

III. 2. Background concepts

As shown in section I. 4, answers lead to different kinds of implicatures, especially regarding exhausitivity, depending on whether they are, intuitively speaking, positive or negative. But this cannot make sense so far, as I have not said precisely what it is for an answer to be "positive". This is the goal of the present section.

I now assume that questions are all equivalent to something like:

Q: "For which x is P(x) true?", where x is of any semantic type, and P is a certain predicate (simple or complex) that can be built in a natural language. I further assume that the domain of quantification is fixed and finite, and known to all participants. Thus any relevant answer to Q can be translated into the following propositional language L_Q: let $(c_i)_{0< i < n+1}$ be an enumeration of names for each of the individuals of the domain. Then L_Q is the propositional language with disjunction and conjunction as its only binary connectors and based on the atomic sentences $(P_i)_{0< i < n+1,}$ where P_i translates $P(c_i)$.

Now, relevant answers to Q can be seen as sets of valuations of $(P_i)_{0< i < n+1}$. And the relevant part of any information state can also be seen as a set of valuations. So we can assimilate information states to sets of valuations, without loosing anything.

Definitions:

1. **Literal**: a **literal** is an atomic sentence or the negation of an atomic sentence. A literal is **positive** if it is an atomic sentence, **negative** otherwise

2. **Sentence P *favours* literal L**: a sentence or a proposition P favours a literal L iff there is a valuation V such that V(P) = V(L) = 1 and $V_{-L}(P) = 0$, where V_{-L} is defined as the valuation which is identical to V except for the value it assigns to L.

3. **Sentence P *essentially mentions* literal L**: A sentence P essentially mentions a literal L iff L occurs without a negation preceding it in every P' equivalent to P and such that the scope of all negations occurring in P' is an atomic sentence.

4. **positive sentence/positive proposition**: a sentence or a proposition is positive (resp. negative) iff it favours at least one positive (resp. negative) literal and no negative (resp. positive) literal.

We can then prove the following theorems (Egré, Gliozzi & Spector):

Theorem 1: For any sentence P and any literal L, P favours L iff P essentially mentions L

Theorem 2: A sentence P is positive (resp. negative) iff P is equivalent to a sentence which belongs to the closure of positive (resp. negative) literals under conjunction and disjunction

Corollary: A sentence P is positive iff it is equivalent to a sentence P' which contains no negation.

We therefore have two characterizations of positive answers: an answer is positive if it is equivalent to a sentence which contains no negation, or, equivalently, if it favors at least one positive literal and no negative literal. This equivalence will prove helpful.

III. 3. The case of positive propositions: predicting exhaustification

The alternative set of any positive proposition is defined as **the set of all positive propositions**[9].

III. 3.1. An Example

Consider the following dialogue:

(10) – Among John, Peter, Mary and Sue, who will come?

- Well, John will come, or Peter and Mary will come

I translate the answer into a propositional language containing four atomic sentences A, B, C and D:

$P = A \vee (B \wedge C)$

P quite clearly implicates Q : $Q = (A \wedge \neg B \wedge \neg C \wedge \neg D) \vee (B \wedge C \wedge \neg A \wedge \neg D)$ i.e. "either only John will come, or only Peter and Mary will", which is exactly what exhaustification in Groenendijk & Stokhof's sense would yield.

What I will now prove is that $Max(P) = \{Q\}$, from which it indeed follows that P implicates Q.

First, I show that $Q \in I(P)$, i.e. P is an optimal answer in S(P) in information state Q. Suppose the speaker's information state is Q. Q can be represented as the following set of valuations, where a valuation is itself represented the set of atomic sentences that this valuation makes true: $Q = \{\{A\},\{B, C\}\}$.

By hypothesis, the speaker has to choose a proposition that belongs to the alternative set. This proposition must be entailed by Q and be such that there is no better proposition in the alternative set. Let Q' be a positive sentence entailed by Q. Necessarily the valuation represented by {A} is in Q'. But then, the valuation {A,B} must be in Q' too: if {A,B} were not in Q', indeed, $\neg B$ would be **favoured** by Q', since there would be a valuation v making $\neg B$ true in Q' (namely v = {A}) and such that the valuation v' identical to v except over B (v'={A, B}) would not be in Q'; so Q' would favour a negative literal and not be positive, contrary to the hypothesis. By the same reasoning, {A,C}, {A,D}

[9] It should be clear that the alternative set is dependent on the question under discussion, since "positivity" is defined in terms of the propositional language derived from the question under discussion *via* the translation procedure defined above.

{A,B,C}, {A,B,D}, {A,C,D} and {A,B,C,D} must belong to Q', and so does {B,C,D} (since {B,C} is in Q and therefore in Q'). So any positive proposition entailed by Q must include the following proposition, i.e. be entailed by it:

{{A}, {A,B}, {A,C},{A,D}, {A,B,C},{A,B,D}, {A,C,D},{A,B,C,D}, {B,C}, {B,C,D}} (= P)

But this set, **which turns out to represent P**, is a positive proposition which is entailed by Q and which entails all other positive propositions that are entailed by Q (as I have just shown). So P is the strongest positive proposition entailed by Q, i.e. $Q \in I(P)$ (recall that I(P) is the set of all information states which make P an optimal answer among positive answers).

Second, I show that Max(P) = {Q}. This amounts to proving that Q entails all the members of I(P). Assume there is an information state i which belongs to I(P) and is not entailed by Q. Since i is not entailed by Q, then either {A} or {B, C} does not belong to i. Suppose {A} does not belong to i. On the other hand, i belongs to I(P) and therefore entails P. From which it follows that i entails P-{A}, i.e. i is included in the following set of valuations:

P - {A} = {{A,B}, {A,C}, {A,D}, {A,B,C}, {A,B,D}, {A,C,D}, {A,B,C,D}, {B,C}, {B,C,D}}. But this set is itself a positive proposition, since it can be checked that P-{A} favours no negative proposition. In fact, P-{A} can be written as: $(A \wedge (B \vee C \vee D)) \vee (B \wedge C)$. So i entails a positive proposition that is stronger than P, namely P-{A}, which contradicts the hypothesis that i belongs to I(P). Things work similarly if {B,C} does not belong to i. Therefore there is no such i. From which it follows that Q entails all the members of I(P). Q.E.D

III. 3. 2. Predicting exhaustification

In the general case, positive answers are predicted to be interpreted as **exhaustive**.

Definitions:

1. Exhaustification:

Let P be any non-negative proposition, then the function Exhaust is defined as follows:

Exhaust(P) = {V | V ∈ P and there is no valuation V' in P such that V' ⊂ V}

This operator is the propositional counterpart of Groenendijk & Stockhof's exhaustivity operator.

2. Positive extension of a proposition P: for any non negative proposition P, there is a unique positive proposition Q such that P entails Q and Q entails all the other positive propositions that P entails (i.e. Q is **the strongest positive proposition that P entails**). This can be shown by using the same reasoning as in the previous section: namely, you get Q by adding to P all the valuations that are needed in order not to favour any negative literal. The result of this operation I call the *Positive Extension of P*, or **Pos (P)**.

For any P, Pos(P) = {V ⎪ there is a valuation V' in P such that V' ⊆ V} (recall that a valuation is seen as a set of atomic sentences)

Facts: for any non negative proposition P,

1. If P is positive, Pos(P) = P
2. Exhaust (P) ⊆ P
3. Exhaust (P) = Exhaust (Pos (P))
4. Pos (Exhaust (P)) = Pos (P)

From these facts, I prove the following theorem:

Theorem: if P is a positive proposition, then Max(P) = {Exhaust(P)}, and therefore P implicates Exhaust(P).

Proof: let P be a positive proposition. Suppose i ∈ I(P), i.e. i is such that a speaker who is in information state i would choose P among the set of positive sentences. Then P must be the strongest positive proposition that i entails, e.g. P = Pos(i) and I(P) = {i⎪Pos(i)=P}. Since Pos(Exhaust(P))=Pos(P)=P, Exhaust(P) ∈ I(P). Let i_1 be a member of I(P) such that Exhaust(P) does not entail i_1. Then there is a valuation V_1 in Exhaust(P) which does not belong to i_1. Therefore V_1 does not belong to Exhaust(i_1), and Exhaust(P) ≠ Exhaust(i_1). But Exhaust(i_1)=Exhaust(Pos(i_1))=Exhaust (P), which is contradictory. Hence there is no such i_1 and Exhaust(P) entails all the members of I(P), from which it follows that: Max(P) = {Exhaust(P)}. Q.E.D

III. 3. 3. Pair-list questions

Consider sentence (3) again ("Each of the students read *Othello* or *King Lear*"). If (3) is understood as an answer to a pair-list question like *"Which students read which plays by Shakespeare?"*, exhaustification predicts an exclusive reading for *or*. Note that the translation of a certain natural language sentence into a sentence of propositional logic will yield different results for different underlying questions (see section **III. 2.**). In the case of the above pair-list question, but not in other cases, atomic sentences represent elementary answers of the type 'x read y', and (3) will be translated as something like (3'):

(3') (A∨B)∧(C∨D) ∧(E∨F)∧…… ∧(G∨H)

Exhaustification of (3') yields the desired result (exclusive reading for all the disjunctions). This context-dependency explains why judgments are not uniform.

III. 4. Non-positive propositions

We have seen in **I. 4.** that negative answers are not exhaustified, but nevertheless trigger some implicatures. This is straightforwardly predicted if the alternative set of a negative proposition P consists in *the closure under disjunction and conjunction of all the literals that P favours.* The asymmetry between negative and positive answers then boils down to the fact that positive answers are compared to all positive answers, while negative answers are compared only to a proper subset of the negative answers.

Regarding answers that are neither positive nor negative, the data are quite complex, and judgments are not very robust. A good strategy is to look at the

clearest cases, find which principles could account for them and then let these principles decide for the other cases:

(11) a. Among Peter, Mary and Jack, who came ?

b. Peter, but not Mary

>> No exhaustivity effect: we infer nothing regarding Jack

(12) a. Among the philosophers, the linguists and the chemists, who came?

b. Between two and five linguists

>> Exhaustivity effect : we infer that no chemist and no philosopher came.

One difference between (11) and (12) is that, even though both are neither positive nor negative, (12) is *quasi-positive* in the following sense:

Def 1: A proposition P ***strongly favours*** *a literal L if P favours L and P does not favour the negation of L*

Def 2: A proposition P is ***quasi-positive*** *if P does not strongly favour any negative literal.*

If we want to predict that only quasi-positive sentences lead to exhaustification, we may adopt the two following rules, which cover all the cases:

If P is quasi-positive, P's alternative set consists in the union of the set of positive propositions and {P} itself.

If P is not quasi-positive, then P's alternative set consists in the closure under union and intersection of all the literals that P favours.

These rules make the following predictions (assuming the question under discussion is the same as in (12)):

(13) Between two and five linguists and no philosopher came.

