Extracting Temporal Information and Ordering Events for Swedish

Master's thesis report

Anders Berglund

27th October 2004

Abstract

In a narrative of a sequence of events, events aren't always presented in chronological order. When a computer to some extent is to understand the narrative, it might be useful or even necessary to understand the order in which the described events occur.

In this Master's thesis report I present a method for extracting such an ordering as well as a method for detecting and interpreting time expressions, information that also can be useful in a program.

The methods are implemented and evaluated for Swedish as well as integrated with the Carsim program.

Acknowledgements

This work wouldn't have been possible without Pierre Nugues. I am grateful for getting an interesting task and grateful for good supervision. I enjoyed getting the opportunity to participate in the writing of a paper, and I enjoyed the discussions we have had about the minutiae of the usage of the English language.

I am also grateful to Richard Johansson for the good support and feedback I've gotten, as well as for the good discussions we have had.

I would also like to thank Inderjeet Mani for providing me with valuable feedback.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Carsim is a program that generates animated 3D scenes from texts that describe road accidents (Johansson et al., 2004a, Johansson et al., 2004b).

Carsim considers authentic texts, generally collected from online newspapers or written by victims of accidents. Carsim has a modular structure that allows for different language processing modules. Modules for French, English, and Swedish are available. Currently, the work is focused on developing a module for Swedish.

To illustrate the goals and challenges of the Carsim program, I wrote a small text in English where I annotated some entities with identifiers:

> *{Two people}$_{o1}$ died$_{e1}$ {late yesterday evening}$_{t1}$ when a car$_{o2}$ drove$_{e2}$ off the road$_{o3}$ and crashed$_{e3}$ into a tree$_{o4}$. The car$_{o5}$ {was overtaking}$_{e4}$ another car$_{o6}$ when the driver$_{o7}$ {lost control}$_{e5}$ of it$_{o8}$.*

To create a good animation, the program needs to extract and understand who the participants are and what they do. In the case of the accident above, it has to

1. Detect and interpret the events $e2$, $e3$, $e4$ and $e5$.
2. Detect and interpret the involved physical entities $o2$, $o3$, $o4$, $o5$, $o6$ and $o8$.
3. Understand that cars $o2$ and $o5$ are the same, and that the pronoun $o8$ refers to it.
4. Link the participants to the events using semantic roles or grammatical functions.
5. Understand that the order of the events is $e4$-$e5$-$e2$-$e3$.
6. Possibly detect the time expression $t1$ to anchor temporally the animation. Evenings generally imply darkness, and such information can at times be useful.

The purpose of this Master's thesis is to address points 5.) and 6.), i.e. detect and order the events as well as detect and interpret the time expressions. Extraction of participants and semantic roles is carried out by other parts of the Carsim program.

1.1 Previous work

Modern work on temporal event analysis probably started with Reichenbach (1947), who proposed the distinction between the point of speech, point of reference, and point of event in utterances. The separation allows for e.g. a systematic description of tenses, and his analysis proved to be very powerful.

The logic of event ordering and automatic extraction of such information has been a research topic for over 20 years. Allen (1984) pioneered the field by creating a formal classification of temporal relations. He identified 13 different relations between pairs of temporal intervals as shown in Figure 1.1. If Allen's relations were to be applied to the accident described above, a graph such as the one in Figure 1.2 could be created.

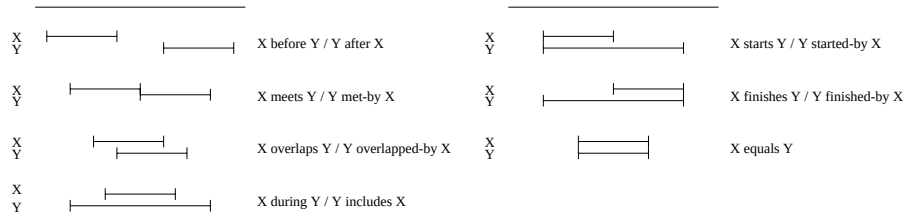


Figure 1.1: Allen's 13 temporal relations.

