RESTARTING AUTOMATA: MOTIVATIONS AND APPLICATIONS

Martin Plátek
KTIML MFF, Charles University, Prague,
Malostranské nám. 25, CZ-118 00 Praha 1 - Malá Strana, Czech Republic
e-mail: platek@ksi.ms.mff.cuni.cz

Markéta Lopatková
Centre for Computational Linguistics, MFF, Charles University, Prague,
Malostranské nám. 25, CZ-118 00 Praha 1 - Malá Strana, Czech Republic
e-mail: lopatkova@ckl.ms.mff.cuni.cz

and

Karel Oliva
Austrian Research Institute for Artificial Intelligence (OeFAI),
Freyung 6, A-1010 Wien, Austria
and
Institute of the Czech Language, Academy of Sciences of the Czech Republic,
Letenská 4, CZ-110 00 Praha 1 - Malá Strana, Czech Republic
e-mail: karel@oefai.at, oliva@ujc.cas.cz

ABSTRACT

Automata with restart operation (henceforth RA) constitute an interesting class of acceptors and reduction systems at the same time, and simultaneously display a remarkable *'explicative power'* wrt. the analysis of formal and natural languages. This paper focuses on RA as a formal-theoretical tool for modelling the *analysis by reduction* of natural languages and on the role of analysis by reduction (i.e. of analysis by RA) in dependency approach to syntax, and it also outlines some practical applications.

1. Restarting automata – bottom up analysers

The aim of this contribution is to present the main motivations for study of restarting automata and outline some applications of restarting automata in linguistics in an informal way. All the formal notions can be found in [1] or in [3], which also contains the bibliographic data.

A *restarting automaton* M can be described as a non-deterministic device with a finite control unit and a head with a lookahead window attached to a linear list. The automaton works in *cycles*. Within one cycle, it performs the following sequence of actions:

- *moves* the head along the word on the list (possibly in both directions);
- *rewrites* — a limited number of times within one cycle — the contents of the list within the scope of its lookahead by another (usually shorter) string defined by instructions of the automaton;
- *continues* by scanning some further symbols;
- *"restarts"* – i.e. resets the control unit into the initial state and places the head on the left end of the (shortened) word.

The computation halts when M enters an accepting or a rejecting state.

M can be viewed as a *recognizer* and as a (regulated) *reduction system*, as well. A *reduction* by M is defined trough the rewriting operations performed during one individual cycle by M, in which M distinguishes two alphabets: the input alphabet and the working alphabet.

For any input word w, M induces a set (due to non-determinism) of sequences of reductions. We call this set the *complete analysis* of w by M ($CA(w, M)$). $CA(w, M)$ is composed of accepting sequences and rejecting sequences of reductions. If, for a given w, $CA(w, M)$ contains at least one accepting sequence, w is accepted. Otherwise, w is rejected.

From the techniques summarized in [1] and quoted in [3] it follows that restarting automata define a taxonomy for types of *regulation* of bottom-up syntactic analysers (parsers) for several types of (originally unregulated) grammars (e.g., CFGs, pure grammars, Marcus grammars, categorial grammars, and for variants of these allowing for discontinuous constituency).

Restarting automata enable direct modelling of correct syntactic phenomena as well as of syntactic incorrectness. This is crucial since in the current techniques of computational linguistics the positive and negative observations (constraints) play equally important roles. Let this be explained in more detail. Syntactic observations of natural languages are collected stepwise, in a slow and uneasy way. At any moment of the development of a realistic syntactic description, the collection of positive syntactic observations is far from complete and precise. The same, then, holds also for negative observations. Because of lack of initial completness and precision, both types of observations have to be collected iteratively and gradually integrated into the analysers (grammars), and their formal description then individually verified and debugged. Such an approach, in turn, can be efficiently supported by (implementing the analysers by means of) restarting automata.

2. Restarting automata – analysis by reduction

A special (restricted) type of restarting automata is the RLW-automata, cf. [3].

