Charles University in Prague
Faculty of Mathematics and Physics

PhD Thesis

Petr Němec

Automatic Analysis of Temporal Relations within a Discourse and Its Application within a Machine Translation Framework

Department of Formal and Applied Linguistics
Supervisor: prof. PhDr. Eva Hajičová, DrSc.
Study Program: I–3 Mathematical Linguistics

I dedicate this work to my wife Jana who was a great support for me while I was working on the thesis. She also annotated great deal of the data this work is based on.

I would like to thank Eva Hajičová for her guidance in my study of computational linguistics, Keith Hall for his ideas regarding the generation component and evaluation metrics of the system of automatic analysis of temporal relation and for providing me with an implemented EM algorithm tool, Jarmila Panevová for many remarks on temporality issues, Jan Hajič and Libuše Brdičková for their great support in practical issues, David McClosky for the annotation of alternative past perfect tense, Petr Sgall for devoting his time to our regular meetings in the beginning stage of my research and many other colleagues for their advice and sharing their ideas and opinions with me.

I would also like to thank the Center of Language and Speech Processing at Johns Hopkins University in Baltimore for supporting my research and the Centrum voor Wiskunde en Informatica in Amsterdam for their hospitality.

The development of the presented work has been partially 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, grants IIS-9982329 and OISE-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.

I declare that I wrote the thesis by myself and listed all sources I used. I agree with lending the thesis.

In Prague on 18.9.2007 Petr Němec

Název práce: Automatická analýza časových vztahů v diskurzu a její využití v kontextu strojového překladu
Autor: Petr Němec
email: xp.nemec@gmail.com
Pracoviště: Ústav formální a aplikované lingvistiky
Školitel: prof. PhDr. Eva Hajičová, DrSc.
e-mail vedoucho: hajicova@ufal.mff.cuni.cz

Abstrakt: V této práci prezentujeme námi vyvinutý systém automatické analýzy časových relací v diskurzu v českém jazyce a diskutujeme možnosti jeho využití pro strojový překlad při generování vět v cílovém (anglickém) jazyce. Představujeme jednotlivé komponenty, které se k tomuto schématu váží: schéma anotace časových relací v diskurzu, vytvořený korpus anotovaný relací, samotný systém automatické analýzy, námi implementovanou generativní komponentu systému strojového překladu z češtiny do angličtiny a experimenty se zvyšováním kvality této komponenty přidáním informace o časových vztazích. V souvislosti s generativní komponentou navíc předkládáme implementovaný stromový přepisovací formalismus umožňující provádět požadované stromové transformace.

Klíčová slova: časový, temporalita, automatický, analýza, diskurz

Title: Automatic Analysis of Temporal Relations within a Discourse and Its Application within a Machine Translation Framework
Author: Petr Němec
email: xp.nemec@gmail.com
Department: Department of Formal and Applied Linguistics
Supervisor: prof. PhDr. Eva Hajičová, DrSc.
Supervisor's e-mail address: hajicova@ufal.mff.cuni.cz

Abstract: In this thesis we present a developed system of automatic analysis of temporal relations within a discourse for Czech and discuss possibilities to use the system to enhance a Czech–to–English machine translation system. We introduce the respective components related to this scenario: the scheme of annotation of temporal relations within a discourse, the created corpus annotated for temporality, the system of automatic analysis of temporal relations itself, the generative component of a Czech–to–English machine translation system producing English sentences and the experiments with improving the quality of this component by adding the information on temporal relations. Furthermore, in connection to the generative component, we present an implemented tree–rewriting formalism that allows for the required tree transformations.

Keywords: temporality, analysis, automatic, discourse

Contents

1 Introduction **7**
1.1 Prague Dependency Treebank . . . 9
1.1.1 Tectogrammatical Representation . . . 9
1.1.2 Analytical Layer . . . 12
1.1.3 Morphological Layer . . . 12
1.2 Prague Czech–English Dependency Treebank . . . 13

2 Annotation of Temporal Relations **14**
2.1 Introduction . . . 14
2.1.1 Event Properties . . . 14
2.1.2 Allen's Temporal Relations . . . 16
2.1.3 Annotation Efforts . . . 17
2.2 Annotation Scheme . . . 19
2.2.1 Basic Principles . . . 19
2.2.2 Functional Formalism . . . 21
2.2.3 Special Markers . . . 29
2.3 Annotated Corpus . . . 30
2.3.1 Extending PCEDT by Temporal Annotation . . . 31
2.3.2 Annotation Format . . . 32
2.3.3 Corpus Statistics . . . 32
2.4 Conclusion . . . 33

3 Automatic Analysis of Temporal Relations **36**
3.1 Introduction . . . 36
3.2 Linguistic Prerequisities . . . 37
3.2.1 Morphology . . . 37
3.2.2 Content Clauses . . . 37
3.3 Recursive Temporal Principle . . . 38
3.4 Evaluation Metrics . . . 39
3.5 Automatic Analysis . . . 40
3.5.1 Corrections to RTP . . . 41
3.5.2 Extensions to RTP . . . 45
3.5.3 Unresolved Issues . . . 48

3.6 Inference of Temporal Relations from Time Expressions 51
3.6.1 Parser 51
3.6.2 Inference Engine 56
3.7 Conclusion 58

4 Text Generation within a Machine Translation Framework 60
4.1 Introduction 60
4.2 Preprocessing Tectogrammatic Data 61
4.3 Generative Process 61
4.3.1 Insertion Model 62
4.3.2 Surface-order Model 63
4.3.3 Morphological Generation 65
4.4 Empirical Evaluation 65
4.4.1 Insertion Results 65
4.4.2 Reordering Results 68
4.4.3 Combined System Results 68
4.5 Conclusion 69

5 Tree Searching/Rewriting Formalism 71
5.1 Introduction 71
5.2 Query Formula 72
5.2.1 Structural predicate 73
5.2.2 Path predicate 74
5.2.3 Other predicates 74
5.3 Substitution Process 75
5.4 Test Examples 78
5.5 Implementation 79
5.5.1 Algorithm 79
5.5.2 Complexity Issues 80
5.5.3 Performance Results 80
5.6 Conclusion 81

6 Using Temporal Information for Text Generation 83
6.1 Tense Generation 83
6.1.1 Unresolvable Tense Issues 83
6.1.2 Tangible Issues 85
6.2 Empirical Evaluation 86
6.2.1 Data Preparation 87
6.2.2 Results 87
6.3 Conclusion 90

7 Conclusion 91

A Grammar for Parsing of Time Expressions 93

B Estimating Extensions of Functional Compositions **99**

C Automatic Subfunctor Assignment **105**

D Overview of the Implemented Software **107**

Chapter 1

Introduction

In this thesis we aim to describe our work in the field of natural language understanding and machine translation. We present a developed system of automatic analysis of temporal relations within a discourse and a text generation component of a machine translation system together with a related tree rewriting formalism. We also discuss possibilities to enhance the performance of the text generation system by the additional information gained by the temporal analysis.

The system of automatic analysis of temporal relations is designed for Czech and its purpose is to identify events expressed in a discourse (e.g. a newspaper article) and, most importantly, to determine relative ordering of these events as presented in the discourse. In addition to being a step towards automatic natural language understanding, the existence of such a system can contribute to various natural language processing tasks. For example, in text–summarization, the knowledge of the relative ordering of events is important for the correct generation of the summarized output. In question–answering, the user can ask when a particular event occurs or what events occurred after a given event. For the machine translation purposes, this information might help in the generation of text in the target language as this thesis tries to demonstrate. Naturally, a temporal annotation scheme has to be devised before one can experiment with automatic analysis of temporal relations. We present such a scheme as well.

The text generation component presented in this thesis is related to a transfer–based Czech–to–English machine translation framework where the transfer layer is a layer of deep-syntactic (tectogrammatical) representation (see Section 1.1.1). Tectogrammatical representation attempts to capture deep syntactic relationships based on the valency of sentence participants as a dependency tree. A key feature of tectogrammatical representation is that dependency relationships are represented only for autosemantic words (content words), meaning that synsemantic words (function words) are encoded as features of the grammatical relationships rather than the actual words. Moreover, tectogrammatical representation does not capture surface order of words, i.e., the order in which the words appear in the sentence. Abstracting away from specific syntactic and lexical items allows for the representation to be less language-specific making the representation attractive as a medium for machine translation. On the other hand, the generation process has to recover all the missing information to produce the translation.

Finally, having the information on the relative ordering of events in the Czech text at hand might help the generation of English translation. For example, in

Čekala tam už dvě hodiny, když přišel.

translated as

She had already been waiting there for two hours when he came.

it is necessary that the event of waiting terminates at the point the event of coming takes place if the past perfect tense is to be used properly. In other words, we have to know that this relation holds so as to be able to produce the translation. However, this relation is not encoded by grammar on the Czech side, it has to be acquired by the system of temporal analysis for Czech and passed over to the English generation component.

The thesis is structured as follows: In the rest of this chapter we describe the tectogrammatical representation scheme together with the Prague Dependency Treebank (Section 1.1) and the parallel Prague Czech–English Dependency Treebank that serves as a data base for the described machine translation experiments (Section 1.2) in greater detail. The first two chapters deal with temporality issues: Chapter 2 introduces the devised temporal annotation scheme and Chapter 3 presents the developed system of automatic analysis of temporal relations and discusses the achieved results. The text generation component is subject of Chapter 4 and the tree rewriting formalism developed to ease tree modification in the generation process is described in Chapter 5. The experiments aimed at the improvement of the generation component by the temporal information are described in Chapter 6. Chapter 7 concludes the thesis. An introduction to the relevant areas of research as well as a comparison to related work is covered in the respective chapters. The appendices provide supplementary material to the main text: Appendices A and B are related to Chapter 3, Appendices C and relates to Chapter 4. Appendix D provides an overview and basic description of the implemented software modules (more detailed information is available on the enclosed DVD).

The presented work (including data annotation and manipulation) is that of the author of this thesis except where explicitly stated otherwise. The topics covered in this thesis are also described in the following papers published (sometimes in cooperation with other authors) by the author of this thesis: the annotation scheme in [55], the functional approach towards capturing the meaning of time expression in [54], the results achieved in the automatic analysis of temporal relations in [57], the generation of text from tectogrammatical tree structures within a machine translation framework in [29], the description of the tree rewriting formalism used in the generation in [56] and [41].

1.1 Prague Dependency Treebank

In this section we aim to briefly describe the annotation scenario of the Prague Dependency Treebank (PDT) [5] we use as a framework for our experiments.

Full annotation of a sentence within the PDT consists of three layers:

- morphological layer
- analytical layer (layer of shallow syntax)
- tectogrammatical layer (layer of deep syntax)

The technical realization differs between PDT versions 1.0 [25] and 2.0 [58] but we will not discuss the differences here. However, note that the experiments described in this thesis are implemented for PDT 1.0 style data as this format is used in the Parallel Czech–English Dependency Treebank (see Section 1.2).

Note that in the subsequent chapters, when referring to the "tectogrammatical annotation" or "annotation on tectogrammatical level" we mean the tectogrammatical representation including the annotation of all the lower layers unless stated otherwise.

Let us briefly describe the respective layers beginning with the most abstract tectogrammatic layer.

1.1.1 Tectogrammatical Representation

The tectogrammatical representation (*TR*) [4, 23, 74] comes out of the Praguian linguistic theory known as the Functional Generative Description of language founded by Sgall [73, 72]. TR attempts to capture deep syntactic properties of a sentence by the so–called tectogrammatical tree structure (*TGTS*). TGTS is a dependency tree whose vertices correspond to autosemantic (content) words of the sentence and whose edges represent either valency participants of verbs (nouns, adjectives) or free modifications. The type of a dependency relation is captured by the *functor* which takes its value from the predefined set of values. A key feature of TR is that dependency relationships are represented only for autosemantic words (content words), meaning that synsemantic words (auxiliary function words) are encoded as *grammatemes* (morphological and grammatical features) rather than the actual words. Abstracting away from specific syntactic and lexical items allows for the representation to be less language-specific making the representation attractive as a medium for machine translation and summarization as shown in Chapter 4.

For example, consider the TGTS from Figure 1.1 representing the sentence

Honza veze Marii do školy.
(John is driving Mary to school.)

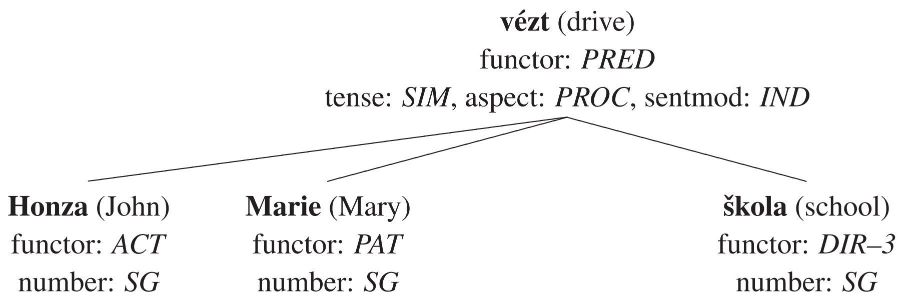


Figure 1.1: Example of a simple tectogrammatical tree. *SIM* stands for present tense, *PROC* denotes processual aspect (see below), *IND* denotes indicative, *ACT* and *PAT* stand for actor and patient, respectively.

The root[1] of the tree represents the main predicate of the sentence, the verb *"vézt" (to drive)*. Its concrete morphological form in the sentence is not preserved but its meaning is captured by the *tense*, *aspect* and *sentmod* grammatemes. The first two participants of the predicate, *Mary* and *John*, are assigned *ACT* (actor) and *PAT* (patient) functors, respectively. The meaning of the non–valency complement *"do školy" (to school)* is captured by the *DIR–3* functor which represents a target direction (answer to the question "where to"). Again, their morphological forms as well as the synsemantic preposition *"do" (to)* are missing.

Note that the (horizontal) order of the nodes in the tree does not correspond to the surface order (i.e., the order in which the words appear in the sentence). Instead, this *deep* order expresses the so–called scale of communicative dynamism closely related to topic–focus articulation [28]. Essentially, the higher (more to the right in a TGTS) a node is, the newer[2] information (from the speaker's point of view) it represents. The items that lie low on the scale form the *topic* of the sentence (what the sentence is about) and those that lie high form its *focus* (what is said about the topic). The division line is assumed to lie either immediately before or after the main predicate (see Hajičová et al. [24] for a complex discussion). The lowest item on the scale is called the *topic proper*, the highest is the *focus proper*. In Czech, the deep order corresponds in prototypical case to the surface word order, i.e., unless some part of the sentence is stressed. As we don't use the deep order in our experiments, we will not describe it in detail, see the cited works for the detailed description.

For the experiments we present in this thesis, the grammatemes of *tense* and *aspect* are crucial so we will describe them in more detail. The *tense* grammateme

[1]Actual TGTSs contain an artificial root bearing information about the represented sentence, its modality, its source, etc. For simplicity, we do not show this artificial root. Instead, we associate the relevant information it carries with the main predicate.

[2]To say that an information is new does necessarily mean that it hasn't yet occurred in the discourse - it means that it is presented as new (emphasized) by the speaker.

may contain one of the values *ANT*, *POST* and *SIM* corresponding to the three morphological tenses (past, future and present). The *aspect* grammateme may contain values *PROC* (*processual*) corresponding to imperfective aspect, *CPL* (*complex*) corresponding to perfective aspect and *RES*, the so–called resultative, which roughly corresponds to explicit perfect constructions in Czech, i.e., events consisting of an event core and its consequent state such as

Mám večeři připravenou.
(I have the dinner ready.)

Another important feature of the TR is the so–called subfunctor that further specifies the meaning of functor. For example, in the sentence

Honza přišel poté, co Marie odešla.
(John came after Mary had left.)

whose TGTS is shown in Figure 1.2, the $TWHEN$ functor specifies that the adverbial phrase is a temporal one, answering the question *when*. However, the meaning of *"after"* is not captured by the functor itself but it is encoded in the subfunctor. Note that the subfunctor's value *after* does not denote the preposition - it is only a string value expressing the meaning of posteriority.

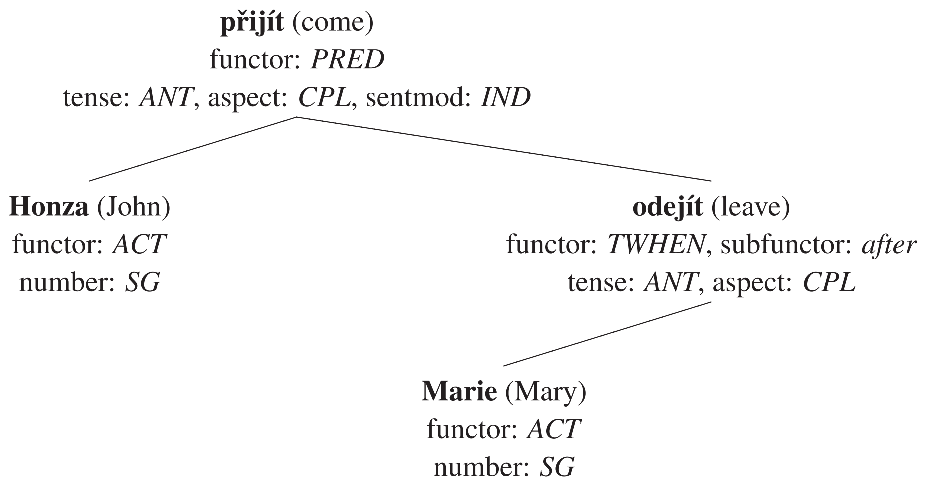


Figure 1.2: TGTS for the sentence *"Honza přišel poté, co Marie odešla."*

Annotation of coreferential relations is also part of the TR [36], although the annotation of nominal anaphora, i.e., the coreference between autosemantic noun phrases, has not yet been finished.

1.1.2 Analytical Layer

The analytical representation [22] of a sentence does not belong to the theoretically based description of the sentence; it is just an auxiliary technical device specifying the so-called shallow syntactic properties. It provides the information that is underspecified in the tectogrammatical representation, i.e., synsemantic words, the surface word order, actual word form and together with the morphological representation (see Section 1.1.3) morphological properties of words (e.g. case). Shallow–syntactic relations between the respective nodes are captured by *analytical functors*. Figure 1.3 shows the analytical tree for the introductory sentence in the previous section repeated here for convenience

Honza veze Marii do školy.
(John is driving Mary to school.)

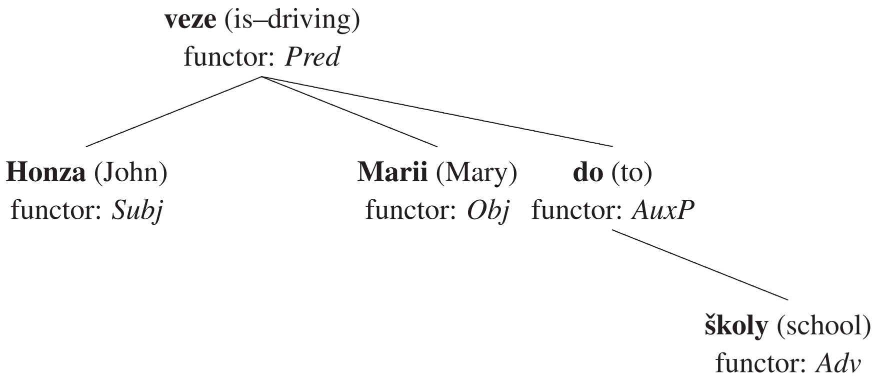


Figure 1.3: Example of an analytical tree. *Subj*, *Obj*, *AuxP* and *Adv* correspond to subject, object, preposition and adverbial, respectively.

1.1.3 Morphological Layer

The annotation on the morphological layer [21] assigns the morphological tag and the lemma to each word of the given sentence. The tag contains information about the morphological properties of the word in a given sentence such as part–of–speech, number, gender, case, person, etc. Each of the 15 positions in the tag corresponds to the value of one particular category. The sample sentence would be annotated as shown in Table 1.1.

Word	**Tag**	**Description**
Honza	NNMS1 – – – – – – – – – –	singular masculine noun in nominative
veze	VB–S – – –3P–AA– – –	active verb in third person singular
Marii	NNFS4 – – – – – – – – – –	singular feminine noun in accusative
do	RR – – – – – – – – – – – – –	preposition
školy	NNFS2 – – – – – – – – – –	singular feminine noun in genitive

Table 1.1: Morphological annotation of the sentence *"Honza veze Marii do školy."*. Lemma contains additional information not shown here.

1.2 Prague Czech–English Dependency Treebank

Prague Czech–English Dependency Treebank 1.0 (PCEDT) [11, 12, 15] is a corpus of Czech-English parallel resources suitable for experiments in machine translation with a special emphasis on dependency-based translation.

The core part of PCEDT is a Czech translation of 21600 English sentences from the Wall Street Journal part of Penn Treebank 3 corpus[3] [47, 46]. Sentences of the Czech translation were automatically morphologically annotated and parsed into the analytical and tectogrammatical level corresponding to the PDT structure (see Section 1.1). The original English sentences were transformed from the Penn Treebank phrase-structure trees into dependency representations. A development and evaluation set of 233 and 231 sentence pairs, respectively, was selected and manually annotated on tectogrammatical level in both Czech and English. For the purposes of quantitative evaluation (e.g. computation of BLEU score as in our experiments) this set has been retranslated from Czech to English by 4 different translation companies.

The included Czech-English translation dictionary consists of approximately 45,000 entry-translation pairs in its lemmatized version and approximately half a million pairs of word forms where for each entry-translation pair all the corresponding word form pairs have been generated.

PCEDT also contains other data sources but we do not introduce them here as they are not used in our experiments. For more information see the cited papers.

The style of manual tectogrammatical annotation within PCEDT corresponds to the obsolete PDT 1.0 annotation style. Moreover, as described in Section 2.3.1, the manual tectogrammatical annotation contains many errors. Nevertheless, during the time we were working on the projects described in this thesis there were no other resources available that would be suitable for machine translation via tectogrammatical layer. Currently, an annotations schema for English is being designed and systematic tectogrammatical annotation of English text is in progress (see [10]).

[3]released by LDC in 1999 as LDC99T42

Chapter 2

Annotation of Temporal Relations

In this chapter we describe the temporal annotation scheme and the annotated corpus we have created and used for our experiments with automatic analysis of temporal relations as described in Chapter 3. The chapter is structured as follows: Section 2.1 introduces several aspects of temporal annotation of events and their classification. It also presents existing schemata for temporal annotation. Section 2.2 describes our annotation schema and Section 2.3 provides information on the annotated corpus. Section 2.4 concludes the chapter and provides comparison of our approach to the related work.

2.1 Introduction

In this section we aim to briefly introduce properties that characterize events and therefore represent information annotation schemata should capture. We also introduce Allen's classification of temporal relations [1] as many of the cited work rely on it. We then present several existing annotation schemata.

In the following text, we refer by *time–of–speech* of an utterance to the time in which the utterance took place. If the discourse is a written document time–of–speech denotes the creation time of each respective utterance.

2.1.1 Event Properties

Tense

Tense generally expresses precedence, posteriority or concurrence of an event to another event or time–of–speech. Languages differ in the complexity of their tense system – some lack tense markers at all (e.g. Mandarin Chinese), some contain only basic set of tenses – past, present and future (e.g. Czech) and some feature a complex tense system (e.g. English). However, a complex tense system featuring various perfect tenses usually also serves as a substitute for lack of aspect information in lexemes themselves and is closely related to grammatical aspect (see below).

In TR, tense is captured by the *tense* grammateme (see Section 1.1.1).

Grammatical Aspect

Grammatical aspect (perfective or imperfective) expresses how an event is viewed by the speaker. Perfective aspect is used if the event is presented as a single unit (completed or not) without emphasis on its internal structure. Imperfective aspect, on the other hand, stresses the internal structure and/or duration of an event. In Slavonic languages (e.g. Czech) the aspect is largely expressed lexically, i.e., aspect lies on the boundary of being a property of a given lexical unit (see Section 3.2) and being a grammatical function. Note that this aspect of "lexical nature" is something quite different from the "lexical aspect" introduced below.

In TR, aspect is captured by the *aspect* grammateme (see Section 1.1.1).