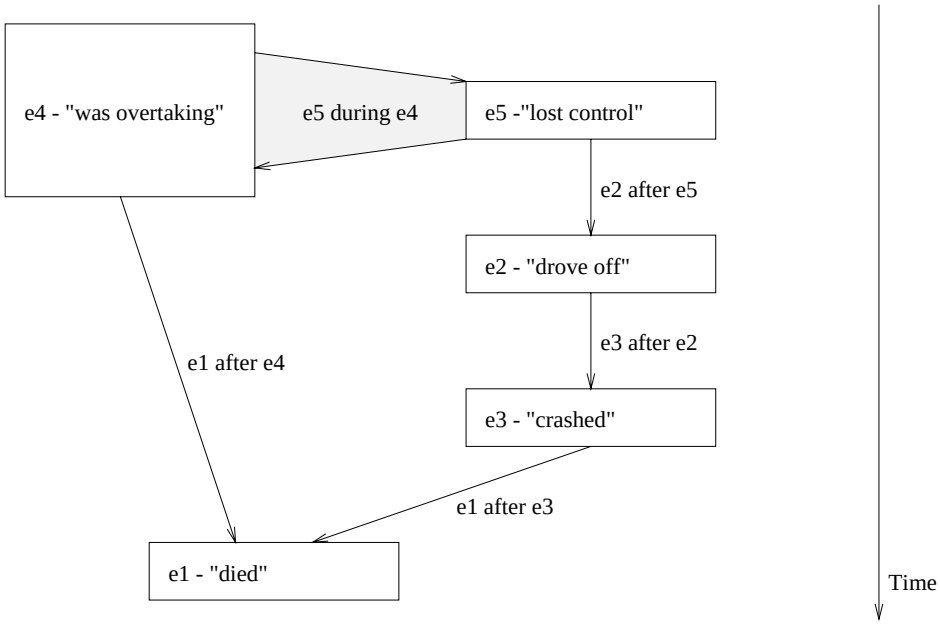


Figure 1.2: The chain of events in the text on page 1 interpreted using Allen's relations.

Later, Dowty (1986) introduced the "narrative convention", the idea that the usage of two verbs in the perfect tense means that the second event occurs after

the first one. In the accident report above, this implies that event $e3$ happens after event $e2$ as well as event $e5$ happening after event $e4$. It also implies that event $e4$ happens after event $e3$, which unfortunately is not true. Webber (1988) continued Dowty's work by creating a larger set of conventions for time stamping and ordering of phrases.

Lascarides and Asher (1993) presented a system that used a wealth of semantic knowledge to order events of phrases in pluperfect. Hitzeman et al. (1995) argued that such an approach is too complex, and work along those lines has been discontinued.

The research community continued investigations into the field and created several incompatible annotation schemes. Pustejovsky et al. introduced the Time Mark-up Language (TimeML) (TimeML Specification, 2002) in 2002 in an effort to reconcile and unify the temporal annotations. TimeML is intended to capture all aspects of temporal relations between events in discourses. TimeML has to some extent been adopted as the interlingua of temporal markup, and its usage makes it relatively easy to combine different peoples' work.

Recently, much work has focused on building a large collection of TimeML annotated discourses. The resulting TimeBank Corpus (2002) currently contains some 300 annotated discourses, and it will probably ease further work in the area.

Late 2003, Mani et al. (2003) introduced the use of decision trees to partially order events in English texts. The events they order are the main events of successive clauses, not necessarily every event. They achieved impressive results, results as of now unmatched.

1.2 Goal for this Master's thesis

The objective for this Master's thesis was to develop a module capable of

1. Ordering the events in the Carsim program.
2. Extracting time expressions from Swedish texts.

Although the approach is similar to Mani et al.'s, this work was an independent development and I became aware of Mani et al.'s papers when I started to write this report.

In this work, events, things "that happen", not clauses, are ordered. When a text is processed, events and time expressions are detected and interpreted. Several different decision trees are used to order the events. Some of the trees span large segments of the text, creating a large number of partial orderings. Some of these orderings may be conflicting, and these are reconciled to create a full ordering of the events in the text.

The resulting full ordering is used to schedule the events in the Carsim program.

1.3 Overview of the report

- Chapter 2 gives an overview of the work.
- Chapter 3 gives an overview of the TimeML annotation scheme.

- Chapters 4–9 give a detailed description of the different parts of the work done.
- Chapter 10 presents some ideas for future work.
- Chapter 11 gives a small summary.
- Appendix A presents some of the terms used in this report. Moreover, the measures of precision, recall and "F-measure" are presented. These measures are used throughout this report.
- Appendix B contains part of a paper from 1940 by Whorf where he discusses how verbs and nouns are separated in natural languages. The excerpt is relevant in Chapter 7 when I declare what words/phrases the engine regards as events.

To keep the report readable, examples are given only in English when general concepts are presented, as their counterparts in Swedish are irrelevant for the purpose of understanding.

Chapter 2

A comprehensive approach to time and event annotation

This chapter serves as a short overview of the approach we adopted to annotate time and events. The following chapters describe the annotation scheme, the tagger, as well as the decision trees in greater detail.

2.1 TimeML

When processing the text, we needed a good annotation scheme. We decided to use TimeML, as it is a generic annotation with large coverage that has been adopted by researchers in the field.

An quick overview of TimeML is given in Chapter 3.

2.2 An automatic TimeML tagger

Regular expressions are a convenient approach for finding time expressions, verb phrases, et cetera. But, as normal regular expression engines don't compute anything when returning from matches, we needed a custom engine that could chain computations.

I wrote an engine capable of doing this. The regular expressions are encoded as rules. The rules, possibly recursive, apply regular expressions, perform computations, and create TimeML entities.

The rules are encoded in an XML file, thus making it easy to extend and refine the set of rules.

2.3 A machine learning approach

The central task, the proper ordering of events, is difficult. Some orderings are easy to detect, but the majority of the temporal relations can't be found using simple rules.

In order to avoid writing complex rules that probably wouldn't perform too well given the previous results in the field, the ungrateful task of event ordering

was handed off to a set of decision trees in the hope that they, given proper training, could generate high-quality temporal orderings of events.

When used, the trees are given a set of features extracted from events as well as some other measures believed to be important. They decide partial orderings between tuples of events, and eventually the entire set of partial orderings is used to create a globally coherent ordering of all events detected.

