Capturing The Meaning of Time Expressions – A Functional Approach

Petr Němec

Institute of Formal and Applied Linguistics,
Faculty of Mathematics and Physics, Charles University
Malostranské náměstí 25
118 00 Praha 1, Czech Republic
nemec@ufal.mff.cuni.cz

Abstract

In this paper we present a functional approach to capture the information conveyed by various time expressions within a discourse. The approach is motivated by annotation scheme that captures general temporal relations between events expressed in a discourse. A parser for Czech as well as an inference engine that makes it possible to compare functional compositions is implemented.

1. Introduction

In this paper we present a functional approach to capture the meaning of various time expressions within a discourse. This endeavor is motivated by the possibility to use the information conveyed by these expressions to draw event ordering inferences, thus improving the performance of an automatic event-ordering system.

An inference engine that decides on the ordering of the events associated with the respective time expressions is also implemented. We also present one particular parsing engine that produces the functional representation of time expressions for Czech.

The paper is structured as follows: Section 2 introduces the basic principles of the temporal annotation framework that encapsulates the functional formalism used to capture meaning of time expressions. Section 3 describes the formalism in detail. Section 4 gives a brief overview of the annotated corpus. Section 5 introduces the implemented inference engine that compares time extensions represented by the respective functional compositions as well as a parser that builds these compositions based on a syntactic representation of Czech sentences. Section 6 compares the presented approach to similar work in the field and Section 7 concludes the paper.

2. Annotation Scheme Overview

In this section we give a very brief overview of the formalism used to capture temporal relations between respective events expressed in a discourse . The scheme is described in greater detail in (Němec, 2006). There are alternative annotation schemata (such as TimeML (Pustejovsky et al., 2005)) but as the role of a particular scheme is only auxiliary for the purposes of this paper we do not compare respective approaches. The described scheme can be viewed as a minimal "temporal core" shared by various annotation schemata.

In accordance with (Novák, 2004) we recognize the starting time point anchor E_s and the ending time point anchor E_e of each event (process, activity, accomplishment, state etc.) E expressed in a discourse. E_s anchors the beginning of the event whereas E_e anchors the time the event is finished. If an event E takes place in one single time point, we take $E_s = E_e$. These anchors are interpreted as time points on the real time axis.

The set of all the anchor pairs and the set of time–of–speech points (one for each discourse utterance) together form the *temporal space* of a discourse. Consider the following example:

1. *A consortium of private investors operating as BPH Funding Co. said yesterday that it could eventually make a $300 million cash bid.*
2. *Today it announced that it no longer considers the possibility.*

The two time–of–speech points (*1.t, 2.t*) are present as well as the starting and ending points for the events expressed by the words *operating* (*op.s, op.e*), *said* (*say.s, say.e*), *make* (*mk.s, mk.e*), *announced* (*anc.s, anc.e*), and (no longer) *considers* (*cnsd.s, cnsd.e*).

The task of the temporal annotation of a discourse is to identify its temporal space and to determine relations between these points.

The following relative ordering relations may hold between two time points p and q: precedence ($p \prec q$), precedence–or–equality ($p \preceq q$), antecedence ($p \succ q$), antecedence–or–equality ($p \succeq q$) and equality ($p = q$). For the example mentioned above some of the relative ordering relations would be as follows:

- $1.t \prec 2.t$ (sentence order)
- $say.s = say.e \prec 1.t$ ($said$ expresses a single time point event)
- $1.t \prec mk.s = mk.e$ ($make$ takes place in the future if it takes place at all)
- $anc.s = anc.e \prec 2.t$
- $cnsd.s \prec anc.s \prec cnsd.e$
- $op.s \prec 2.t \prec op.e$ (the state of affairs is understood to be true even in the time–of–speech of the other sentence)

3. Functional Formalism

Some time points are more specifically determined by functions of other time points or are even specified absolutely. For example, in the sentence

Last year we spent our holiday in Austria and it was very similar to our vacation in Germany in February 1980.

the event of spending the holiday in Austria is determined as a function of the time–of–speech point (returning the extension of last year relative to time–of–speech) whereas the event of spending the holiday in Germany has been positioned absolutely to the interval of February 1980. Note that although we may not know the exact value of speech time of the utterance we still understand the sentence and should be able to annotate it.

To capture this kind of information we have developed an apparatus based on the operators and functions described below[1]. It represents the contents of expressions such as "last Friday", "beginning of the next month", "the middle of 80s" etc. It allows for the construction of an efficient algorithm for the computation of partial ordering of these expressions on the real time axis as described in Section 5.2..