>> no exhaustivity effect: nothing should be implicated regarding linguists

(14) Between two and five linguists and three philosophers came

>>Exhaustivity effect: suggests no linguist came

(15) Three philosophers but less than two chemists came

>>No-exhaustivity effect: nothing should be inferred regarding linguists.

Though judgments are not so clear, an informal inquiry seems to indicate that most people have the expected intuitions. More work needs to be done in order to understand what is really going on here[10].

Conclusion

I have offered a precise formalization of the gricean reasoning that underlies scalar implicatures, and exhaustification of answers. I have shown that the facts regarding exhaustification can be directly derived from the gricean reasoning[11].

[10] I do not give the proof that my two rules achieve the results I claim they do, due to lack of space.

[11] As an anonymous reviewer noticed, I have not addressed all the cases that Chierchia pointed out as problematic for the standard neo-gricean procedure. Once again, limitation of space prevents me from doing so. Let me mention that a

The only stipulations that were needed concern the rules according to which alternative sets are built. Yet the original notion of "scalar alternatives" is also stipulative. It remains to be seen whether the role played by polarity (namely, the distinction between positive and non-positive answers) can be derived in a more principled way. I suspect that introducing a notion of *utility* in our model of information processing, as Nilsenova & Van Rooy (2002) do in order to account for the pragmatic effects of polar questions, could help explain why "positive" and "negative" answers pattern asymmetrically. Another topic that must be investigated is the following one: a sentence like "John will come or John and Mary will" expresses the same proposition as "John will come", but is not interpreted in the same way. This shows the limitations of any procedure that only takes into account the literal semantic values of sentences, and not their actual phonological and syntactic form. I will explore some possible solutions to this problem.

Finally, let me point out that while the procedure I have defined is context-dependent (since implicature computation depends on what the question under discussion is), it is possible to devise a very similar procedure that would not be context-dependent. These two procedures, taken together, can provide us with an analytic tool for investigating to what extent scalar implicatures are *generalized* rather than extremely sensitive to context.

Selected references

Chierchia, G., 2001, "Scalar Implicatures, polarity phenomena, and the syntax/pragmatics interface", University of Milan, Ms.

Egré, P., Gliozzi, V. & Spector, B., “Mentioning, Involving and Favouring”, Ms., IJN, forthcoming.

Groenendijk, J. & Stokhof, M., 1984, *Studies in the Semantics of Questions and the Pragmatics of Answers*, PhD Thesis, University of Amsterdam

Nilsenova, M. & Van Rooy, R., 2002, “No’s no good alternatives”, Stuttgart Workshop on Information Structure, Ms.

Sauerland, U., 2001, "Scalar Implicatures in Complex Sentences”, university of Tübingen, Ms.

Van Rooy, R., 2002, "Relevance Implicatures", ILLC, Amsterdam, Ms.

Stechow, A. von & Zimmermann T. E., 1984, “term answers and contextual change”, *Linguistics,* **22**, 3-40

Zeevat, H., 1994, “Questions and Exhaustivity in update Semantics”, in Bunt & al., *Proceedings of the International Workshop in Computational Semantics*, Institute for Language technology and Artificial Intelligence, Tilburg.

---

more sophisticated version of my proposal is able to predict the phenomenon of *conditional strengthening* (inference from“if A, then B”to“B if and only if A”).

Formalization of Morphosyntactic Features of flective language as exemplified by Croatian

TOMISLAV STÒJANOV
Faculty of Philosophy, Zagreb
tstojan@ffzg.hr

ABSTRACT. The article is about a morphosyntactic model of Croatian language with which help the formal processing is being approached. It presents grammatical basis of the project 'Machine Understanding of Croatian Language' (MZT-RH-130440) in tagger developing for Croatian language which is based on morphologic generator included in Microsoft Word's Spelling Checker (Silić, 1996). Besides the researches on the removal of ambiguity in formal processing of natural languages, based in the first place on qualitative (grammatical) criteria, classes of morphosyntactic units obtained by formal analysis can also serve as a theoretical approach to the typology of word classes.

There are 15 classes of lexical units in Croatian language, divided according to 31 morphosyntactic features of 9 formal categories.

The work is divided in two parts: the first one contains a summary of formal features, their structure and methodology applied for the division into classes of units or parts of speech. The second part provides a concise explanation of each unit class, and points out problematic situations with the modes applied for their solving.

It is emphasized that (i) the introduction of the full morphosyntactic constraints in the classification of units into classes and types is possible, (ii) formal procession of natural language could use features that traditional linguistics does not find descriptive, (iii) due to the morphosyntactic predetermination of syntax, parsing should start from the morphosyntactic features seen as the minimal units, (iv) every formal analysis which takes into consideration the grammatical categorial features is marked by *language-specific* approach.

1 Introduction

In morphosyntactic and syntactic sentence analysis, the acceptance of methodology that will by its results be able to satisfy the functional coherence, mathematical explicitness and computational efficiency is being demanded of contemporary linguistic science more than ever.[1] Satisfying these aspirations caused a splitting into 'traditional' and 'formal' linguistics.

[1] "Fulfilling these requirements will take hard, systematic, goal-oriented work, but it will be worth the effort." Hausser (1999: 3)

Proceedings of the Eighth ESSLLI Student Session.
Balder ten Cate (editor)

Applying the exact principles, grammar had to be redefined in order to solve the same questions that were occupying the first antique grammarians – parts of speech. Plato, Aristotle and Stoics were the first grammarians who considered the language as the *logical form of Logos* [Gr. *λόγος*] that was resulted in parts-of-speech dividing. (cf. Dionizije Tračanin [Dionysius the Thrax], 1995)

Computational linguistics demands as prerequisite condition well-defined grammatical units, which is a particular problem because many linguistic terms are not only polysemantic and imprecise, but also non-exact and non-empirical.

It is paradoxical to talk about computational linguistics without previously defined unit of natural language – word (word form, morphosyntactic word [Spencer 1991:45; Trask 1999:343], morphosyntactic unit, morphosyntacteme).[2]

The initial presumption is that it is possible to create a consistent model that will in the first place use morphosyntactic and syntactic features in parsing, and that will also tend towards descriptive adequacy of describing a specific language.

2 Survey of formal features

2.1 Formal features in relation to the traditional ones

Formal processing requires complete defining of values for each lexical unit by its category features. Categorial features can be:

a) morphosyntactic (besides syntactic and other grammatical features that are not an issue of interest here), and
b) formal.

Regarding terminology, it is worth mentioning that formal linguistics use the terms *feature* and *value* for terms *category* and *feature* used by traditional linguistics. Thus the feature [NUMBER] has the values [SINGULAR] and [PLURAL], unlike traditional nomenclature in which number category is divided into features of singular and plural.

Formal features and values are written in capital letters and in angle brackets, in order to be distinguished from morphosyntactic categories and features.

Corbett (2000:4) says the following regarding these terminological differences: "In many theoretical frameworks number, like comparable categories such as

[2] After numerous explained theories and approaches, Spencer (1991: 453) at the end of his book as the answer to the question "Where's morphology?" concludes: "I conclude this discussion with a rephrasing of this question, which will serve as the background to my own answer to it. At various times I've suggested that one of the key unresolved questions in morphology is 'what is a word?".
According to his suggestion, we could assume that every method unfailingly implies theory about minimal units, and that grammarians, before their elaborating, should answer to the basic question: what are, and which are the minimal units of their approach?

gender, case and person, is treated as a 'feature'. This feature is said to have certain 'values' (for number, these include singular and plural, and we have already come across others too). (...) An alternative terminology has number as a 'category' and singular as a 'property' or 'feature' (...). We retain 'category of number' as a wider term, to include all manifestation of number, including number words (...), as opposed to the category of gender, tense and so on."
Corbett does not state that terminological difference is determined by the different methodology.

The difference between certain formal and morphosyntactic feature (gender, for instance) is in the following:

1. formal gender can have three features: [+GENDER], [–GENDER], [±GENDER], while morphosyntactic gender can have only one;
2. formal gender has values ([MASCULINE], [FEMININE], [NEUTER]) and valency, while morphosyntactic gender has features (masculine, feminine, neuter);
3. formal gender is expressed in capital letters and in angle brackets, while morphosyntactic gender in small letters without brackets.

2.2 Structure of the formal features

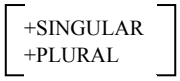
Formal feature is made of two pieces of information: (i) valency and (ii) title of the feature or value, and it is written as [xTITLE] where x represents valency or valency feature, and TITLE the name of valency feature or valency value.
There are four possible ways to annotate formal feature:

[FEATURE], e.g. [NUMBER];
[xFEATURE], e.g. [–NUMBER];
[VALUE], e.g. [SINGULAR];
[xVALUE], e.g. [–SINGULAR].

Valency feature is the common name for univalent, bivalent, polyvalent and extravalent features.

Bivalent features have opposite, binary, or-or value. They have a label [+] or 2v. These are, for instance, affirmation category (affirmative and negative) and number (singular and plural): bivalent feature [+NUMBER] has the values [+SINGULAR] [+PLURAL].

Graphically representing, bivalent feature [+NUMBER] means (1):



Bivalency indicates that unit can have any one of the two values in the feature.

Univalent features have only one of the values (e.g. nominal gender in 'kuća' [Eng. 'house'] has feminine gender exclusively, without the possibility of masculine and neuter alteration). They have a label [–] or 1v. For instance, univalent feature [–GENDER] has the values [–MASCULINE] [–FEMININE] [–NEUTER].

Univalent feature [–GENDER] means:

$$\begin{bmatrix} +\text{MASC.} \\ -\text{FEMIN.} \\ -\text{NEUTER} \end{bmatrix} \text{ or } \begin{bmatrix} -\text{MASC.} \\ +\text{FEMIN.} \\ -\text{NEUTER} \end{bmatrix} \text{ or } \begin{bmatrix} -\text{MASC.} \\ -\text{FEMIN.} \\ +\text{NEUTER} \end{bmatrix}$$

Univalency indicates that, out of all values in the feature, unit can have only one.

Polyvalent features have more than two values – e.g., case category that has seven features (seven cases in Croatian). They have a label [+] or Pv. For instance, polyvalent feature [+CASE] has the values [+NOMINATIVE] [+GENITIVE] [+DATIVE], etc.

Polyvalent feature [+CASE] means:

$$\begin{bmatrix} +\text{NOMINATIVE} \\ +\text{GENITIVE} \\ +\text{DATIVE} \\ + \ldots \end{bmatrix}$$

Polyvalency indicates that unit can have any value in the feature.