The motivation for using decision trees is that they are easy to train from an annotated corpus and that they are effective as proven by Ng and Cardie (2002) when they achieved the best results yet for coreference resolution of noun phrases.

2.4 Resolving conflicting temporal orderings

Using several different decision trees enables us to take into account event dependencies between events relatively far apart. However, it also introduces conflicts.

To examplify, consider three events A, B and C. One set of relations might say `"A before B, B before C"` while another might say `"C before A"`. The resulting temporal ordering is `"A->B->C->A"`; clearly a temporal loop. After all temporal orderings have been generated, such loops are resolved by removing of some of the conflicting temporal orderings. Section 8.2 describes how this is done.

2.5 Domain independence

I attempted to make the tagger as natural language agnostic and program independent as possible. The architecture of the implementation is presented in Figure 2.1.

The only part of the time ordering functionality that depends on the Carsim program is the “Interface” box. If the POS tagger data were supplied in some portable format, the event ordering and time detection functionality would be completely program independent.

The “Engine” box lies in a separate Java package and it is in this report referred to as the “TimeCore module”.

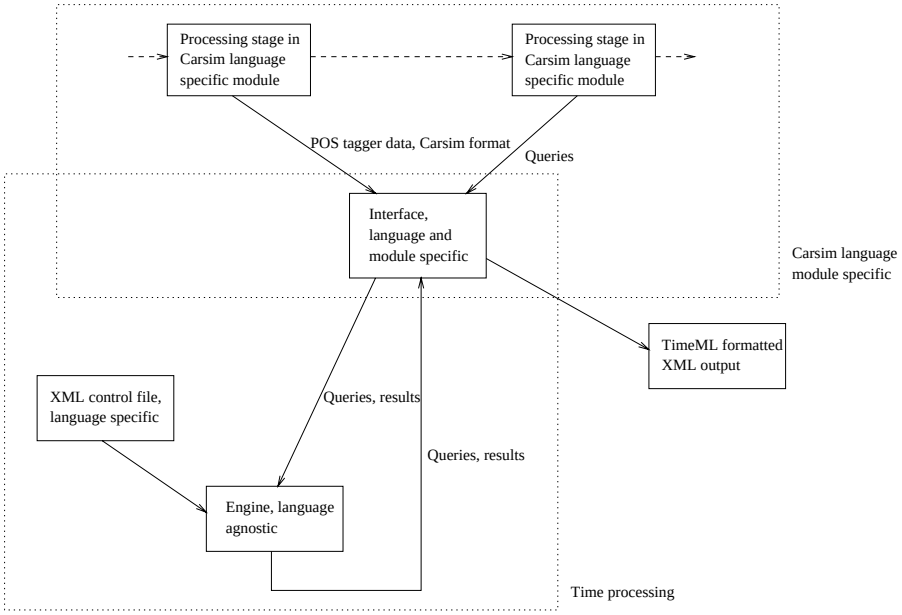


Figure 2.1: Architecture of the time ordering module.

Chapter 3

TimeML

In this chapter, I give a short overview of the TimeML annotation scheme (The TimeML Specification, 2002). It is an XML-based language capable of marking up the flow of events in a text.

TimeML defines tags for events (e.g. *he slept*), for time expressions (*at four o'clock*), for function words (*he did* not *sleep*), and tags that connect entities in different ways.

In TimeML, the sentence *John died after the accident* would be marked up as follows:

```
John <EVENT eid="e1">died</EVENT>
<SIGNAL sid="s1">after</SIGNAL> the
<EVENT eid="e2">accident</EVENT>.
<MAKEINSTANCE eiid="ei1" eventID="e1"/>
<MAKEINSTANCE eiid="ei2" eventID="e2"/>
<TLINK eventInstanceID="ei1" relatedToEvent="ei2"
signalID="s1" relType="AFTER"/>
```

The idea is that all entities believed to be important for the understanding of the temporal flow are marked up. In the example, two events are annotated and instantiated. They are connected by a temporal link, a `TLINK`. The temporal signal *after* can also be relevant, and it is tagged with a `SIGNAL` tag.

In the following sections, I describe cursorily each TimeML tag. The examples I give are in general shortened, as the intention is to present the principles, not the entire mark-up language. A more verbose and complete overview can be found in the *The Specification Language TimeML* overview (Pustejovsky et al., 2004).

3.1 Events

Events in TimeML have a tense, an aspect, and a "class". The class classifies the *type* of event, whether it is e.g. a state or an instantaneous event. Only the class feature is mandatory for TimeML events; the tense and aspect are optional.

Verb group	Aspect
taught	NONE
was teaching	PROGRESSIVE
had taught	PERFECTIVE
had been teaching	PERFECTIVE_PROGRESSIVE

Table 3.1: TimeML's aspects for verbs in the past tense.

3.1.1 Tense and aspect for TimeML events

In TimeML, the tenses are *past, present, future*, and *none* (for e.g. infinitives), the aspects *none, progressive, perfective,* and *perfective_ progressive*. This allows for 4×4 "general tenses" – a classification more flexible and powerful than the "standard" tense system.

Table 3.1 shows how the aspects are used for verbs in the past tense (the table is taken from the *TimeML Specification* (2002)).