Let us present the type system for these functions and operators first:

3.1. Types

- t_point is a concrete point on the real time axis.
- $t\_interval$ is a concrete closed interval on the real time axis, e.g. the time period between two time points including the points
- t_range is any amount of time (e.g. two seconds, four months etc.)
- t_etype represents a time entity type, currently it is one of the following constants: $millenium$, $century$, $year$, $month$, day, $hour$, $minute$, $second$
- t_pe_name is a named time entity representing a time point such as $midnight$ or $noon$
- t_ie_name is a named time entity representing a time interval such as day parts ($morning$), seasons ($spring$), weekdays ($tuesday$) and months ($january$)
- t_int,t_uint represent a signed and unsigned integer respectively
- t_bool is the boolean type

There are also set types - appending s to the type name yields the set of the objects of the given type, e.g. t_points denotes a set of points.

[1]Using these operators and functions we have been able to capture all the absolute time expressions within the annotated data. However, we do not claim that this list is sufficient to capture all time specifications. Its extension may be needed in the future.

3.2. Functional Apparatus

We can now list the functions[2] (the type of arguments and result follow after colon, t denotes time–of–speech point).

- $const(Y, [M, D, H, M, S]) : (t\_int, [t\_uint, ...]) \rightarrow t\_interval$
- $constM(Millenium) : (t\_int) \rightarrow t\_interval$
- $constC(Century) : (t\_int) \rightarrow t\_interval$

  The constructors make it possible to construct a time interval by specifying the respective parts (year (Y), month (M), etc.). Only the year is obligatory in the first version of the constructor. Note that all constructors return an interval, i.e. *const(1980,3,15,10,40,25)* does not denote the point 15.3.1980 10:40:25, but rather the entire interval of the 25th second. To retrieve the point it is possible to use the $start$ function: $start(const(1980, 3, 15, 10, 40, 25))$
- $entityRange(EntityType, Number) : (t\_etype, t\_uint) \rightarrow t\_range$

  returns the time range represented by $Number$ time entities of type $EntityType$, e.g.

  $$entityRange(year, 2)$$

  returns time range of two years.
- $shift(Point, Distance, InPast) : (t\_point, t\_range, t\_bool) \rightarrow t\_point$

  returns the time point that succeeds or precedes (depending on the value of $InPast$) $Point$ by $Distance$. For example

  $$shift(t, entityRange(hour, 3), false)$$

  returns the the time point exactly three hours after t.
- $span(Point, EntityType) : (t\_point, t\_etype) \rightarrow t\_interval$

  returns the concrete time interval of the time entity of type $EntityType$ that contains the time point $Point$, e.g.

  $$span(start(const(2006, 11, 5)), month)$$

  returns the interval corresponding to November 2006.
- $findEntityType(Point, EntityType, Index) : (t\_point, t\_etype, t\_int) \rightarrow t\_interval)$

  finds the $Index$-th occurence of $EntityType$ succeeding or preceding (if $Index$ is negative) $Point$. For example,

  $$find(t, day, 1)$$

  returns the day following the day containing t (i.e. corresponds to "tomorrow").

[2]We only present the substantial functions, trivial operations on the given types are also supported, e.g., $start$ and end functions that retrieve the starting and ending point of an interval, respectively.

- $findETByOrd(Point, EntityType, Order, SupET, Index, This)$: $(t\_point, t\_etype, t\_uint, t\_etype, t\_int, t\_bool) \rightarrow t\_interval)$

  finds the $Index$-th occurence of the $Order$-th $EntityType$ within $SupET$ (superior entity type) succeeding or preceding (if $Index$ is negative) $Point$. $This$ determines whether the time entity containing $Point$ is taken into account or not. For example,

  $$findETByOrd(t, day, 8, month, -1, true)$$

  returns the interval corresponding to the last 8th day in a month. If t lies in such a day, that day is returned (the occurrence is counted).

- $findIEByName(Point, Entity, Index, This)$: $(t\_point, t\_ie\_name, t\_int, t\_bool) \rightarrow t\_interval)$

- $findPEByName(Point, Entity, Index, This)$: $(t\_point, t\_pe\_name, t\_int, t\_bool) \rightarrow t\_point)$

  Both versions (for point and interval entities, respectively) find the $Index$-th occurence of $Entity$ succeeding or preceding (if $Index$ is negative) $Point$. $This$ determines whether the time entity containing $Point$ is taken into account. For example,

  $$findIEByName(t, january, 1, false)$$

  finds the "next January" from t regardless of whether t lies in January.