Extravalent features have an unspecified valency – they indicate the presence of the features only at a part of units of a certain class or non-existence of the features in case of all units. For instance, only two verbs in third person in present tense can have gender feature, i.e. only the verb 'biti' (Eng. 'to be') and 'htjeti' (Eng. 'to want') has the emphasis category. They have a label [±] or Ev: extravalent feature [±EMPHASIS] has the values [±EMPHATIC] [±NONEMPHATIC]. Information on which unit contains an extravalent feature, and which one does not is to be found in the lexicon.

Extravalent feature [±EMPHASIS] means:

$$\begin{bmatrix} +\text{EMPHATIC} \\ +\text{NONEMPHATIC} \end{bmatrix} \text{ or } \begin{bmatrix} -\text{EMPHATIC} \\ -\text{NONEMPHATIC} \end{bmatrix}$$

Extravalency indicates that only certain lexical units in the class have the value stated in the feature.

2.3 Valency

Mark [+] of some feature indicates that *all* lexical units of a certain type or class *have all* the values of that feature well-formed (for instance, nouns have [+CASE] because they are declined by all cases).

Mark [–] of some feature indicates that all lexical units of a certain type or class *have not all* the values of that feature well-formed, i.e. they *have* only *one* value of that feature well-formed. For instance, number nouns 'dvojica' (Eng. 'two of them'), 'trojica' (Eng. 'three of them') in [NUMBER] have only plural but not singular form. So, they have value [–NUMBER], or in fact [–PLURAL]).

Mark [±] of some feature indicates that *just some* lexical units of a certain type or class *have* the values of that feature well-formed, i.e. *not all* lexical units of a certain type or class *have* the values of that feature well-formed.

We can also say like this:

[+] means that each unit in a class has all values;

[–] means that a) each unit has just one value;
b) not each unit has all values;

[±] means that a) some units have certain feature;
b) not all units have certain feature.

2.4 From formal features to word classes

Although it does not seem so at the first sight, implementation of formal feature has been justified by the optimum argument of computational system. It says that it is minimal, and thus more optimal, to place units into common classes without repeating the same formal features.

If we classify verbs 'napišem' (Eng. 'I write'), 'napišeš' (Eng. 'you write') and 'napišu' (Eng. 'they write') according to their morphosyntactic features, we can talk about three classes,

(i) [PRESENT] [FIRST] [SINGULAR];
(ii) [PRESENT] [SECOND] [SINGULAR];
(iii) [PRESENT] [THIRD] [PLURAL],

where 'napišem' belongs to the first class, 'napišeš' to the second one, and 'napišu' to the third class.

More economical system would start from the fact that words are derived and that they have their basic (conjugational or declinational) form. This way we would place the whole paradigm into one formal class. The basic form 'napišem'[3] and all its derived present units would thus belong to the class (iv).

[3] The basic form of 'napišem' (Eng. 'I write'), 'napišeš' (Eng. 'you write'), 'napiše' (Eng. he/she/it writes'), 'napišemo' (Eng. 'we write'), 'napišete' (Eng. 'you write'), 'napišemo (Eng. 'they write') is not 'napisati' (Eng. 'to write'), but 'napišem' (Eng. 'I write') – due to relatively often discrepancies of the infinitive and present stem, e.g. 'biti-budem' (Eng. 'to be-I will be'), 'žeti-žanjem' (Eng. 'to harvest-I harvest'), etc.

(iv) [PRESENT] [+PERSON] [+NUMBER] [±GENDER]

If every (derived) unit would have its own class, classification form system would get significantly compound and complicated (in case of nouns we would get fourteen instead one class, cf. Appendix 2).

For instance, instead [+CASE] [+NUMBER] [NEUTER] for noun 'more' (Eng. 'sea'), we would have (v):

(v) [NOMINATIVE] [SINGULAR] [NEUTER], 'more'
[GENITIVE] [SINGULAR] [NEUTER], 'mora'
[DATIVE] [SINGULAR] [NEUTER], 'moru'
[ACCUSATIVE] [SINGULAR] [NEUTER], 'more', etc.
(…)

It is assumed that language generalization (without the loss of accuracy and purpose) represents the nature of formal processing of natural languages in better fashion than thoroughly performed formalization that would start from each lexical unit individually. Non-connection of lexical units into paradigms might turn out as a flaw in later analyses. Systematization might turn out useful for future lexicographical, morphological, discursive and other researches.

Placing of verb forms in one class is analogous to the understanding that all case forms of a certain unit belong to the same class.

Unlike non-paradigmatic approach, by acceptance of the assumption that there are *basic* and *derived* forms, meaning that morphology actively takes part in computational system, instead of fourteen noun classes we get only one: [+CASE] [+NUMBER] [–GENDER]. This way we also get a more economic formal system.

The optimum of computational system relates to the number of introduced formal classes, or in fact the level to which formalization can be raised in relation to its purpose.

Formal system represented in this paper indicates the acceptance of morphology as a part of computational system, and the starting from (linguistic) categorial features in that processing.

So, conjugation begins from the first person singular, and not from the infinitive, similarly to declination that begins from its first, nominative form.

Present thus belongs to the separate class than infinitive and 'glagolni prilog' (Eng. 'present and past verbal adverb', kind of participle) does. See Appendix 1.

3 Morphosyntactic features in Croatian

3.1 Analysis of pronouns

In traditional definition, pronouns were divided into nominal and adjectival pronouns. Nominal pronouns are personal ('ja', 'ti', 'on', itd.) [Eng. 'I', 'you', 'he', etc.], reflexive ('se/sebe') [Eng. 'myself, yourself, itself, etc.'], part of interrogative and relative pronouns ('tko', 'što') [Eng. 'who', 'what'] and part of indefinite pronouns (all compound ones with 'tko', 'što').

Adjectival pronouns are all other pronouns: reflexive-possessive, demonstrative, possessive, interrogative and relative (except 'tko', 'što'), and indefinite pronouns (all except those formed of 'tko', 'što'). (cf. 3.4)

In Gramatičkom tezaurusu ('Grammar Thezaurus', Batnožić & Ranilović & Silić, 1996) morphosyntactic criteria of gender category has caused division of these groups into two completely individual word groups. Adjectival pronouns are thus considered as *adjectives* because they have the same formal morphosyntactic features as adjectives: polyvalent case, number and gender. Nominal adjectives are placed as 'real' pronouns because they do not have polyvalent gender feature.

According to what is left from them, pronouns can be added the following formal features and divided into four classes (vi): reflexive pronoun has [+CASE], [–SINGULAR] [±EMPHASIS], interrogative and relative have [+CASE], [–SINGULAR] indefinite [+CASE] [–SINGULAR], first and second person of personal pronouns have [+CASE], [–NUMBER], [–PERSON], [±EMPHASIS], and third person personal pronouns have [+CASE], [+NUMBER], [–GENDER], [–PERSON], [±EMPHASIS].

(vi) 1C [+CASE] [–SINGULAR]
(interrogative, relative and indefinite pronouns)
2C [+CASE] [–SINGULAR] [±EMPHASIS]
(reflexive pronoun)
3C [+CASE] [–NUMBER] [–PERSON] [±EMPHASIS]
(first and second person of personal pronouns)
4C [+CASE] [+NUMBER] [–GENDER] [–PERSON] [±EMPHASIS]
(third person of personal pronouns is included)

3.2 Analysis of numbers

Numbers are traditionally divided into nominal, adjectival and adverbial numbers depending on the fact whether they can or cannot be changed, and if they can be changed do they have nominal or adjectival declinable paradigm (nominal and adjectival numbers in contrast to adverbial ones)?

The difference between nominal and adjectival paradigm is in gender category: nominal numbers have univalent [–GENDER] (e.g. ‘dva’, ‘oba’, ‘dvije’, ‘petero’) [Eng. ‘two’-nom.pl., ‘both of them’-nom.pl.m., ‘two of them’-nom.pl.f., ‘five of them’-nom.sg.n.]), and adjective ones polyvalent feature of gender [+GENDER] (e.g. ‘dvoji/dvoja/dvoje’ [Eng. ‘two’-nom.pl.m./f./n.), ‘prvi/prva/prvo’, [Eng. ‘first’-nom.sg.m./f./n.], etc.).

Since adverbial numbers do not have morphosyntactic categories, they are excluded from numbers, so that only nominal and adjectival numbers are considered as numeral type.

Therefore, numbers have two classes: [+CASE], [–NUMBER] [–GENDER] (so called nominal numbers, first class numbers, or 1C numbers) and 2C [+CASE] [–NUMBER] [+GENDER] or adjectival numbers.

3.3 Analysis of nouns

Nouns are also various in their formal features. There are nouns (i) that have only singular (*singularia tantum*) – e.g., personal names like ‘Ilija’, ‘Ines’, ‘Marko’, nouns like ‘tele’, ‘prase’ (Eng. ‘calf’, ‘piglet’), or nouns (ii) that have only plural (*pluralia tantum*) – e.g., ‘gaće’, ‘grablje’, ‘ljestve’, ‘naočale’, etc. (Eng. ‘underpants’, ‘rake’, ‘ladder’, ‘glasses’), and (iii) numeral nouns like ‘dvojica’, ‘trojica’ (Eng. ‘two of them’-nom.pl.n., ‘three of them’-nom.pl.n.).

[–NUMBER] is common to all of them.

Indeclinable nouns belong to the separate class (female name ‘Ines’, for instance).

Thus, nouns are formally divided into three classes: the first one with features [+CASE], [+NUMBER], [–GENDER] (first class nouns or 1C nouns), the second one with [+CASE], [–NUMBER], [–GENDER] (second class nouns or 2C nouns), and the third one with [–NOMINATIVE] [–SINGULAR] [–GENDER] (third class nouns or 3C nouns).

3.4 Analysis of adjectives

Adjectives are divided into four classes; they all have common features of case, number and gender.

Except the indeclinable adjectives (e.g. ‘braun’ [Eng. ‘brown’]), all other adjectives have difference only in comparison and aspect.

At division of adjectives into formal classes the difference between traditional and computational linguistics is what is the most noticeable. It is most visible in differentiation of form and meaning of adjectival aspect category (definite and indefinite aspect). As contrary to the expectations, formal approach determines

adjectival aspect as a semantic category and in dealing with it acts accordingly[4], because it was not possible to establish formal classes by observing forms. Besides, adjectives do not have different paradigms for aspect category for all their forms – they differ in only five masculine singular cases and in three neuter singular cases, whereas all feminine forms are equalized. Compromise was impossible since the observation of relation between form and meaning did not bring us to the conclusion – for instance, adjectives with definite form, e.g. 'dalmatinski' [Eng. 'Dalmatian'-nom.m.sg.], 'šivaći' [Eng. 'sewing'-nom.m.sg.] can be both definite and indefinite in their meanings, whereas indefinite 'sinov' [Eng. son's-nom.m.sg.] is definite in meaning.