In TimeML, the tense pluperfect doesn't exist. Phrases in pluperfect are encoded in TimeML as having the tense *past* and the aspect *perfective*.

3.1.2 Classes for TimeML events

The class attribute is the most important feature of an event in TimeML. Unfortunately, it is also the most complex and ambigous part of the event annotation.

TimeML differentiates between events of different types, be they aspectual, states, occurrences, hypothetical events or of other types. It also distinguishes intentional events from events "that just happened". The class system in TimeML is quite complex, and the best description there is is available in the TimeML annotation guidelines (Pustejovsky et al., 2002).

3.2 Time expressions

Time expressions in TimeML are tagged with Timex3 (TimeML DTD, 2002) tags. Timex3 is a "standard" for encoding possibly fuzzy time expressions. A time expression in Timex3 can be just time-of-day, just date or both. The value of the time-expression is stored in the ISO8601 (2000) format. As ISO8601 can represent dates, times, periodic events, and temporal intervals, so can Timex3:s.

An example Timex3-tag (if "today" is 2002-08-05):

```
<TIMEX3 type="time"
        value="2002-08-04T10:00">yesterday at 10 am</TIMEX3>
```

3.3 The `MAKEINSTANCE` tag

In expressions such as *John was here Monday and Tuesday*, there are two events anchored in time; John's presence on Monday and his presence on Tuesday. In TimeML, two "instances" of the *was here* event are to be created. The instances are declared using the `MAKEINSTANCE` tag, and they are used separately. This can be useful at times. If the previous example continued with *On Tuesday he*

was late, the *was late* event happens after the "Monday instance", but not after the "Tuesday instance".

3.4 Signals

Signals are tags for function words that specify modality (e.g. modal auxiliaries, markers for negation) or words with other functions, such as the word *after*, which can be a temporal signal.

Below is an example of two different kinds of signals, one modal and one temporal:

```
It <SIGNAL>might</SIGNAL> arrive <SIGNAL>before</SIGNAL>
you [do].
```

The only attribute of a signal in TimeML is an identifier. The semantic meaning of the signal is to be encoded using links, something that might be problematic if a parser can't find the event(s) to connect the signal to.

3.5 Subordinate links

Subordinate links (`SLINKs`) are used to connect signals to event instances or to connect pairs of event instances. A signal can be connected to an event instance to e.g. imply negativity, i.e., to indicate that something didn't take place.

An SLINK connecting one event instance to another might couple the events in a phrase such as *he said he slept*. It is clear that there is a relation between the two events, and in TimeML that relation could be encoded with an `SLINK` having the relation *evidential*.

3.6 Aspectual links

An aspectual link (`ALINK`) connects two event instances, one of which is aspectual and the other one its "argument". The overview *The Specification Language TimeML* (Pustejovsky et al., 2004) gives the following examples that explain the concept well: *John started to read* (initiation), *John finished assembling the table* (culmination), *John stopped talking* (termination) and *John kept talking* (continuation).

The difference between culmination and termination is perhaps not clear to the casual eye, but the focus is for the culmination the "climax" of the event, whereas the focus in the case of the termination is on the fact that the action ended.

3.7 Temporal links

Temporal links (`TLINKs`) connect time expressions to event instances or pairs of event instances. `TLINKs` are appropriate in phrases such as *he came at ten o'clock* and *after eating, he went to bed*. In the first case, the meaning of the `TLINK` would be "at same time", in the second the `TLINK` would connect the two events with an "after"-relation.

Chapter 4

A regular expression engine for TimeML markup

This chapter gives an overview of the engine used by the system. The engine lies in a module of its own, where it is largely independent of the rest of the Carsim program. The separate module is called "TimeCore".

The engine itself is simple and natural language agnostic. The parts of the functionality that are language and domain specific are separated from the TimeCore module, with one part of the logic lying in an XML control file that controls the engine's workings, and another part of the logic lying in an interface class.

The XML file follows a DTD that the engine understands, and the interface class supplies implementations for methods in an abstract superclass found in the TimeCore module, thus presenting an interface the engine can work with.

I architectured it this way to keep the engine itself clean and language independent, and all language specific features are found in the XML control file and in the interface class.

4.1 Basic idea

The basic idea is that the engine starts by building a dictionary of expressions. It then processes "stages" defined in the XML file. In each stage, it tries to match a set of rules (defined in the XML file) against a list of tokens. When it finds a match, it forwards the result to the interface class, and then continues matching rules against the rest of the tokens.

To summarize, the workings of the engine is defined by the XML file. The interface class handles the results. The engine itself is simple and knows nothing of the language in question.

4.2 Concepts used in the XML control file

The engine is best described by explaining the concepts used in the XML file, all of which correspond to some part of the engine. The concepts are "data views", "decision tree specifications", "enumerations", "derivations", "rules", "multiresult

```
<enumeration class="DOWenum">
  <item time="DOW=1" string="måndag"/>
  ...
  <item time="DOW=7" string="söndag"/>
</enumeration>
```

Figure 4.1: An example enumeration declaration. "DOW" means "day of week".

```
<derivation class="DOWsPOD">
  <string class="DOWenum" ref="R1"/>
  <string value="s"/>
  <string class="PODenum" ref="R2"/>
  <simpletime source="R1^R2"/>
</derivation>
```

Figure 4.2: Example derivation declaration. "DOW" means "day of week", "POD" means "part of day".

rules", and "processing stages". The concepts are each described in separate sections below.