Lexical Aspect

The classification of events (eventualities) into classes based on their lexical aspect was first introduced by Vendler in [79] and further extended and refined by other authors, e.g. Moens and Steedman [50], Smith [75], Dorr and Olsen [16], Dowty [18]. The basic division is as follows:

states (statives) States are durative eventualities that do not contain an internal structure – an entity simply is or is not in the given state (with certain level of approximation). A state has the so–called *subinterval property*: if a state holds for a period of time (p), it must also hold for any subinterval within p. For example,

John loves Mary.

is a state.

activities (processes) Activities are durative events that lack a culmination point. They usually have an internal structure, i.e., they consist of primitive events. An example of an activity is

John was walking in a park.

accomplishments Accomplishments are durative events that culminate such as

John repaired the radio in two hours.

achievements Achievements are "instantaneous accomplishments", i.e., only the culmination point is present. For example,

John reached his goal.

is an achievement.

Novák [53] argues against this distinction and proposes to represent all eventualities in a uniform manner – by their starting and ending point. While we do not fully agree with Novák that the distinction is irrelevant for the representation of discourse content, we believe that this uniform representation is a good starting point for temporal annotation and our annotations scheme reflects that. Problems arise only with accomplishments as they in fact express two events - the preparatory process and the culmination (see Section 2.2.1). Moreover, we consider the distinction between (content of) accomplishments and achievements rather unclear. For example, in

John died.

it is questionable whether there is an associated process of dying associated with the culmination. The context might provide a clue, e.g.

John died in two days. (accomplishment)
John died suddenly on Thursday at 15:30. (achievement)

but in general we find the distinction hard to draw.

Event Structure

Another property (tightly connected to lexical semantics) of an event is its structure, i.e., its decomposition into primitive events. For example, meaning of *"to wash"* can be (informally) captured as "to cause to become clean". There are several accounts on event decomposition, e.g. that of Dowty [17], Jackendoff [31] and Pustejovsky [63]. Unlike the previous properties of events, event structure of a particular event expressed in a discourse does not depend on the context and is therefore not to be annotated for each separate event. Instead, this information belongs to an event ontology.

2.1.2 Allen's Temporal Relations

Allen [1] proposes to classify temporal relations between two intervals (t_1,t_2) that are extensions of respective events as follows (quotation from [1]):

$DURING(t_1, t_2)$
time interval t_1 is fully contained within t_2

$STARTS(t_1, t_2)$
time interval t_1 shares the same beginning as t_2, but ends before t_2 ends

$FINISHES(t_1, t_2)$
time interval t_1 shares the same end as t_2, but begins after t_2 begins

$BEFORE(t_1, t_2)$
time interval t_1 is before interval t_2, and they do not overlap in any way

$OVERLAP(t_1, t_2)$
time interval t_1 starts before t_2, and they overlap

$MEETS(t_1, t_2)$
time interval t_1 is before interval t_2, but there is no interval between them, i.e., t_1 ends where t_2 starts

$EQUAL(t_1, t_2)$
t_1 and t_2 are the same interval

Taking inverses of these relations into consideration we obtain 13 possible relations. Allen also introduces derived relations such as

$$IN(t_1, t_2) \Leftrightarrow DURING(t_1, t_2) \vee STARTS(t_1, t_2) \vee FINISHES(t_1, t_2)$$

2.1.3 Annotation Efforts

Temporal annotations, started virtually a few years ago, are one of the most recent parts of the research on temporality. In many of the works we cite below, the annotation scheme is rather implicit and the attention is paid mostly to an automatic analysis system. We therefore also introduce the analysis part of the cited work in Section 3.1.

The earliest approaches dealt with shallow annotation (and associated extraction) of time expressions in plain texts. The DARPA Message Understanding Conference tagging included a subtask aimed at the recognition of absolute time expressions (i.e. time expressions whose time extension is independent on the time–of–speech such as *"1.5.1998"*) [51] and later also relative [52] time expressions. The reference set was manually annotated by human annotators. However, the annotation scheme contained no interpretation (determination of the denoted time extension), the task was simply to identify an expression as a *date*, *time*, etc.

TIMEX2 [82] represents a step towards interpretation of time expressions as each time expression has calendrical value denoting the time extension of the time expression associated with it. To further explicate the meaning of a time expression (rather than only its extension) and also to allow for separation of the recognition and the interpretation of a time expression, TIMEX2 has been further enriched by a functional approach [65] to TIMEX3 specification. The functions take the time–of–speech point as their input argument and yield the extension of the given time expression. For example, the meaning of *"last week"* can be captured as

$$predecessor(week(time - of - speech))$$

where the $week$ function returns the extension of the week time–of–speech lies in and the $predecessor$ function returns the extension of the previous week.

Let us turn our attention to complex approaches that address the annotation of events and, most importantly, temporal relations between the respective events.

Perhaps one of the most restrictive approaches is that of Katz and Arosio [33]. They only annotate verbs and draw temporal relations strictly within single sentences. Their work is primarily motivated by the wish to study how temporal relations are conveyed by syntactic and lexical information within complex sentences. This is similar to the Recursive Temporal Principle (see Section 3.3) that answers some of these questions for Czech and that we use in our system of automatic analysis of temporal relations.

Another fairly restrictive approach is that of Filatova and Hovy [19] that aims to time–stamp events, i.e., to assign a calendrical value to events expressed in a discourse. In their approach, events are represented by clauses containing a (verbal) predicate and a subject.

Schilder and Habel [71] consider verbal events and events expressed by a noun phrase (nominalization). They only consider temporal relations between an event and its anchoring time expression which is determined largely by prepositions and use Allen's classification of temporal relations. They also provide another way of representing meaning of time expressions by means of Prolog [6] predicate underspecification.

Li et al. [39] deal both with event anchoring (time–stamping) and temporal relations between events for Chinese. Again, the classification of the relation types is based on Allen's classification.

To our best knowledge, the most complex approach is the TimeML annotation scheme developed by Pustejovsky et al. [65]. TimeML captures all kinds of events expressed in a discourse (even by nominals, adjectives and infinitive verbs). Unlike the approaches discussed so far, it classifies events into several groups, e.g. *occurrence, state, reporting, perception* etc. Features such as tense, aspect, polarity, modality, etc. are part of the annotation, too. The types of temporal relations between events are largely motivated by Allen's approach. TIMEX3 specification for capturing meaning of time expressions (described above) is also part of the TimeML specification.

2.2 Annotation Scheme

2.2.1 Basic Principles

In accordance with Novák [53] we recognize the starting time point anchor E_s and the ending time point anchor E_e of each event (state, activity, accomplishment, achievement) 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 ($p \preceq q$), posteriority ($p \succ q$), posteriority–or–equality ($p \succeq q$) and equality ($p = q$). For the example mentioned above the basic possible set of 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)
- $anc.s = anc.e \prec 2.t$
- $cnsd.s \prec anc.s \prec cnsd.e$

These relations correspond to the information provided by grammar. Nevertheless, more relations can be inferred with various levels of confidence:

- $op.s \prec say.s$ (the consortium was probably operating as BPH Funding before it made a statement yesterday)
- $op.s \prec 1.t \prec op.e$ (the consortium is understood to operate as BPH Funding in the time the sentence was written)

- $say.s \prec anc.s$ (follows from the temporal expressions *yesterday*, *today*)
- $op.s \prec 2.t \prec op.e$ (the state of affairs is understood to be true even in the time–of–speech of the other sentence)
- $cnsd.s \prec 1.t \prec cnsd.e$ (the same)

In our approach, the temporal space of a discourse consists in the points of events that are explicitly expressed (by a verb, a deverbative noun or adjective). We do not consider any other related events with the following exception. If an event is an accomplishment (in Vendler's sense) modified by a temporal adverbial phrase with *za* (*in*) as in

Petr postavil dům za dva roky.
(Petr built a house in two years.)

we introduce the process (building of the house) associated with the culmination event (*built*) so as to be able to anchor the adverbial phrase (see Section 2.2.2). However, we do not introduce processes associated with accomplishments in general because we consider the boundary between accomplishments and achievements rather fuzzy (see Section 2.1) – we would not be able to make the decision on whether there is a preparatory process related to a given event systematically. Moreover, these "hidden" events would significantly and, in our view, artificially alter the distribution of the annotated events and therefore even the results of a system of automatic analysis of temporal relations.

The relative ordering relations between the respective events are to be annotated from the viewpoint of the speakers of the respective utterances.

The annotator should annotate all relations which correspond to a natural interpretation of the expressed events, not only relations which are obvious or logically necessary. We have chosen this approach because we believe that all these relations represent a part of the human understanding of the given discourse and a system of automatic analysis should be able to determine even the relations which are only plausible but not directly implied by the discourse if it is to be truly a part of a content understanding system.

The final annotation is obtained as the transitive closure of the annotated relations. We do not deal with "unclosed" annotations and analyses because we consider them inconsistent: An annotator can (and should) never annotate all the relations that follow from the relations he/she has entered (the core) and the choice of this core largely depends on the annotator as it would be very hard to design some canonical form of the core and force the annotator to uphold it. All the corpus statistics and errors of a system of automatic analysis would then depend on the (arbitrary) choice of a core by a particular annotator – two annotations could differ even if their content would be the same. On the other hand, as far as the evaluation of a system of automatic analysis is concerned, the evaluation based on the closed set of relations leads to propagation of errors, thus altering their impact.

The final principle deals with the number of potential relations within a discourse. It would be infeasible for the annotator to consider all of them as their number rises quadratically with the size of the temporal space (e.g. in an article containing 30 simple sentences, each with only a single verb, the temporal space consists of 90 points $(30*2+30)$ and there are 4005 $(90*89/2)$ potential relations in total). To make the annotation process reasonably fast we have annotated only relations between points in one sentence and between two adjacent sentences. Because the final annotation is formed by transitive closure of the respective relations, this principle allows us to retain some degree of global discourse coverage.

2.2.2 Functional Formalism

Some of the time points within a discourse 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, we should be able to annotate it and draw inferences from it.

To capture this kind of information we have developed an apparatus based on the operators and functions described below[1]. It represents content of the expressions such as *"last Friday"*, *"beginning of the next month"*, *"the middle of 80s"* etc. It allows for the construction of efficient algorithm for the computation of partial ordering of these expressions on the real time axis as described in Section 3.6.2.

Let us present type system for these functions and operators first (see Appendix A for the complete list of supported constants for the relevant types):

Types

- t_point is a concrete point on the real time axis
- $t\_interval$ is a concrete closed interval on the real time axis, i.e., the time period between two time points including the points
- t_etype represents a time entity type. There are following basic types:

[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.

  - $millenium$
  - $century$
  - $decade$
  - $year$
  - $month$
  - day
  - $hour$
  - $minute$
  - $second$

  These types can be modified by $quarter$ and $half$ prefixes to yield derived types such as $quarteryear$[2]. A type may be further modified by a modifier such as $fiscal$ or $academic$ to yield e.g. $halfyear\_fiscal$ or $year\_academic$

- t_range is any amount of time (e.g. two seconds, four months, etc.)
- t_pe_name is a named time entity representing a time point such as $mid-night$ 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 ($jan-uary$)
- t_vague_part is either $beginning$, end, or $middle$
- t_int,t_uint represent a signed and unsigned integer, respectively
- t_bool is the boolean type realized as 1 for true and 0 or -1 for false[3]

  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.

Functions and Operators

We may now list the operators and functions (the type of arguments and result follow after colon, t denotes time–of–speech point):

$\cup, \cap(Interval_1, Interval_2)$:
$(t\_interval, t\_interval) \rightarrow t\_interval$

are the usual union and intersection operations on the interval type.

[2]The reason we treat these specific parts of basic types as types themselves is that they correspond to one–word expressions (*halfyear*) which makes the correspondence between a time expression and the resulting functional representation more direct.

[3]We use -1 for false in the $InFuture$ parameter of the functions described below. -1 then means "in the past" and 1 means "in the future".

$start, end(Interval)$:
$(t\_interval) \rightarrow t\_point$

retrieve the starting and the ending point of the specified interval, respectively.

$intervalByPoints(Point_1, Point_2)$:
$(t\_point, t\_point) \rightarrow t\_interval$

constructs the interval from the specified points

$intervalSize(interval)$:
$(t\_interval) \rightarrow t\_range$

returns the entity range represented by the specified $interval$

$+, -(Range_1, Range_2)$:
$(t\_range, t\_range) \rightarrow t\_range$

are the usual addition and subtraction operators on the time range type.

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

$constM(Millenium)$:
$(t\_int) \rightarrow t\_interval$

$constC(Century)$:
$(t\_int) \rightarrow t\_interval$

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. It is possible to use the $start$ function to retrieve the point:

$$start(const(1980, 3, 15, 10, 40, 25))$$

$entityRange(EntityType, Number)$:
$(t\_etype, t\_uint) \rightarrow t\_range$

returns the time range represented by $Number$ time entities of type $Entity-Type$, e.g.

$$entityRange(year, 2)$$

returns time range of two years.

$shift(Point, Distance, InFuture)$:
$(t\_point, t\_range, t\_bool) \rightarrow t\_point$

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

$$shift(t, entityRange(hour, 3), 1)$$

returns 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 occurrence 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., it corresponds to "tomorrow").

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

finds the $Index$-th occurrence 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,

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

returns the interval corresponding to the last 8th day in a month (i.e., the month containing t or the month before). If t itself 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 occurrence of $Entity$ succeeding or preceding (if $Index$ is negative) $Point$. $This$ determines whether the time entity containing $Point$ is taken into account. For example,

$$findIEByName(t, january, 1, 0)$$

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 and is therefore equivalent to

$$partByPEntity(const(1970, 3, 2), noon, 1)$$

$partVague(Interval, VaguePart)$:
$(t\_interval, t\_vague\_part) \rightarrow t\_interval$

returns the given vague subinterval of the specified $Interval$ – $beginning$, end or $middle$[4]. For example,

$$partVague(const(1970), beginning)$$

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.

[4]Note that there is actually no difference between $partVague$ and $partByIEntity$ functions as the latter also accepts vague expressions such as $evening$ or $summer$. However, vague periods are not entities.

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

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

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

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.

$seriesETByCount(Point, EntityType, Count)$:
$(t\_point, t\_etype, t\_uint) \rightarrow t\_intervals$

$seriesIEByCount(Point, Entity, Count)$:
$(t\_point, t\_ie\_name, t\_uint) \rightarrow t\_intervals$

$seriesPEByCount(Point, Entity, Count)$:
$(t\_point, t\_pe\_name, t\_uint) \rightarrow t\_points$

All the three versions return $Count$ occurrences of the specified $Entity$ (or $EntityType$) from $Point$. For example,

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

returns the set of 4 days after t.

$[exp_1, exp_2, ...]$

denotes an alternative. For example,

$$[findIEntityByName(t, September, 1),$$

$$findIEntityByName(t, October, -1)]$$

is either the "next September" or the "last October".

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 *time specification.* For example, 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 the absolute distance between two time points (event duration) or even more complicated relations between time points.

If a time expression is interpreted as underspecified in a given context, as in

The president will definitely leave on Monday but the exact date is still unknown.

the proper version of $part$ function with unspecified input interval is used:

$$depart.s(= depart.e) \in partByIEntity(X, Monday, 1)$$

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.

Let us present a few more examples of time specifications:

1. *He will return in five hours from now.*

$$return.s(= return.e) = shift(t, entityRange(hour, 5))$$

2. *The committee discussed the report on Thursday.*

$$\begin{aligned} &discussed.s, discussed.e \in \\ &\qquad partByIEntity(X, Thursday, 1) \\ &\qquad (findIEntityByName(t, Thursday, -1)) \end{aligned}$$

3. *The shipment will arrive no sooner than January of next year.*

$$\begin{aligned} &arrive.s(= arrive.e) \geq \\ &\qquad partIByName(findEntityType(t, year, 1), January) \end{aligned}$$

4. *The board will be discussing the new business strategy on 4.5.2006 in the morning.*

$$\begin{aligned} &discussing.s, discussing.e \in \\ &\qquad partByIEntity(const(2006, 5, 4), morning) \end{aligned}$$

5. *The plane America West ordered will be delivered next Monday or Tuesday.*

$$\begin{aligned} deliver.s(= deliver.e) \in \; &[findIEntityByName(t, Monday, 1), \\ &findIEntityByName(t, Tuesday, 1)] \end{aligned}$$

6. *The company will be sponsoring the project for at least two years.*

$$sponsoring.s \leq shift(sponsoring.e, entityRange(year, 2))$$

7. *The meeting of the presidents will take place on Monday.*

$$\begin{gathered} meeting.s, meeting.e, take.s, take.e \in \\ partByIEntity(X, Monday, 1) \\ (findIEntityByName(t, Monday, 1)) \end{gathered}$$

Note that in Examples 2, 4 and 7 the starting and ending points of the respective events are placed into the time extension denoted by the given time expressions although this is incorrect from the strict logical viewpoint – e.g. from the fact the committee discussed the report on Thursday it does not follow that the report was discussed only on Thursday as the anchoring of the time points suggests. However, when reading such a sentence in a newspaper (in a neutral context) we conclude that the discussion took place only on Thursday, because if not, the author of the article would have added the information. In other words, not providing the complete information on the extension of such an event (while not informing the reader that it is incomplete) would be viewed as a misinformation. The annotation of the presented examples is therefore in accordance with our principle of "natural interpretation" annotation (see Section 2.2.1). Nevertheless, in certain situations we really only wish to state that there is a common point between an event and a time expression as in

The committee discussed the report also on Thursday.

which can be annotated as follows:

$$\begin{aligned} discussed.s &\preceq end(partByIEntity(X, Thursday, 1)) \\ discussed.e &\succeq start(partByIEntity(X, Thursday, 1)) \end{aligned}$$

Example 7 (and similarly Example 2) also demonstrates the difference between an underspecified reading and a reading that involves pragmatic inference. In the former case, the time specification only asserts that the meeting would take place on some Monday. However, in a common context of a newspaper article, the preferred reading of the sentence is that *"on Monday"* refers to the next Monday relative to the time–of–speech (document time) of the article. If so, the more specific composition

$$findIEntityByName(t, Monday, 1)$$

is the appropriate one. The important point to note here is that the determination of the correct (most specific) time specification requires not only syntactic and semantic analysis but also pragmatic inference. There are no explicit “intermediate layers” in our annotation scheme, only the final content of the time expression is captured.

A consequence of this strategy is that our annotation schema also does not contain a layer of “shallow” time expression recognition. The expression *on Monday* in Example 7 anchors four time points – the starting and the ending points of *meeting* and *take place*[5] and only the corresponding four time specifications are present in the annotation – the information about its source, the surface time expression, is lost. (In special cases, a time specification may not even correspond to a surface time expression, see Section 2.2.3 where an example of plan repeat period specification is presented.)

The reason our approach is not explicitly a stratificational one is that it aims to be the core of a content representation of temporal expressions, not a general time specification language. However, in combination with tectogrammatical annotation (see Section 1.1.1) the annotated temporal corpus contains intermediate syntactic layers as described in Section 2.3. Moreover, the layer of semantic properties (i.e. no pragmatical inference) of time expressions is, to certain extent, also implicitly present. In our approach, semantics of a time expression corresponds roughly to the underspecified time specification which can often be inferred from the final specification, e.g.

$$x \in findIEntityByName(t, Monday, 1)$$

entails

$$x \in partIByName(X, Monday, 1)$$

2.2.3 Special Markers

Some discourse expressions do not express a single event taking place in a single interval of time.

The event may be iterative as in

Last month, I used to wake up every morning and run 10 miles.

and it is necessary to capture this information. However, it is not enough to mark each of the two events separately as the connection between them would be lost. Instead, we introduce the notion of a *plan* to capture the fact that the events are iterative and one follows after another in a “single run”: the ordering of the two events is annotated as usual but the events are declared to be part of one particular box – the plan. The plan itself has a start time point and end time point (plan

[5]This verb might be considered temporal auxiliary as it only anchors time specification of another event (*meeting*). However, in accordance with our annotation principles, we consider it an event and annotate it as well.

boundaries) which denote the interval of plan's validity (in the example, these are directly specified as the start and end of the *"last month"*). Repeat period of the plan may also be specified (*"one day"* in our example). The complete annotation is as follows:

- $P.s = start(findEntityType(t, month, -1))$ (the plan starts in the beginning of the last month)
- $P.e = end(findEntityType(t, month, -1))$ (the plan ends in the end of the last month)
- $P.repeat = entityRange(day, 1)$ (repeat period)
- $P.wake.s(= P.wake.e) \prec P.run.s \prec P.run.e$ (sequence of events within the plan)
- $P.wake.s(= P.wake.e) \in partByIEntity(X, morning, 1)$ (waking up in the morning)

Another special situation similar to iterativeness arises when an event occurs separately for each actor in distributive readings such as

Each company built its own headquarters in Boston.

This is indicated in the annotation by a special "distributive" marker:

$$D.built.s(= D.built.e) \prec t$$

An event may be both iterative and distributive as in

Many people wake up every morning and run 10 miles.

In this case the plan is marked as distributive as well:

$$D.P.s = start(findEntityType(t, month, -1)$$
$$\ldots$$

2.3 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 layer of tectogrammatical representation (TR) described in Section 1.1.1. The temporal annotation can be therefore viewed as an extension to TR.

We have annotated the Czech translation of portion (see below) of Wall Street Journal (WSJ) as present in the PCEDT (see Section 1.2) corpus. The development testing set was annotated by the author of this thesis and the evaluation testing set was annotated by a different annotator[6]. However, due to the complexity of the annotation procedure, the time specifications corresponding to the respective time expressions i the evaluation testing set were also annotated by the author of this thesis[7].

The primary reason for the decision to use these data for temporal annotation (in contrast to annotating the original PDT data) was the possibility to integrate the resulting automatic temporal information retrieval system into this machine translation framework. This allows us to directly evaluate the contribution to the existing machine translation system (see Chapter 4). However, this decision also has a major disadvantage – due to the PDT 1.0 style of PCEDT tectogrammatical annotation, our implemented experiments cannot be directly run on PDT 2.0 style data.

2.3.1 Extending PCEDT by Temporal Annotation

Our temporal corpus consists in the extension of the manual TR annotation of the Czech translation of WSJ as present in the PCEDT. This manual TR annotation covers parts of WSJ sections 22,23 and 24 and is divided (within PCEDT itself) into development and evaluation testing sets as follows:

DTest
subsections *2303, 2308, 2309, 2313, 2315, 2332, 2338, 2393, 2399, 2406, 2435, 2436* (**233** sentences in total)

ETest
subsections *2201, 2202, 2203, 2211, 2212, 2214, 2222, 2246, 2248, 2249* (**231** sentences in total)

We keep this division in our experiments as well. Note that there is no training set manually annotated on tectogrammatical level within PCEDT.

Generally, even the manual tectogrammatical annotation (not to speak of the automatic data) within PCEDT contains many errors, namely

- anomalies and errors in the tree structure
- errors in or absence of values of some grammatemes
- absence of subfunctors

[6]Jana Němcová

[7]The evaluation testing set should be as much a "black box" for the author of this thesis as possible because he tests the performance of his system of automatic analysis of temporal relations on this data set. As far as the meaning of time expressions is concerned, we were unable to avoid seeing the evaluation data.

- missing and inconsistent punctuation, especially direct speech markers (“, ”, *:* etc.)

Some of these errors would render automatic analysis of temporal relations virtually impossible, it was therefore necessary to repair the data. We have modified the original data as follows:

- We have corrected and added values of the grammatemes *tense* and *aspect*.
- We have corrected and added punctuation related to direct speech.
- We have non-systematically repaired incorrect functors.
- For the purposes of text generation we have developed an automatic procedure to assign subfunctors (see Appendix C). This insertion takes place “on the fly” so subfunctors are not a permanent part of the modified PCEDT data.

We have not changed the structure of TGTSs in any way as we consider such a modification to be a too grave intervention. However, this causes errors in our experiment that are only due to incorrect tectogrammatical annotation.

2.3.2 Annotation Format

The annotation is provided in a plain text format very similar to the presented annotation examples. Events are identified by their unique ordering number (ord attribute) within the corresponding TGTS. Example 2.1 shows a TGTS tree and its temporal annotation. This simple annotation format can be simply converted e.g. to XML.