Abolishing of adjectival aspect as a formal category would not solve the problem. Therefore, the following methodology was applied: definiteness and indefiniteness are viewed as semantically determined categories that have their own forms but cannot change them (in such a way that is a case of gender in adjectives, for instance). Determining which adjective is definite, and which one indefinite was done indirectly with the help of comparison category, like in Silić' Grammar Thesaurus. Semantic analysis of Croatian language indeed prove that only indefinite adjectives can have comparison category (since only quality is compared, and not substance), whereby a conclusion that all comparable adjectives are indefinite can be drawn.

It is also worth emphasizing that established formal classes could not encompass an example of superlative in definite use[5], which is explained by semantic transformation of higher order, like even clearer example of possessive (definite) adjectives ('majčino mlijeko' [Eng. mother's milk] or 'Ivanov auto' [Eng. Ivan's car]) that can be indefinite only with a special explanation of the context.

All comparable adjectives that are therefore indefinite go into the first class. All incomparable (definite) and possessive adjectives go into the second class. The third class encompasses all pronominal adjectives: possessive pronominal adjectives (e.g., 'moj', 'tvoj' [Eng. 'my', 'your']), possessive-reflexive pronominal adjective 'svoj' [Eng. one's own], interrogative and relative pronominal adjectives (e.g., 'tko', 'čiji' [Eng. 'which one', 'whose']), demonstrative pronominal adjectives (e.g., 'taj' [Eng. 'that one']), and indefinite pronominal adjectives (e.g., 'ikoji' [Eng. 'any one']).
Their formal features are:
1C [+CASE] [+NUMBER] [+GENDER] [+COMPARISON] [–INDEFINITE]

[4] Unlike traditional grammars that place the form on the first place although the issue of aspect is in fact completely a semantic one.
[5] 'najpametniji neka odu' (koji? a ne kakvi?)
the smartest-nom.m. let-particle leave-3.pl.pres. (which? not what like?)
'the smartest ones may leave' (who may leave? not what like may leave?)

2C [+CASE] [+NUMBER] [+GENDER] [–POSITIVE] [–DEFINITE]
3C [+CASE] [+NUMBER] [+GENDER]
4C [–NOMINATIVE] [–SINGULAR] [–GENDER]

1C pronoun, 2C pronoun	**1C, 2C, 3C adjectives**	**1C noun, 4C pronoun**	**2C noun, 1C number**	**2C number**	**3C noun 4C adjective**	**3C pronoun**
[+CASE] [–SINGULAR]	[+CASE] [+NUMBER] [+GENDER]	[+CASE] [+NUMBER] [–GENDER]	[+CASE] [–NUMBER] [–GENDER]	[+CASE] [–NUMBER] [+GENDER]	[–NOMINATIVE] [–SINGULAR] [–GENDER]	[+CASE] [–NUMBER]

Illustration 1. Scheme of nominal features according to case, number and gender categories

3.5 Analysis of verbs

Verbs are, unlike other words with morphosyntactic categorial features, significantly various according to the much greater number of derived forms and it is therefore difficult to talk about some clearly noticeable formal classes like in case of nouns for instance. The criteria by which only two classes have been defined (depending on conjugation of verb forms) is found on the connecting spot of formal grammatical accuracy, optimal computational system and units listed in Silić' Grammar Thesaurus.

Thus verbs have six morphosyntactic categories: finiteness, infiniteness, person, number, gender and emphasis.

Finiteness category presupposes all general (tense/mood/participle) inherent features that can be added to individual verb lexical unit in Croatian language: present, aorist, imperfect, imperative, verb adjective I (active participle) and verb adjective II (passive participle).

Since compound tenses and moods do not derive from an individual verb but are conditioned by a connection, they are not found under morphosyntax but are regarded as syntactic or relational-motivated categories.

Out of all categories of finiteness only verb adjectives can change by gender.[6] The feature of emphasis is found only in 'auxiliary' verbs 'biti' [Eng. 'to be']7 and 'htjeti' [Eng. 'to want']8.

[6] radio/radila/radilo; viđen/viđena/viđeno
worked-present part.3.m.sg./f/n; seen-past part.3.m.sg/f/n
he/she/it worked; he/she/it is seen
[7] sam/jesam
am-pres.1.sg.nonemphatic/emphatic
I am
[8] ću/hoću
want-pres.1.sg.nonemphatic/emphatic
I want

Infinitive, verb adverb I (present) (e.g., 'kopajući' [Eng. 'digging']) and verb adverb II (past) (e.g., 'kopavši' [Eng. 'having dug']) go into conjugatable verb class.
1C v. [–FINITENESS] [+PERSON] [+NUMBER] [±GENDER] [±EMPHASIS];
2C verbs [–INFINITENESS].

4 Conclusion

Formal morphosyntactic analysis deals with the formalization of morphosyntactic features of Croatian language. The aim of this paper is to work on the frame for description of morphosyntax, i.e. inflective morphology, for the needs of tagging and parsing.[9]

Unlike the traditional approach where description significantly included semantic component, here morphosyntactic valencies have been used for grammatical and conceptual-forming categorial features in defining of lexical units.[10]

Thus instead of classical definition, adjectives are here defined as well-formed lexical units that have categories of case (Pv), gender (Pv), number (2v), comparison (ø/1v/Pv) and aspect (ø/1v).

Formally speaking, there are no nouns, numbers, pronouns, adjectives and verbs in morphosyntax, but formal features that define classes. However, although the criteria for division of types and classes is of formal nature, we still use the marks from traditional terminology (nouns, verbs, etc.; Van Valin, 2000: 13) which directs us to the need for further studying of syntactic as well as semantic categorial features.

This is the basis on which it was spotted that basic division of changeable and non-changeable forms should be replaced by division of units with and without morphosyntactic categories. Lexical units without morphosyntactic features have to be able to be determined with the help of syntactic categorial features.

In Croatian language, inflective forms are divided into 5 types and 16 classes. Morphosyntactic analysis of Croatian language encompasses 46 features in 10 morphosyntactic categories

[9] For more on computational application of inflective morphology see Daille, Fabre & Sebillot (2002: 210-234)

[10] "In traditional grammar, lexical categories are given notional definitions, i.e. they are characterized in terms of their semantic content. For example, 'noun' is defined as 'the name are of a person, place or thing', 'verb' is defined as an 'action word', and 'adjective' is defined as a 'word expressing a property or attribute'. In modern linguistics, however, they are defined morphosyntactically in terms of their grammatical properties.", Van Valin, Jr (2000: 6)

Appendix 1 Division of formal morphosyntactic features[11]

1. feature **[TYPE]** has values [1C NOUN], [2C NOUN], [3C NOUN], [1C ADJECTIVE], [2C ADJECTIVE], [3C ADJECTIVE], [4C ADJECTIVE], [1C PRONOUN], [2C PRONOUN], [3C PRONOUN], [4C PRONOUN], [1C VERB], [2C VERB], [1C NUMBER], [2C NUMBER]
2. feature **[GENDER]** has values [MASCULINE], [FEMININE], [NEUTER]
3. feature **[NUMBER]** has values [SINGULAR], [PLURAL]
4. feature **[CASE]** has values [NOMINATIVE], [GENITIVE], [DATIVE], [ACCUSATIVE], [VOCATIVE], [LOCATIVE], [INSTRUMENTAL]
5. feature **[PERSON]** has values [FIRST], [SECOND], [THIRD]
6. feature **[FINITENESS]** has values [PRESENT], [IMPERFECT], [AORIST], [IMPERATIVE], [VERB ADJECTIVE I], [VERB ADJECTIVE II]
7. feature **[INFINITENESS**] has values [INFINITIVE], [VERB ADVERB I], [VERB ADVERB II]
8. adjective feature **[ASPECT]** has values [DEFINITE], [INDEFINITE]
9. feature **[EMPHASIS]** has values [EMPHATIC], [NONEMPHATIC]
10. feature **[COMPARISON]** has values [POSITIVE], [COMPARATIVE], [SUPERLATIVE]

Appendix 2 Survey of formal morphosyntactic features of lexical units

1. nouns
   1C [+CASE] [+NUMBER] [–GENDER]
   2C [+CASE] [–NUMBER] [–GENDER]
   3C [–NOMINATIVE] [–SINGULAR] [–GENDER]
2. verbs
   1C [–FINITENESS] [+PERSON] [+NUMBER] [±GENDER] [±EMPHASIS]
   2C [–INFINITENESS]
3. adjectives
   1C [+CASE] [+NUMBER] [+GENDER] [+COMPARISON] [–INDEFIN.]
   2C [+CASE] [+NUMBER] [+GENDER] [–POSITIVE] [–DEFINITE]

[11] It is hard to translate the system of so-called Croatian participles. We could explain it literally as verb adjectives and verb adverbs. Verb Adjective I is the ‘glagolni pridjev radni’ or ‘active participle’ – e.g. ‘radio’, ‘kopao’, in English ‘worked’, ‘digged’. Verb Adjective II is the ‘glagolni pridjev trpni’ or ‘passive participle’ – e.g. ‘ubijen’, ‘viđen’, in English ‘killed’, ‘seen’. Verb Adverb I is the ‘glagolni prilog sadašnji’ or present participle

3C [+CASE] [+NUMBER] [+GENDER]
4C [–NOMINATIVE] [–SINGULAR] [–GENDER]

4. pronouns
1C [+CASE] [–SINGULAR]
2C [+CASE] [–SINGULAR] [±EMPHASIS]
3C [+CASE] [–NUMBER] [–PERSON] [±EMPHASIS]
4C [+CASE] [+NUMBER] [–GENDER] [–PERSON] [±EMPHASIS]

5. numbers
1C [+CASE] [–NUMBER] [–GENDER]
2C [+CASE] [–NUMBER] [+GENDER]

References

Batnožić & Ranilović & Silić (1996) *Hrvatski računalni pravopis*, MH, SYS, Zagreb

Corbet (2000) *Number*, Cambridge

Daille, Fabre & Sébillot (2002) *Applications of Computational Morphology*. In *Many Morphologies*, ed. Paul Boucher, 210-234, Somerville, MA: Cascadilla Press

Dionizije Tračanin [Dionysius the Thrax] (1995) *Gramatičko umijeće*, edition and translation by Dubravko Škiljan, biblioteka Latina et graeca, book XXXVIII, Zagreb

Franks (1995) *Parameters of Slavic Morphosyntax*, Oxford

Hausser (1999) *Foundations of Computational Linguistics, Man-Machine Communication in Natural Language*, Springer

Silić (1996), cf. Batnožić & Ranilović & Silić

Silić (2001) *Hrvatski jezik 2, udžbenik za II. razred gimnazije*, ŠK, Zagreb

Spencer (1991) *Morphological Theory*, *An Introduction to Word Structure in Generative Grammar*, Blackwell

Tadić (1994) *Računalna obrada morfologije hrvatskoga književnog jezika*, unpublished Ph.D. thesis, Zagreb

Trask (1999) *Key Concepts in Language and Linguistics*, Routledge

Van Valin, Jr (2001) *An introduction to Syntax*, Cambridge

Key words: natural language processing, computational linguistics, flective languages, Croatian language, morphosyntax, formal features, valency features, morphosyntactic features, word classes, tagging.