To give an understanding of how the engine is used, I give examples from the XML file used for Carsim's "New Swedish" language processing module.

4.2.1 Enumerations

Enumerations are quite simple concepts, they are just listings of strings having certain values. Figure 4.1 shows an example enumeration that connects the Swedish names of the days of the week to the equivalent TimeCore encoding.

4.2.2 Derivations

Derivations creates time expressions from enumerations. In Swedish, a concept such as *Monday evening* is a compound noun (*måndagskväll*). Figure 4.2 shows a derivation that creates such compound nouns.

The separate entities can be referred to by declaring "references" using the *ref* attribute. In Figure 4.2, the meanings of the involved entities are referred to by using their identifiers and their meanings are merged in the final *simpletime* element using a caret (^).

The caret is an ASCII-rendering of "∩" (intersection). The caret is used as the commonly used "&":s aren't easily available in XML as they are reserved characters.

The motivation for using the intersection concept for merging time expressions is that the terms can be considered to have some attributes set while the other attributes can be regarded as being the "full set". In the derivation in Figure 4.2, $R1$ and $R2$ have their day-of-week respectively part-of-day attributes set and all other attributes are undefined, i.e. they are the "full set". The intersection between them is thus a time expression with the day-of-week and part-of-day attributes specified.

```
<!-- strax före halvåttatiden,
     strax före åttatiden,
     strax före midnatt,
     strax före lunchtid,
     strax innan nio -->
<rule class="JUST BEFORE [halv]NUMtiden/NAMED-HOD/KLOCKAN ?">
  <string value="strax"/>
  <string value="före|innan"/>
  <string class="GENERAL TIME-OF-DAY|NUM as H:M" ref="R1"/>
  <time>
    <computation>
      <term source="R1" op="setgiven"/>
      <term values="MIN=-5" op="add"/>
    </computation>
  </time>
</rule>
```

Figure 4.3: A rule matching expressions such as *just before nine o'clock*.

The derivations can be simpler. Some derivations have just one string element, in which case they are simple mappings from enumerated strings to time expressions.

The derivations allows easy generation of compound nouns carrying meaning. The derived elements can be viewed as a "Cartesian product" of their constituents, and thus an exhaustive listing of all compound expressions *along with their meaning* can be avoided.

4.2.3 Rules

Rules are the primitives of the matching. They can be very simple, they can be recursive, they can create objects, and sometimes they perform computations. Figure 4.3 shows a rule that matches expressions such as *just before nine o'clock* and carries out the necessary computations.

The string elements in the rules (see Figure 4.3) are matched to tokens from the input. Several types of matches can be performed. The tokens in the input can be matched to listed strings, using the *value* attribute as in the first two string elements in Figure 4.3. The tokens can also be matched to other rules, using the the *class* attribute as in the third string element in the example rule. Tokens can also be matched to regular expressions, using a *regexp* attribute.

Several alternative matches can be made in each string element by separating the alternatives using bars (|). In the example above, the second string element matches either the string *före* or the string *innan* (both meaning *before*).

Some elements return a single entity with the result of the matching. In the example rule in Figure 4.3, a "time" object will be returned by the third string element if a match was found.

The returned entities can be referred to by creating references (using the *ref* attribute). This is done in the example rule in Figure 4.3 so that the entity returned by the third string element can be used in the final computation.

```
<dataview name="splitbasicview">
  <dimension type="splittoken"/>
  <dimension type="lemma"/>
  <dimension type="postag"/>
</dataview>
```

Figure 4.4: Example data view declaration.

4.2.4 Data views

In some cases, it can be useful to use e.g. both the POS tag as well as the lemma of the tokens in the input when matching. Thus, there is a need to be able to supply several features for each token. A "data view" is such a mechanism. Each processing stage can use a different data view. The interface class is expected to understand the name of the data view (given in the XML file) and return an array populated properly with entities.

Figure 4.4 shows the data view that is used in the first processing stages of the implementation. For each "split" token, the token itself, its lemma, and its POS tag will be available. The original tokens are "split", separated into several parts, if they contain white spaces, which they sometimes do.

Regarding the use of data views, the rules default to match to the first "dimension" in the data view, but they can be made to match against other dimensions if necessary. If the contents of a certain dimension are used for matching, the contents have to be strings. But the slots can also be used as general purpose containers, to e.g. allow rules to use references to already created entities.

As noted above, the interface class is responsible for populating an array according to the specifications of the data view. The engine uses the contents of that array for all matching, and there is no need to store the text in the engine itself. This allows for a dynamic "input set" that can change over time, as nothing is stored in the engine and as the array is regenerated at the start of each processing stage.

In the case of the Carsim's "New Swedish" module, the available data is initially just the data from the POS tagger. When entities are detected in the engine, these are stored by the interface class, and in later stages, these entities are reused in the matching.