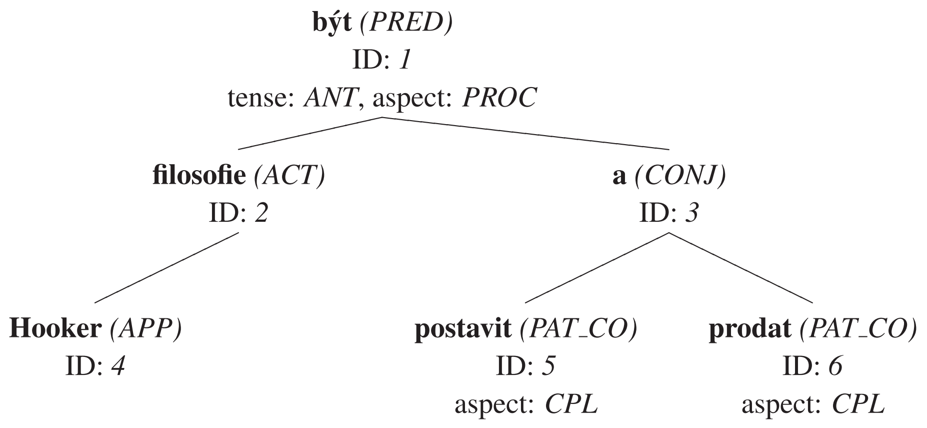
2.3.3 Corpus Statistics

In this section we provide several statistics related to the morphological types of the lexical items expressing events as well as distribution of plans and distributive markers and the number of time specifications related to time expressions. We present most of the statistics only for the development testing set because we believe that this sort of analysis of the evaluation testing set internal structure is in fact a violation of its supposed “black box” nature[8]. As far as the evaluation testing set is concerned, we only present overall statistics used for the evaluation of system performance. A more detailed statistics can be found in the corresponding files[9] but we have never studied it nor used it.

In the development testing set, there are **1691** points (corresponding to 962 entities) in total. **146** of these are marked by the distributive marker and **184** belong

[8]A potential developer a system of automatic analysis might for example decide not to handle certain morphological types or special markers because their frequency is too low in the evaluation testing set.

[9]available on the enclosed DVD



$P.s = 1.s$ (the validity of the plan is determined by the verb *"být" (to be)*
$P.e = 2.s$

$P.e \prec 0.p$ (the plan has ended in the past, before the time–of–speech)
$P.5.s = P.5.e$ *"postavit" (to build)* is a complex (instantaneous) event
$P.6.s = P.6.e$ *"prodat" (to sell)* is also a complex event

$P.5.s \prec P.6.e$ *"postavit"* precedes *"prodat"* in a single run of events

Figure 2.1: Example of temporal annotation for the sentence *"Hookerova filosofie byla postavit a prodat."*.

to a plan. Tables 2.1 list the distribution of events expressed by the respective morphological types.

Table 2.2 list the distribution of temporal relations between the respective morphological types after the transitive closure (**73517** relations in total). We also list the distribution of the original ("unclosed") set in Table 2.3 for the sake of illustration of the closure effect(**2075** relations in total). In the evaluation testing set, there are **87043** relations in total.

There are **92** and **115** time specifications present in the development and evaluation training set, respectively.

2.4 Conclusion

We have presented a scheme for temporal annotation featuring a pure functional approach for capturing meaning of time expressions. Although the scheme is very simple, it is more general than many of the schemes presented in Section 2.1 mainly in terms of types of events that are covered and the general intensional approach toward capturing meaning of time expressions. Unlike most of the annotation

	Time–of–speech	**Verbs**	**Nouns**
Entities	233 (24.22)	544 (56.55)	100 (10.40)
Points	233 (13.78)	1088 (64.34)	200 (11.83)

	Adjectives	**Plan Boundaries**
Entities	51 (5.30)	34 (3.53)
Points	102 (6.03)	68 (4.02)

Table 2.1: Distribution of time entities and associated time points as present in the development testing set. The percentage with respect to the sum of the respective rows is given in parentheses.

	TOS	**Verbs**	**Nouns**	**Adjectives**	**PB**
TOS	3124 (4.25)	X	X	X	X
Vrbs	23698 (32.23)	21239 (28.89)	X	X	X
Nns	4196 (5.71)	8334 (11.34)	976 (1.33)	X	X
Adjs	2232 (3.04)	3355 (4.56)	461 (0.63)	439 (0.60)	X
PBs	1517 (2.06)	2861 (3.89)	647 (0.88)	262 (0.36)	176 (0.24)

Table 2.2: Distribution of temporal relations between events expressed by the respective morphological types in the development testing set after the transitive closure.

	TOS	**Verbs**	**Nouns**	**Adjectives**	**PB**
TOS	221 (10.7)	X	X	X	X
Verbs	432 (20.8)	737 (35.5)	X	X	X
Nouns	32 (1.5)	104 (5.0)	121 (5.8)	X	X
Adj.	61 (2.9)	22 (1.1)	5 (0.2)	34 (1.6)	X
PB	24 (1.2)	182 (8.8)	56 (2.7)	4 (0.2)	40 (1.9)

Table 2.3: Distribution of temporal relations between events expressed by the respective morphological types in the original development testing set (before the transitive closure).

schemata, we also annotate inter-sentential relations (between adjacent sentences). Note that the scheme for relative ordering of events is able to capture each of the 13 Allen's relations. However, it is not able to express all relations derived from this basic set e.g. by logical disjunction.

To our best knowledge, the only scheme that is in almost every aspect more general than our approach is TimeML [65] mainly because it classifies events to groups (roughly corresponding to their lexical aspect). It also features a functional approach toward capturing the meaning of time expressions but we have not been able to determine how does the functional system look like (in the examples the authors present, the compositions are underspecified). It is also obvious that only some indexical time expressions are represented by functional compositions while other are only assigned their extension (calendrical value).

Another difference between our approach and the cited work is that our schema extends the existing deep–syntactic representation that already contains much of the information that other approaches annotate from scratch.

Chapter 3

Automatic Analysis of Temporal Relations

In this chapter we present the results achieved by our system of automatic analysis of temporal relations for Czech. The chapter is structured as follows: Section 3.1 introduces previous research in the field. Section 3.2 provides a brief overview of relevant points in the Czech morphologic and syntactic system. The so–called “Recursive Temporal Principle” (RTP), the core algorithm of the system, is described in Section 3.3. Section 3.4 discusses the metrics we use to evaluate the performance of the system. The main Section 3.5 presents the respective corrections and extensions to the RTP that yield the final system of automatic analysis whose temporal expression processing part is described in Section 3.6 in greater detail. Section 3.7 compares our results to the related work and concludes the chapter.

3.1 Introduction

The work on the automatic analysis of temporal relations is closely related to the annotation schemata introduced in Section 2.1.3.

Using TIMEX annotation scheme [51, 52], Mani and Wilson [44] attempted to determine event ordering by a naive rule–based approach based on blind proximity propagation of the extensions of time expressions that occur in a text (the event ordering is then given by the ordering of these extensions). They were able to obtain 59.4% accuracy on a small testing set of 8,505 words from New York Times articles and transcripts from Voice of America. Later, Mani et al. [42] trained decision tree for the problem of anchoring events to reference times yielding 80.2 F-measure. Extending the approach to deal with temporal relations between events within TimeML framework, Mani et al. [43] used a Maximum Entropy classifier on the union of the Time–Bank corpus [64] and the Opinion Corpus. They used event features such as aspect, tense, negation, modality etc. The classifier yields results as high as 93% accuracy which is significantly higher than the performance of a baseline rule–based approach they present. Another work related to TimeML framework is that of Boguraev and Ando [3] who trained a classifier on Timebank corpus for event anchoring for events and times within a single sentence that yielded 53.1 F-measure.

Schilder and Habel [71] used a set of rules (implemented as a cascade of Finite State Transducers) to anchor events expressed by verbs and nouns in German financial texts to time expressions. They were able to achieve 84.49 accuracy.

Li et al. [39] used a rule–based approach to determine event ordering in Chinese texts within a single sentence based on presence of temporal connectives (*before, after*). They approach yielded approximately 93% accuracy but its coverage was very low. Later, Li et al. [40] also used several machine–learning techniques for complex determination of temporal relations between events and achieved 78-88% accuracy.

3.2 Linguistic Prerequisities

To ease the understanding of the subsequent text, we give a very brief overview of the relevant points regarding morphologic system of Czech (which is substantially different from, for example, English tense morphology). We also introduce the notion of "content clauses" – a basis on which the RTP is formulated.

3.2.1 Morphology

In Czech, a verb bears either perfective or imperfective aspect. The former corresponds primarily to a complex event (taking place in a single time point), the latter corresponds primarily to a processual event (event or state with a time duration). The aspect information is therefore partly encoded lexically (in contrast to English where it is encoded purely gramatically), see Section 2.1.1. The following example demonstrates the difference:

Petr dělal domácí úkol. (Petr was doing his homework.)
Petr udělal domácí úkol. (Petr has done his homework.)

A finite verb bears one of the three basic tenses – past, present or future. There are no perfect tenses. A perfective verb can only bear past or future tense.

3.2.2 Content Clauses

The division between content and non–content clauses represents an alternative classification to the classification based on sentence constituents. The classification criterion is the relationship between the considered clause and its governing clause. Some classical grammars ([20, 30]) consider this division secondary to the primary classification according to sentence constituents, other studies argue for its primary position ([77, 78]).

Roughly speaking, content clauses correspond to indirect speech and some similar types of clauses. Object clauses form the core of content clauses but there are also content clauses among subject and attributive clauses. Classical grammars often try to define the content clause based on whether the clause expresses the so–called relative time (the tense is related to the tense of the governing clause) in contrast to the absolute time (its tense is related to the time–of–speech). The Recursive Temporal Principle described in the next section represents correction and generalization of this simplified distinction. However, if the content clauses were defined based only on their temporal properties the entire principle would be a tautology (or more precisely, the definition itself). See Panevová [59] for the detailed analysis of this notion and an attempt to provide an independent classification of content clauses. See also Hajičová et al. [27] for a discussion on the validity of the described classification for English.

3.3 Recursive Temporal Principle

RTP can be described as follows: The morphological tense of a finite verb – past, present, future – determines its temporal relation – precedence, overlap (i.e. at least one common point), posteriority – with its reference point.

- The reference point of the matrix clause of the sentence is the time–of–speech of the sentence.
- The reference point of a content clause is the point of the event of its governing clause.
- The reference point of a non–content clause is (recursively) the reference point of its governing clause.

For example, consider the sentence

Honza řekl,že Petr našel řešení, které Marie schvaluje.”
(lit. Honza said that Petr found solution that Marie approves.)

whose TGTS is shown in Figure 3.1 (the clause headed by *“najít” (find)* is a content clause, the clause headed by *“schvalovat” (approve)* is not). According to RTP

- *řekl* precedes time–of–speech, thus
  řekl.s (=řekl.e) $\leq$ *tos*
- *naš*el precedes *řekl*, thus
  našel.s (=našel.e) $\leq$ *řekl.s (=řekl.e)*
- *schvaluje* is simultaneous (overlaps) with *řekl*, thus
  schvaluje.s $\leq$ *řekl.s (=řekl.e)* $\leq$ *schvaluje.e*

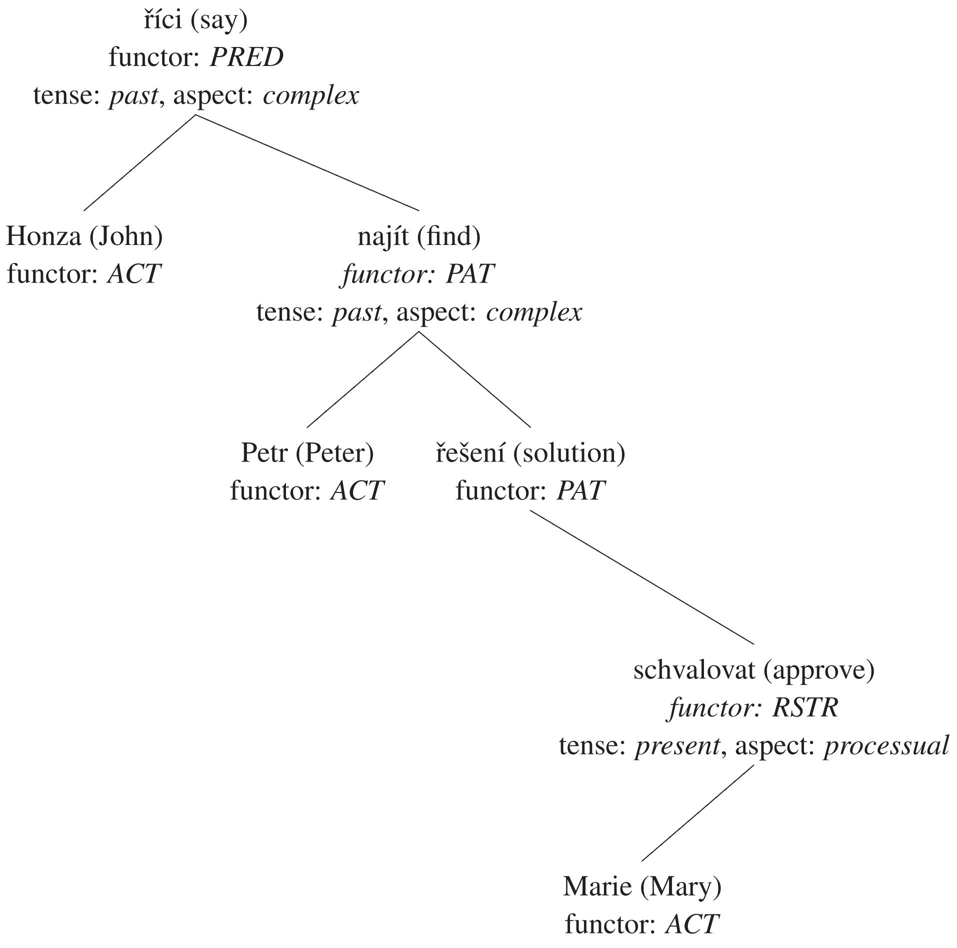


Figure 3.1: TGTS representing the sentence “Honza řekl, že Petr našel řešení, které Marie schvaluje.”.

As the annotation of content clauses is not part of TR, we take content clauses as identical to object clauses. As shown in Section 3.5, this simplification did not cause any additional errors.

3.4 Evaluation Metrics

The evaluation of the performance of the presented system is based on the comparison between the reference (annotated test set) matrix and the hypothesis matrix of temporal relations for the respective discourses (WSJ articles). The transitive closure of both is computed before the evaluation takes place.

We compute precision P and recall R defined as

$$P = \frac{\#Correct}{\#Hypothesis}$$
$$R = \frac{\#Correct}{\#Reference}$$

where $\#Correct$ denotes the number of correctly determined relations, $\#Hy-pothesis$ is the number of determined relations and $\#Reference$ is the number of relations within annotation.

A relation between two points (p and q) is considered correct if its value (i.e. one of $\prec$, $\preceq$, $\succ$, $\succeq$, $=$) is exactly the same within the hypothesis and the reference. If the hypothesis contains a weak relation ($\preceq$,$\succeq$) instead of the (correct) corresponding "strong" relation ($\prec$,$\succ$, or $=$) we call the result "weakly correct". We denote P_w and R_w the versions of P and R that consider weakly correct relations as correct.

By default, if p or q is marked as distributive and unrecognized as such or is part of an unrecognized plan the relation is considered incorrect. We also denote P_f and R_f the versions of P and R that do not take these markers into account. We provide these supplementary metrics because we believe that providing a weak (defensive) analysis or a correct analysis on an unrecognized plan or distributive marker is still an achievement that should be taken into account. On the other hand, it is obvious that in these cases the system did not succeed entirely.

Finally, P_{wf} and R_{wf} combine both aspects and are therefore the most permissive metrics. We also list F–measure (F, F_w, F_f and F_{wf}) for the respective metrics in the evaluation tables.

3.5 Automatic Analysis

In this section we list the respective rules used in the system of automatic analysis of temporal relations together with their performance. The rules are divided into two groups: those that correct the errors produced by the RTP are listed in Section 3.5.1 and those that extend it are listed in Section 3.5.2. Section 3.5.3 discusses unresolved issues (either errors or types of relations we were not able to determine).

Table 3.1 gives complete overview of the system performance on the development testing set: the unrefined RTP baseline, the performance with all rules on and the contribution of each respective rule by switching it off and providing difference to the all–on state. We do not provide difference of each single rule against the RTP baseline because the rules are not independent – application of some of them requires prior application of others to yield correct results. The rules are described in the following subsections.

Table 3.2 list the overall performance of the system on the evaluation testing set. As we were able to process deverbative adjectives and nouns only to a limited extent (see Section 3.5.2) we also report the performance measured only on verbs in Table 3.3.

3.5.1 Corrections to RTP

Direct Speech

If a sentence is part of a direct speech segment its reference point (i.e. the reference point of its matrix verb) obviously is not the time–of–speech point (as assumed by the unrefined RTP) but rather the speech verb introducing the direct speech segment. A TGTS from PCEDT – one per sentence – provides by itself no clue as to whether the sentence it represents belongs to a direct speech segment or not. To correct the produced errors it is necessary to detect direct speech segments (based on quotation markers) and speech verbs which we did.

Adverbial Aim Clauses

It turns out that – in contrast to the assumptions of the RTP – the behavior of adverbial clauses of aim conjoined by the conjunction *"aby"* (so that) is different from the expected behavior of adjunct clauses. Consider the following example:

> *Pan Shidler řekl, že firma propustila zaměstnance, aby ušetřila.*
> *(lit. Mr. Shidler said that company fired employees so that (it) saves (money).)*

According to the RTP, the reference point of the event *ušetřila (saves*[1]*)* is the point of *řekl (said)*, but there is no relation whatsoever – the company may save the money before or after Mr. Shidler made the statement. Instead, the event in the subordinate clause follows the event – *propustila (fired)* – in the governing clause.

The reference point of an aim clause is therefore its governing clause itself rather than the reference point of this governing clause as predicted by the RTP. In this aspect, the behavior of the aim clauses is identical to the behavior of content clauses rather than adjunct clauses. Moreover, because of the semantics of the aim clauses, such a clause expresses posteriority to its governing clause. It seems that this is the general behavior of the aim clauses. (However, one might argue that the reference point is in fact still the point predicted by the RTP but because the clause expresses no tense there is no temporal relationship. The posteriority relation to the governing clause is then given only by the semantic properties of the conjunction *"aby"*. In any case, it is important that the temporal relation can be drawn.)

The rule *Aim Clauses* reflects the proper behavior of this type of clauses.

Historic Present

Primarily, the verbal aspect corresponds to the duration of the denoted event: complex verbs denote events taking place in one single time point (with certain level of

[1]This is actually a conditional in Czech so its tense meaning is not expressed.

All	P	R	F	P_w	R_w	F_w
	75.39	39.39	**51.74**	77.12	40.29	**52.92**
Without Rule	dP	dR	dF	dP_w	dR_w	dF_w
Historic Present	3.17	0.91	**1.54**	3.21	0.91	**1.54**
Common Complex Anchor	0.33	1.54	**1.42**	0.27	1.53	**1.39**
Zero Conditionals	2.31	0.15	**0.68**	2.36	0.15	**0.69**
Modality	-0.06	1.03	**0.88**	-0.10	1.03	**0.87**
Aim Clauses	2.69	0.21	**0.83**	2.43	0.03	**0.61**
Direct Speech	2.23	0.13	**0.65**	2.09	0.03	**0.52**
Infinitives	-0.67	0.01	**-0.15**	-0.57	0.07	**-0.08**
Inference (full)	0.25	0.43	**0.43**	0.14	0.37	**0.35**
Inference (underspecified)	0.16	0.34	**0.33**	0.12	0.32	**0.30**
Negated Complex Aspect	0.18	0.20	**0.22**	0.17	0.20	**0.21**
RTP	5.48	4.20	**4.93**	5.14	4.06	**4.73**
All	P_f	R_f	F_f	P_{wf}	R_{wf}	F_{wf}
	86.23	45.05	**59.18**	88.10	46.03	**60.46**
Without Rule	dP_f	dR_f	dF_f	dP_{wf}	dR_{wf}	dF_{wf}
Historic Present	3.87	1.17	**1.93**	3.91	1.18	**1.94**
Common Complex Anchor	0.39	1.76	**1.63**	0.33	1.77	**1.62**
Zero Conditionals	2.89	0.30	**0.95**	2.95	0.31	**0.97**
Modality	-0.32	1.05	**0.84**	-0.37	1.05	**0.83**
Aim Clauses	3.02	0.20	**0.90**	2.76	0.03	**0.69**
Direct Speech	2.51	0.13	**0.72**	2.38	0.03	**0.59**
Infinitives	-0.07	0.37	**0.31**	0.05	0.44	**0.39**
Inference (full)	0.18	0.42	**0.41**	0.07	0.38	**0.34**
Inference (underspecified)	0.10	**0.34**	0.32	0.05	0.32	**0.29**
Negated Complex Aspect	0.14	0.19	**0.20**	0.14	0.20	**0.20**
RTP	7.27	5.30	**6.31**	6.94	5.18	**6.12**

Table 3.1: The overall performance of the system with all rules on the development testing set and the list of differences to this performance for each respective rule off. The last row corresponds to the unrefined RTP algorithm (baseline).

	P	R	F	P_w	R_w	F_w
RTP	69.84	42.50	**52.84**	73.74	44.87	**55.79**
Rules	71.28	47.40	**56.93**	74.28	49.40	**59.33**
	P_f	R_f	F_f	P_{wf}	R_{wf}	F_{wf}
RTP	75.07	45.68	**56.79**	79.28	48.24	**59.98**
Rules	78.08	51.93	**62.37**	81.37	54.11	**64.99**

Table 3.2: The overall performance of the system with all rules on and off (RTP baseline), respectively, on the evaluation testing set.

	P	R	F	P_w	R_w	F_w
RTP	69.84	59.66	**64.34**	73.74	62.99	**67.94**
Rules	71.46	66.49	**68.88**	74.46	69.28	**71.77**
	P_f	R_f	F_f	P_{wf}	R_{wf}	F_{wf}
RTP	75.07	64.13	**69.17**	79.28	67.72	**73.04**
Rules	78.28	72.84	**75.46**	81.57	75.90	**78.63**

Table 3.3: The overall performance of the system with all rules on and off (RTP baseline), respectively, on the evaluation testing set taking only the verbs into account.

approximation) while processual verbs denote events that last over a certain time period. Occasionally, this correspondence is broken as is the case of the so–called "historic present" (described in this temporal context already by Panevová in [60]). For example, in

Pan Shidler v rozhovoru říká, že společnost změní svou investiční strategii.
(lit. Mr. Shidler in interview says that company will_change its investment strategy.)

říká (says) actually means *řekl (said)* and the processual present is used for stylistic reasons. Note, however, that sentences such as

Zpráva říká, že Martin přijede zítra.
(lit. Message says that Martin will_come tomorrow.)

are not instances of the historic present. We have attempted a correction of errors that stem from this phenomenon for speech verbs only. To do so, it is necessary a) to detect an animate actor of the speech verb, b) to ensure that the instance really is a single point event and not a recurrent event as in

Maminka říká, že nemám chodit do lesa.
(lit. Mum says that I should not go to forest).

As far as a) is concerned, we use a simple animate actor detection algorithm: if the actor is not a proper noun it consults the Czech EuroWordNet ontology [80] whether the actor is an animate entity. If the actor is a proper noun, the algorithm scans for all the occurrences of that name within the discourse and checks whether there is a description associated via apposition, e.g.

Tom Baker, president of Machinists District 751)

The description is again checked against the ontology. We have not found any examples of the mentioned recurrent event usage in our domain of newspaper articles (WSJ) so we have not even attempted to fulfil the second condition. The current solution is therefore a domain–specific rule.

Negated Complex Verbs

Another case where the verb's aspect may not correspond to the nature of the event is the negation of a complex verb:

Boeing zásilku zatím nedostal.
(lit. Boeing package(accusative) so_far not_got.)

obdržet (to receive) is a complex verb but its negation in this context denotes a progressive state. However, in context such as

> *Boeing zásilku včera v 16:00 nedostal.*
> *(lit. Boeing package(accusative) yesterday in 16:00 not_got.)*

the event is still taking place in a single point (16:00 yesterday). Currently, we detect the presence of the specific adverbs such as *zatím* or *dosud (so far)* to correct this type of errors.

Conditionals

Although the conditional in Czech has two sets of forms (past conditional and present conditional), this distinction is not present in the annotation of the PCEDT corpus. The information on the tense of a conditional expression – required by the RTP – is therefore not available. Worse yet, as the autosemantic part of the conditional expression is identical to the ordinary past tense, the tense attribute of the conditionals in the PCEDT corpus is incorrectly set to past. This naturally produces many errors.

The decision on the proper "content" tense of the event denoted by a conditional seems to be hard. Consider the following examples:

- (future) *Ty projekty jsou velké, ale nešli bychom do nich sami.*
  (lit. Those projects are big but (we) would_not_go into them alone.)
- (present) *Nezdá se, že by stávka měla nějaký efekt.*
  (lit. It_does_not_seem that strike would have any effect.)
- (past) *Přišel by, ale ona nechtěla.*
  (lit. He_would_come but she did_not_want (him to).)