– e.g. ‘kopajući’, ‘radeći’, in English ‘digging’, ‘working’. Verb Adverb II is the ‘glagolni prilog prošli’ – e.g. ‘kopavši’, ‘radivši’, in English ‘having dug’, ‘having worked’.

In Search of Theme and Rheme Pitch Accents in Estonian

Maarika Traat
The University of Edinburgh
M.Traat@ed.ac.uk

Abstract. It has been proposed that certain pitch accents in English always go with theme, while others always accompany rheme (Steedman 2000a), where theme and rheme are defined in terms of contextual "alternative sets". The goal of the present pilot study was exploring whether any regularities of this sort could be found in the Estonian language, which being a "free word-order" language, differs considerably from English. On the basis of the data studied the shapes of the theme and rheme pitch accents in Estonian do not seem to be remarkably different, but the rheme accents always reach a higher maximum pitch than the theme pitch accents in the same sentence.

1 Introduction

The present paper was motivated by the claim ((Steedman 2000a), (Steedman 2000b), (Steedman 2002)) that in the English language certain pitch accents always accompany theme, while others always go with rheme. The above raised the question whether anything of the kind could be detected in the Estonian language.

The present paper analyses several prosodically annotated Estonian examples from information structural point of view, the principal goal being finding out whether there are any regularities regarding the use of pitch accents in connection with theme and rheme linguistic material.

Very little work has been done on both Estonian intonation and the information structure of Estonian sentences. Nobody has explored the possible connection between the pitch accents used and the information structure of the sentence.

Word order in Estonian being considerably freer than that in English, there is a good chance that some effects in meaning achieved by means of intonation in English may be achieved by variation in word order in Estonian. The above claim is supported by (Asu 2002) analysis of Estonian questions according to which the intonation of the questions which use very straightforward morpho-syntactic marking does not differ from that of statements,

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

while the intonation of the morpho-syntactically unmarked ones does.

1.1 Motivations behind the study of intonation

The study of intonation is interesting firstly from cognitive point of view, because the findings shed light to how people process and present information. Secondly, the knowledge about intonation is important in speech technology — both speech recognition and synthesis applications benefit from it. In speech recognition intonation can make available additional information to be retrieved from the spoken text; in speech synthesis knowing how people use intonation helps to make the speech output more natural, and also, even more importantly, helps to convey the intended meaning.

2 About the "meaning" of intonation

It turns out that people use intonation in a systematic way. The intonation contour used depends on the context and on the intended meaning of the utterance. The same string of words uttered with a different intonation contour can be appropriate in certain contexts while not in others. The following example illustrates the point (the words carrying the main pitch accent of the sentence are printed in capital letters):

(1) Q: Who taught Alexander the Great?
A1: ARISTOTLE taught Alexander the Great.
A2: *Aristotle taught ALEXANDER THE GREAT.
A3: *Aristotle TAUGHT Alexander the Great.

Even though the three answer candidates to the question in the above example contain exactly the same string of words, only the first answer is appropriate. The second and third answer candidate, even though conveying the same propositional meaning, are not appropriate answers to the question above. But each of them would be an appropriate answer to a different question. Thus, intonation seems to carry some kind of "pragmatic meaning".

2.1 The relationship between information structure and intonation

Information structure refers to the way information is packaged in an utterance. There are several proposals about the constituents of information structure. According to (Vallduví and Engdahl 1996) in spite of the big variety of proposed constituent names, there are two main bipartite informational articulations:

- **ground/focus** division
- **topic/comment** division

(Vallduví 1992) defines a tripartite hierarchical articulation, according to which a sentence is composed of focus and ground, the latter being further divided into link and tail.

The present paper uses Steedman's (Steedman 2000a) theme/rheme, an instance of ground/focus division according to (Vallduví and Engdahl 1996), to describe the information structure. The terms *theme* and *rheme* have previously been used in (Mathesius 1929), (Firbas 1964), (Firbas 1966), (Halliday 1967), (Halliday 1970).

The **theme** is a known, expected, or noninformative part of the sentence. It is the link that is presupposed to the preceding discourse, or as Halliday put it "the peg on which the message is to hang" and the element in which "the speaker announces his intentions" (Halliday 1970). The remaining part of the utterance is what moves the discourse onward — the **rheme** — a newsy, informative, or dominant part of the sentence. The following example illustrates the notions of theme (th) and rheme (rh).

(2) Q1: Who taught Alexander the Great?
A1: (ARISTOTLE)^{rh} (taught Alexander the Great.)th

(3) Q2: Who did Aristotle teach?
A2: (Aristotle taught)th (ALEXANDER THE GREAT.)^{rh}

Intonation appears to be closely related to the information structure. As proposed in (Steedman 2000a), (Steedman 2000b), (Steedman 2002) the specific pitch accent or the subset of the pitch accents that can be used in a certain position in an utterance is determined by the "themeness" or "rhemeness" of the part of the utterance.

Besides the choice of intonation contour, information structure can also play an important role in determining the appropriate word order, especially in free word order languages.

3 Some comments on related research

The works most directly related to the present study are (Steedman 2000a), (Steedman 2000b), (Steedman 2002), where Steedman discusses his compositional approach to the information structural meaning of intonation. He establishes there being two principal types of intonational tunes - theme tunes and rheme tunes. According to him the most common intonation pattern for a marked theme[1] is L+H* LH% and the most common tune for

[1]Themes can be divided into marked themes and unmarked themes. The intonation of an unmarked theme is ambiguous as far as information structure is concerned, and it does

the rheme is H* LL%[2]. He also proposes that the choice of pitch accents used conveys whether the theme/rheme is mutually agreed or not, and the boundary tones indicate whether the speaker or the hearer is "responsible for" the corresponding information unit.

A practical implementation of Steedman's information structural theory in a speech generation application was provided by Prevost (Prevost 1995). Prevost also examines the issue of contrastive stress, and how it influences the choice of the word to carry the principal pitch accent in an intonational phrase. He develops an algorithm for making this choice using alternative set semantics.

Similarly to Steedman, Pierrehumbert and Hirschberg (Pierrehumbert and Hirschberg 1990) pursue a compositional approach towards the meaning of intonation. In addition they claim that:

> /- - -/ a speaker (S) chooses a particular tune to convey a particular relationship between an utterance, currently perceived beliefs of a hearer (H), and anticipated contributions of subsequent utterances.

As for the study of the intonation of Estonian language not much has been done. Asu ((Asu 2001),(Asu 2002)) has been working out the inventory of Estonian pitch accents using the approach of autosegmental-metrical theory. But nobody has examined the relationship between intonation and information structure.

4 A quick quide to the Estonian language

The Estonian language belongs to the Finnic branch of the Finno-Ugric language family. It is characterised by a relatively free word order, a rather elaborate declinational and conjugational system — nouns, adjectives, pronouns, numerals have as many as fourteen cases, the verbs are used in four different tenses, and the ending of the verb depends on the tense, person and number. The stems of words tend to change in declination and conjugation. Estonian has no prepositions, but there are several postpositions. Estonian uses no articles and no gender. Word stress is always on the first syllable, except in the case of some relatively recent loan words. Another peculiarity of the Estonian language is that it makes lexical distinctions based on quantity - the meaning of the word depends on the length of the vowel in its stressed syllable. The vowel can have three different lengths: short, long, very long.

(4)	koli (short)	kooli (long)	kooli (very long)
	$\text{junk}_{(nom.sing.)}$	$\text{school}_{(gen.sing.)}$	$\text{school}_{(part.sing.)}$

not contrast with any previous theme. A marked theme is "marked" by one of the specific theme intonational tunes, and stands in contrast with a theme from previous discourse. See (Steedman 2000a).

[2]This notation is taken from autosegmental-metrical theory (Pierrehumbert 1980).

vala (short)	vaala (long)	vaala (very long)
pour$_{(2ndpers.sing.imperative)}$	whale$_{(gen.sing.)}$	whale$_{(part.sing.)}$

Word order in Estonian being considerably freer than that in English, there is a good chance that some effects in meaning achieved by means of intonation in English may be achieved by variation in word order in Estonian.

5 Experiment

When studying intonation two approaches are possible:

- to make subjects read handcrafted examples,
- to analyse spontaneous speech data.

The positive side of the first approach is that it allows the experimenter to have control over her data; spontaneous speech data is difficult to handle because of its enormous variety and complexity. However, the in-lab approach also has downsides - the subjects are put into an artificial environment, and the sentences to be read are set in artificial context. This can shed doubt to the naturalness of the results.

For the present preliminary study of the Estonian theme and rheme pitch accents the approach using handcrafted examples was chosen. The main reason for the choice was the desire to be able to draw conclusions from the study of a limited amount of controlled data. Altogether 80 examples read by a single speaker were studied. In order to put the sentences of the experiment into context, they were presented as answers to questions. Both marked (a clearly distinguishable change in pitch in intonation contour) and unmarked themes were viewed. The examples contained both utterances with theme at the sentence initial position, and those with rheme preceding theme. Also as in Estonian both SVO and OVS word orders are possible, examples with both word orders were studied.

The shape of the intonation contour on the stressed syllable of the word carrying the pitch accent depends on the quantity of the vowel in the syllable. Therefore, to be able to draw any sound conclusions, great care was taken about having words with the same stressed syllable vowel length in theme and rheme positions in the examples.

The following two examples illustrate the kinds of examples that were used in the study.

(5) Q: Mina müün maasikaid. Aga kes vaarikaid müüb?
I$_{nom}$ sell strawberries$_{acc}$. But who$_{nom}$ raspberries$_{acc}$ sells?
I sell strawberries. But who sells raspberries?

A1: (Vaarikaid müüb)th (Maarika.)^{rh} (OVS)
Raspberries$_{acc}$ sells Maarika$_{nom}$.

(Maarika)^{rh} (sells raspberries.)th

A2: (Maarika)^{rh} (müüb vaarikaid.)th (SVO)
Maarika$_{nom}$ sells raspberries$_{acc}$.
(Maarika)^{rh} (sells raspberries.)th

A3: (Maarika.)^{rh} (S)
Maarika$_{nom}$.
(Maarika does.)^{rh}

(6) Q: Mina müün maasikaid. Aga mida Maarika müüb?
I$_{nom}$ sell strawberries$_{acc}$. But what$_{acc}$ Maarika$_{nom}$ sells?
I sell strawberries. But what does Maarika sell?

A1: (Maarika müüb)th (vaarikaid.)^{rh} (SVO)
Maarika$_{nom}$ sells raspberries$_{acc}$.
(Maarika sells)th (raspberries.)^{rh}

A2:* (Vaarikaid)^{rh} (müüb Maarika.)th (OVS)
Raspberries$_{nom}$ sells Maarika$_{acc}$.