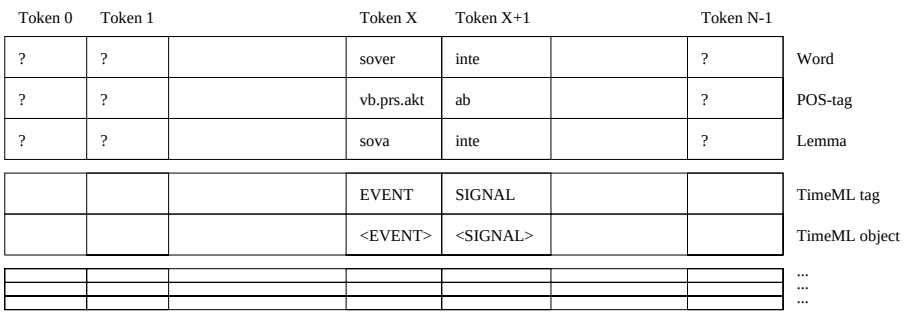


Figure 4.5: Internal dynamics of the interface class.

Figure 4.5 shows some of the data available. The TimeML tags of detected entities are especially useful when matching: In one of the last stages of processing, the engine tries to connect events to time expressions. All it has to do is to look for `EVENT` and `TIMEX3` tags in the TimeML tag dimension, and when it finds a match it simply creates a link between the corresponding entries in the TimeML object dimension of the data view.

To summarize, the data views allows for powerful matching as well as for dynamic change of the input set, making the engine flexible while at the same time keeping the engine itself very simple.

4.2.5 Multiresult rules

Some rules need to generate two or more results, and in those cases, "multiresult rules" can be used. The multiresult rules can return several entities. It would be quite difficult to use the set of values returned in other rules, and thus they can only be "top level" rules, i.e. these rules can't be used for matching by other rules.

Figure 4.6 shows a multiresult rule that matches verb group such as *[inte] kunde [inte] ha sovit* (*could [not] have slept*) and creates all necessary `SIGNALs` and `SLINKs`.

The *opt* attribute means that a match is optional; if the engine isn't able to match an element it continues the matching with using the next element in the rule. In the multiresult rule example (in Figure 4.6), the match process continues even if no signals for negation are found. The optional `SLINKs` will only be created if the corresponding signals are found.

4.2.6 Processing stages

The processing stages are the starting point for the rule matching. A stage lists the data view that it wants, and such a data view is requested from the interface class when the processing of the stage begins. The engine tries to match each listed rule (the attribute giving the name of the rule is *class*) to every token, starting from the first token. If a match is found, the object returned by the rule is passed to the interface-class along with a description of the operation to perform, encoded in the attribute *lcbOpIfMatch* ("Language [module] CallBack (a.k.a. Interface) Operation If Match"). If a match is found, the parser normally does the next matching attempt at the end of the last match.

Figure 4.7 shows a simple processing stage that tries to find events consisting of stand-alone verbs.

4.2.7 Decision tree specifications

Rules can call the interface class directly, and that mechanism is used mainly for creating `TLINKs` using decision trees. Figure 4.8 shows a rule that generates a `TLINK` from a decision tree.

In order to keep the specification of the decision trees clear, they are encoded in the XML files using decision tree specifications. Figure 4.9 shows how this is done.

The reason for explicitly declaring the attributes given to the trees in the XML file is to have it in one place clearly visible. Both the manually written

```
<!-- Pattern <inte> kan <inte> ha sovit -->
<multiresultrule class="MODAL.PRS ... AUX ... SUP">
  <string regexp="signal:neg.*" dimension="postag" ref="S1"
          opt="yes" reftodimensionvalue="yes"
          refdimension="signal"/>
  <string regexp="vb\.prs.*\.mod" dimension="postag"
          ref="R1"/>
  <string class="NO VERBS/MID/MAD/KN/SN/SIGNAL" opt="yes"/>
  <string regexp="signal:neg.*" dimension="postag" ref="S2"
          opt="yes" reftodimensionvalue="yes"
          refdimension="signal"/>
  <string class="NO VERBS/MID/MAD/KN/SN/SIGNAL" opt="yes"/>
  <string regexp="vb\.(prs|prt|inf)\.akt\.aux"
          dimension="postag" ref="R2"/>
  <string class="NO VERBS/MID/MAD/KN/SN/SIGNAL" opt="yes"/>
  <string regexp="vb\.sup\..*" dimension="postag" ref="R3"/>

  <!-- Resulting entities: -->
  <signal tokens="R1" type="modal" ref="SR1"/>
  <event tense="present" class="ND" tokens="R3"
         aspect="perfective" ref="EV1"
         structure="modal + verb group"/>
  <metaphrase type="ignore" tokens="R2"/>
  <slink signal="S1" subordinatedevent="EV1"
         reltype="negative" opt="yes"/>
  <slink signal="S2" subordinatedevent="EV1"
         reltype="negative" opt="yes"/>
</multiresultrule>
```

Figure 4.6: Example multiresult rule.

```
<stage stagename="stand-alone verbs"
       dataview="entitieswithsplitbasicview">
  <startelement class="PRS stand-alone verb"
                lcbOpIfMatch="addmultiresult"/>
  <startelement class="PRT stand-alone verb"
                lcbOpIfMatch="addmultiresult"/>
</stage>
```