We were unable to find any reliable[2] principle that would allow us to distinguish between the respective cases. TGTSs of PDT distinguish at least between past conditionals and present[3] conditionals. However, this distinction is not present in PCEDT trees. We therefore do not determine any relation when considering a conditional. The rule is denoted as *Zero Conditionals*.

3.5.2 Extensions to RTP

Common Complex Anchor

If the reference point of a present processual verb (A) is again a processual verb (B) it is not possible to determine any relations in terms of the starting and the

[2]It is possible to set, for instance, future defaults and gain a higher F–measure as the precision loss is less than the recall gain but we have decided not to as the rules should be as reliable as possible.

[3]Present conditionals subsume conditionals both with present and future meaning.

ending points of the two events because they may overlap in a variety of ways. If B is in the present tense we have tried to extend RTP by taking the reference point of B (C) as the reference point of A. If C is again a processual present verb we use its reference point (D) and so forth. For example, in

Pavel řekl, že si myslí, že pracuje dobře.
(lit. Pavel said that he thinks that he works well.)

this principle allows us to determine the overlap relation between *"pracuje" (works)* and *"řekl" (said).*

We have encountered no errors due to this extension. A potential counterexample would consist in the possible intransitivity of the overlap relation.

Modality

Let us consider the situation where an active[4] modal verb M is associated with an infinitive verb V whose reference point (according to the RTP) is R. In this situation, M expresses the tense of the compound $M - V$ expression whereas V expresses the aspect. We have tried to identify the cases where a relation between V and R can be drawn for a complex V.

A reliable determination does not seem to be possible for the past tense of M *"moci" (can),"mít" (should),"chtít" (want)* and *"smět" (may)* probably as the following examples demonstrate:

(Dnes) řekl, že na vycházku mohl/měl/směl/chtěl vyrazit včera/zítra.
(He said (today) that he could/should//might (have)/wanted (to) go for a walk yesterday/tomorrow.)

All variants of the sentence are plausible[5] but the relation between *"řekl"* and *"vyrazit"* differs and cannot be determined without the respective adverbs. An exception is the modal verb *"muset" (must)* whose past tense seems to always indicate that V precedes R.

Present or future tense of all the modal verbs seems to indicate that V follows R:

Petr může/má/chce/smí/musí přijít.
(Petr can/should/wants(to)/may/must come.)

We have not found any real counterexample to this rule in our corpus, the precision decrease is caused by relations that are part of plan, i.e., there are more

[4]In Czech, a modal verb is an ordinary verb from the viewpoint of morphology and it is conjugated normally.

[5]However, it is true that the variant with "směl" and "zítra" sounds a bit strange.

instances where the "normal" relationship between V and R is broken as in

(Jiní staví a prodávají.) My chceme postavit a ponechat si.
(Others build and sell.) We want to build and keep.

Infinitives

We have discussed the relations between finite verbs (and infinitives surrounded by auxiliary finite verbs) so far. Let us consider the situation where an infinitive (I) is dependent on an autosemantic finite verb (F). Our hypothesis is as follows: If I is complex then $F.s \leq I.s$. For example, in

"Přišel nám pomoci."
(He came to help us.)

Note that the hypothesis obviously does not hold if I is processual as the following examples demonstrate:

"Odmítl nadále pracovat."
(He denied to work further.)

"To dělníkům pomáhá vyvíjet tlak na společnost."
(That helps workers to put pressure on the company.)

We are aware that this is quite an ad-hoc hypothesis, nevertheless we have not found any real counterexample to it in the corpus. In spite of that, this rule actually decreases the resulting F–measures numbers. As for modalities, the precision loss is caused by iterative verbs (parts of plans).

Temporal Clauses

Czech temporal subordinate clauses determine the relative ordering relation of the expressed event not only with respect to its reference point (according to the RTP) but also to its governing clause. TGTS captures various types of temporal complements such as "before", "after", "parallel to" etc. For certain types of complements, this information can be quite straightforwardly used to determine relations between the head verb of a temporal subordinate clause and the head verb of its governing clause.

This rule brought only a small F–measure increase due to the small number of affected temporal clauses. The rule is denoted as *Temporal Clauses* in Table 3.1.

Inference from Time Expressions

As shown in Section 2.2.2, the annotation scheme provides a functional approach to capture meaning of various time expressions. In order to infer relative ordering relations based on these determinations it is necessary to construct the functional compositions and to compare the respective compositions. Both issues are described in greater detail in Section 3.6.

The *Inference (underspecified)* row in Table 3.1 corresponds to the comparison of time specifications without the knowledge of the exact value of time–of–speech point as described in Section 3.6. The *Inference (full)* row lists the results for the comparison where this value is known which therefore makes it possible to draw more precise estimates (exact values in most cases). The performance increase is low due to the fact that only the inferences based on time expressions within one sentence or in adjacent sentences may be drawn as the annotation does not contain relations between events across more than one sentence – we would not be able to evaluate full analysis based on time expressions. There are only a few adjacent time expressions in the corpus in total.

Additionally, we also detect events expressed by nouns by the analysis of time specifications (see below).

Non–verbal Events

We detect events that are expressed by a non–verb to a very limited extent only. Firstly, every noun that has a temporal adverbial recognized as a time specification attached to it is considered an event. For all these events we only draw the trivial $s \preceq e$ relation.

Secondly, the morphological tag allows for detection of certain deverbative adjectives. In Czech, there are two basic categories of deverbative adjectives: “active’ (corresponding to the English -ing verb form) and “resultative” (corresponding to the English perfect verb form). Adjectives of the former category are marked by G subtype in their morphological tag. A very small fraction of the adjectives of the latter category (those corresponding to a transgressive verb form) is marked by M subtype. For these adjectives, we draw $s \prec e$ and $s = e$ relations, respectively.

The overall effect of this extension is negligible so we do not list it as a separate rule.

3.5.3 Unresolved Issues

Deverbative Adjectives and Nouns

Both the “active” and “resultative” form of adjectives (see the previous section) seem to be counterparts of the subordinated attributive clauses. Unfortunately, a reliable determination of a temporal relation for any of these adjectives does not seem to be possible as the following examples demonstrate:

- *Řekl, že (namísto současné trosky) předá synovi prosperující podnik. (= podnik, který bude prosperovat)*

  (He said that (instead of the current ruin) he would pass a well-performing company on his son.) (= a company that will perform well)

- *Slíbil jí, že její prosperující podnik neprodá. (= podnik, který prosperuje)*

  (He promised her that he would not sell her well-performing company.) (= a company that performs well)

- *Policie prohlásila, že ukradené zlato bude navráceno bance. (= zlato, které bylo ukradeno)*

  (The policie announced that the stolen gold will be returned to the bank.) (= gold that was stolen)

- *Před loupeží slíbil, že ukradené peníze budou rozděleny rovným dílem. (= peníze, které budou ukradeny)*

  (He promised before the robbery that the stolen money would be distributed evenly.) (= money that will be stolen)

The sentences are of the same pattern but the time meaning of the deverbative items change in accordance with the context.

Exceptional Clauses

We have identified two interesting types of clauses that violate the RTP. The first type is represented by sentences containing certain clauses where two different tense forms could be used: the future and the present tense. Consider the following example:

Otec chce synovi předat (namísto současné trosky) podnik, který prosperuje (= bude prosperovat).
(lit. Father wants (on his) son pass (instead_of current ruin) company that prospers (= will prosper).

Future tense corresponds to the "real" meaning of the clause predicted by the RTP, whereas the present tense is a marked variant yielding incorrect results in the context of the RTP.

The other case (described already by Panevová in [60]) is the so–called "commentary" of the speaker as in

Pan Shidler řekl, že společnost chce odkoupit pozemky v Ohiu, které nyní vláda plánuje rozdělit.
(lit. Mr. Shidler said that company wants (to) buy properties in Ohio that now goverment plans (to) divide.)

where the last subordinate clause is not part of the reported speech but rather a commentary of the author of the article.

We do not see any feasible way to identify and correct these instances.

Iterative Events

As some of the described rules demonstrate, appurtenance of an event to a plan leads to many errors in general because the nature of the described recurrent series of events is different. It represents our greatest source of errors. A very common situation looks as follows:

BPH Funding Co. běžně nejprve složí zálohu a poté koupé pozemek.
(lit. BPH Funding Co. usually first will_make deposit and then will_buy property.)

where the future tense of both verbs wrongly suggests that the events take place in the future.

The appurtenance of an event to a plan can also lead to aspect deviations where the processual aspect of a verb can (but does not have to) be used to express iteration of a complex event as in:

BPH Funding Co. běžně nejprve skládá zálohu a poté kupuje pozemek.
(lit. BPH Funding Co. usually first makes deposit and then buys property.)

where the respective verbs express instantaneous events in the single run of the plan. (However, it is often not clear what the nature of an event in the single run is and both variants seem to be plausible).

Iterative events can be detected by the presence of relevant temporal adverbs such as *obvykle (usually)*, *často (often)*, etc., but this handles only a little fraction of (incomplete) parts of plans so there is essentially no improvement.

Remaining Errors and Unresolved Issues

The errors that remain after the application of all the described rules are caused partly by counterexamples to RTP we were unable to correct as described in Section 3.5.3, partly by (common) errors in the structure (or grammatemes) of the underlying TGTSs and partly by the errors in the temporal annotation.

The rest of the unresolved types of relations are largely represented by inferences based heavily on context and world knowledge. An automatic analysis of these types of relations seems to be very difficult.

3.6 Inference of Temporal Relations from Time Expressions

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. Both are described in the subsequent sections.

3.6.1 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 consists 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 can be an interpretation function associated with each rule. 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.

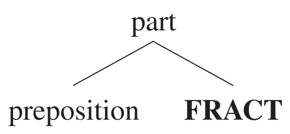
An example of the grammar is shown in Figure 3.2. S, $FRCT$, EXP and $PART$ are non–terminal symbols representing the respective subtrees (S is the start symbol and $PART$ is EXP, recursively). $part$, $preposition$, $numerator$, $denominator$, $modifier$, $ordinal$, $count$, $quantifier$ represent nodes in the parsed TGTS that fulfill corresponding lexical conditions (see below), most of them are optional. *InterpretVaguePart*, $InterpretFraction$ and $InterpretCore$ are interpretation functions creating the functional compositions that correspond to the information provided by the given layer. For example, $InterpretVaguePart$ applies the $partVague$ function (see Section 2.2.2) on the interpretation (of type $t\_interval$) resulting from the subtree $FRCT$ and supplies the second argument $part$.

To illustrate how the parsing works consider the TGTS subtree shown in Figure 3.3 that corresponds to the adverbial phrase

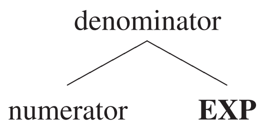
na začátku druhé poloviny příštího měsíce
(lit. in (the) beginning of (the) second half (of the) next month)

Figure 3.4 captures the parsing process: S is the starting symbol of the grammar so its righthand side must match the subtree. And it does: $part$ matches the word "beginning", $preposition$ matches "in" and the remaining branch must

S (interpreted by $InterpretVaguePart$):



FRACT (interpreted by $InterpretFraction$):



EXP (interpreted by $InterpretCore$):

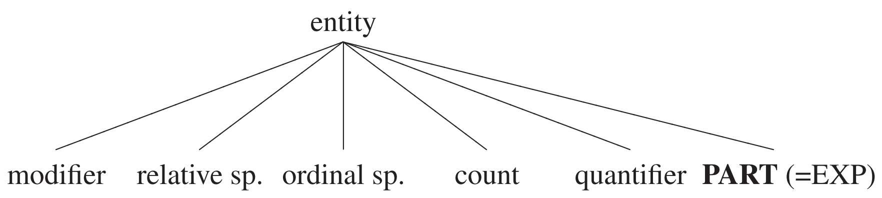


Figure 3.2: Examples of rules recognized by the tree grammar. The horizontal ordering of nodes is arbitrary – it is not related to the surface ordering of the respective lexical units.

match the subtree determined by $FRACT$ (the interpretation waits till the interpretation of $FRACT$ is available). Again, *numerator* matches "half", *denominator* matches "second" and the remaining branch must match the subtree determined by EXP and so on. When the entire subtree is matched we can proceed with interpretations in the bottom–up manner. Firstly, $InterpretCore$ interprets the lowest subtree (EXP) corresponding to "next month" and yields the following composition:

$$findEntityType(t, month, 1)$$

This composition is passed to the $InterpretFraction$ that yields

$$partByFraction(findEntityType(t, month, 1), 1, 2, 1, 1)$$

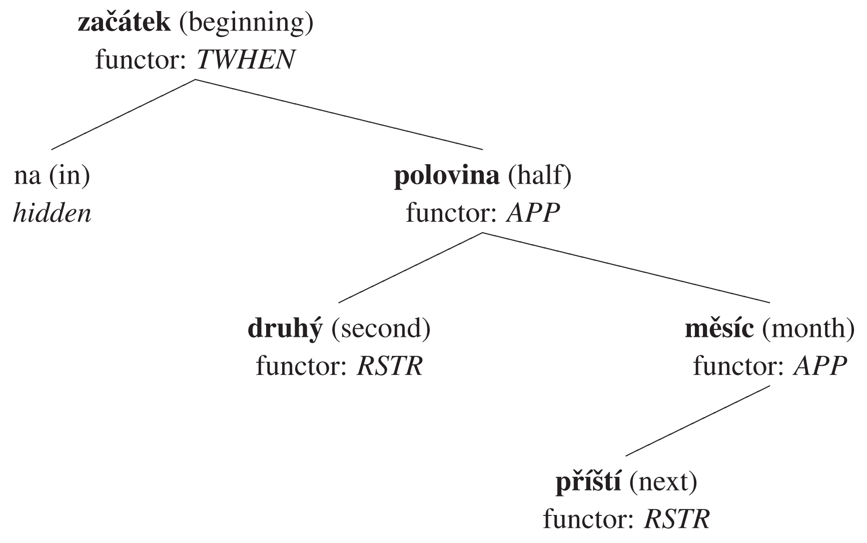


Figure 3.3: A simplified TGTS representing the adverbial phrase *"na začátku druhé poloviny příštího měsíce"*. The English auxiliaries (*the*, *of*) are not displayed as they are not present in the Czech sentence.

and, finally, $InterpretVague$ returns

$$\begin{aligned} &partVague(\\ &\qquad partByFraction(findEntityType(t, month, 1), 0, 1, 1, 2), \\ &\qquad beginning) \end{aligned}$$

The presented example is a simplification of the actual grammar. It is clear that a time expression does not have to contain all the described layers. For example, the phrase

na začátku příštího měsíce
(lit. in (the) beginning (of the) next month)

should also be accepted by the grammar. Therefore, there can be multiple right–hand side alternatives for a single non-terminal on the left–hand side, one of which may be just a simple transition to another non–terminal (bypassing the given tree layer). The actual form of the rules from Figure 3.2 is as shown in Figure 3.5. Note that because the top layers may be missing the preposition node has to be repeated in every rule.

The set of allowed lexical items for a given terminal node in the grammar is specified by an associated procedure, it is therefore very general. The grammar also allows for specification of additional constraints on the children in the respective

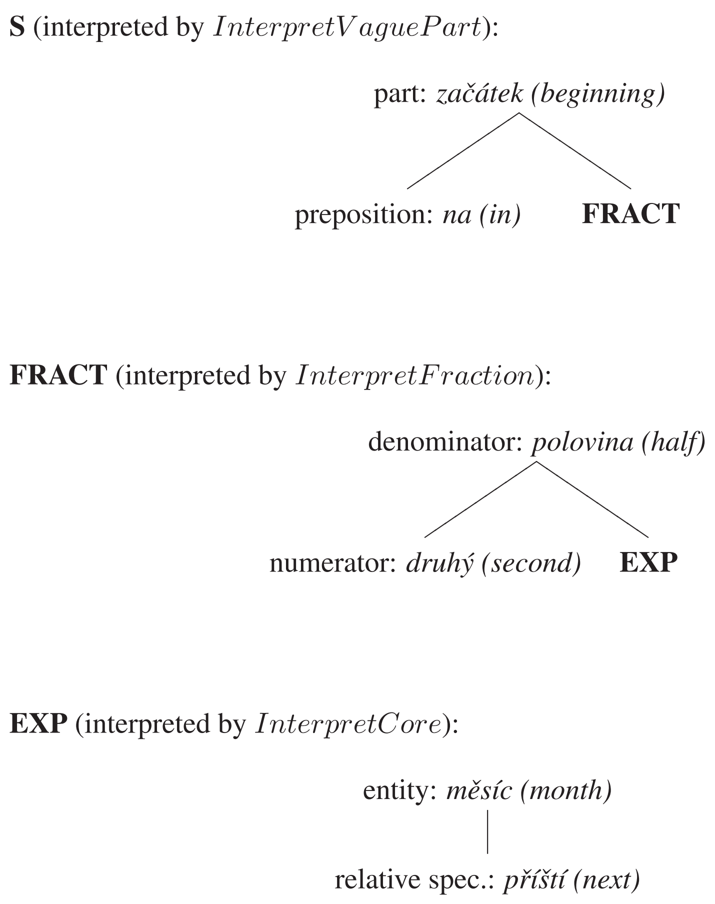

S (interpreted by $InterpretVaguePart$):
part: *začátek (beginning)*
preposition: *na (in)*
FRACT
FRACT (interpreted by $InterpretFraction$):
denominator: *polovina (half)*
numerator: *druhý (second)*
EXP
EXP (interpreted by $InterpretCore$):
entity: *měsíc (month)*
relative spec.: *příští (next)*

Figure 3.4: Parsing process of the TGTS subtree shown in Figure 3.3 by the grammar from Figure 3.2.

rules such as surface order, logical conditions on their co–occurrence etc. Additionally, because some embedded subtrees cannot be properly interpreted without the knowledge of structure that embeds them, *parameters* make it possible to pass parsing information "top–down".

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 accepted. Appendix A lists the complete grammar.

Performance Evaluation

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 ex-

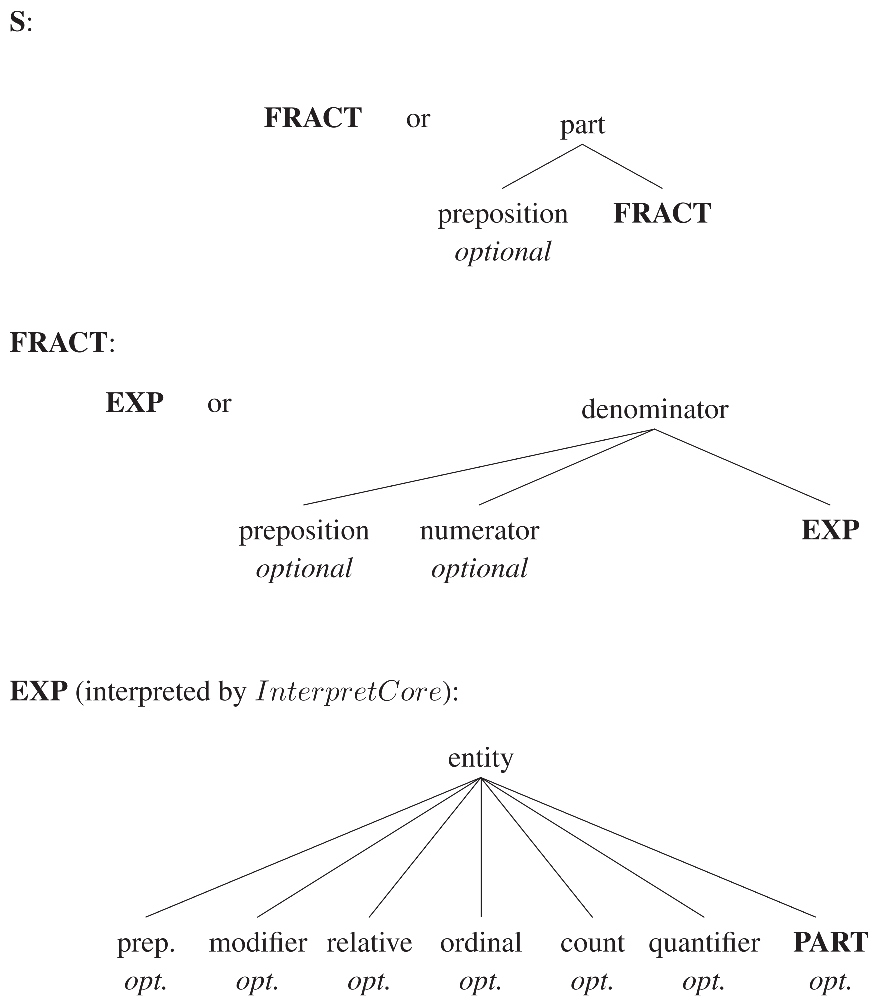


Figure 3.5: Actual form of the example rules recognized by the tree grammar.

pression *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

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

Table 3.4: The performance of the time expression recognition system.

and R_p respectively that: a) take misplaced functional compositions (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 F and F_p.

Table 3.4 lists the results. The errors are caused mainly by the errors in the TGTS annotation.

3.6.2 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 tracing the composition "inside–out" (from the innermost function) and using 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 can be inferred.

Let us present the inference mechanism in greater detail. After a function has been processed, the beam structure carries the following information (*FE* denotes the extension of the processed function – set of intervals in general):

- Lower and upper estimate of the lowest (leftmost) point of the resulting set of intervals[6] relative to the source input interval (constructed by a constructor) or point variable, i.e., the innermost argument in the functional composition.
- Lower and upper estimate of the highest (rightmost) point of the resulting set of intervals relative to the source input interval or point variable.
- The absolute time versions of the items described above which are only computed if the information is available. In general, they are set to exact time points using CPAN DateTime package[7] if possible. As their use is similar to

[6]In general, a function returns a set of intervals such as $series$ family of functions. However, in most cases we only deal with a single interval or a time point.

[7]by Dave Rolsky

their relative estimate counterparts, we do not include them in the description list.

- Lower and upper estimate of the size of the resulting time extension if it is a single interval.
- Indication of whether the lowest and highest point, respectively, of the resulting set of intervals lie on a boundary of an entity such as day, week etc. A value of these items is a set of entities on whose boundaries the given point lies. It is only present, if it applies for both the lower and upper estimate for the given point.

This information may in general allow for a more precise estimate of extension of the next (outer) processed function.

To illustrate the inference process, let us consider the following sentence:

My mother will visit us no sooner than September of next year but my father will come this Friday.

We will show how the inference engine determines the relative ordering of the events (without the knowledge of time–of–speech of the utterance). The parser (interpreter) produces the following temporal specifications:

$$\begin{aligned}&visit.s(=visit.e)\geq\\&\quad start(partIByName(findEntityType(t,year,1),September,1))\end{aligned}$$

$$come.s(=come.e)\in findIEntityByName(t,Friday,1)$$

The extension of the first functional composition is estimated as follows:

1. $findEntityType$ is estimated. The estimate of the leftmost point (beginning of the next year)) is set to the interval

   $$(0,entityRange(year,1))$$

   relative to t as the next year may start from almost immediately after t (if t is the last moment of a year) to almost a year from t (if t is the first moment of a year). Similarly, the estimate of the rightmost interval is set to

   $$(entityRange(year,1),entityRange(year,2))$$

   The size the interval is set to $entityRange(year,1)$ as it represents exactly one year. The boundary markers are also set to the beginning and end of a year.

2. $partIByName$ is estimated, the estimate structure (beam) from the preceding point is supplied as its first parameter. The beam is updated as follows: Since we know that the interval the September part is taken from is a calendrical year, the estimate of the leftmost point is shifted by 9 months to

   $$(entityRange(month, 9),$$

   $$entityRange(year, 1) + entityRange(month, 9))$$

   whereas the estimate of the rightmost point is shifted by 10 months to

   $$(entityRange(month, 10),$$

   $$entityRange(year, 1) + entityRange(month, 10))$$

   Again, the information that the outcome represents a month is recorded.

3. The $start$ function is processed which consists simply in taking the leftmost point from the input beam as the estimate and recording that the outcome is a point. The final extension estimate is therefore

   $$(entityRange(month, 9),$$

   $$entityRange(year, 1) + entityRange(month, 9))$$

The extension of the second functional composition is estimated to lie within

$$(0, entityRange(day, 8)$$

Comparing the estimates of the extensions of the two functional compositions it is clear that the second one is always less than the first and because *"visit"* takes place after the interval of the first composition we may draw the desired inference:

$$come.s (= come.e) \prec visit.s (= visit.e)$$

See Appendix B for detailed description on how respective functions are estimated.