A3: (Vaarikaid.)^{rh} (O)
Raspberries$_{acc}$.
(Raspberries.)^{rh}

The examples studied seem to indicate that in Estonian an object cannot be placed sentence initially if it is a rheme (see 6:A2). However, an answer to a question containing just a rheme fragment (it is the only element in the answer, hence it is initial) is perfectly acceptable (see 6:A3).

6 Intonation contours and results

Figures (1)–(3) along with Examples (7)–(9) illustrate some of the examples analysed. All these examples contain a marked theme — the stressed syllable of the "marked" word is accompanied by a rise in the intonation contour. However, in Figure 2 the pitch rise on theme is much smaller than in the other two figures presented. Since the tendency was the same whenever a marked theme followed a rheme, it is probable that this rise in pitch may have been caused by a concious effort made by the subject to always produce marked themes as well as unmarked ones, and in reality a marked theme can never follow the rheme (supported by the native speaker's intuition of the author). In order to mark the theme with intonation, it may be necessary that it be placed also syntactically in a more prominent position, i.e. sentence initially.

(7) (Meie MAArika müüb)th (magusaid VAArikaid)^{rh}.
(Our Maarika$_{nom}$ sells)th (sweet raspberries$_{acc}$)^{rh}.

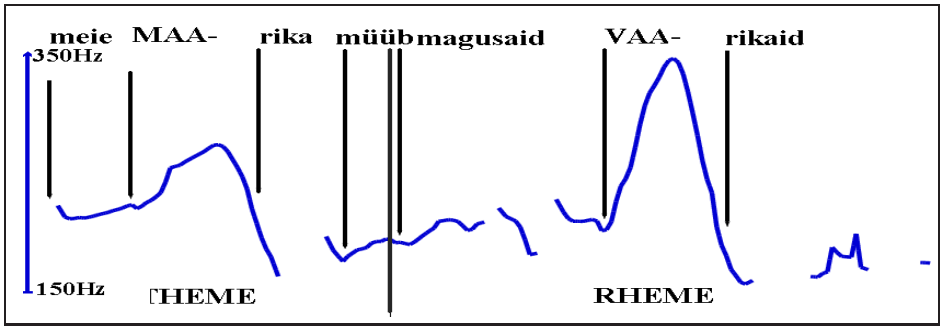


Figure 1: Illustration to Example 7

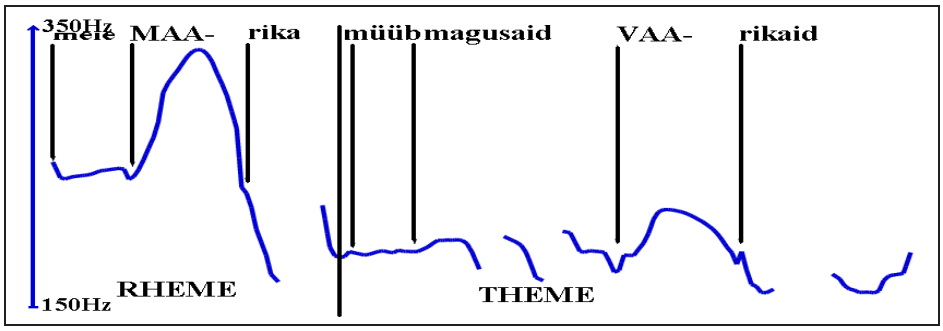


Figure 2: Illustration to Example 8

(8) (Meie MAArika müüb)^{rh} (magusaid VAArikaid)th.
(Our Maarika$_{nom}$ sells)^{rh} (sweet raspberries$_{acc}$)th.

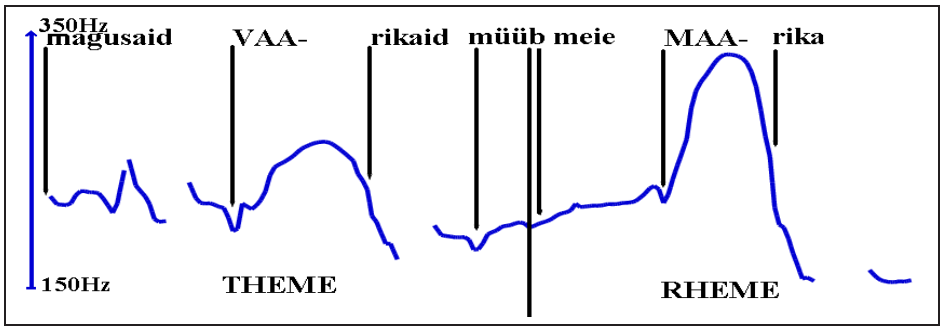


Figure 3: Illustration to Example 9

(9) (Magusaid VAArikaid)th (müüb meie MAArika)^{rh}.
(Sweet raspberries$_{acc}$)th (sells our Maarika$_{nom}$)^{rh}.

Visually, the only remarkable persistent difference between the theme and rheme pitch accents seemed to be the relative frequency of the maximum pitch. However, the picture was somewhat obscured by intra-speaker

variability. In order to discover any other systematic differences, if present, besides the difference in pitch, between the theme and rheme pitch accents, we carried out several measurements. Some of the features we measured were: the accented syllable duration in theme and rheme, the relative time when the maximum pitch was reached, and the beginning, maximum, minimum and final frequency of pitch on the main accented syllable in the theme and the rheme part of the utterance.

The measurements confirmed the visual observation that the principal significant systematic difference between theme and rheme pitch accents was their relative maximum frequency. Also the pitch at the beginning of the accented syllable was higher in the case of rheme than theme. Tables 1 and 2 illustrate the average beginning and maximum pitch of the accented syllable for themes and rhemes.

THEME	beginning pitch	max pitch
marked theme followed by rheme	237.17	276.40
unmarked theme followed by rheme	220.76	226.78
rheme followed by marked theme	190.84	213.15
rheme followed by unmarked theme	198.52	200.83

Table 1. Average beginning and maximum pitch for theme

RHEME	beginning pitch	max pitch
marked theme followed by rheme	261.24	333.56
unmarked theme followed by rheme	272.47	329.80
rheme followed by marked theme	264.81	324.12
rheme followed by unmarked theme	282.63	324.19
single rheme	260.30	286.40

Table 2. Average beginning and maximum pitch for rheme

The difference between the maximum frequency of theme and rheme pitch accents clearly serves the purpose of contrasting these two parts of the utterance with each other. The rheme has to be always more prominent than the theme. Since in the case when a question is answered using only the rheme, there is no confusion about which part is rheme and which theme, there is no need to use extremely high pitch. The maximum pitch in the single rheme answers was on average about 40 Hz lower than that in the rhemes of full sentence utterances. This observation could make us expect to encounter a lower maximum rheme accent pitch also in the case the theme in the utterance is unmarked, but the results of our study do not confirm this anticipation.

7 Conclusions and future work

According to the results of the present study theme and rheme pitch accents in Estonian:

- do not (need to) have a different shape. However, it is not excluded that there might be some pitch accents not encountered in the present study exclusively reserved for use on theme or rheme.
- differ in terms of the relative frequency of their beginning and maximum pitch.

Another observation of the present study was that it is likely that a sentence final theme cannot be marked by a pitch accent.

As for future work, an analysis of a corpus of spontaneous Estonian speech would be necessary to find all the naturally occurring intonation contours used with theme and rheme. Also the interpersonal similarities and differences in the use of intonation by different Estonian speakers need to be examined — thus the future experiments need to include more subjects, preferably unaware of the goal of the research. There has been some skepticism about whether the context setting provided by the questions in the examples used in the present study was enough to cause the subjects elicit truly contrastive marked themes, therefore different ways of setting sentences into context need to be explored. Other features besides the ones studied in the present study need to be measured — for example, the slope for theme and rheme intonation contours. The future study using a greater amount of more varied data, it would be possible to present a reliable statistical analysis. Last but not least, the interplay between word order and intonation remained out of the scope of the present paper, but there is no doubt that this is an important issue in the case of the Estonian language — therefore this problem also needs to be explored in the future.

References

Asu, E. L. (2001). An autosegmental-metrical analysis of estonian intonation: preliminary results. In *Proceedings of the Ninth International Congress for Finno-Ugric Studies*, Tartu, Estonia.

Asu, E. L. (2002). Downtrends in different types of question in estonian. In *Proceedings of the Speech Prosody 2002 Conference, 11-13 April 2002*, Aix-en-Provence, pp. 143–146. Laboratoire Parole et Langage.

Firbas, J. (1964). On defining the theme in functional sentence analysis. *Travaux Linguistiques de Prague 1*, 267–280.

Firbas, J. (1966). Non-thematic subjects in contemporary english. *Travaux Linguistiques de Prague 2*, 229–236.

Halliday, M. (1967). Notes on transitivity and theme in english, part ii. *Journal of Linguistics 3*, 199–244.

Halliday, M. (1970). Language structure and language function. In J. Lyons (Ed.), *New Horizons in Linguistics*, pp. 140–165. Harmondsworth: Penguin.

Mathesius, V. (1929). Functional lingusitics. In J. Vachek (Ed.), *Praguiana: Some Basic and Less Well-known Aspects of the Prague Linguistics School*, pp. 121–142. Amsterdam: John Benjamins, 1983.

Pierrehumbert, J. (1980). *The Phonology and Phonetics of English Intonation*. Ph. D. thesis, Massachusetts Institute of Technology. Published 1988 by Indiana University Linguistics Club, Bloomington, IN.

Pierrehumbert, J. and J. Hirschberg (1990). The meaning of intonational contours in the interpretation of discourse. In P. Cohen, J. Morgan, and M. Pollack (Eds.), *Intentions in Communication*, pp. 271–312. Cambridge, MA: MIT Press.

Prevost, S. A. (1995). *A Semantics of Contrast and Information Structure for Specifying Intonation in Spoken Language Generation*. Ph. D. thesis, University of Pennsylvania.

Steedman, M. (2000a, July). Information structure and the syntax-phonology interface. *Linguistic Inquiry 31*.

Steedman, M. (2000b). *The Syntactic Process*. Cambridge, Massachusetts: The MIT Press.

Steedman, M. (2002, June). Information structural semantics of english intonation. Draft.

Vallduví, E. (1992). *The informational component*. New York: Garland.

Vallduví, E. and E. Engdahl (1996). The linguistic realization of information packaging. *Linguistics 34*, 459–519.