Figure 4.7: Example stage declaration.

```
<multiresultrule class="DT:EVENT.MAIN ... EVENT.REL">
  <string regexp="EVENT.*" dimension="timeml-tag" ref="EV1"
          reftodimensionvalue="yes" refdimension="event"/>
  <string class="TLINK-DT: SKIP-ALL-BUT-TIMEXES-AND-EVENTS"
          opt="yes"/>
  <string regexp="EVENT.*" dimension="timeml-tag" ref="EV2"
          reftodimensionvalue="yes" refdimension="event"/>

  <entityfromlcb lcbmethod="dt:tlinkfrom2events">
    <entityref ref="EV1"/>
    <entityref ref="EV2"/>
  </entityfromlcb>
</multiresultrule>
```

Figure 4.8: Example decision tree multiresult rule.

```
<decisiontreespecification name="TLinkDecisionTree2Events"
                           filestem="tlinkdt2events">
  <dataelement name="mainEventTense"/>
  <dataelement name="mainEventAspect"/>
  <dataelement name="mainEventStructure"/>

  <dataelement name="relatedEventTense"/>
  <dataelement name="relatedEventAspect"/>
  <dataelement name="relatedEventStructure"/>

  <dataelement name="temporalSignalInbetween"/>

  <dataelement name="tokenDistance"/>
  <dataelement name="sentenceDistance"/>
  <dataelement name="punctuationSignDistance"/>
</decisiontreespecification>
```

Figure 4.9: Example decision tree specification.

1. Read the control XML file. In reality, this file is partitioned into several different XML files, and all files have to be read and merged.
2. Store all enumerations.
3. Generate all derivations.
4. Store all rules.
5. Process all stages.

Figure 4.10: Algorithm for engine

code and the automatically generated code will protest if they aren't in sync with the decision tree specifications, and thus problems can be avoided.

4.3 Algorithm

Figure 4.10 shows the main algorithm for the engine. It uses the concepts in the XML file, and it is very straight-forward. The engine is very simple, and it just follows the instructions in the XML file.

In the next chapter, I'll go on to give an overview of how the engine is used to annotate a text with TimeML entities.

Chapter 5

Strategy for TimeML mark up

This chapter gives an brief overview of how the engine is used to annotate the text using TimeML. Only what TimeML tags are used and the order in which the engine detects the entities is presented. Details are given in Chapters 6–8.

5.1 Entities to create

TimeML, described in Chapter 3, defines the `EVENT`, `TIMEX3`, `SIGNAL`, `MAKEINSTANCE`, `ALINK`, `SLINK`, and `TLINK` tags. The engine annotates the texts with all those tags except the `ALINKs`, as they are of too little use in the Carsim framework to motivate an annotation attempt.

5.2 Strategy

This section describes in what order the different tags are detected and how the different links are created.

The first entities the engine detects are time expressions. It then detects temporal signals and signals for polarity (negativity). The next stages detect events and finally it creates `TLINKs`. When all `TLINKs` are created, some of them are removed to remove temporal loops.

The following sections describe the stages cursorily.

5.2.1 Time expressions

First, the engine inteprets the document time and stores it. This time expression is important, as all relative time expressions are computed relative to this time expression.

Next, the engine detects all time expressions in the text. The time expressions are interpreted and stored.

The process of time expression detection and interpretation is described in detail in Chapter 6 (“Detection of time expressions”).

5.2.2 Temporal and polarity signals

The engine then tries to find temporal signals (e.g. *after*) as well as polarity signals (e.g. *not*). These signals are detected using simple string matching, and they are stored in the interface class along with a description of their "meaning". The semantic meaning is not a part of TimeML, but it is needed in the engine.

The engine overdetects signals. Prepositions such as *after* are always detected as temporal signals. This is not too big a problem in the Carsim context, as words such as *after* actually are used almost exclusively as temporal signals in the Nyheter corpus (one of the Carsim corpora).

5.2.3 Events and `SLINKs`

The process of event detection is complex. The engine first detects and interprets predicative clauses (copulas) (e.g. *was injured*), then verb phrases using modal and temporal auxiliaries (e.g. *could walk*, *had crashed*). The next stage looks for the remaining "stand alone" verbs, verbs such as *run*.

When searching for matches for these phrases, the rules look for signals for negativity (e.g. *not*) and links these with `SLINKs` to the events it detects when appropriate. The module tags the modal auxiliaries as signals and creates `SLINKs` to connect these signals to their respective event.

The final stage of the event detection finds the nouns that the engine regards as events and stores these.

The detection and interpretation of events are further described in Chapter 7 ("Event Detection").

5.2.4 `TLINKs`

The first `TLINKs` the engine creates connects events to time expressions "close by". It is a simple matter of matching for tags in the TimeML tag dimension of the data view used and of linking the corresponding entries in the TimeML object dimension.

It then creates `TLINKs` by applying decision trees to sequences of events. The final `TLINKs` are created using domain knowledge, common sense, and knowledge from the Carsim program.

Finally, temporal loops are resolved by removing `TLINKs` until all conflicts are resolved.

The process of event ordering is described in Chapter 8 ("Event Ordering").