3.7 Conclusion

The results achieved by our system of temporal analysis have shown that the RTP provides vast majority of the gained information. In comparison to the RTP backbone, the respective rules that correct and extend the RTP do not affect the overall performance very significantly. Nevertheless, we believe that this experiment showed how far a reliable rule–based approach can go. Moreover, some of the rules them are of linguistic relevance per se.

The inferencing mechanism based on the meaning of time expressions is one of the key aspects of our temporal analysis scheme although it turned out that it is quantitatively rather insignificant. Although the inferencing subsystem deals with a variety of simple expressions many feasible issues are still not implemented such as the analysis of ranges (from–to), disjunctions, comparison between standard and derived entity time types (e.g. between calendric and fiscal years), etc.

The comparison to the related work introduced in Section 3.1 is difficult as the reported success rates are measured on different testing data, use different annotation schemata, annotation principles and evaluation metrics. Nevertheless, it seems clear that machine-learning approaches can highly increase the performance of a system of automatic temporal analysis. Mani et al. [43] is a nice example of this. Many of the problems mentioned in Section 3.5.3 (conditionals, deverbative adjectives etc.) as well as many other unresolved relations could be possibly resolved by such an approach.

As far as inferencing from time expressions is concerned, there has been a substantial previous work on the subject. Some approaches such as [45] or [71] work only with extensions of time expressions and are therefore unable to process discourses whose time-of-speech is unknown. The only schema we are aware of that does deal with time expressions intensionally (via functional compositions) is TimeML [65]. The annotation scheme introduces time functions for some indexical expressions but to our best knowledge there is no implemented inferencing engine that would compare the functions directly. Systems that extract and normalize time expressions from raw text (for example [82]) usually achieve high F-measure scores. As explained in Section 3.6.1 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 nor they address pragmatic inference.

Chapter 4

Text Generation within a Machine Translation Framework

In this chapter we present a generation component of a transfer–based Czech–to–English machine translation system. The system itself is still under development. The transfer layer is represented by the tectogrammatical representation. The chapter is structured as follows: Section 4.1 introduces our model and previous research on generation from various sorts of abstract structures. Section 4.2 describes which PCEDT data were used and how they were modified. The generation process itself is subject of Section 4.3 and the achieved results are listed in Section 4.4. Section 4.5 concludes the chapter.

4.1 Introduction

Syntactic models for language are being reintroduced into language and speech processing systems thanks to the success of sophisticated statistical models of parsing [8, 13]. Representing deep syntactic relationships is an open area of research; examples of such models are exhibited in a variety of grammatical formalisms, such as Lexical Functional Grammars [7], Head-driven Phrase Structure Grammars [61] and the tectogrammatical representation of the Functional Generative Description (see Section 1.1.1). For the purpose of high level transfer–based machine translation frameworks and also given the large number of publications on the analytical processes for deriving deep representation (e.g., dependency parsing and verb frame prediction) we believe it is important to show that surface structure and sentences can be recovered from abstract deep representations.

Augmenting models of machine translation (MT) with syntactic features is one of the main fronts of the MT research community. The Hiero model has been the most successful to date by incorporating syntactic structure amounting to simple tree structures [9]. Synchronous parsing models have been explored with moderate success [83, 66]. An extension to this work is the exploration of deeper syntactic models, such as TR. However, a better understanding of the synthesis of surface structure from the deep syntax is necessary.

We present a generative model for surface syntax and strings of English given tectogrammatical trees. Sentence generation begins by inserting auxiliary words associated with autosemantic nodes; these include prepositions, subordinating conjunctions, auxiliary verbs, and articles. Following this, the linear order of nodes is modelled by a similar generative process. These two models are combined in order to synthesize a sentence. The insertion of auxiliaries and surface ordering takes place within TGTSs using hidden nodes.

We have also preliminarily experimented with a variant of the model where the surface ordering phase is carried out on the analytical representation layer. A TGTS with inserted auxiliaries is transformed to the analytical tree using a simple list of tree transformations written as a program recognized by the interpreter described in Chapter 5. However, the quality of the resulting analytical trees is not good and the final results are much worse that those for the main model as listed in Section 4.4. We have therefore decided to abandon this variant.

4.2 Preprocessing Tectogrammatic Data

In order to evaluate the efficacy of the generation model, we construct a dataset from both manually annotated data and automatically generated data of PCEDT (see Section 1.2). The information contained in the originally manually annotated TGTSs all but specifies the surface form. We have modified the annotated data by removing all features except those that could be directly transferred across languages. Specifically, we preserve the following features: lemma, functor, subfunctor and (Penn Treebank) morphological tag which is part of the PCEDT.

Although the morphological tag is definitely not part of the pure TR, we believe we would be able to obtain it with high accuracy in the Czech–English transfer process as it only encodes simple English morphological properties that are (with the exception of the verb gerund form) encoded by TGTS grammatemes . Nevertheless, we have not tested this assumption.

Subfunctor information is not part of PCEDT but as it is crucial for determination of the complete meaning of a given phrase (and the associated preposition) we have generated subfunctors automatically according to rules listed in Appendix C.

4.3 Generative Process

In this section we describe the generative process that inserts the synsemantic auxiliary words, reorders the trees, and produces a sentence. Our evaluation will be on English data, so we describe the models and the model features in the context of English. While the model is language independent, the specific features and the size of the necessary conditioning contexts is a function of the language.

Given a TGTS T, we wish to predict the correct auxiliary nodes A and an ordering of the words associated with $\{T \cup A\}$, defined by the function $f(\{T \cup A\})$.

The functions f determine the surface word order of the words associated with nodes of the auxiliary-inserted TGTS: $N = \{T \cup A\}$. The node features that we use from the nodes in the TGTSs and ATs are: the word lemma, the part-of-speech (POS) tag, the functor and the subfunctor. The objective of our model is:

$$\arg\max_{A,f} P(A, f|T) \tag{4.1}$$

$$= \arg\max_{A,f} P(f|A, T)P(A|T) \tag{4.2}$$

$$\approx \arg\max_{f} P(f|T, \arg\max_{A} P(A|T)) \tag{4.3}$$

In Equation 4.3 we approximate the full model with a greedy procedure. First, we predict the most likely A according to the model $P(A|T)$. Given A, we compute the best ordering of the nodes of the tree, including those introduced in A.

There is an efficient dynamic-programming solution to the objective function in Equation 4.2; however, in this work we experiment with the greedy approximation.

4.3.1 Insertion Model

The specific English auxiliary nodes which are not present in TR include articles, prepositions, subordinating conjunctions, and auxiliary verbs. For each node in the TGTS, the generative process predicts which synsemantic word, if any, should be inserted as a dependent of the current node. We make the assumption that these decisions are determined independently.

Let $T = \{w_1, \ldots, w_i, \ldots, w_k\}$ be the nodes of the TGTS. For each node w_i, we define the associated node a_i to be the auxiliary node that should be inserted as a dependent of w_i. Given a tree T, we wish to find the set of auxiliary nodes $A = \{a_1, \ldots, a_k\}$ that should be inserted[1]:

$$P(A|T) \tag{4.4}$$

$$= \prod_i P(a_i|a_1, \ldots, a_{i-1}, T) \tag{4.5}$$

$$\approx \prod_i P(a_i|T) \tag{4.6}$$

$$\approx \prod_i P(a_i|w_i, w_{g(i)}) \tag{4.7}$$

Equation 4.5 is simply a factorization of the original model, Equation 4.6 shows the independence assumption, and in Equation 4.7 we make an additional conditional independence assumption that in order to predict auxiliary a_i, we need only know the associated node w_i and its governor $w_{g(i)}$.[2]

[1]Note that we include the auxiliary node labeled NOAUX to be inserted, which in fact means a node is not inserted.

[2]In the case of nodes whose governor is a coordinating conjunction, the governor information comes from the governor of the coordination node.

We further divide the model into three components: one that models articles, such as the English articles *the* and a; one that models prepositions and subordinating conjunctions; and one that models auxiliary verbs. The first two models are of the form described by Equation 4.7. Additionally, each model is independent of the other and therefore up to two insertions per TGTS node are possible (an article and another syntactic modifier). In a variant of our model, we perform a small set of deterministic transformations in cases where the classifier is relatively uncertain about the predicted insertion node (i.e., the entropy of the conditional distribution is high).

The auxiliary verb insertion model is preliminary. We are unable to determine correct modal auxiliary as the modality of a verb grammateme is not present in the PCEDT data. Some of the remaining auxiliary verbs can be inserted based on the tense, aspect (resultative or not) and negation of the verb but in general the knowledge of the proper English tense is necessary for the determination of the correct auxiliary (see Chapter 6).

Insertion Features

Features for the insertion model come from the current node being examined and the node's governor. When the governor is a coordinating conjunction, we use features from the governor of the conjunction node. The features used are the lemma, morphological tag, functor and subfunctor for the current node, and the lemma, morphological tag, and functor of the governor.

$$\prod_i P(a_i|w_i, w_g) \tag{4.8}$$

$$= \prod_i P(a_i|l_i, t_i, f_i, l_g, t_g, f_g) \tag{4.9}$$

The left-hand side of Equation 4.8 is repeated from Equation 4.7 above. Equation 4.8 shows the expanded model for auxiliary insertion where l_i is the lemma , t_i is the morphological tag, and f_i is the functor of node w_i

4.3.2 Surface-order Model

The node ordering model is used to determine a projection of the tree to a string. We assume the ordering of the nodes in the input TGTSs is arbitrary, the reordering model proposed here is based only on the dependency structure and the node's attributes (words, morphological tags, etc.). In a variant of the reordering model, we assume the deep order of coordinating conjunctions to be the surface order.

Algorithm 1 presents the bottom-up node reordering algorithm. In the first part of the algorithm, we determine the relative ordering of child nodes. We maximize the likelihood of a particular order via the precedence operator $\prec$. If node $c_i \prec c_{i+1}$, then the subtree of the word associated with c_i immediately precedes the subtree of the word associated with c_{i+1} in the projected sentence.

Algorithm 1 Subtree Reordering Algorithm

```
    procedure REORDER($T, A, O$)                                    ▷ Result in $O$
        $N \leftarrow$bottomUp($T \cup A$);    $O \leftarrow \{\}$
        for $g \in N$ do
            bestScore $\leftarrow 0$;    $o_g \leftarrow \{\}$
5:          for $C \leftarrow$permutation of $g$'s children do
                for $i \leftarrow 1 \ldots |C|$ do
                    $s \leftarrow s * P(c_i \prec c_{i+1} | c_i, c_{i+1}, g)$
                end for
                if $s >$ bestScore then
10:                 bestScore $\leftarrow s$;    $o_g \leftarrow C$
                end if
            end for
            bestScore $\leftarrow 0$;    $m \leftarrow 0$
            for $i \leftarrow 1 \ldots |bestOrder|$ do
15:             $s \leftarrow P(c_i \prec g \prec c_{i+1} | c_i, c_{i+1}, g)$
                if $s >$ bestScore then
                    $s \leftarrow$ bestScore ;    $m \leftarrow i$
                end if
            end for
20:         Insert governor $c_g$ after m$^{th}$ child in $o_g$
            $O \leftarrow O \cup o_g$
        end for
    end procedure
```

In the second half of the algorithm (starting at line 13), we predict the position of the governor among the previously ordered child nodes. Recall that this is a dependency structure; knowing the governor does not tell us where it lies on the surface with respect to its children. The model is similar to the general reordering model, except we consider an absolute ordering of three nodes (left child, governor, right child). Finally, we can reconstruct the total ordering from the subtree ordering defined in $O = \{o_1, \ldots, o_n\}$. The procedure described here is greedy; first we choose the best child ordering and then we choose the location of the governor.

Reordering Features

Our reordering model for English is based primarily on non-lexical features. We use the morphological tag and functor from each node as features. The two distributions in our reordering model (used in Algorithm 1) are:

$$P(c_i \prec c_{i+1} | c_i, c_{i+1}, g) \tag{4.10}$$
$$= (c_i \prec c_{i+1} | f_i, t_i, f_{i+1}, t_{i+1}, f_g, t_g) \tag{4.11}$$
$$P(c_i \prec g \prec c_{i+1} | c_i, c_{i+1}, g) \tag{4.12}$$
$$= P(c_i \prec g \prec c_{i+1} | f_i, t_i, f_{i+1}, t_{i+1}, t_g, f_g) \tag{4.13}$$

In both Equation 4.10 and Equation 4.12, only the functor and morphological tag of each node is used.

4.3.3 Morphological Generation

In order to produce true English sentences, we convert the lemma and morphological tag to a word form. We use John Carroll's morphg tool[3] to generate English word forms given lemma/morphological tag pairs. This is not perfect, but it performs an adequate job at recovering English inflected forms. In the complete-system evaluation, we report scores based on generated morphological forms.

4.4 Empirical Evaluation

We have experimented with the above models on both manually annotated TGTSs and automatically generated trees from PCEDT.

All models were trained on the PCEDT data set, approximately 49,000 sentences, of which 4,200 were randomly selected as held-out training data, the remainder was used for training. We estimate the model distributions with a smoothed maximum likelihood estimator, using Jelinek–Mercer EM smoothing (i.e., linearly interpolated backoff distributions). Lower order distributions used for smoothing are estimated by deleting the rightmost conditioning variable (as presented in the above models).

4.4.1 Insertion Results

For each of the two insertion models (the article model and the preposition and subordinating conjunction model), there is a finite set of values for the dependent variable a_i. For example, the articles are the complete set of English articles as collected from the Penn Treebank training data (these have manual morphological tag annotations). We add a dummy value to this set which indicates no article should be inserted.[4] The preposition and auxiliary model assumes the set of possible modifiers to be all those seen in the training data that were removed when modifying the manual TGTSs.

The classification accuracy is the percentage of nodes for which we predicted the correct auxiliary from the set of candidate nodes for the auxiliary type. Articles are only predicted and evaluated for nouns (determined by the morphological tag). Prepositions and subordinating conjunctions are predicted and evaluated for all nodes that appear on the surface. We have experimented with different features sets and found that the model described in Equation 4.8 performs best when all features are used.

In a variant of the insertion model, when the classifier prediction is of low certainty (probability less than .5) we defer to a small set of deterministic rules. For

[3]Available on the web at:
`http://www.informatics.susx.ac.uk/research/nlp/carroll/morph.html.`

[4]In the classifier evaluation we consider the article *a* and *an* to be equivalent.

Model	Manual Data			
	Ins. Rules		No Rules	
Model	Articles	Prep & SC	Articles	Prep & SC
Baseline	N/A	N/A	77.93	76.78
w/o g. functor	87.29	89.65	86.25	**89.31**
w/o g. lemma	86.77	89.48	85.68	89.02
w/o g. POS	87.29	89.45	86.10	89.14
w/o functor	86.10	85.02	84.86	84.56
w/o subfunctor	87.49	88.65	**86.62**	88.20
w/o lemma	81.34	89.02	80.88	88.91
w/o POS	84.81	88.01	84.01	87.29
All Features	**87.49**	**89.68**	86.45	89.28
Model	Automatic Data			
	Ins. Rules		No Rules	
Model	Articles	Prep & SC	Articles	Prep & SC
Baseline	N/A	N/A	78.00	78.40
w/o g. functor	**88.07**	91.83	**87.34**	91.06
w/o g. lemma	87.53	90.95	86.55	91.16
w/o g. POS	87.68	**91.86**	86.89	**92.07**
w/o functor	86.01	85.60	84.79	85.65
w/o subfunctor	87.82	91.35	87.19	91.54
w/o lemma	81.28	91.03	81.42	91.33
w/o POS	85.53	91.08	84.69	90.98
All Features	87.87	91.83	87.24	92.02

Table 4.1: Classification accuracy for insertion models on development data from PCEDT 1.0. Article accuracy is computed over the set of nouns. Preposition and subordinating conjunction accuracy (P & SC) is computed over the set of nodes that appear on the surface (excluding hidden nodes in the TGTS – these will not exist in automatically generated data). Models are shown for all features minus the specified feature. Features with the prefix "g." indicate governor features, otherwise the features are from the node's attributes. The Baseline model is one which never inserts any nodes (i.e., the model which inserts the most probable value – NOAUX).

Model	Manual Data			
	Coord. Rules		No Rules	
	All	Interior	All	Interior
Baseline	N/A	N/A	68.43	21.67
w/o g. functor	**94.51**	**86.44**	**92.42**	**81.27**
w/o g. tag	93.43	83.75	90.89	77.50
w/o c. functors	91.38	78.70	89.71	74.57
w/o c. tags	88.85	72.44	82.29	57.36
All Features	94.43	86.24	92.01	80.26
Model	**Automatic Data**			
	Coord. Rules		No Rules	
	All	Interior	All	Interior
Baseline	N/A	N/A	69.00	21.42
w/o g. functor	94.90	87.25	93.37	83.42
w/o g. tag	93.82	84.56	91.64	79.12
w/o c. functors	91.91	79.79	90.41	76.04
w/o c. tags	88.91	72.29	83.04	57.60
All Features	**95.21**	**88.04**	**93.37**	**83.42**

Table 4.2: Reordering accuracy for TGTSs on development data from PCEDT. We include performance on the interior nodes (excluding leaf nodes) for the Manual data to show a more detailed analysis of the performance. “g.” are the governor features and “c.” are the child features. The baseline model sorts subtrees of each node randomly.

infinitives, we insert “to”; for origin nouns, we insert “from”, for actors we insert “of”, and we attach “by” to actors of passive verbs. In the article insertion model, we do not insert anything if there is another determiner (e.g., “none” or “any”) or personal pronoun; we insert “the” if the word appeared within the previous four sentences or if there is a *suggestive* adjective attached to the noun.[5]

Table 4.1 shows that the classifiers perform better on automatically generated data, but also perform well on the manually annotated data. Prediction of articles is primarily dependent on the lemma and the tag of the node. In predicting the prepositions and subordinating conjunctions, the node’s functor is the most critical factor. It turns out that the rules are not reliable for the preposition and subordinating conjunction model.

Table 4.3 presents a confusion set from the best article classifier on the de-

[5]Any adjective that is always followed by the definite article in the training data.

% Errors	Reference→Hypothesis		
41	the	→	NULL
19	a/an	→	NULL
16	NULL	→	the
11	a/an	→	the
11	the	→	a/an
2	NULL	→	a/an

Table 4.3: Article classifier errors on development data.

Manual		Automatic	
Det.	P & SC	Det.	P & SC
85.53	89.18	85.31	91.54

Table 4.4: Accuracy of best models on the evaluation data.

velopment data. Our model is relatively conservative, incurring 60% of the error by choosing to insert nothing when it should have inserted an article. The model requires more informed features as we are currently being overly conservative.

In Table 4.4 we report the overall accuracy on evaluation data using the model that performed best on the development data. The results are consistent with the results for the development data; however, the article model performs slightly worse on the evaluation set.

4.4.2 Reordering Results

Evaluation of the final sentence ordering was based on predicting the correct words in the correct positions. We use the reordering metric described in [26] which computes the percentage of nodes for which all children are correctly ordered (i.e., no credit for partially correct orderings).

Table 4.2 shows the reordering accuracy for the full model and variants where a particular feature type is removed. These results are for ordering the correct auxiliary-inserted TGTSs (using deep-syntactic functors and the correctly inserted auxiliaries). In the model variant that preserves the deep order of coordinating conjunctions, we see a significant increase in performance. The child node tags are critical for the reordering model, followed by the child functors.

4.4.3 Combined System Results

In order to evaluate the combined system, we used the multiple-translation dataset in the PCEDT corpus. This data contains four retranslations from Czech to Eng-

Model	Manual	Automatic
TR w/ Rules	**.4614**	**.4777**
TR w/o Rules	.4532	.4657

Table 4.5: BLEU scores for complete generation system for TGTSs (with and without rules applied).

lish of each of the original English sentences in the development and evaluation datasets. In Table 4.5 we report the BLEU scores on development data for our TR generation model (including the morphological generation module). The results for the TR model with the additional rules are consistent with the previous results; the rules provide only a marginal improvement. Finally, we have run the complete system on the evaluation data and achieved a BLEU score of **.4633** on the manual data and **.4750** on the automatic data. These can be interpreted as the upper-bound for Czech-English translation systems based on TR tree transduction.

4.5 Conclusion

We have provided a model for sentence synthesis from tectogrammatical trees. We provide a number of models based on relatively simple, local features that can be extracted from impoverished TGTSs. We believe that further improvements will be made by allowing for more flexible use of the features. The current model uses simple linear interpolation smoothing which limits the types of model features used (forcing an explicit factorization). The advantage of simple models of the type presented in this paper is that they are robust to errors in the TGTSs – which are expected when the TGTSs are generated automatically (e.g., in a machine translation system).

Very similar experiments were performed at the 2002 Johns Hopkins summer workshop. The results reported here are substantially better than those reported in the workshop report [26]; however, the details of the workshop experiments are not clear enough to ensure the experimental conditions are identical.

Another work that is very closely related to ours is that of Ptáček and Žabokrtský [62] who developed a system that generates Czech sentences from the PDT 2.0 trees. The authors used a purely rule–based approach using wide variety of PDT 2.0 features and obtained BLEU score result of **0.477** which is almost the same result we obtained for automatic data. However, we feel that the quality of sentences generated by [62] is substantially better than our results[6]. On the other hand, PDT 2.0 trees are of better quality and contain more information than PCEDT trees we work with.

The Amalgam system also provides a similar model for generation from a *logical form* [14] compared to our model. The primary difference between our ap-

[6]The files with sentences generated by our system are available on the enclosed DVD.

proach and that of the Amalgam system is that we focus on an impoverished deep structure (akin to logical form); we restrict the deep analysis to contain only the features which transfer directly across languages; specifically, those that transfer directly in our Czech-English machine translation system. Amalgam targets different issues. For example, Amalgam's generation of prepositions and subordinating conjunctions is restricted as many of these are considered part of the logical form. Amalgam's reordering model is similar to the one presented here; their model reorders constituents in a similar way that we reorder subtrees.

Both the model of Amalgam and that presented here differ considerably from the n-gram language models of [38], the TAG models of [2], and the *stochastic generation from semantic representation* approach of [76]. In our work, we order the local-subtrees[7] of an augmented deep-structure tree based on the syntactic features of the nodes in the tree. By factoring these decisions to be independent for each local-subtree, the set of strings we consider is only constrained by the projective structure of the input tree and the local permutation limit.

Yet another work similar to ours is that of [37] on the Halogen system. The differences that distinguish their work from ours stem from the type of deep representation from which strings are generated. Although their syntactic and semantic representations appear similar to the Tectogrammatical Representation, more explicit information is preserved in their representation. For example, the Halogen representation includes markings for determiners, voice, subject position, and dative position which simplifies the generation process. We believe their *minimally specified* results are based on input which most closely resembles the input from which we generate in our experiments.

[7]A local subtree consists of a parent node (governor) and its immediate children.

Chapter 5

Tree Searching/Rewriting Formalism

In this chapter we aim to introduce a Tree Searching/Rewriting Formalism (TSRF) and its implementation. Originally, we developed the formalism for searching complicated linguistic phenomena in the tectogrammatical trees [41] and then we extended it to a rewriting formalism for the purpose of converting TGTSs to analytical trees as described in Chapter 4 (however, this approach did not prove useful).

The formalism recognizes rules whose left side specifies a forest of subtrees to be found within a tree by imposing a set of constraints encoded as a query formula. The optional right side then contains respective substitutions for the found subtrees (or may be omitted in the searching mode). The searched structures have to be rooted trees whose nodes may be labeled by a set of $(attribute - value)$ pairs.

There are several reasons for the development and implementation of such formalism. First, it can serve as a corpus searching tool allowing linguists to search for relevant linguistic phenomena. Second, its rewriting capabilities provide an elegant and intuitive way to perform rule-based tree transformations. Moreover, the rules can be reversed, i.e. given a rule that transforms tree A to tree B, it is possible to determine a rule that transforms B back to A.

An interpreter for the proposed formalism is fully implemented.

The chapter is organized as follows: Section 5.1 briefly introduces the formalism, Section 5.2 describes its searching part and Section 5.3 explains the substitution process. Section 5.4 lists several examples of rules recognized by TSRF. Section 5.5 discusses the implementation issues and Section 5.6 concludes the chapter.

5.1 Introduction

The proposed formalism is designed to allow searching for a specified forest of subtrees within queried trees. In the substitution mode, the found matches are then replaced by specified substitution trees. The nodes of the queried trees may be labelled by a set of *(attribute,value)* pairs[1] (such as e.g. TGTS trees).