Formal Representation of Property Grammars

Tristan VanRullen
Laboratoire Parole et Langage, CNRS - Université de Provence, Aix-en-Provence - France
tristan.vanrullen@lpl.univ-aix.fr

Marie-Laure Guénot
Laboratoire Parole et Langage, CNRS - Université de Provence, Aix-en-Provence - France
marie-laure.guenot@lpl.univ-aix.fr

Emmanuel Bellengier
LPL, CNRS - Université de Provence, Aix-en-Provence - France, and Semantia, ADER-PACA
emmanuel.bellengier@lpl.univ-aix.fr

Abstract. A problem for the development of constraint-based formalisms concerns (i) the maintainability of the grammars (ii) the modification of the programs according to this maintenance operations. We propose a solution to these difficulties by formally describing the global behaviour of the constraint satisfaction process and by expressing mathematically the locale functioning of each constraint. The proposed solution allows modifying the grammar without modifying the code of the program.

1 Introduction

In this paper, we propose a formal representation of a constraint-based formalism called *Property Grammars* (presented in [Bla01]). This formalism is totally based on constraints (also called *properties*): they represent all linguistic information, which allows parsing to be only conceived as a constraint satisfaction problem. The aim of this article is to show that we can provide a sufficiently well-defined logical representation of the formalism. This logical representation permits to modify the grammar without modifying programs.

We first present the role of properties in the Property Grammars formalism (hereafter PG). Then we describe and build the logical representation of PG. Finally we show a set of properties defined in the ground of a currently developed French grammar.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

2 Constraint-based formalisms

Almost all linguistic formalism make use of the notion of constraint[1] which indicates, in its broadest meaning, a property, relying two or more objects, that has to be satisfied. Constraints are extremely useful (and then extremely usual), both to describe linguistic information, and to control its parsing process. Nevertheless, the room given to these constraints may differ substantially from one approach to another: in certain cases they only form an auxiliary mechanism, though in other ones the entire theory revolves around them.

In this section, after having explained the reasons why a totally constraint-based approach is so valuable for natural language representation and processing, we will illustrate some of the problems it states through the case of current linguistic formalisms and indicate how the PG formalism allows to resolve some of these problems.

2.1 The role of constraints in current linguistic theories

From an implementation point of view, constraints present several restrictions of use. More precisely, if parsing can easily be considered as a constraint satisfaction problem, it is obviously much more complicated to concretely implement this idea. Most of contemporary approaches claim to really implement their parsing algorithm as a constraint satisfaction problem. Their way to represent it is (roughly) to specify a set of variables, their domain and a constraint system; the satisfaction mechanism consists then in constantly verifying the consistency of this system. Although, the main outcoming problem comes from the specification of the constraint variables, that usually defines high-level structures and/or partially constructed objects, and thus prohibits an immediate access to their values. More precisely, one can distinguish *sequential methods* on the one hand, which makes use of rules systems and constructs separately different representation levels (like in [PS93]), and *incremental methods* on the other hand, in which parsing is considered as a parallel process (like in HPSG: see [PS94]). In the former, one has to select, among the convergent (well-formed) derivations, the optimal one; in this approach, constraint satisfaction is necessary but not sufficient to evaluate grammaticality. In the latter, typically a constraint-based one, one has to replace the set of rules by an interactive constraint system (see [Pol96]), which allows a more direct connection between the grammar theory and its implementation.

[1] For example, see [SW99] concerning HPSG, [PS93] or [AL97] concerning Optimality Theory, and [DT99] or [Mar98] concerning Constraint Dependency Grammars.

2.2 The role of constraints in Property Grammars

Actually, we think that the implementation problem we have just pointed out comes from the incompatibility of classical generative interpretations with a totally constraint-based approach. We put forward in this paper a parsing framework, based upon PG, allowing to handle this difficulty by maintaining scrupulously a direct connection between linguistic theory and computational interpretation (cf. [JL99]). Thus, similar concepts are used both in a linguistic and in a computational outlook, which avoids developing multiple *ad hoc* (and therefore limited) parsing solutions. Moreover, this constraint satisfaction perspective allows to handle incomplete data, and therefrom to conceive, at last, a unique robust parser so as to deal with unrestricted texts, being able to treat, at least partially, any kind of input.

In PG, constraints play both roles to represent information, and to constitute the centre of the parsing process. Moreover, building a local structure is not necessary for the verification of the constraint system. Finally, this approach relies on the notion of *characterization*[1](in place of grammaticality), which allows the verification of properties on any kind of input (including non-grammatical ones).

In this model, all information concerning relations between categories is represented by means of a set of (unordered) constraints. Basically, we currently use the following set of properties for the description of a large-scale grammar for French (under development): *requirement* (cooccurrence between categories or sets of categories), *exclusion* (cooccurrence restrictions between categories or sets of categories), *linearity* (linear precedence constraints), *dependency* (lexical or semantic relations), *obligation* (a list of possible heads of a category[2]), and *uniqueness* (one category cannot occur twice). These properties are introduced in the table 1. Every property contains some clauses that express relations between the mentioned categories or sets of categories. The parsing result (*i.e.* the characterization) is composed of two lists: the first one ($P+$) contains the satisfied clauses, the second one ($P-$) contains the violated clauses.

Each property has got its own satisfaction mechanism. The following enumeration of each property's satisfaction mechanism is the basis for the developments driven in the next section. Properties express constraints over the input (during the parsing process) by means of logical hybrid clauses where variables are terms referencing the tokens with optional feature patterns waiting to match the tokens' features. In the following descriptions, an available term defines a term matching up a token and its features. More information about these notions can be read in the next section.

In this section, we have shown the handling of constraint satisfaction in current constraint-based formalisms, and in Property Grammars. We will

[1] A characterization is the state of the constraint system at the end of the parse.

[2] Possible heads of a category can also be handled by the dependency properties.

see in details below how the difficulties we have seen can be solved with PG.

Property	Semantics of the Property's clauses	Sample clauses

Requirement
$term_1 \Rightarrow term_2$
	if $term_1$ is not available **then**
	the clause is not available;
	else
	if $term_2$ is not available **then**
	put the clause in P_i- (violated);
	else
	put the clause in P_i+ (satisfied);
	end if
	end if
		For the category "Adj"
		(adjective or adjectival phrase):
		$Adj \Rightarrow Adj$ or Adv or $Coord$ *
		(i.e. in an adjectival phrase, an adjective requires the presence of another adjective or an adverb, or a coordonner)
		*: Such a clause is not a tautology because left and right "Adj" are two different variables

Exclusion
$term_1 \neq term_2$
	if $term_1$ is not available **then**
	the clause is not available;
	else
	if $term_2$ is available **then**
	put the clause in P_i- (violated);
	else
	put the clause in P_i+ (satisfied);
	end if
	end if
		For the category "Adj"
		(adjective or adjectival phrase):
		$Coord \neq Adv$ or $V_{participle}$
		(i.e. in an adjectival phrase, a coordonner cannot co-occur with an adverb or a participle verb)

Linearity
$term_1 \ll term_2$
	if $term_1$ is not available **or** $term_2$ is not available **then**
	the clause is not available;
	else
	if $term_1.rank \geq term_2.rank$ **then**
	put the clause in P_i- (violated);
	else
	put the clause in P_i+ (satisfied);
	end if
	end if
		For the category "NP" (noun phrase)
		Det or $Num \ll Noun$ or Adj or Adv
		(i.e. in a noun phrase, a determiner or a numeral must occur before a noun, an adjective or an adverb)

Dependency
$term_1 \approx term_2$
	if $term_1$ is not available **or** $term_2$ is not available **then**
	the clause is not available;
	else
	if $term_1.feature \neq term_2.feature$ **then**
	put the clause in P_i- (violated);
	else
	put the clause in P_i+ (satisfied);
	end if
	end if
		For the category "NP" (noun phrase):
		$Noun_{\{gender\}} \approx Det_{\{gender\}}$
		$Noun_{\{number\}} \approx Det_{\{number\}}$
		(i.e. in a noun phrase, a noun and a determiner must agree in gender and number)

Table 1: *Formal description of the properties' satisfaction mechanism*

3 Logical and mathematical background

3.1 Parsing with constraints

Before exploring the main point of this section, an overview of former works on parsing with PG should demonstrate the advantages of retrieving a logical frame in their mechanisms. Parsers programmed in the paradigm of PG have in fact been built with different goals and for different uses: some to permit speech synthesis (see [BH01]), others to parse large corpora or to extract rich information (see [BGV03]), and some to test the PG formalism and to compare it with other ones (see [BV02]). We can distinguish them in terms of determinism, deepness, dynamism, as well as in terms of programming language, heuristics and complexity. This is the way we usually compare parsers. In the current consideration, the only comparison will be

done in terms of genericity. From shallow parsers to deep ones, many make use of logico-mathematical fundaments and each program offers its own interpretation (see for different examples [BM01] and [BBV02]). With CHR or PROLOG rules, with graphs, trees or tables, heuristics are intimately linked to data structures and generally to the formalism's implicit semantic fundings.

The observation of how the already built parsers deal with constraints and with the evolution of the formalism itself allows to catalogue their similarities and principles. Such an inventory can lead to find neither general nor universal rules, but a formal system of constraint mechanisms, homomorphic with other potential systems. The following points characterize the parsing processes with PG, whatever the program and its heuristics:

- Parsers use bottom-up, tree walking strategies with a pattern matching step for constraint satisfaction.
- It is possible to enumerate different kinds of patterns to match:
  - constants, such as feature values, to recognize in the inputs,
  - variables referencing linguistic tokens,
  - variables referencing tokens' features (even domains of these features),
  - second order typed calculus functions expressing relations and constraints over patterns.
- Several common points in the mechanism of the properties allowing a generalization of their writing.

Writing a generic parser based on these points can now be imagined. This is what we explain hereafter.

3.2 Computational perspective

Many constraints interfere simultaneously for the choice of the grammar's constraints representation. The problematic governing these constraints should be given in epistemological terms (so that a large overview of relations between linguistic models and theories is kept), as in mathematical terms (by choosing a model, totally representative of its set of objects without implicit knowledge, and canonical for the treatments it allows). As shown on figure 1, different representations of a grammar lead to different outputs, which can differ in their organisation, but which should contain the same (or similar) information. Choosing one kind of representation is thus a procedure driven by a need of genericity.

Another question in the choice of a representation is given by the computational perspective: a program that would have to be changed (partially

or totally) anytime the grammar changes, would be too heavy to maintain and to develop. That is why between different kinds of representations, for a given grammar, we have to choose the simplest and the one that avoids *ad hoc* programming.

The convergence point for these requirements can only pass through a logical and mathematical consideration:

- Firstly to describe the global behaviour of constraint satisfaction in the chosen formalism, by giving the set of mathematical objects needed for this description. This will be directly used to program the specific tools able to deal with the grammar.
- Secondly to express the local functioning of each kind of clause to be satisfied. This part consists in writing the constraints by using the objects of the same mathematical set as hereupon. In the computational perspective, this task will lead to write a separate file containing the mathematical description of each kind of constraint to be satisfied.