- $partByEntityType(Interval, Part, Order)$: $(t\_interval, t\_etype, t\_int) \rightarrow t\_interval$

- $partByIEntity(Interval, Part, Order)$: $(t\_interval, t\_ie\_name, t\_int) \rightarrow t\_interval$

- $partByPEntity(Interval, Part, Order)$: $(t\_interval, t\_pe\_name, t\_int) \rightarrow t\_point$

  All the three versions (for entity type and point and interval entities, respectively) retrieve the $Order$-th $Part$ within the specified $Interval$. ($Order$ is counted from the end if negative.) For example,

  $$partByPEntity(const(1970, 3), noon, 2)$$

  returns the interval corresponding to the noon of 2.3.1970. Vague subintervals are also supported, e.g.

  $$partByIEntity(const(1970), beginning)$$

  corresponds to the expression "beginning of 1970".

- $partByFraction(Interval, N_1, D_1, N_2, D_2)$: $(t\_interval, t\_uint, t\_uint, t\_uint, t\_uint) \rightarrow t\_interval$

  returns the $Interval$'s subinterval determined by the two fractions (numerators N_1, N_2 and denominators D_1, D_2). For example

  $$partByFraction(const(1980), 1, 4, 1, 2)$$

  returns the second quarter of 1980.

- $seriesETByInterval(EntityType, Interval)$: $(t\_etype, t\_interval) \rightarrow t\_intervals$

- $seriesIEByInterval(Entity, Interval)$: $(t\_ie\_name, t\_interval) \rightarrow t\_intervals$

- $seriesPEByInterval(Entity, Interval)$: $(t\_pe\_name, t\_interval) \rightarrow t\_points$

  All the three versions return all the occurrences of the specified $Entity$ (or $EntityType$) within the specified $Interval$. For example,

  $$seriesIEByInterval(monday, const(1980, 4))$$

  returns the set of all Mondays within April 1980.

- $seriesETByCount(Point, EntityType, C, Pst)$: $(t\_point, t\_etype, t\_uint, t\_bool) \rightarrow t\_intervals$

- $seriesIEByCount(Point, Entity, C, Pst)$: $(t\_point, t\_ie\_name, t\_uint, t\_bool) \rightarrow t\_intervals$

- $seriesPEByCount(Point, Entity, C, Pst)$: $(t\_point, t\_pe\_name, t\_uint, t\_bool) \rightarrow t\_points$

  All the three versions return C occurrences of the specified $Entity$ (or $EntityType$) from $Point$ (to the past if Pst is set). For example,

  $$seriesETByCount(t, day, 4)$$

  returns the set of 4 days before t.

These functions and operators may be composed to form the resulting *functional composition*. A time point can then be related to this functional composition yielding the complete *specification*. E.g. the starting point $sp.s$ of *spend* from our introductory example can be positioned to "last year" as follows

$$sp.s \in findEntityType(t, year, -1)$$

The same mechanism can be used to specify absolute distance between two time points etc. Note that the set of provided functions is not parsimonious (e.g. some $part$ functions can be replaced by their $find$ counterparts) but it corresponds more directly to the syntactic structure of time expressions which leads to less complicated compositions.

4. Annotated Corpus

Although the annotation scheme itself is language independent and can be used to annotate plain texts, it is particulary convenient to link the temporal annotation with the existing level of tectogrammatic (deep–syntax) annotation within the framework of the Prague Dependency Treebank (PDT) (Bohmová et al., 2002).

The tectogrammatic representation (TR) of a sentence captures its deep–syntax properties and relations as a tectogrammatical tree structure (TGTS). A TGTS is a dependency tree, the nodes of which represent the autosemantic (content, i.e., not auxiliary) words of the sentence. Each node is labelled with an inflectionally reduced word–form called the lemma and a functor that describes the deep syntactic relationship to its governor (parent node) such as actor (ACT), patient (PAT), temporal specification (TWHEN), locative etc. Additionally, the nodes are labelled with grammatemes that capture further morphological and semantic information corresponding to the autosemantic words such as tense, aspect, gender, number, mood, modality etc. An example of a simplified tectogrammatic tree is depicted in Figure 1. For a detailed description of the TGTS annotation scheme see (Bohmová et al., 2002). The temporal annotation can be therefore viewed as a natural extension to the PDT framework.