The operation of an RLW-automaton can be viewed as implementation of a set of meta-instructions, each of which describes the actions to be performed within an individual cycle of the automaton. Each meta-instruction has the form $(R_1, u \to v, R_2)$, where R_1, R_2 are regular expressions, u, v are strings over the input alphabet ($|v| < |u|$), and $u \to v$ is a rewriting rule. In one cycle, the RLW-automaton rewrites (according to the meta-instruction) the contents u of its lookahead by the string v (only one rewriting operation can be performed within one cycle). An important property of a meta-instruction is that it can be verified (debugged) individually.

A very important feature of operation of RLW-automata is the "error preserving property": if an input sentence is incorrect then the reduced sentence is also incorrect (after any number of cycles). In addition, deterministic RLW-automata have "correctness preserving property": if a sentence is correct then the reduced sentence is correct as well. Obviously, both these properties are important for the description (and build-up of syntactic structures) of natural languages.

Due to the above, the RLW-automata can serve as a framework for formalization of the analysis method called *analysis by reduction* (AR), which constitutes a natural and above all solid basis for building dependency grammars of natural languages.

In particular, AR helps to find (define) the relations of:

- *vertical dependency*, which holds in case one of the two words constituting the dependency pair can be omitted without harming the grammaticality of the sentence, while the other word cannot be omitted;

- *horizontal dependency*, which holds if none of the two words constituting the dependency pair can be omitted without making the result ungrammatical;

- *coordination* and other non-dependency relations;

as well as to state which pairs of words have no syntactic relation to each other.

Let it be remarked that in the visualization of dependency trees (grammars), both vertical and horizontal dependencies are usually represented as one type, e.g., as oblique dependencies (see the Figs. below for examples of dependency trees). The transformation of vertical dependencies into the oblique dependencies is trivial. On the other hand, in order to obtain the oblique dependencies from the horizontal ones (i.e. to determine the 'governors' of these dependencies) finer methods are to be used, (e.g., analogy).

Let us outline some principles of the analysis by reduction and its relation to syntactic (in)dependencies and coordinations:

- The *AR* consists of stepwise simplification of a complete sentence in such a way that its syntactic correctness is preserved; at the end, the 'core' of the sentence is obtained.
- The reduction is the basic operation of *AR*: in each step (cycle) the simplification is performed by deleting at least one word of the sentence and possibly rewriting other words in the sentence. The deletion is 'justified' by linguistically motivated rules.
- The process is non-deterministic, mutually *independent* words (parts) can be deleted in any order (preserving syntactic correctness is the only criterion for the success of the reduction).
- All the *vertically dependent* words (parts) must be deleted before their governor (from the corresponding dependency structure) is deleted; all the *horizontally dependent* words must be deleted simultanously (i.e. within one cycle).
- Two parts of a sentence can be considered as a *coordination* only if these parts are not (syntactically) dependent on each other.

The following three examples illustrate the way how dependency structures are obtained via *AR*. Note the contrast between the (superficial) similarity of the three sample sentences and the differences among the resulting types of *AR*'s.

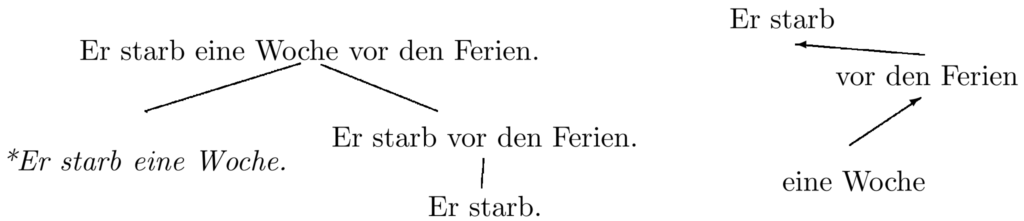

Figure 1: First example of AR and the corresponding dependency structure.