In the following text we take a tree to be of the form (V, E), where V is the set of vertices[2] and E is the set of edges. Additionally, we will denote $\overline{V_v}$ and $\overline{E_v}$ the set of all vertices and edges, respectively, within the subtree whose root is $v \in V$.

In the following we will describe TSRF in the substitution mode. In the search mode, TSRF works the same way, but it accepts only the left side of a rule (query formula) and yields the list of all the matches.

TSRF operates on a list of rules (a *program*) that are sequentially applied on a given tree. Schematically, a substitution rule looks as follows

$$F- > [S_1, S_2, ..., S_n],$$

where F is a query formula (see Section 5.2) containing (among other predicates) positive, i.e. non-negated, occurrences of structural predicates specifying subtrees (*templates*) T_1, T_2, ..., T_n to search for and S_1, S_2, ..., S_n are the respective substitution trees (*substitutions*) for the found subtrees. The template vertices are associated with actual tree vertices after a successful match. Substitutions represent new tree structures on the matched vertices.

Each vertex v of a template or substitution is assigned a *label* $l(v)$. l must be injective for the set of all template vertices (i.e. each vertex $v \in T_i, 1 \leq i \leq n$, is assigned a unique label). S_i, the corresponding substitution, then defines a new tree structure by means of these labels. Additionally, for the substitution vertices l may introduce a new label (node adding), a single template label may be repeated (node copying), or a template label may be omitted whatsoever (node drop).

5.2 Query Formula

In this section we will describe the query formula, the searching part of the formalism, in detail.

As mentioned in the previous section, the purpose of the query formula is define a set of templates T_1, T_2, ..., T_n to be searched for. Each such template is defined by one non-negated occurrence of the structural predicate (described in the next subsection). The query result is the set of all matches of these templates that satisfy the query formula (see Section 5.4 for examples of complete query formulae).

The query formula is a propositional formula consisting of standard logical operators (and, or, not) that can be arbitrarily nested (i.e. unrestricted use of parentheses is supported). Its atoms are the predicates described in the following subsections.

[1]Note that it is straightforward to convert e.g. edge-labelled trees to this representation as the label on an edge may be viewed as a value of a (special) attribute of the child node this edge leads to.

[2]For brevity reasons, we will also assume that each $v \in V$ contains its labelling as well and that exactly one $r \in V$ is designated as the root of the tree.

The arguments of some of the predicates (e.g. attribute value or absolute position test) can be variables that are then subject to the standard unification procedure over the entire query formula.

From these facts a very important point regarding the overall expressive power of the query formula follows: because an occurrence of the structural predicate can viewed as an implicit existential quantifier on the vertex variables this predicate contains and because this occurrence can be arbitrarily nested and/or negated in the query formula, we in fact get the full power of first order logic over vertex variables, not just its existential fragment. (This follows from the fact that for any formula of first order logic there exists an equivalent formula without universal quantifiers.)

5.2.1 Structural predicate

The structural predicate specifying a template is of the form

$$(VertexVar, VertexConditions, SubTrees),$$

where $VertexVar$ is an (arbitrary) vertex variable identifying the template vertex, $VertexConditions$ is a propositional formula imposing conditions on the vertex, and $SubTrees$ is a list of other structural predicates specifying the children vertices of this vertex, i.e., the structural predicate is fully recursive on the members of $SubTrees$.

The propositional formula ($VertexConditions$) consists of arbitrarily nested standard logical operators (and, or, not) and the following predicates-atoms:

- Attribute value test

  (Attribute Operator Value)

  where $Operator \in \{=, =\sim, \neq, \neq\sim, <, >\}$ and $Value$ is either a number, a string, or a variable. The last option allows for the unification of attribute value variables throughout the entire query formula. $=\sim$ ($\neq\sim$) allows to test match (mismatch) against the given regular expression.

- Children count test

  $$(\#\ \ Operator\ Number),$$

  where $Operator \in \{=, <, >\}$

For example, the structural predicate specifying a template consisting of a conjunction (a) that coordinates two nouns (its children, b and c) that share the grammatical number would look as follows:

(a,func = 'CONJ', [(b, tag =∼ 'N' and number = X, []), (c, tag =∼ 'N*' and number = X, [])])*

5.2.2 Path predicate

The $path$ predicate allows to impose constraints on the path between two nodes - $Start\_VertexVar$ and $End\_VertexVar$ - specified in the structural predicates. It is of the following form

$$path(Start\_VertexVar, End\_VertexVar,$$

$$Segments, VertexConditions),$$

$Segments$ is the list of respective path segments. Each segment specifies the "movement" within the tree with respect to the previously visited node (which is $Start\_VertexVar$ if this is the first segment, otherwise it is the last node of the previous segment). The form of a segment is as follows:

$$(Distances, Direction),$$

where $Distances$ is a list of intervals specifying the possible number of steps-nodes in the given direction. $Direction$ is either up (towards the parent), $down$ (towards a child), $left$ (towards the nearest node on the same tree level to the left), $right$ (towards the nearest node on the same tree level to the right). For example,

$$[((1 - inf), right), ((0, 2), up))]$$

denotes a two segment path, i.e., one node lies between $Start\_VertexVar$ and $End\_VertexVar$: it is located on the same level and to the right of $Start\_Ver-texVar$ and $End\_VertexVar$ is either this node itself or its grandfather.

$VertexConditions$ specify the attribute values constraints imposed on the nodes of the segment and are identical to attribute tests in the structural predicate (see the previous subsection).

5.2.3 Other predicates

There are several other predicates present:

- $testAttribute(VertexVar, Attribute, Operator, Value)$
- $testChildrenCount(VertexVar, Operator, Value)$
- $testDistance(VertexVar_1, VertexVar_2, Attribute, Operator, Value)$

The first two predicates correspond to their counterparts in the structural predicate. This way, however, they can appear independently in the query formula, thus increasing the expressive power of the formalism. For instance,

$$(a, , []) \ and \ (b, , []) \ and \ (testAttribute(a, [attr =' value'])$$

$$or \ testChildrenCount(b,' >', 1))$$

allows to search for nodes a and b, where either $a \rightarrow attr$ equals to $value$ or b has more than 1 child. This would be impossible to express with only the structural predicates at hand.

$testDistance$ allows to compare the difference in numerical attribute $Attri-bute$ of the specified nodes $VertexVar_1$ and $VertexVar_2$. This predicate compensates to some extent[3] the impossibility to use arithmetic expressions within predicate calls. The predicate holds if the difference is in the $Operator$ relation to the $Value$. For example,

$$testDistance(a, b, 'sentord', =, 1)$$

tests whether the difference in $sentord$ attribute between nodes a and b is 1 (for a TGTS, this tests whether the words represented by the two nodes lie next to each other in the sentence).

5.3 Substitution Process

In this section we will describe the substitution process, the optional substitution part of the formalism, in detail. Throughout this section we will provide examples based on the situation shown in Figure 5.1 for easier understanding.

Let $Q = (V^Q, E^Q)$ be a tree. Let R be a rule that is to be applied on Q, let T_1, T_2, ..., T_n be the respective templates and S_1, S_2, ..., S_n the corresponding substitutions. We will call any set of nodes that form a matching subtree for T_i within Q a *match* for T_i. Additionally, we will call the set of assignments of the vertices of template T to the vertices of a match M for $T = (V^T, E^T)$ the *map* m_{TM} between T and M. Let $val_{m_{TM}}(v), v \in V^T$, be a function that returns the matching node for v according to m_{TM}.

R is applicable on Q if and only if the following conditions hold [4]:

1. For each $v \in V^Q$, v appears at most in one match from the set of all matches for the templates that are to be altered (i.e. the templates whose corresponding substitution is not identical to the templates themselves).

2. Let S_i be a substitution containing a vertex v such that $l(v) = l(w)$, $w \in T_j, i \neq j, 1 \leq i, j \leq n$ (i.e. v is defined in a *foreign* template T_j). Then T_j has only a single match in Q or T_i has no match in Q.

These conditions[5] on applicability ensure that the substitution process may be carried out as intended.

[3]In our experience, all the linguistically relevant queries over PDT that require a numerical operation are limited to the comparison of attribute value difference.

[4]If R is not applicable on Q, it is ignored.

[5]Note that both of them represent "run-time" conditions - their fulfillment depends on not only on the rules themselves but also on the queried tree. It is the responsibility of the user to create meaningful rules that are applicable on all the processed trees.

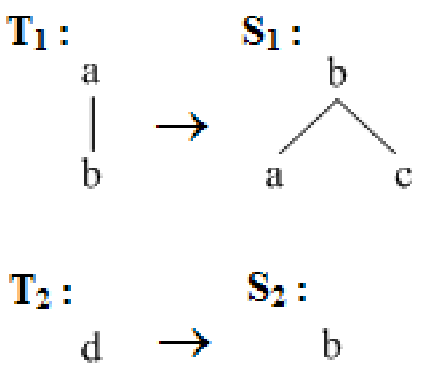


Figure 5.1: Example of a substitution rule featuring two templates T_1 and T_2 and the corresponding substitutions S_1 and S_2

The first one prevents a part of the tree to be subject to "multiple substitution" by specifying that the respective matches do not overlap (it is harmless for a template whose substitution is identical to it). Figure 5.2 illustrates such a malformed situation: M_1 and M_2 are two matches for the template T_1 from example shown in Figure 5.1. Node 2 is contained in both matches and, as the substitution for S_1 is not identical to the template T_1, its application is not well defined at all.

The second condition ensures that a substitution featuring foreign template vertices is defined unambiguously. For the example shown in Figure 5.1 this requires that either T_1 has only a single match or T_2 has no match within the queried tree as the node b appears also in S_2 (T_1 is a foreign template with respect to S_2).

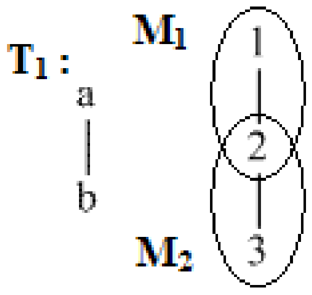


Figure 5.2: Example of an unapplicable rule.

If R is applicable the following substitution step is performed for each match M of each template $T = (V^T, E^T) \in \{T_i, 1 \leq i \leq n\}$:

Let $S = (V^S, E^S)$ be the substitution associated with T. Let us define the partitioning of V^S into the set of nodes occurring in the corresponding template (V^S_O), the set of nodes occurring in foreign templates (V^S_F) and the set of new nodes (V^S_N):

$$V_O^S = \{v \in V^S : \exists w \; ; \; w \in V^T \; \wedge \; l(v) = l(w)\}$$

$$V_F^S = \{v \in V^S : \exists T', w \; ; \; T \neq T' \; \wedge \; w \in V^{T'} \; \wedge$$
$$l(v) = l(w)\}$$

$$V_N^S = V^S - V_O^S - V_F^S$$

For the example shown in Figure 5.1, node $c \in S_1$ is a new node and node $b \in S_2$ is a node from the foreign template T_1, nodes $a \in S_1$ and $b \in S_1$ lie in their corresponding template T_1.

Let us now define the graph G that is induced by S. Informally, G consists of the copies of match nodes connected according to S (with the values of the specified attributes rewritten), copies of all the descendants of these nodes in Q (except for those that are already in some match for T) and the added nodes (according to S).

Formally, let $c_v(u), v \in V^S$, be a vertex *copy* function that returns a vertex with the same valuation as u, v is a substitution vertex that induces the copy[6]. We denote $\{c_v(u) : u \in V\}$ as $c_v(V)$, where V is a set of vertices. Similarly, $c_v(E)$ is $\{(c_v(u), c_v(w)) : (u, w) \in E\}$, where E is a set of edges. Let us introduce two more abbreviations $val(v)$ and $c(v)$:

$$val(v) = val_{m_{TM}}(v) \Leftrightarrow v \in V_O^S$$
$$val(v) = val_{m_{T'M}}(v) \Leftrightarrow v \in V_F^S$$

$$c(v) = c_v(val(v)) \Leftrightarrow v \in V_O^S \cup V_F^S$$
$$c(v) = c_v(v) \Leftrightarrow v \in V_N^S$$

where T' is the one foreign template in which v occurs. Then S induces the following subtree $G = (V^G, E^G)$:

$$V^G = \bigcup_{v \in V_N^S} c(v) \cup \bigcup_{v \in V_O^S \cup V_F^S} c_v\Big(\overline{V_{val(v)}^Q} - \bigcup_{u \in (V_O^S \cup V_F^S) - \{v\}} \overline{V_{val(u)}^Q}\Big)$$

$$E^G = \{ \; (c(v), c(u)) : (v, w) \in E^S \; \} \; \cup \bigcup_{v \in V_O^S \cup V_F^S} \Big(c_v(\overline{E_{val(v)}^Q}) \; - \; \{ \; (u, c_w(w)) :$$
$$u \in c_v(\overline{V_{val(v)}^Q}) \wedge w \in V^S\}\Big)$$

[6]We index c function(s) by this vertex in order to be able to identify the copy unambiguously.

Additionally, the valuation of each vertex $val(v), v \in V^S$, is changed according to the labelling of v so that the values of the specified attributes are rewritten (the values of other attributes remain unchanged).

The substitution process then consists in removal of M from Q and attachment of G (if the root of M is identical to the root of Q then the updated tree is G itself).

Figure 5.3 presents an example of a tree transformation via the rule shown in Figure 5.1. M_1 and M_2 are the only matches of T_1 and T_2 respectively. Node $3'$ denotes a copy of node 3 and new is a new node (corresponding to template vertex c).

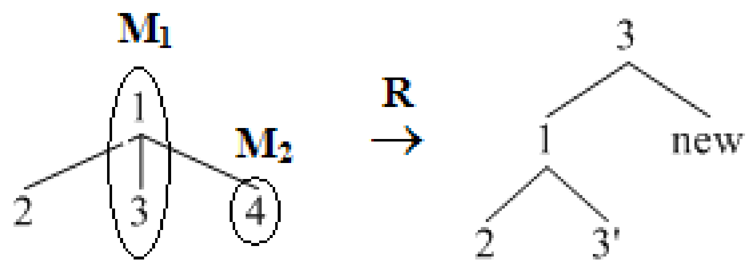


Figure 5.3: Example of a tree transformation via rule R shown in Figure 5.1.

5.4 Test Examples

This section provides a few examples of the replacement rules. These examples have been selected because their query parts (left side of the rules) represent relevant types of linguistic queries and because they well demonstrate the possibilities of TSRF. We have also used these types of substitution rules in the experiments with machine translation. Additionally, we will test the performance of the implemented software tool on these examples (see Section 5.5).

1. Copy the value of $form$ to $lemma$ for each preposition or conjunction (tag begins with *R* and *J* respectively).

   (a, tag =∼ '[RJ]' and form = X, []) →*
   (a, lemma = X, [])

2. Search for a template consisting of an auxiliary word ($afun$ begins with *Aux*) and its daughter - a locative adverbial ($func = LOC$). Swap the two nodes and mark the auxiliary as hidden:

   (a, afun =∼ 'Aux', [(b, func = 'LOC',[])]) →*
   (b,,[(a,TR = 'hidden',[])])

3. Delete all subtrees whose root is a noun and which do not contain an adjective that 1) matches the gender or the number of the noun and 2) precedes the noun at most in two positions in the linear surface order (value of $sentord$ attribute). Two templates are needed[7] as the adjective is not necessarily a direct dependant of the noun.

   (a, tag =∼ 'N' and gender = X and*
   number = Y, [ ]) and not
   ((b, tag =∼ 'A' and (gender = X or number = Y), [ ]) and*
   testDistance(a,b,'sentord',1-2) and path(b,a,[vu,1-INF])) → ()

4. Repair a possibly incorrect parsing result: a noun phrase, being a dependant of the main predicate ($func = Pred$), contains a temporal determination ($func = TWHEN$) which is more likely to be a modifier of the main predicate.

   (a, func = 'Pred', [ ]) and (b, tag =∼ 'N', [ ]) and path(b,a,[vu,1-INF]) and*
   (c, func = 'TWHEN', [ ]) and path(c,b,[vu,1-INF]) → (a,,[(c,,[ ])]),(b,,[ ]),()

5.5 Implementation

In this section we present the software tool that implements the described formalism. First, we will describe some of its basic characteristics, then we discuss the complexity issues, and finally we will present performance results of the implemented tool.

The tool was developed in Mercury[8] programming language. Mercury is a declarative language similar to Prolog but it goes above first order logic and provides a strict type system. Moreover, the Mercury compiler first translates the code into the programming language C and then compiles it as standard C code. This generates fast running code. Many optimizations, especially those connected to the backtracking backbone, are thus performed by Mercury.

The tool was implemented to process PDT 1.0 style trees and currently features no support for PDT 2.0 style trees. However, the tool relies on its own tree representation so only the conversion routines have to be changed in order to incorporate PDT 2.0 data. Appendix D provides more information regarding the software.

5.5.1 Algorithm

The algorithm used to process TSRF queries is quite simple. The query formula is being evaluated (with backtrack to get all the possible matches and variable instantiations) and the truth value of each predicate atom is tested. The free and ground

[7]Note that only the first template is subject to substitution as the second is negated.
[8]http://www.cs.mu.oz.au/research/mercury

variables from the already evaluated predicates are passed through and upon each successive predicate test these might be subject to unification.

When a structural predicate occurrence is being tested, i.e., there is an attempt to find a match for the corresponding template within the queried tree, the tree is traversed with backtrack starting with the root of a template.

All the possible matches (and variable instantiations) that fulfill the query formula are then collected and returned. If the tool is used in the substitution mode, the respective substitutions are then performed on the returned matches (providing the rules are applicable).

The nodes of the tree to be queried are first indexed so as to minimize the access time when evaluating the respective predicates (mainly the structural predicate) in the subsequent search run. There is large number of other optimizations (also for the optimal performance of the tests of respective designed predicates) but essentially the search run is optimized for a single tree (no optimizations are made for the entire treebank in advance).

5.5.2 Complexity Issues

Let us discuss the computational complexity of the respective parts of the algorithm presented in the previous section.

The initial node indexing is performed once for each node, its time complexity is thus linear. The query formula is being evaluated in exponential time with respect to the number of occurrences of structural predicates contained in it.

The structural predicate test runs generally also in exponential time (searching for a subtree within a tree is known to be NP-hard), but not independently of the query formula evaluation - each successive match found by the structural predicate test run leads to a backtracking step of the query formula evaluation. The path predicate test runs in $O(k * n)$, where k is the number of segments specified by the predicate and n is the number of nodes of the queried tree. All other predicates listed in Section 5.2 run in constant time.

As each node of the queried tree is subject to at most one substitution (applicability condition), all the substitutions are performed in linear time with respect to the queried tree size.

In summary, the entire algorithm runs in exponential time (which cannot be avoided as there are in general exponentially many subtrees within a tree).

5.5.3 Performance Results

We have chosen the queries from Section 5 to measure the computation time over the PDT corpus. The primary purpose of doing so is to ensure that even such a large treebank as PDT (containing cca 50000 TGTS) can be queried in an acceptable time. The tests were performed on AMD Athlon 64 3800+, 1 GB RAM, running Windows XP. We have divided the tectogrammatical trees of PDT into three groups according to the number of their nodes. The resulting average computation times per tree from the given size range are listed in Table 5.1.

nodes	**1**	**2**	**3**	**4**
0 - 9	0.4	0.4	0.6	0.4
10 - 20	1.1	1.0	2.1	1.0
> 20	2.1	2.1	4.9	0.20
any size	1.2	1.1	2.5	1.1

Table 5.1: Average computation time per tree in milliseconds.

These results show that the tool is able to perform the relevant queries in acceptable times even for large corpora, in our case in the order of minutes for the entire PDT. Moreover, the results are being retrieved sequentially so the user does not have to wait till the search is complete.

5.6 Conclusion

The expressive power of presented formalism is able to capture complex linguistic structures in the tree structures and is comparable to the currently most advanced tools available. However, there are still structures which cannot be captured (such as two templates connected by a potentially infinite set of vertices with properties that cannot be expressed by the $path$ predicate). Likewise, there are substitutions that cannot be performed (such as reversion of potentially infinite path within a tree).

The implemented tool has shown acceptable performance which makes it possible to use it to process even large data sources. However, many optimizations (such as indexing of the entire corpus in advance) can yet be performed.

To our best knowledge, there is no similar tool that could be straightforwardly used both as a searching and rewriting tool. However, there is a large number of treebank searching tools, e.g. CorpusSearch [68], ICECUP III [81], TGrep2 [70], TIGERSearch [35], VIQTORYA [32], and Finite Structure Query[34]. Specifically for PDT querying purposes, NetGraph [49] tool was previously developed. All these tools implement some form of predicates for basic tree relations between nodes. As far as their overall expressive power is concerned, they rank from the most restricted ones featuring only limited possibilities to combine respective constraints by logical operators (CorpusSearch, ICECUP III, NetGraph) via more general ones (TGrep2, TIGERSearch, VIQTORYA) up to those that use the full power of first order logic (Finite Structure Query). Additionaly, some of these tool can be used to query more general structures than strict trees (VIQTORYA, Finite Structure Query).

We will compare the presented tool to Finite Structure Query, the tool featuring the most powerful query expressiveness so far, and NetGraph, the tool directly designed to query PDT treebank, in greater detail.

TSRF is comparable to the query language used by Finite Structure Query. Although TSRF does not use overt quantification (and is therefore not strictly first order logic based), for any first order logic formula (over vertex variables) there exists an equivalent formula that can be expressed by TSRF. However, TSRF is much less powerful than Finite Structure Query in terms of generality of the queried structures - TSRF operates only on strict trees whereas Finite Structure Query can operate on an arbitrary finite structure. The expressive power of a formalism depends of course also on the set of supported predicates. We believe that in this aspect the set of predicates present in TSRF at least matches the set offered by Finite Structure Query, we were at least able to express all the examples presented at [34] in the TSRF query formula[9].

NetGraph uses its own form of query formula. This formula in fact directly represents an underspecified tree template rather than being a logical formula containing predicates. As described in [49] , NetGraph is obviously less powerful than TSRF as its query formula (when stated in logical terms) features only a restricted disjunction and no negation at all (thus it forms only a positive existential fragment). However, NetGraph was recently substantially improved [48] as it developed a powerful functionality within its underspecified template (e.g. a specific type of negation, a form of attribute value unification etc.).

In summary, we see the contribution of the presented work mainly in the expressiveness of the query formula, in the elegant and intuitive way the rules are written (and their easy reversibility), and in the performance of the implemented tool.

[9]For example, we are not sure whether unification of attribute value variables supported by TSRF is supported by Finite Structure Query,too.

Chapter 6

Using Temporal Information for Text Generation

In this chapter we aim to enhance the English generation system described in Chapter 4 by the results of automatical analysis obtained from the Czech trees. Section 6.1 discusses the theoretical background of English tense generation. The experimental results are then described in Section 6.2. Chapter 6.3 concludes the chapter.

6.1 Tense Generation

We will begin the discussion with the account on what aspects of English tense cannot be determined (Section 6.1.1) using only information on event ordering and then turn to issues where this information might help (Section 6.1.2).

6.1.1 Unresolvable Tense Issues

Simple vs Perfect

It seems clear that the event ordering information alone does not allow for determination of perfect or simple tense (with the partial exception of past perfect discussed below). The difference between these tenses does not consist in the time placement of these events. Consider the following pairs of sentences:

John did his homework (yesterday).
John has done his homework.

John had done his homework before his mother returned.
John did his homework before his mother returned.

John will do his homework in the afternoon.

John will have done his homework by the afternoon.

The relative ordering of events and time–of–speech points is the same in each pair of sentences (although in case of future perfect it is possible that John had already made the homework but then the utterance is rather misleading).

Although all the three tenses (past, present and future perfect) express or emphasize completion of an event, each of the tenses requires different set of conditions to be fulfilled before the proper tense can be used (such as the condition of recentness for present perfect), see [67] for detailed account on English tenses. According to Reichenbach [69] the perfect and simple tenses differ in the position of their respective *reference time* which is the time a given event is related to. For example, in past simple, the reference point is identical to the time of the event whereas in present perfect it is simultaneous with the time–of–speech point, i.e., the event is viewed from the perspective of the time–of–speech point. However, an automatic determination of the reference time is difficult.

To be able to make a decision on whether a perfect tense should be used, the tectogrammatical representation would have to contain a grammateme for perfectiveness [1] (as even the authors of the TR admit in [27]) which is not present in any of the current corpora annotated on tectogrammatical layer.