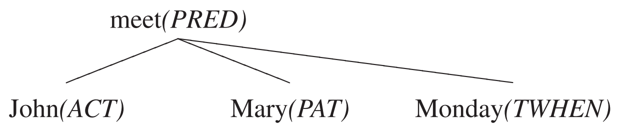

Figure 1: A simplified tectogrammatic tree representing the sentence "John met Mary on Monday." Only the functors are displayed.

We have annotated the Czech translation of a part[3] of the Wall Street Journal (WSJ) as present in the parallel Prague Czech–English Dependency Treebank (PCEDT) corpus (Čmejrek et al., 2004). Currently, the development testing data set (233 sentences) and the evaluation testing data set (231 sentences) are annotated for temporality and used as testing data for our experiments. Both contain approximately 100 time specifications (and much more event ordering relations).

5. Parsing and Inferencing

In order to infer relative ordering relations based on time expressions it is necessary to construct the functional composition and to compare the respective compositions. The former task is addressed by the parser, the latter by the inference engine.

5.1. Parser

The construction of the compositions is handled by a parser module that scans TGTSs for occurrences of subtrees that are accepted by a tree grammar. The functional composition is then built incrementally.

The tree grammar consist of rules whose left–hand side is a non–terminal representing a dependency subtree and the right–hand side represents the head of that subtree and its immediate descendants – children. A child (or the head) might be either a terminal node (corresponding to a node in the parsed tree) or a non–terminal defined by another rule. In this way the grammar makes it possible to process layers (a head and its children) of the time expression subtree one by one. There is an interpretation function associated with each rule (which corresponds to a tree layer). It combines the interpretations – functional compositions of the respective non–terminal children with the lexical and structural information contained in the processed layer to yield the interpretation for this layer.

[3]parts of sections 22, 23 and 24

An example is depicted in Figure 2. S, $FRCT$ and EXP are non–terminal symbols representing the respective subtrees (S is the start symbol). $part$, $preposition$, $numerator$ and $denominator$ represent nodes in the parsed TGTS that fulfill corresponding lexical conditions. $InterpretVaguePart$ and $InterpretFraction$ are interpretation functions creating the functional compositions that correspond to the information provided by the given layer. For example, $InterpretVaguePart$ applies the $partByIEntity$ function (see Section 3.2.) on the interpretation resulting from the subtree $FRCT$ supplying the argument $part$.

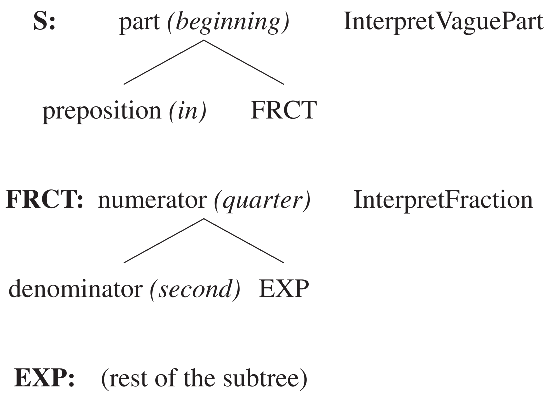

Figure 2: Examples of rules recognized by the tree grammar.

The majority of simple expressions such as "two days ago", "tomorrow", "in the beginning of the next week", "in 1987", "in the end of the last summer" etc. are parsed by our parser.

The evaluation of the performance of the time expression identification system is not straightforward. Our annotation schema does not contain a level of "shallow" time expression recognition which would allow us to count the number of recognized time expressions. For example, consider the sentence *"The meeting of the presidents will take place on Monday."* appearing in a newspaper article. The expression *on Monday* anchors four time points – the starting and the ending points of *meeting* and *take place* and only the corresponding four specifications are present in the annotation. Moreover, the determination of the correct functional composition ($findIEntityByName(t, Monday, 1)$ in this case) requires pragmatic inference – we have to know that we speak of the next Monday. The evaluation is therefore based on the complete specifications rather than the separate time expressions. We measure the performance of the time expression identification system by the following metrics. Precision P denotes the ratio of the correctly determined specifications and all the determined relations. Recall R denotes the ratio of the correctly determined specifications and all the existing (annotated) specifications. In order to somehow demonstrate the "shallow time expression recognition capability" of the system we also introduce the versions of precision and recall P_p and R_p respectively that: a) take misplaced functional compo-

P	R	F	P_p	R_p	F_p
65.33	44.95	**53.25**	82.66	56.88	**67.38**

Table 1: The performance of the time expression recognition system.

sitions (i.e. compositions which are correct but attached to an incorrect time point) as correct and b) take underspecified compositions as correct (e.g. $part(X, Monday, 1)$ instead of $findIEntityByName(t, Monday, 1)$). We also compute the corresponding F–measures[4] F and F_p. Table 1 lists the results. The errors are caused mainly by the errors in the TGTS annotation.