The sentence in Fig.1 can undergo two different reductions in the first reduction step (in the left part of Fig.1): either the deletion of the string of horizontally dependent words *vor den Ferien* or the deletion of the horizontally dependent pair *eine Woche*. In the first case the syntactic correctness of the sentence is violated (which is marked off by an asterisk and italics – in this case the reduction fails), in the second case the correctness is preserved. The second reduction step is then determined unambiguously: it consists in the deletion of the words *vor den Ferien*. In such a way we have obtained only one correct sequence of reductions and one incorrect reduction. This result determines unambiguously the dependency structure from the right part of Fig.1. The oblique (in this example, originally vertical) dependencies are represented by arrows, the horizontal dependencies are represented by the remaining strings of words in the 'nodes'.

Let us consider the second example. In this case only the first reduction is correct, while the second one yields an incorrect string. In this way we obtain only one arrow denoting a vertical dependency in the resulting dependency structure (right half of the picture). It is the one between the nodes with *Angst* and *vor den Ferien*. The other one (originally horizontal), between *Er hatte* and *Angst*, is obtained by analogy, i.e. via the observation that the parts with finite verbs occur "usually" on the top of a dependency structure, similarly as in the previous example.

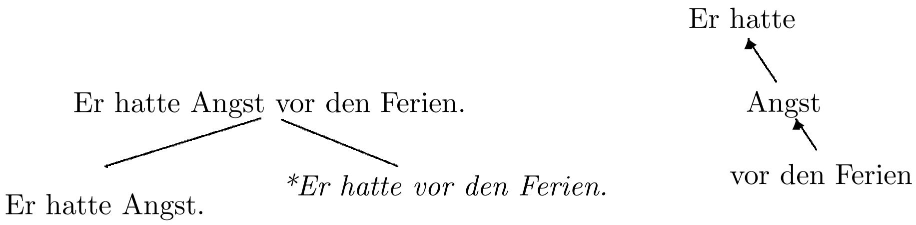

Figure 2: Second example of AR and the corresponding dependency structure.

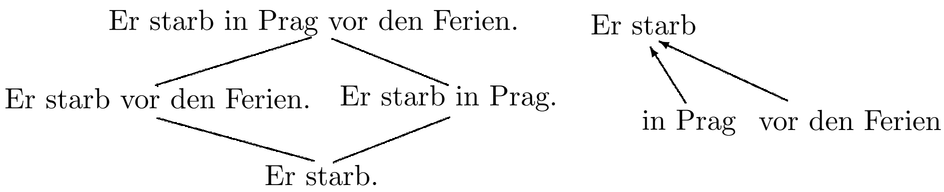

Figure 3: The example of AR for an independence.

The third example serves as an example of how to discover independence between two parts of the sentence. The deletions of the strings *in Prag* and *vor den Ferien* can be performed in both orders. It means that this two pairs are mutually (syntactically) independent (and, in this particular case, are both dependent on the finite verb).

It is the aim of the approach described here to find transformations from restarting automata modelling *AR* into dependency analyzers.