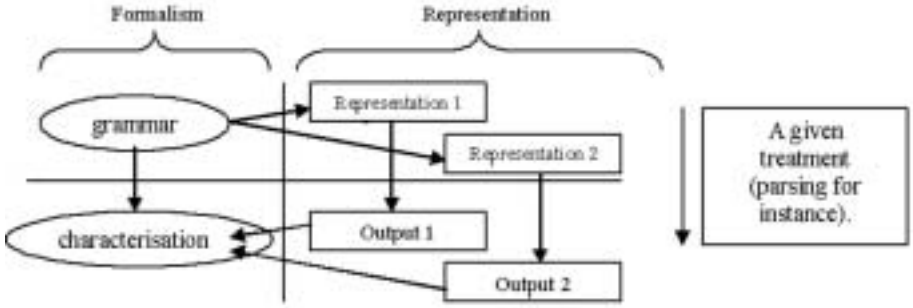


Figure 1: *Need of genericity for grammar representations and of similarity between outputs of a given treatment*

4 Building a constraint description

4.1 Levels of logic and hybrid constraint systems

Constraints in PG are not hierarchical, but are organized like sets of clauses. Each set of clauses is called a property. All clauses of all properties have similar characteristics even if their behaviour is different, that is why it will be possible to enumerate and define a set of attributes necessary for all properties. All clauses belonging to a property have the same satisfaction procedure, which can be expressed using the second order logic and typed calculus.

As clauses are expressed according to an input and make reference to categories of the grammar, we must define the grammar and the input before

expressing the clauses, which can be done in an extended Backus-Naur Form.

- Grammar G ::=
  [[set C of Categories],
  [set D of property definitions]]

Let $card(C)$ be the cardinality of C, let c_i (i$\in$ $[0 \dots card(C)]$) be the i^{th} category of C,

- Category c_i ::=
  [name of the category (*label*),
  [set of Properties P_{c_i}],
  [set M_{c_i} of features]]

Grammatical categories can be lexical categories (like *noun*, *verb*, etc.) as well as syntagmatic categories (NP, VP, PP, ...). The sets of features are not described here, because their importance is minimal for the logical representation of properties. An input for the grammar is a vector of heterogeneous information, supposed to describe enough each parsed token:

- Input I_r ::=
  [a word (*label*),
  r (rank order, position in the sentence)$\in$ $[0 \dots \infty[$,
  a disambiguated category c_r belonging to C,
  [a set M_{c_r} of features]]

Each property given for each category in the grammar is a set of property clauses (called *clauses* because of their satisfying procedure, but we can define them as truth functions over hybrid second order arguments). We can separate the enumeration of the clauses of a property from the definition of a property's behaviour (which does not depend on the category).

- Property P_c ::=
  [Name of the property (*label*),
  [a set K_{p_c} of clauses]]

A given clause k_{p_c} in a given property p of a given category c can be defined like a set of terms:

- Clause k_{p_c} ::=
  [ordered set T of a_p terms]]
  (a_p, the arity of a clause is defined hereunder)

- Term t ::=
  [Reference or Logical expression]

- Reference ρ ::=
  [Name of the referenced category ($label$)$\in C$,
  [a pattern of features M_ρ (optional)]]

- Logical Expression E ::=
  [Logical operator O,
  [a set of term (its cardinality agrees with O's arity)]]

A property definition furnishes the symbol (name) of its clauses (*e.g.* the symbol for a clause of linearity is $\ll$) and describes the context of their satisfiability with a mathematical expression. The cardinality of the set of clauses can be fixed or variable, and an order relation between the clauses of a property may have to be observed (see table 1).

With the definitions given here, we can now write the main definition (for this article), dealing with the goal of mathematical and logical genericity.

- Property definition d_p ::=
  [Name of the property ($label$),
  Symbol for its clauses ($label$),
  (a_p (arity of the clauses)$\in$ (0, 1, 2 ...) (finite integer),
  Expression of satisfiability E_p for the clauses,
  cardinality of the set of clauses (fixed or not) $\in$ (0, 1, 2 ... ∞),
  a boolean $Ordered \in (true, false)$]

The expression E_p describes how a clause can be satisfied in a property p by using second order logic and typed calculus (see below). Considering a grammar description as a preliminary for a parsing process, we can define some characteristics of the references and the logical expressions, in accordance with a supposed input characteristics:

Definition 1. availability of a reference: A *reference* has to be interpreted as a variable declaration; a variable has a name and refers to an existing category of the grammar. It can also be sub-defined by describing a pattern of features referring to the category's set of potential lexical or semantic features. During the parse, an input will be presented to each variable referring to it. If the input features match the variable pattern, the reference is then declared available.

Definition 2. availability of a logical expression: A *logical expression* is available when all of its terms are available.

Notation:
Let t_1, t_2, t_i, ... t_{a_p} be the terms of a clause in K_p verifying i $\in [1...a_p]$.

Definition 3. availability of a term: We define a truth value ω_i (*availability of a term i*) which is *true* when the term t_i, is available and $false$ when not available (a term can be a logical expression or a single reference).

Definition 4. availability of a clause: According to the semantics of each property p, the availability of a clause k_p is a logical expression Ω of the a_p truth values (ω_1, ω_2, ω_i, ..., ω_{a_p}) such as $\Omega(\omega_1, \omega_2, \omega_i, \ldots, \omega_{a_p}) \supset$ (k_p is available).

Definition 5. satisfiability of a clause: A clause can be evaluated as satisfied or not when and only when it is available. An available clause is satisfied if its terms verify a given condition over its terms $\Psi(t_1, t_2, t_i, \ldots, t_{a_p})$. Because a term consists in a reference or in a logical expression based on references, all available information in these references can be used in the expression of the condition Ψ, like the *rank* of the i^{th} token or even a function of potential values or features of the inputs.

We now can write the definition of an expression of satisfiability:
- Expression of satisfiability E_p ::=
  [Truth function of *availability* $\Omega(\omega_1, \omega_2, \omega_i, \ldots, \omega_{a_p})$,
  Condition of *satisfiability* $\Psi(t_1, t_2, t_i, \ldots, t_{a_p})$]

4.2 Definition of some properties

Requirement

name =	"Requirement"
symbol of clauses =	'⇒'
arity of clauses =	2
Satisfiability of clauses =	$\lvert\ availability = \omega_1 \wedge \omega_2$ $satisfiability =$ '*true*' $\rvert$
cardinality =	∞,
ordered =	*false*

Exclusion

name =	"Exclusion"
symbol of clauses =	'≠'
arity of clauses =	2
Satisfiability of clauses =	$\lvert\ availability = \omega_1 \wedge \neg\omega_2$ $satisfiability =$ '*true*' $\rvert$
cardinality =	∞,
ordered =	*false*

Linearity

name =	"Linearity"
symbol of clauses =	'≪'
arity of clauses =	2
Satisfiability of clauses =	$\lvert\ availability = \omega_1 \wedge \omega_2$ $satisfiability = t_1.rank < t_2.rank$ $\rvert$
cardinality =	∞,
ordered =	*false*

Dependency

name =	"Dependency"
symbol of clauses =	'≈'
arity of clauses =	2
Satisfiability of clauses =	$\lvert\ availability = \omega_1 \wedge \omega_2$ $satisfiability = t_1.feature < t_2.feature$ $\rvert$
cardinality =	∞,
ordered =	*false*

Table 2: *Formal description of properties with the introduced notations*

The table 2 expresses the semantics of properties in the grammar, corresponding to the table 1. It presents the advantage of being extensible. With the introduced notions, it will be possible to describe semantic or prosodic phenomena without building new parsers. With the fundaments described here, the pattern matching step still has to be defined in a compatible formulation, which will be possible by designing data structures like graphs where nodes and edges carry dynamic semantic objects. In this direction, several computational paradigms propose realizable solutions, like classical CSPs, parallel processing, reactive machine and hybrid models or graph walking. Our current work is based on the third one. It should lead to a deep parser with variable granularity (see [BBV02] and [OCC+00]).

5 Conclusion

Constraints offer several advantages, both for linguistics and computational reasons. We have seen that a Property Grammar representation can express constraints directly over categories and use constraints at any level of the parsing process.

We have shown that a well defined logical representation of the properties allows the modification of the grammar without any modification of the program (in a parsing strategy). Property Grammars seem to be a satisfying tool for the representation of linguistics information and for a computational implementation. The perspective of working with hybrid constraints within the formalism added to the possibility of relaxing violated constraints without giving up the parsing process will be decisive in the future of symbolic parsing paradigm.

References

[AL97] Diana Archangeli and Douglass Terence Langendoen, *Optimality theory*, Blackwell, 1997.

[BBV02] Jean-Marie Balfourier, Philippe Blache, and Tristan VanRullen, *From shallow to deep parsing using constraint satisfaction*, Proceedings of the 27th International Conference on Computational Linguistics (COLING'02), 2002, pp. 36–42.

[BGV03] Philippe Blache, Marie-Laure Guénot, and Tristan VanRullen, *Corpus-based grammar development*, Proceedings of Corpus Linguistics 2003, 2003, pp. 124–131.

[BH01] Philippe Blache and Daniel Hirst, *Aligning prosody and syntax in property grammars*, Proceedings of EuroSpeech 2001, 2001.

[Bla01] Philippe Blache, *Les grammaires de propriétés : des contraintes pour le traitement automatique des langues naturelles*, Hermès Science, 2001.

[BM01] Philippe Blache and Franz Morawietz, *A non-generative constraint-based formalism*, Travaux Interdisciplinaires du Laboratoire Parole et Langage d'Aix-en-Provence **19** (2001), 11–26.

[BV02] Philippe Blache and Tristan VanRullen, *An evaluation of different shallow parsing techniques*, Proceedings of LREC-2002, 2002.

[DT99] Denys Duchier and Stefan Thater, *Parsing with tree descriptions: a constraint-based approach*, Proceedings of Sixth International Workshop on Natural Language Understanding and Logic Programming(NLULP'99), 1999.

[JL99] David Johnson and Shalom Lappin, *Local constraints vs. economy*, CSLI, 1999.

[Mar98] Hiroshi Maruyama, *Structural disambiguation with constraint propagation*, COLINGACL'98, 1998.

[OCC+00] Stephan Oepen, Ulrich Callmeier, Ann Copestake, Dan Flickinger, and Robin Malouf, *Scalable grammar software for computational linguists*, Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL'00), 2000.

[Pol96] Karl Pollard, *The nature of constraint-based grammar*, Proceedings of the 10th Pacific Asia Conference on Language, Information and Computation (PACLING'96), 1996.

[PS93] Alan Prince and Peter Smolensky, *Optimality theory: Constraint interaction in generative grammar*, Rutgers University Centre for Cognitive Science, 1993.

[PS94] Karl Pollard and Ivan Sag, *Head-driven phrase structure grammars*, CSLI, Chicago University Press, 1994.

[SW99] Ivan Sag and Tom Wasow, *Syntactic theory. a formal introduction*, CSLI, 1999.