Continuous Tenses

The event ordering information is also not sufficient for the decision on the continuous (extended) tense. The continuous tense may be used to report an ongoing activity as in

Jack is playing football.

or to indicate repetition as in

Women are wearing larger hats this year. (Reichenbach [69])

or, generally, to stress the progressive nature of an event. Unfortunately, an ongoing event (state) can be expressed by a non–continuous tense as well:

He never takes any chances.

Therefore, knowing that an event is simultaneous with the time–of–speech and is non–iterative does not suffice for the decision on the correct variant. The same holds true for past and future tenses

[1]The *aspect* grammateme may contain the value *"resultative"* which captures explicit perfective constructions in Czech. However, this rare case represents only a fraction of cases where a perfect tense would be used in English.

He was (will be) running. He ran (will run).

On the other hand, a repetition can be (and usually is) expressed also by a non–continuous tense:

I play football whenever I have time.

so, again, the knowledge that an event is iterative, i.e., part of a plan (see Section 2.2.3), therefore does not help to determine the right variant (sometimes both variants are possible and the difference is very little).

6.1.2 Tangible Issues

There are two situations in which one would expect that the information on event ordering would help to generate the correct English tense: determination of past perfect tense and determination of the correct tense in indirect speech.

Past Perfect Tense

The past perfect tense has a special position among English perfect tenses – in addition to its perfective aspect (which we cannot capture, see Section 6.1.1) it also expresses precedence of an event in the past to another past event as in

Mary looked around but John had left.

Using past perfect is usually optional and there is rarely an occurrence of this tense in our corpus. On the other hand, past perfect conveys more information than simple past so it might be desirable to generate past perfect whenever possible for for certain applications.

As we are unable to automatically determine the reference time (in Reichenbach's sense) to use as an anchor for generating past perfect, we attempt to generate this tense for a verb V if and only if there is a verb W such that

1. the event expressed by W precedes time–of–speech of the given utterance
2. the event expressed by W follows the event expressed by V
3. either W is a descendant of V or V is a descendant of W in the given TGTS
4. W lies in the vicinity of V – there is no verb U that lies between V and W in the surface order

Indirect Speech

In English, a tense present in a sentence is changed when this sentence appears in indirect speech (as a content clause) introduced by a speech verb in past simple.

Original tense	**Indirect speech**
present simple	past simple
present perfect	past perfect
past simple	past perfect
future tenses	past with *would*

Table 6.1: Indirect speech tense transition. Continuous tenses remain continuous. Other tenses not listed in the table do not change.

The overview of the transition rules is shown in Table 6.1. Some of these transitions are often optional or even improper:

Past simple often does not have to be changed to past perfect. Moreover, the transition must not violate perfective nature of the matrix verb in the indirect speech. If so, the indirect speech retains past simple.

Present simple does not have to be changed to past simple if the content conveyed by the indirect speech is believed to be true even in the time–of–speech of the reporting utterance.

The knowledge of the relation between a speech verb introducing an indirect speech and the matrix verb of the indirect speech allows for the determination of the correct tense. Again, as we cannot reliably determine perfect and continuous tenses, we attempt to generate only simple tenses.

6.2 Empirical Evaluation

We have tried to verify whether the information about relative ordering of events really contributes to determination of correct tense as discussed in Section 6.1.2. Firstly, we have linked the corresponding Czech and English trees as described in Section 6.2.1. Then we have used the manual temporal annotation to test the respective methods. The results are listed in Section 6.2.2. Because, as discussed later, the numbers themselves are hardly informative, we have not even attempted to provide similar analysis for the output of the automatic system and to compare the results. Given the number of verb instances affected by the described rules, such comparison would be insignificant. Instead, we try to discuss the counterexamples to draw some linguistic conclusion out of it.

Although we focus on the analysis of performance of the event instances affected by rules described in Section 6.1.2 we also provide generation results for all linked verbs – the verbs unaffected by any of our rules are assigned simple past, present and future tense based on the tense of the associated Czech counterpart.

6.2.1 Data Preparation

To be able to test the potential gain of the event ordering information (determined on the Czech side) for the English generation component, it is necessary to link the corresponding Czech and English TGTSs, i.e., to pair the tree nodes that represent events (other nodes are not relevant). Ideally, this would be done automatically given a Czech-English dictionary (such as the probabilistic dictionary included in the PCEDT distribution). However, a fully automatic assignment is not possible due to

- ambiguities (i.e. a sentence to be linked may contain several identical or similar lexical items that all match a plausible translation of an item in the source sentence)
- the fact that the translation of an event may be so highly context–specific that it does not correspond to a usual equivalent translation
- low coverage of the used dictionary

Note that it is also not possible to convey all the temporal information from the Czech side. The mapping is not one–to–one as the translation is not literal and some events expressed in the English text may be omitted on the Czech side and vice versa.

We have therefore decided on a semi-automated approach. Firstly, the preliminary mapping is created by a simple but highly accurate automatic assignment algorithm which only links a Czech verb to the English verb that is the only plausible candidate according to the probabilistic dictionary. This algorithm was able to correctly link approximately 40% of the links and produce report on the remaining ambiguities and unmatched instances which were subsequently processed manually to yield the final mapping. We have only used manual (both Czech and English) TGTSs corresponding to the development and evaluation testing set, respectively.

The correct tense of verbs within the testing sets was determined in advance by an automatic procedure based on the presence of the relevant auxiliaries. Additionally, because of the rare occurrence of past perfect in the corpus, for the verbs in the past tense in the vicinity of other past tense verbs it has been indicated whether they could also appear in past perfect in the given context. This annotation was performed by a native American English speaker[2].

6.2.2 Results

Having the mapping between Czech and English TGTSs described in the previous section at hand, we can transfer the event ordering information to events on the English side.

[2]David McClosky

	Instances	Correct	Accuracy	Bl. Corr.	Bl. Accur.
IS Past	17	13	76%	12	71%
IS Present	15	0	0%	15	100%
IS Future	4	3	75%	2	50%
Past Perfect	5	4	80%	2	40%

Table 6.2: Performance of the respective methods in tense generation for affected instances only. *“Bl.”* stands for the baseline.

	Instances	Correct	Accuracy	Bl. Corr.	Bl. Accur.
DTest	337	289	85.8%	286	84.9%
ETest	362	306	84.5%	309	85.3%

Table 6.3: Performance of the respective methods in tense generation for all linked verbs. *“Bl.”* stands for the baseline.

Table 6.2 lists the performance of the respective methods on the development testing set. IS Past (Present, Future) stand for the generation of tense of the main verb in indirect speech that precedes (is simultaneous, follows) the verb introducing the indirect speech. As described earlier, baseline results are obtained simply by generating simple past, present and future based on the tense of the Czech counterpart.

It is evident that the results are not statistically significant which is also demonstrated by the evaluation on the evaluation testing set (33 instances): The combined method generating past perfect and indirect speech in past and future (the methods that performed well on the development testing set) yielded **66%** (22 correct instances) accuracy whereas the performance of the baseline was **75%** (25 correct instances).

Table 6.3 shows the results for all verbs that could be linked to their respective Czech counterparts. Rules affect only the instances listed in Table 6.2, the outcome for the remaining instances is set as identical to the baseline.

Apparently, the evaluation brings no decisive results but it suggests that our rules are not reliable. Let us discuss the performance of the respective methods in detail to discover the counterexamples to our rules.

Indirect Past Speech

Although using past perfect for the main verb in indirect past speech yielded negligibly better result than using past simple, there are counterexamples in which the transition to past perfect was deemed inappropriate by a native speaker, e.g.[3]

Kaufman & Broad Home Corp. said it ***formed (had formed$*$)*** *a $53.4 million limited partnership subsidiary to buy land in California suitable for residential development.*

It seems the event lacks the "perfective" nature which apparently has to be retained even in the indirect speech transition.

Indirect Present Speech

As shown in Table 6.2, all the instances of indirect speech originally in the present tense remain in the present tense, for example

Reached in Honolulu , Mr. Shidler said that he ***believes (believed$*$)*** *the various Hooker malls can become profitable with new management.*

This systematic use of present tense in indirect speech seems to be connected to the recentness of the information but it violates to some extent the usual grammatical assumptions.

Indirect Future Speech

Some occurrences of indirect speech in future tense were transformed to "would" clauses correctly whereas other retain the future tense:

Ripples from the strike by 55,000 Machinists union members against Boeing Co. reached air carriers Friday as America West Airlines announced it ***will (would$*$)*** *postpone its new service out of Houston because of delays in receiving aircraft from the Seattle jet maker.*

It seems the future tense can be preserved if the event follows not only the reporting utterance but even the time–of–speech point.

Past Perfect

Our method of generating past perfect has yielded a single counterexample:

Against that backdrop , UAW Vice President Stephen P. Yokich , who recently ***became (had become$*$)*** *head of the union 's GM department , issued a statement Friday blasting GM 's " flagrant insensitivity " toward union members.*

[3]In the presented examples, asterisk denotes a different (possibly incorrect) variant determined by the system.

Again, the context apparently does not grant use of the perfect tense - either because of the nature of the event or because the events are not explicitly related to each other (*“became”* governs the attributive clause related to Mr. Yokich).

6.3 Conclusion

We have tested a preliminary approach to English verb tense generation based on the traditional assumptions regarding English tense system. Although the obtained results are statistically insignificant due to the small number of affected instances, it seems clear that the traditional assumptions often do not hold and that the rules based on them are unreliable. The tested tense generation subsystem cannot improve the text generation system described in Chapter 4. We believe that further research on the automatic analysis of event properties that determine usage of the respective English tenses is necessary.

Chapter 7

Conclusion

We have presented our work in different yet related areas.

In the area of natural language understanding and content representation we have developed a rule–based system of automatic analysis of temporal relations for Czech. The system relies heavily on the recursive temporal principle introduced by Panevová while correcting some of the errors produced by the application of this principle and extending it in various ways namely by the analysis the lexical information provided by time expressions. To capture their meaning, a functional approach has been designed that allows for comparison of time expressions appearing in discourses whose time–of–speech is unknown. The corresponding software tools are implemented. However, the related work shows that a rule–based approach is probably insufficient, in particular, that its recall is too low. As pointed out, there are many types of relations determination of which is impossible by a reliable rule and which therefore remain unresolved in our system. The distribution of the respective variants might be highly context–dependent in real texts making a high–precision determination by using machine–learning techniques possible. For this, a large training temporal corpus is necessary.

Within a machine translation framework, we have developed a generative component that generates English text from the corresponding tectogrammatical trees. The components shows relatively good performance (demonstrated also by the BLEU score) but it can be further enhanced. One way of improving the component consists in implementing the dynamic solution to the problem instead of the presented greedy approach, i.e., to select the best auxiliary insertion–surface order combination instead of selecting the best auxiliary insertion to be reordered. Another possibility is to enrich the reordering model by a language model trained on huge amount of plain text. Yet another way consists in a purely rule–based approach which turned out to be very successful in generation of Czech text from Czech tectogrammatical trees. However, in comparison to the Czech PDT 2.0 data, the manual English data from PCEDT 1.0 are of poor quality with many features missing which can significantly hinder a rule–based approach.

Finally, we have attempted to use the temporal information to improve the generation process but were unable to gain any improvement.

Appendix A

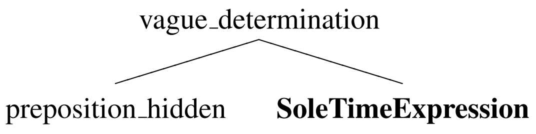
Grammar for Parsing of Time Expressions

This appendix lists the grammar used to parse the time expressions. We do not describe the respective interpretation functions please see the corresponding software module described in Appendix D.

Main Grammar

Start ($InterpretStart$):

SoleTimeExpression $\vee$



vague_determination:

beginning: počátek, začátek
end: konec, závěr

preposition_hidden:

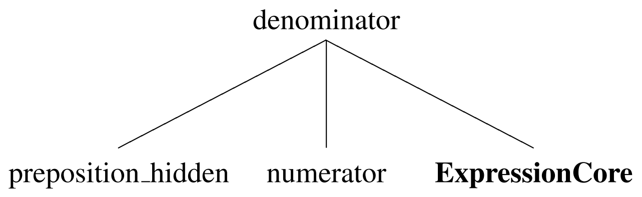
after: po, za
before: před
in: v, k, během, na, o, průběhu
not_before: nejdříve, od, počínaje
not_after: nejpozději, do, konče

beginning: počátkem, začátkem

end: koncem, závěrem

SoleTimeExpression ($InterpretFraction$):

ExpressionCore ∨



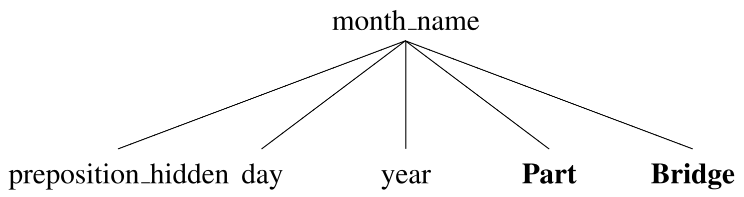
denominator: a fraction numeral[1]

numerator: an ordinal numeral

ExpressionCore:

DateConstructor ∨ **TimeConstructor** ∨ **SingleIndexicalExp** ∨ **AbsoluteSpecification** ∨ **GeneralSpecification**

DateConstructor: ($InterpretDateConstructor$)

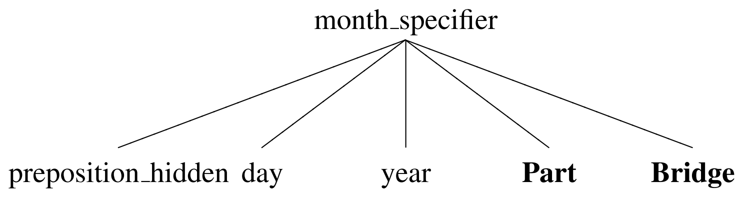


month_name: name of a month

day: an ordinal numeral

year: a cardinal numeral

∨



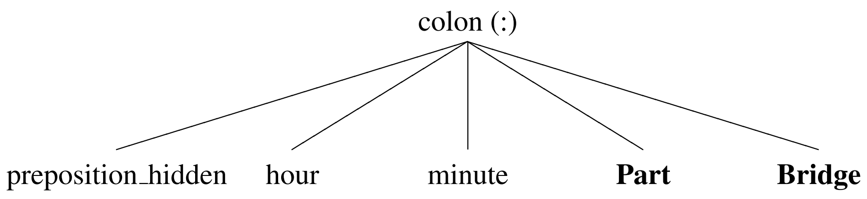
[1]Numeral denotes either a textual or numeric representation of a number.

month_specifier: an ordinal number specifying a month

Part: **SoleTimeExpression**

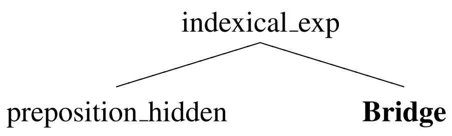
Bridge: a separate clause connected to the time expression

TimeConstructor: $(InterpretTimeConstructor)$



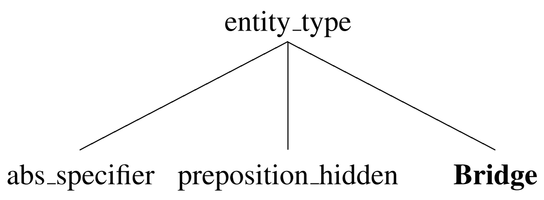
hour: a cardinal number
minute: a cardinal number

SingleIndexicalExp: $(InterpretSingleIndexicalExp)$



indexical_exp:
today: dnes, dnešek, dneska
tomorrow: zítra
tomorrow_tomorrow: pozítří
yesterday: včera
this_year: letos
prev_year: loni, vloni
prev_prev_year: předloni
next_year: napřesrok

AbsoluteSpecification $(InterpretETAbsoluteSpecification)$:
ExpressionCore $\vee$

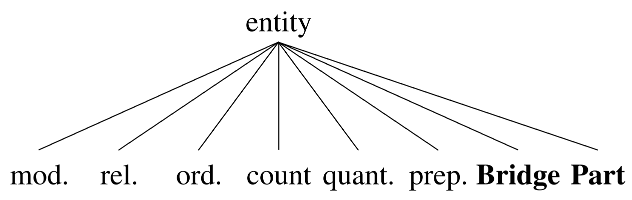


entity_type:

year: rok
century: století
millennium: tisíciletí

abs_specifier: an ordinal numeral

GeneralSpecification ($InterpretGeneralSpecification$):



entity:

millennium: tisíciletí
century: století
decade: desetiletí
(half/quarter)year: (půl/čtvrt)rok
month: měsíc
week: týden
(half)day: (půl)den
(half/quarter)hour: (půl/čtvrt)hodina
(half/quarter)minute: (půl/čtvrt)minuta
year_part: jaro, léto, podzim, zima
week_part: víkend
day_part: ráno, dopoledne, poledne, odpoledne, večer, noc, půlnoc
month_name: name of a month
day_name: name of a day

mod. (modifier):

fiscal: fiskální
school: školní
academic: akademický

work: pracovní

rel. (relative specifier):

next: další, následující, příští, budoucí, zítřejší
next_next: přespříští
this: tento, dnešní, letošní
previous: předešlý, minulý, uplynulý, loňský, předchozí, předcházející, včerejší, loňský
previous_previous: předvčerejší, předloňský

ord. (ordinal specifier): an ordinal numeral
count: a cardinal numeral
quant. (quantifier): an ordinal numeral

every: každý
some: některý

prep. (preposition hidden))

Grammar for Attributes

Start:

indexical_expression ($InterpretSingleIndexicalExp$) $\vee$
entity_name ($InterpretEntityNameAttr$)

indexical_expression:

today_attr: dnešní
tomorrow_attr: zítřejší
yesterday_attr: včerejší
this_year_attr: letošní
prev_year_attr: loňský
prev_prev_year_attr: přloňský

entity_name:

month_name_attr: month name adjective (e.g. *únorový*)
day_name_attr: day name adjective (e.g. *úterní*)
day_time_point_attr: polední, půlnoční
day_time_period_attr: ranní, dopolední, odpolední, večerní, noční
week_time_period_attr: víkendový
year_time_period_attr: jarní, letní, podzimní, zimní

Grammar for Durations

Start ($InterpretDuration$):

$$\begin{tikzcd}
 & \text{entity\_type} \arrow[dl, no head] \arrow[d, no head] \arrow[dr, no head] & \\
\text{modifier} & \text{count} & \text{specifier}
\end{tikzcd}$$

entity_type:

millennium: tisíciletí
century: století
decade: desetiletí
(half/quarter)year: (půl/čtvrt)rok
month: měsíc
week: týden
(half)day: (půl)den
(half/quarter)hour: (půl/čtvrt)hodina
(half/quarter)minute: (půl/čtvrt)minuta

modifier:

fiscal: fiskální
school: školní
academic: akademický
work: pracovní

count:

cardinal: a cardinal numeral
vague count: několik pár mnoho málo

specifier:

in_preposition: za

Appendix B

Estimating Extensions of Functional Compositions

In this appendix we present estimates for the time extensions of the respective time functions. The beam (time extension estimate) consist of the following items:

SL, SU
: Lower and upper estimate of the lowest (leftmost) point of the resulting set of intervals[1] relative to the source input interval (constructed by a constructor) or point variable, i.e., the innermost argument in the functional composition.

EL, EU
: Lower and upper estimate of the highest (rightmost) point of the resulting set of intervals relative to the source input interval or point variable.

ASL, ASU, AEL, AEU
: The absolute time versions of the items described above which are only computed if the information is available. In general, they are set to exact time points using CPAN DateTime package if possible. As their use is similar to their relative estimate counterparts, we do not include them in the description list.

WL, WU
: Lower and upper estimate of the size of the resulting time extension if it is a single interval.

SB, EB
: Indication of whether the lowest and highest point, respectively, of the resulting set of intervals lie on a boundary of an entity such as day, week etc. A value of these items is a set of entities on whose boundaries the given point

[1]In general, a function returns a set of intervals such as $series$ family of functions. However, in most cases we only deal with a single interval or a time point.

lies. It is only present, if it applies for both the lower and upper estimate for the given point.

$start(Interval)$:
$(t\_interval) \rightarrow t\_point$

SL = EL: $Interval \rightarrow SL$

SU = EU: $Interval \rightarrow SE$

WL = WU: 0

SB = EB: $Interval \rightarrow SB$

$end(Interval)$:
$(t\_interval) \rightarrow t\_point$

SL = EL: $Interval \rightarrow EL$

SU = EU: $Interval \rightarrow EU$

WL = WU: 0

SB = EB: $Interval \rightarrow EB$

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

$constM(Millenium)$:
$(t\_int) \rightarrow t\_interval$

$constC(Century)$:
$(t\_int) \rightarrow t\_interval$

SL = SU = EL = EU : 0

WL = WU: size of the constructed entity

SB = EB: the constructed entity

$shift(Point, Distance, InPast)$:
$(t\_point, t\_range, t\_bool) \rightarrow t\_point$

SL = EL: $Point \rightarrow SL + / - Distance$ (depends on the value of $InPast$)

SU = EU: $Point \rightarrow SU + / - Distance$ (depends on the value of $InPast$)

WL = WU: 0

SB = EB: check the resulting point based on the input boundary information

$span(Point, EntityType)$:
$(t\_point, t\_etype) \rightarrow t\_interval$

SL: $Point \rightarrow SL - |EntityType|$

SU: $Point \rightarrow SU$

EL: $Point \rightarrow EL$

EU: $Point \rightarrow EU + |EntityType|$

WL = WU: $|EntityType|$

SB = EB: $EntityType$

$findEntityType(Point, EntityType, Index)$:
$(t\_point, t\_etype, t\_int) \rightarrow t\_interval$

SET (Superior Entity Type): $EntityType$

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

SET (Superior Entity Type): $SupET$

SL: $Point \rightarrow SL + (Index - 1) * |SET|$

SU: $Point \rightarrow SU + Index * |SET|$

EL: $Point \rightarrow EL + Index * |SET|$

EU: $Point \rightarrow EU + (Index + 1) * |SET|$

WL = WU: $|EntityType|$

SB = EB: $EntityType$

Holds for $This = false$, for $This = true$ it has to be changed by 1/-1 for positive and negative indices accordingly.

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

SET (Superior Entity Type): the (basic) entity type to which $Entity$ is equivalent if there is such a type, otherwise it is the lowest entity type that safely (i.e. always) contains $Entity$

LET (Lower Entity Type): the (basic) entity type to which $Entity$ is equivalent if there is such a type, otherwise it is the entity type directly under SET

SL: $Point \rightarrow SL + (Index - 1) * |SET|$

SU: $Point \rightarrow SU + Index * |SET|$

EL: $Point \rightarrow EL + Index * |SET|$

EU: $Point \rightarrow EU + (Index + 1) * |SET|$

Holds for $This = false$, for $This = true$ it has to be changed by 1/-1 for positive and negative indices accordingly.