3. Two other applications: disambiguation and grammar-checking

Among the topical problems of current corpus linguistics (i.e. the branch of linguistics dealing with large bodies of running texts - so called *corpora*) is the *resolution of morphological ambiguity* of words in a text corpus - either on the level of Part-of-Speech (PoS) of a word (e.g., the German wordform *sein* has to be disambiguated between the pronominal and the verbal reading) or on some more fine-grained level (e.g., the noun *Leiter*, ambiguous between feminine and masculine gender). There exist different approaches to this *disambiguation*, including statistical ones, memory-based learning ones etc. The most accurate among them (at least up to date) is the rule-based disambiguation. This kind of disambiguation presupposes preprocessing the text by a morphological analyser, which assigns the set of all its potential morphological readings to each word of the input. In the main processing phase, the disambiguation then employs explicit rules (created by linguists) for the (stepwise) elimination of those readings which are impossible in the context of the word token. In particular, the elimination of a reading of a word relies on finding (for the very reading) a *disambiguation context*, which is to be understood as a context in which a particular reading of a particular word is impossible (and hence the reading can be safely removed, by means of which disambiguation is performed). Thus, e.g., in the clause *weil niemand meinen Eltern einen lebenswürdigen Unterhalt versprechen konnte* the finite verb reading of *meinen* can be eliminated based on the occurrence of the unambiguous finite verb *konnte* and on a rule postulating that in any correct German sentence, a comma or a coordinating conjunction must occur somewhere between two finite verbs (for a complete disambiguation, an application of another rule, leaning on some other context, would be needed in order to remove the infinitival reading of *meinen*). While the creation of such rules is a linguistic-theoretical task, it is quite obvious that their application is an issue which is in fact identical to the idea of restarting automata. This can be seen from the following two points:

- each rule looks for the configuration it describes (i.e. for a particular reading with its context), and when such a configuration is detected, the rule serves for deleting the particular reading (leaving the context untouched); after the application of the rule, the whole process has to start again on the newly defined input (i.e. input after the deletion), using some other rule (or maybe the same one on another configuration within the sentence)

- successful application of one rule can trigger the application of another rule (the first rule disambiguates and hence gives rise to a context needed for the application of the second one); the subtle point here is that this is obviously dependent not only on the rules, but also on the sentence to be disambiguated: two rules R_1 and R_2 applied in this order can work well for the sentence S_1, while for another sentence S_2 this order does not work, but the order R_2, R_1 would work. This problem has a straightforward and easy solution, however: given a set of rules $R_1, R_2, ..., R_n$, this set should be applied such that all possible rule permutations (non-determinism) are simulated. Thus, any sentence may be processed, since also the order needed for this sentence is applied (if all are, then also the one needed) - and this is exactly what a non-deterministic restarting automaton with deletion does.

Let it be mentioned in passing that the approach described above for disambiguation can also be used for grammar-checking purposes. In particular, if the morphological analysis and the set of disambiguation rules are applied on a sentence, and if this results in deleting all readings of one word supplied by the morphological analyser, then it is obvious that the readings of this word are in disaccord with the readings of other words of the sentence, i.e. the sentence is syntactically ill-formed. Thus, the corresponding restarting automaton is in fact able to discover (at least some) cases of ill-formed sentences, and in fact also the source of the error can be located as the "last configuration before deletion" (at least in a "reasonable" number of cases). Some grammar-checking techniques based on restarting automata are outlined in [2] and some new techniques of *error recovery* are currently studied.

Acknowledgements

We are indebted to J. Panevová for stimulating discussions concerning analysis by reduction. M. Plátek and M. Lopatková are supported by grants No. 201/02/1456 of GAČR and LN00A063 of MŠMT ČR. K. Oliva is supported by Grant No. P16614-G03 of FWF. OeFAI is supported by the Austrian Federal Ministry of Education, Science and Culture.

References

[1] P. Jančar, F. Mráz, M. Plátek, J. Vogel: On Monotonic Automata with a Restart Operation. *J. Autom. Lang. Comb.* 4 (1999), 287-311.

[2] V. Kuboň, M. Plátek: A Grammar Based Approach to Grammar Checking of Free Word Order Languages, In: *Proceedings COLING 94, Vol.II,* Kyoto, August, 1994, 906 - 910.

[3] F. Otto: Restarting Automata And Their Relations to the Chomsky hierarchy. In: Z. Esik, Z. Fülöp (eds.): *Developments in Language Theory, Proceedings of DLT'2003*, LNCS 2710, Springer Verlag, Berlin, 2003, 55 - 74.

[4] K. Oliva, P. Květoň and R. Ondruška: The Computational Complexity of Rule-Based Part-of-Speech Tagging. To appear in: *Proceedings of TSD 2003*, to be published in the LNAI Series by Springer Verlag.