5.2. Inference Engine

The purpose of the inference engine is to compare time specifications in order to be able to determine relative ordering relation between the corresponding time points without having to know the precise value of time–of–speech. Two time specifications are compared against each other by another module that traces the composition "inside–out" (from the innermost function) and uses a "beam" structure to keep record of the distance (or absolute value) of the outcome of the last visited function to the source interval or point. If the source point is a variable only the estimates are provided while tracing the composition. If the compositions are comparable (e.g., the highest estimate carried by the beam of one composition is lower or equal to the lowest estimate carried by the beam of the other composition) then the ordering of the events anchored by the corresponding time expression may be inferred.

6. Related Work

There has been a substantial amount of previous work on the processing of time expressions. Some approaches such as (Mani et al., 2001) or (Schilder and Habel, 2001) work only with extensions of time expressions and are therefore unable to process discourses whose time–of–speech is unknown. More recent TimeML (Pustejovsky et al., 2005) annotation scheme introduces time functions for indexical expressions but to our best knowledge there is no implemented inference engine that would compare the functions directly.

Systems that extract and normalize time expressions from raw text (for example (Schilder and Habel, 2001), (Wilson et al., 2001)) usually achieve high F–measure scores. As explained in Section 5.1. these results are not directly comparable to the performance of our system as they do not fully address the issue of the attachment of the information provided by a time expression to respective events.

7. Conclusion

We have introduced a functional approach to capture the meaning of various time expressions and showed its usability for automatic inference of event ordering relations. The main advantage of this approach in comparison to systems that rely on computing the extensions (normalized values) of time expressions is its ability to process discourses (documents, speeches etc.) whose time–of–speech is unknown. The disadvantage of the presented approach lies in the lack of intermediate layers. As mentioned earlier there is no "shallow" representation of an isolated time expression. The existence of such layers would contribute to greater modularity of the system making it possible to separate e.g. semantic properties of a time expression and pragmatic inferences. It would also allow for a more direct comparison between alternative approaches.

[4]the harmonic mean of precision and recall

8. Acknowledgements

The development of the presented work has been supported by the following organizations and projects: the LC536 grant of the Ministry of Education of the Czech Republic, Information Society Project No. 1ET201120505 of the Grant Agency of the Academy of Sciences of the Czech Republic, Grant No. 0530118 of the National Science Foundation of the USA, and Grant No. 352/2006 of the Grant Agency of Charles University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the respective grant agencies.

9. References

Bohmová, Alena, Jan Hajič, Eva Hajičová, and Barbora Vidová-Hladká, 2002. The Prague Dependency Treebank: Three-level annotation scenario. *Treebanks: Building and Using Syntactically Annotated Corpora*.

Mani, Inderjeet, George Wilson, Beth Sundheim, and Lisa Ferro, 2001. Guidelines for annotating temporal information. In *HLT 2001, First International Conference on Human Language Technology Research*.

Novák, Václav, 2004. Towards logical representation of language structure. *The Prague Bulletin of Mathematical Linguistics*, (82):5–86.

Němec, Petr, 2006. Annotation of temporal relations within a discourse. In *Proceedings of Text, Speech and Dialogue*. Brno, Czech Republic.

Pustejovsky, James, Bob Ingria, Roser Sauri, Jose Castano, Jessica Littman, Rob Gaizauskas, Andrea Setzer, G Katz, and I Mani, 2005. The specification language TimeML. *I.Mani, J. Pustejovsky, and R. Gaizauskas, (eds.), The Language of Time: A Reader*.

Schilder, Frank and Christopher Habel, 2001. From temporal expressions to temporal information: Semantic tagging of news messages. In *ACL-2001 Workshop on Temporal and Spatial Information Processing*.

Čmejrek, Martin, Jan Cuřín, and Jiří Havelka, 2004. Prague czech-english dependency treebank: Any hopes for a common annotation scheme ? *Proceedings of HLT-NAACL Workshop: Frontiers in Corpus Annotation*:47–54.

Wilson, George, Inderjeet Mani, Beth Sundheim, and Lisa Ferro, 2001. A multilingual approach to annotating and extracting temporal information. In *ACL-2001 Workshop on Temporal and Spatial Information Processing*.