WL: $|LET|$

WU: $|SET|$

SB = EB: SET if $SET == LET$

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

SL:

$Index < 0$: $Point \rightarrow SL + Index * |SET|$
$Index > 0$: $Point \rightarrow SL + (Index - 1) * |SET|$

SU:

$(Index < 0)$: $Point \rightarrow SU + (Index + 1) * |SET|$
$(Index > 0)$: $Point \rightarrow SU + Index * |SET|$

EL:

$(Index < 0)$: $Point \rightarrow EL + Index * |SET|$
$(Index > 0)$: $Point \rightarrow EL + (Index - 1) * |SET|$

EU:

$(Index < 0)$: $Point \rightarrow EU + (Index + 1) * |SET|$
$(Index > 0)$: $Point \rightarrow EU + Index * |SET|$

WL = WU: 0

SB: boundaries implied by $Entity$

EB: boundaries implied by $Entity$

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

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

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

All the three versions of the $part$ function are evaluated as

$$AppropriateFindVersion(start(Interval), Part, Order, true)$$

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

SL: $Interval \rightarrow SL + Interval \rightarrow WL * N_1/D_1$

SU: $Interval \rightarrow SU + Interval \rightarrow WU * N_1/D_1$

EL: $Interval \rightarrow EL + Interval \rightarrow WU * (1 - N_2/D_2)$

EU: $Interval \rightarrow EU + Interval \rightarrow WL * (1 - N_2/D_2)$

WL: $Interval \rightarrow WL * (N_2/D_2 - N_1/D_1)$

WU: $Interval \rightarrow WU * (N_2/D_2 - N_1/D_1)$

SB = EB: none (sophisticated checks can be implemented)

$partVague(Interval, VaguePart)$:
$(t\_interval, t\_vague\_part) \rightarrow t\_interval$

$start$: $fract_1 = 0$, $fract_2 = 1/2$

end: $fract_1 = 1/2$, $fract_2 = 1$

$middle$: $fract_1 = 1/4$, $fract_2 = 3/4$

SL: $Interval \rightarrow SL + Interval \rightarrow WL * fract_1$

SU: $Interval \rightarrow SU + Interval \rightarrow WU * fract_2$

EL: $Interval \rightarrow EL + Interval \rightarrow WU * (1 - fract_2)$

EU: $Interval \rightarrow EU + Interval \rightarrow WL * (1 - fract_1)$

WL: $Interval \rightarrow WL * (fract_2 - fract_1)$

WU: $Interval \rightarrow WU * (fract_2 - fract_1)$

SB = EB: none (sophisticated checks can be implemented)

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

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

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

SL: $Interval \rightarrow SL$

SU: $Interval \rightarrow SU$

EL: $Interval \rightarrow EL$

EU: $Interval \rightarrow EU$

WL, WU, SB, EB: N/A

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

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

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

SL: $Interval \rightarrow SL$

SU: $Interval \rightarrow SU$

EL: $Interval \rightarrow EL$

EU: $Interval \rightarrow EU$

WL, WU, SB, EB: N/A

$seriesETByCount(Point, EntityType, Count, InFuture)$:
$(t\_point, t\_etype, t\_uint, t\_bool) \rightarrow t\_intervals$

SET (Superior Entity Type): $EntityType$

$seriesIEByCount(Point, Entity, Count)$:
$(t\_point, t\_ie\_name, t\_uint) \rightarrow t\_intervals$

$seriesPEByCount(Point, Entity, Count)$:
$(t\_point, t\_pe\_name, t\_uint) \rightarrow t\_points$

SET (Superior Entity Type): the (basic) entity type to which $Entity$ is equivalent if there is such a type, otherwise it is the lowest entity type that safely (i.e. always) contains $Entity$

SL: $Point \rightarrow SL$

SU: $Point \rightarrow SU + |SET|$

EL: $Point \rightarrow EL + (Count - 1) * |SET|$

EU: $Point \rightarrow EU + (Count + 1) * |SET|$

WL, WU, SB, EB: N/A

Appendix C

Automatic Subfunctor Assignment

In this appendix we list subfunctor values that are automatically assigned to given functors based on the respective preposition compounds containing the listed prepositions (in PCEDT, prepositions consisting of multiple words are merged into preposition compounds).

We have only assigned typical prepositions to a given subfunctor, some subfunctors are not assigned to. Note that the most common preposition are not assigned to any subfunctor[1] as they do not convey a very distinct meaning.

ACMP

incl: including

wout: without

BEN

agst: against

CPR

than: than

wrt: unlike

DIR–2

across: across

along: along

around: round

between: between

[1]There is a technical subfunctor *basic*.

near: near

DIR–3

above: above

behind: behind

below: below

elsew: outside

ext: to_up

near: near

to: to

EXT

approx: round, around

less: less, below

more: more, over

LOC

above: above

along: along

around: round

behind: behind

below: below

between: between

elsew: outside

in: inside

near: near

TWHEN

after: after

approx: round

before: before

between: between

flow: during

Appendix D

Overview of the Implemented Software

In this appendix we provide a brief overview of the structure of the software that implements the experiments described in this thesis. It consists of Perl scripts and relies heavily on using BTred[1] – a Perl engine allowing for reading, traversing and manipulating TGTSs. The software is part of the enclosed DVD.

More detailed information regarding the installation of the software, data description and manipulation, how the software is run etc. is available on the DVD, please see the *README.txt* file in the root directory.

Automatic Analysis of Temporal Relations

Main Scripts

analyse_temporality.btred
: is the main btred macro file that is to be run on the TGTSs representing discourses in which temporal relations are to be determined. It contains the implementation of RTP (see Section 3.3) and uses other libraries for the complex subparts of the analysis such as dealing with time expressions.

eval.btred
: is the evaluation script. It reads the given TGTSs, the reference and hypothesis (output of the analysis) annotation of the corresponding temporal relations and outputs detailed statistics. It uses the *Closure* module to compute transitive closure of the relations contained in the annotations. It also uses auxiliary comparison routines (*FunctionEval.pm*) to decide whether a time expression was (weakly) correctly interpreted (see Section 3.6.1).

[1]http://ufal.mff.cuni.cz/ pajas/tred/btred.html

Library Packages

In this section we briefly describe the content of the respective scripts and libraries and outline the main dependencies.

General Knowledge Libraries

TimeKnowledge.pm
: contains some common sense knowledge about time entities as well as some more specific routines tailored to be used by the time expression reasoning subsystem.

NumeralsKnowledge.pm
: allows for recognition of Czech numerals and their conversion to the numeric representation. It is used by time expression parsing subsystem.

CommonKnowledge.pm
: contains routines related to general knowledge (e.g. inverses of ordering relations).

OntologyInterface.pm
: represents a general interface to an ontology for certain types of queries (e.g. appurtenance of an entity to a class). Currently, the EuroWordNet ontology is the only one installed. Queries to the ontology are posed by the subsystem of detection of animate actors (see Section 3.5.1).

Temporal Annotation Manipulation

Annotation.pm
: contains main routines that allow for the reading of plain text temporal annotation files (using *ExpString.pm*), processing the information (which includes determining the morphological types of the respective points using *TempTypes.pm*) and creating the corresponding temporal relation matrix object (*TempMatrix.pm*).

ExpString.pm
: contains various string manipulation and verification routines connected to the recognition of the respective time entities in temporal annotation files.

TempTypes.pm
: contains routines that detect morphological type of the respective time points and record that information for a given time point.

TempMatrix.pm
: represents the temporal relation matrix which also contains attributes of the respective time points (appurtenance to a plan, etc.).

Tempo.pm
contains various useful routines related to tree querying that is specific to the time processing subsystem. This package is widely used by other libraries.

Closure
contains implementation of the computation of transitive closure . *Closure.pm* is a Perl implementation, the project located in *Exe/Closure* is a C++ implementation. Currently, C++ implementation is used as it is much faster.

FunctionEval.pm
contains comparison routines that determine the relationship between two functional compositions, namely whether one if a less specific than the other.

Interpreting Time Expressions

EntityIdentification.pm
is the entry point to time expression interpretation subsystem. The main function gathers the interpretations (functional compositions) of the child subtrees of the given node which correspond to the respective time expressions. It does so by running the parser (*EIEngine.pm*) for the given grammar (*EIGrammar.pm*). It then builds the final time specification for the given node.

EIEngine.pm
contains an implementation of the parser described in Section 3.6.1. The parser accepts a grammar to parse with and a subtree to parse.

EIGrammar.pm
is a grammar describing time expression subtrees. Lexical conditions the respective terminal nodes must match are present in *EIParsing.pm*. *EIInterpretation.pm* contains the interpretation functions for the respective grammar layers.

EIParsing.pm
contains predicates determining whether a node of the parsed subtree fulfills given lexical constrains.

EIInterpretation.pm
contains the interpretation functions for the respective grammar layers. It uses *EIParseNavigation* to process the parsing result.

EIParseNavigation.pm
contains auxiliary routines that make it easier to traverse parsing results.

Comparing Extensions of Time Expressions

Beam.pm
: contains beam estimation routines described in Appendix B. It uses time functions type system (*TimeFunctions.pm*) and the general time knowledge library (*TimeKnowledge.pm*).

TimeFunctions.pm
: contains definition of the respective time functions as well as validation routines that allow for valid construction of functional compositions.

Detection of Animate Actors

Inquiry.pm
: contains the detection routines. It uses *OntologyInterface.pm* to query the EuroWordNet ontology.

Auxiliary Routines[2]

PCEDT.pm
: represents an abstraction layer between tree operations supported by BTred and those used by our systems. The module encapsulates BTred calls by its own functions and our system uses these functions instead of the original BTred calls. It also provides some extensions. The motivation is to somehow shield the code from different tectogrammatic annotation schemata. However, our code still heavily relies on the PCEDT (PDT 1.0) tree structure so it would take much more than simply rewriting *PCEDT.pm* to get the system functional under PDT 2.0.

LexMorph(EN).pm
: contains various tests related to lexical and morphological properties of tree nodes for Czech and English, respectively.

TreeExtensions.pm
: contains various extensions to tree querying.

Pearls.pm
: is a general Perl utility package.

CFGParser.pm
: is a user friendly interface to the Earley[3] context free grammar parser. It is used by the functional composition comparison subsystem to parse the compositions.

[2]Some of these routines are also used by the tense generation system.
[3]CPAN module

Text Generation

Main Scripts

create_data.pl
: processes the training, heldout and testing TGTSs and creates the corresponding n–gram (each position corresponds to a feature) files to be used by the EM module. This is a wrapper around BTred scripts that create the file for the respective subsystems: *dist_det.btred* (insertion of determiners), *dist_pp.btred* (insertion of prepositions and subordinating conjunctions) and *dist_surf.btred* (surface ordering).

run_stat.pl
: runs the EM model on the created data files for the given subsystem. It calls the *NGram* executable[4] (written in C++) to do the actual work.

run_test.pl
: is the evaluation script that can also physically modify TGTSs according to the results for the given experiment (subsystem). It is a wrapper around the scripts for the respective experiments: *eval_det.btred*, *eval_pp.btred* and *eval_surf.btred*. These scripts also contain all the rule–based postprocessing of the results obtained by EM.

tri.pl
: is the TSRF (see Chapter 5) Perl wrapper. TSRF itself is implemented in Mercury and contains large number of modules, see the enclosed DVD.

Libraries

Common.pm
: contains auxiliary routines connected to the wrapper scripts.

tri_parsing.pm
: contains auxiliary routines connected to the TSRF wrapper script.

Generation of English Tenses

Main Scripts

match.pl
: runs the automatic matching procedure between the nodes in Czech and English trees.

verb_tense.pl
: runs the tense generation.

[4]EM computation module was written by Keith Hall and his colleagues at Center for Speech and Language Processing, Johns Hopkins University, Baltimore, USA.

Libraries

MTTemporality.pm
: represents a map object between the Czech and English temporal annotation, i.e., it allows to query temporal relations between the matched English nodes based on the temporal annotation of their Czech counterparts.

VTGSupport.pm
: contains various auxiliary routines.

Bibliography

[1] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] S. Bangalore and O. Rambow. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING'2000), July 31 - August 4 2000*, pages 42–48, Universität des Saarlandes, Saarbrücken, Germany, 2000.

[3] B. Boguraev and R. K. Ando. Timeml-compliant text analysis for temporal reasoning. In *Proceedings of 19th International Joint Conference on Artificial Intelligence (IJCAI'2005), July 30 - August 5 2005*, pages 997–1003, Edinburgh, Scotland, 2005.

[4] A. Bohmová, S. Cinková, and E. Hajičová. A manual for tectogrammatical layer annotation of the prague dependency treebank (english translation). Technical report, ÚFAL MFF UK, Prague, Czech Republic, 2005.

[5] A. Bohmová, J. Hajič, E. Hajičová, and B. Vidová-Hladká. The prague dependency treebank: Three-level annotation scenario. In A. Abeille, editor, *Treebanks: Building and Using Syntactically Annotated Corpora*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[6] M. Bramer. *Logic Programming with Prolog*. Springer, 2005.

[7] J. Bresnan and R. Kaplan. Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA, 1982.

[8] E. Charniak and M. Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 173–180, Ann Arbor, USA, 2005.

[9] D. Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 263–270, Ann Arbor, USA, 2005.

[10] S. Cinková, J. Hajič, M. Mikulová, L. Mladová, A. Nedolužko, P. Pajas, J. Panevová, J. Semecký, J. Šindlerová, J. Toman, Z. Urešov, and Z. Žabokrtský. Annotation of english on the tectogrammatical level. Technical Report 35, ÚFAL MFF UK, Prague, Czech Republic, 2006.

[11] M. Čmejrek, J. Cuřín, J. Hajič, and J. Havelka. Prague czech-english dependency treebank: Resource for structure-based mt. In J. Hutchins, B. Kis, and G. Prószéky, editors, *Proceedings of the 10th EAMT Conference, May 30-31 2005*, pages 73–78, Budapest, Hungary, 2005.

[12] M. Čmejrek, J. Cuřín, and J. Havelka. Prague czech-english dependency treebank: Any hopes for a common annotation scheme? In A. Meyers, editor, *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*, pages 47–54, Boston, USA, 2004.

[13] M. Collins. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637, 2003.

[14] S. Corston-Oliver, M. Gamon, E. Ringger, and R. Moore. An overview of amalgam: A machine-learned generation module. In *Proceedings of the International Natural Language Generation Conference*, pages 33–40, New York, USA, 2002.

[15] J. Cuřín, M. Čmejrek, J. Havelka, and V. Kuboň. Building a parallel bilingual syntactically annontated corpus. In K.-Y. Su and J. Tsujii, editors, *LNCS/Lecture Notes in Artificial Intelligence/Proceedings of the First International Joint Conference on Natural Language Processing (IJCNLP), March 22-24, 2004*, volume 3248, pages 168–176, Hainan Island, China, 2005. Springer Verlag, Heidelberg.

[16] B. Dorr and B. Olsen. Deriving verbal and compositional lexical aspect for nlp applications. In P. R. Cohen and W. Wahlster, editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 151–158, Somerset, New Jersey, 1997. Association for Computational Linguistics.

[17] D. Dowty. *Word Meaning and Montague Grammar*. D. Reidel, Boston, USA, 1979.

[18] D. Dowty. The effects of aspectual class on the temporal structure of discourse: Semantics or pragmatics? *Linguistics and Philosophy*, 9:37–61, 1986.

[19] E. Filatova and E. Hovy. Assigning time–stamps to event–clauses. In *Proceedings of the ACL-2001 Workshop on Temporal and Spatial Information Processing*, Toulouse, France, 2001.

[20] J. Gebauer. *Mluvnice česká pro školy střední a stavy učitelské*. Česká grafická akciová společnost "Unie", Prague, Czech Republic, 1905.

[21] J. Hajič. *Disambiguation of Rich Inflection (Computational Morphology of Czech)*. Karolinum, Charles University Press, Prague, Czech Republic, 2004.

[22] E. Hajičová, Z. Kirschner, and P. Sgall. A manual for analytic layer annotation of the prague dependency treebank (english translation). Technical report, ÚFAL MFF UK, Prague, Czech Republic, 1999.

[23] E. Hajičová, P. Pajas, and K. Veselá. Corpus annotation on the tectogrammatical layer: Summarizing the first stages of evaluations. *Prague Bulletin of Mathematical Linguistics*, (77):5–18, 2002.

[24] E. Hajičová, B. Partee, and P. Sgall. *Topic-focus articulation, tripartite structures, and semantic content*. Kluwer Academic Publishers, Dordrecht, 1998.

[25] J. Hajič. Building a syntactically annotated corpus: The prague dependency treebank. In E. Hajičová, editor, *Issues of Valency and Meaning. Studies in Honor of Jarmila Panevová*, pages 12–19. Prague Karolinum, Charles University Press, 1998.

[26] J. Hajič, M. Čmejrek, B. Dorr, Y. Ding, J. Eisner, D. Gildea, T. Koo, K. Parton, D. Radev, and O. Rambow. Natural language generation in the context of machine translation. summer workshop final report. Technical report, Center for Language and Speech Processing, Johns Hopkins University, Baltimore, 2002.

[27] E. Hajičová, J. Panevová, and P. Sgall. Recursive properties of tense in czech and english. *The Prague Bulletin of Mathematical Linguistics*, (13):9–42, 1970.

[28] E. Hajičová, P. Sgall, and E. Buráňová. Topic-focus articulation and degrees of salience in the prague dependency treebank. In A. Carnie, H. Harley, and M. Willie, editors, *Formal Approaches to Function in Grammar. In honor of Eloise Jelinek, Arizona*, pages 165–177. John Benjamins, Amsterdam/Philadelphia, 2003.

[29] K. Hall and P. Němec. Generation in machine translation from deep syntactic trees. In *HLT-NAACL Workshop on Syntax and Structure in Statistical Translation*, pages 57–64, Rochester, USA, 2007.

[30] B. Havránek and A. Jedlička. *Česká mluvnice*. SPN, Praha, 1960.

[31] R. Jackendoff. *Semantic Structures*. MIT Press, Cambridge, USA, 1979.

[32] L. Kallmeyer and I. Steiner. Querying treebanks of spontaneous speech with viqtorya. *Traitement Automatique des Langues*, 43(2):55–57, 2003.

[33] G. Katz and F. Arosio. The annotation of temporal information in natural language sentences. In *Proceedings of the ACL-2001 Workshop on Temporal and Spatial Information Processing*, pages 104–111, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[34] S. Kepser. Finite structure query: a tool for querying syntactically annotated corpora. In *EACL '03: Proceedings of the 10th conference on European chapter of the Association for Computational Linguistics*, pages 179–186, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[35] E. Konig and W. Lezius. A description language for syntactically annotated corpora. In *Proceedings of the COLING Conference*, pages 1056–1060, 2000.

[36] L. Kučová and E. Hajičová. Coreferential relations in the prague dependency treebank. In *Proceedings of the 5th International Conference on Discourse Anaphora and Anaphor Resolution 2004*, pages 97–102, San Miguel, Azores, Spain, 2005.

[37] I. Langkilde-Geary. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the 2nd International Natural Language Generation Conference*, pages 17–24, 2002.

[38] I. Langkilde-Geary and K. Knight. The practical value of n-grams in generation. In E. Hovy, editor, *Proceedings of the 9th International Natural Language Generation Workshop*, pages 248–255, New Brunswick, New Jersey, 1998. Association for Computational Linguistics.

[39] W. Li, K. Wong, and C. Yuan. A model on temporal and spatial information processing. In *In Proceedings of the ACL-2001 Workshop on Temporal and Spatial Information Processing*, pages 81–87, 2001.

[40] W. Li, K.-F. Wong, G. Cao, and C. Yuan. Applying machine learning to chinese temporal relation resolution. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 582–588, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

[41] V. Ljubopytnov, P. Němec, P. Michaela, J. Reschke, and J. Stuchl. Orakulum - complex linguistic query system. In *SOFSEM Student Forum Proceedings*, Milovy, Czech Republic, 2002.

[42] I. Mani, B. Schiffman, and J. Zhang. Inferring temporal ordering of events in news. In *Proceedings of HLT-NAACL*, pages 55–57, 2003.

[43] I. Mani, M. Verhagen, B. Wellner, C. M. Lee, and J. Pustejovsky. Machine learning of temporal relations. In *Proceedings of ACL*, pages 753–760, 2006.

[44] I. Mani and G. Wilson. Robust temporal processing of news. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL 2000)*, pages 69–76. Association for Computational Linguistics, 2000.

[45] I. Mani, G. Wilson, B. Sundheim, and L. Ferro. Guidelines for annotating temporal information. In *HLT 2001, Proceedings of the First International Conference on Human Language Technology Research*, pages 299–302, 2001.

[46] M. P. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The penn treebank: Annotating predicate argument structure. In *HLT '94: Proceedings of ARPA Human Language Technology Workshop*, pages 114–119, Morristown, NJ, USA, 1994. Association for Computational Linguistics.

[47] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1994.

[48] J. Mírovský. Netgraph: A tool for searching in prague dependency treebank 2.0. In *Proceedings of the Fifth Workshop on Treebanks and Linguistic Theories (TLT)*, number 5, pages 211–222, 2006.

[49] J. Mírovský, R. Ondruška, and D. Pruša. Searching through prague dependency treebank. In *Proceedings of Treebanks and Linguistic Theories*, pages 114–122, 2002.

[50] M. Moens and M. Steedman. Temporal ontology and temporal reference. *Computational Linguistics*, 14(2):15–28, 1998.

[51] MUC. *Proceeedings of the Sixth Message Understanding Conference (MUC-6)*. Morgan Kaufmann, USA, 1995.

[52] MUC. *Proceeedings of the Seventh Message Understanding Conference (MUC-7)*. Morgan Kaufmann, USA, 1998.

[53] V. Novák. Towards logical representation of language structure. *The Prague Bulletin of Mathematical Linguistics*, 82:5–86, 2004.

[54] P. Němec. Capturing the meaning of time expressions: A functional approach. In *To appear*.

[55] P. Němec. Annotation of temporal relations within a discourse. In *Proceedings of Text, Speech and Dialogue*, pages 181–188, Brno, Czech Republic, 2006.

[56] P. Němec. Tree searching/rewriting formalism. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, pages 2194–2199, Genova, Italy, 2006.

[57] P. Němec. Automatic analysis of temporal relations within a discourse. In *Proceedings of 14th International Symposium on Temporal Representation and Reasoning*, Alicante, Spain, 2007.

[58] P. Pajas and J. Štěpánek. A generic xml-based format for structured linguistic annotation and its application to prague dependencytreebank 2.0. Technical Report 29, ÚFAL MFF UK, Prague, Czech Republic, 2005.

[59] J. Panevová. Vedlejší věty obsahové. *Slovo a slovesnost*, 32(4):289–300, 1971.

[60] J. Panevová, E. Benešová, and P. Sgall. *Čas a modalita v češtině*. Charles University, Prague, 1971.

[61] C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, 1994.

[62] J. Ptáček and Z. Žabokrtský. Synthesis of czech sentences from tectogrammatical trees. In *Proceedings of the 9th International Conference, TSD 2006*, pages 221–228, 2006.

[63] J. Pustejovsky. The syntax of event structure. *Cognition*, 41, 1991.

[64] J. Pustejovsky, P. Hanks, R. Sauri, A. See, D. Day, L. Ferro, R. Gaizauskas, M. Lazo, A. Setzer, and B. Sundheim. The timebank corpus. *Corpus Linguistics*, pages 647–656, 2003.

[65] J. Pustejovsky, B. Ingria, R. Sauri, J. Castano, J. Littman, R. Gaizauskas, A. Setzer, G. Katz, and I. Mani. The specification language timeml. In I. Mani, J. Pustejovsky, and R. Gaizauskas, editors, *The Language of Time: A Reader*. Oxford University Press, 2005.

[66] C. Quirk, A. Menezes, and C. Cherry. Dependency treelet translation: Syntactically informed phrasal smt. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 263–270, Ann Arbor, USA, 2005.

[67] R. Quirk, S. Greenbaum, G. Leech, and J. Svartvik. *A Comprehensive Grammar of English*. Longman, Harlow, UK, 1985.

[68] B. Randall. Corpussearch user's manual. Technical report, University of Pennsylvania, 2000.

[69] H. Reichenbach. The tenses of verbs. In *H. Reichenbach : Elements of Symbolic Logic*, pages 287–298. MacMillan, London, 1947.
[70] D. Rohde. Tgrep 2. Technical report, Carnegie Mellon University, 2001.
[71] F. Schilder and C. Habel. From temporal expressions to temporal information: Semantic tagging of news messages. In *Proceedings of ACL-2001 Workshop on Temporal and Spatial Information Processing*, pages 65–72, 2001.
[72] P. Sgall. *Generativní popis jazyka a česká deklinace*. Academia, Prague, 1967.
[73] P. Sgall, E. Hajičová, and J. Panevová. *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Kluwer Academic Publishers, Boston, 1986.
[74] P. Sgall, J. Panevová, and E. Hajičová. Deep syntactic annotation: Tectogrammatical representation and beyond. In A. Meyers, editor, *Proceedings of the HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*, pages 32–38, Boston, Massachusetts, USA, 2004. Association for Computational Linguistics.
[75] C. Smith. *The Parameter of Aspect*. Kluwer, Dordrecht, 1991.
[76] R. Soricut and D. Marcu. Stochastic language generation using widl-expressions and its application in machine translation and summarization. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics*, pages 1105–1112, 2006.
[77] K. Svoboda. *Souvětí spisovné češtiny (scriptum)*. Charles University, Prague, 1970.
[78] F. Trávníček. *Mluvnice spisovné češtiny. I, II.* SPN, Prague, 1951.
[79] Z. Vendler. Verbs and times. *Linguistics in Philosophy*, pages 97–121, 1967.
[80] P. Vossen. *EuroWordNet: a multilingual database with lexical semantic networks for European Languages*. Kluwer Academic Publishers, Dordrecht, 1999.
[81] S. Wallis and G. Nelson. Exploiting fuzzy tree fragment queries in the investigation of parsed corpora. *Literary and Linguistic Computing*, 15(3):339–361, 2000.
[82] G. Wilson, I. Mani, B. Sundheim, and L. Ferro. A multilingual approach to annotating and extracting temporal information. In *Proceedings of the ACL-2001 Workshop on Temporal and Spatial Information Processing*, pages 81–87, 2001.
[83] D. Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404, 1997.