Chapter 6

Detection of time expressions

The detection of Swedish time expressions is a simple thing given the TimeCore engine.

The initial goal was to place the *accident proper* in time, and thus I only attempted to detect time expressions relevant for that purpose.

Table 6.1 shows some time expressions that are relevant for this purpose. They are actual examples from the Nyheter corpus, and the table lists the document time of their respective report. It also lists how the TimeCore module interpreted these time expressions. Actually, only the last time expression was incorrectly interpreted.

6.1 Time expressions detected

The engine currently finds almost all time expressions that refers to dates and times during the day. Expressions that contain both dates and times are evaluated fully, as the first three entries in Table 6.1 shows.

"Stand-alone" time expressions just meaning a time during the day are not coupled with a date. The fourth entry in Table 6.1 examplifies this. The reason why I refrained from guessing the date based on e.g. the date of other time expressions in the text is that I felt that such an approach would be a bit fragile.

6.2 Time expressions not detected

In this section, I list the classes of time expressions I didn't attempt to detect nor interpret.

- Dates that because of limitations in the engine would have required a separate rule each, are not detected. One example for such expressions are those using fixed multi-word phrases such as *tjugondag Knut* (January 13th).

- Expressions such as *for six hours* almost always dealt with the time victims spent in hospitals. They are not detected.

Doctime	Time expression	Value
2002-04-24	strax före kl 18 på onsdagskvällen	2002-04-24T17:55
2002-05-19	strax före midnatt på lördagskvällen	2002-05-18T23:55
2002-06-22	vid 23-tiden på midsommarafton	2002-06-21T23:00
2002-04-18	kvart i åtta	07:45, PM/AM?
2002-12-26	under annandagens eftermiddag	2002-12-26THH:MM PM
2002-04-11	under fredagen	2002-04-05

(In English: *just before 6 PM Wednesday evening, just before midnight Saturday evening, quarter to eight, 23 o'clock Midsummer eve, during the afternoon of the day after the Christmas day, during the Friday.*)

Table 6.1: Example time expressions.

- Expressions of the type *summer of 1992* are not detected as they are uncommon, and also irrelevant for the purposes of the Carsim program (generally they were in the form *the number of accidents during the Easter holiday of this year was lower than that of the same holiday 1992*).
- Time expressions that are encoded as attributes to nouns (such as *Yesterday's accident*) are not detected, as those didn't show up in any of the examples in the TimeML Specification(2002) nor in any of the annotation guidelines found.
- Misspelt time expressions are not detected. Unfortunately, quite a lot of Swedes don't know how to spell properly, and it would in hindsight have been a good idea to try to find the classes of time expressions written using the most common spelling errors. To examplify, quite often expressions such as *vid halv tretiden* were encountered, but as it ought to be spelled *vid halv tre-tiden* (*around 02.30/14.30*), it will not be detected.

6.3 A grammar for time expressions in Swedish

In this section, I present the idea for the structuring of the time expression detection.

The first thing the engine does is to create a small dictionary with the primitives of Swedish time morphology. These primitives are combined according to the morphological rules for Swedish time expressions, and a large dictionary is built. Rules using regular expressions applied to strings decode temporal expressions encoded in strings such as *2002-02-13*. A set of more complex rules use these basic time expressions to detect more complex time expressions, and a small set of rules at the top level "chains" time expressions, allowing for "long" time expressions.

The following sections present the rules in the different layers.

```
Day-of-week := måndag (DOW=1), ..., söndag (DOW=7)
Month := januari (MON=1), ..., december (MON=12)
Part-of-day := morgon (AM), ..., eftermiddag (PM), ...,
               kväll (PM)
Named-day-of-year := nyårsdag (DATE=01-01), ...,
                     luciadag (DATE=13-12), ...

DOWsPOD := {Day-of-week}s{Part-of-day}
  ( => måndagsmorgon (DOW=1, AM), ...,
       söndagskväll (DOW=7, PM) )
DOWsPODen := {DOWsPOD}en
  ( => måndagsmorgonen (DOW=1, AM), ...,
       söndagskvällen (DOW=7, PM) )
NAMED-DOYen := {Named-day-of-year}en
  ( => nyårsdagen (DATE=01-01), ...,
       luciadagen (DATE=01-01) )
NAMED-DOYsPODen := {Named-day-of-year}s{Part-of-day}en
  ( => nyårsdagsmorgonen (DATE=01-01, AM), ...,
       luciadagskvällen (DATE=13-12, PM), ... )
```

Figure 6.1: Idea for building the dictionary of time expressions.

6.3.1 Morphological time primitives

The morphological time primitive encode semantic information, and when larger expressions are built, the meanings of the terms are merged.

Figure 6.1 illustrates this process informally. First, the primitives are enumerated along with their semantic meaning, then more complex time expressions are created from these primitives. The more complex time expressions (`DOWsPOD, DOWsPODen, ...`) carry the meaning of both their constituents.

6.3.2 Simple rules decoding strings using regular expressions

In addition to the basic time expressions in the dictionary, rules that use regular expressions to extract the time expressions are used. Figure 6.2 illustrates two rules using regular expressions.

Both rules supply a "decoding-hint" for the engine. The engine ignores anything but H/HH (hour), mm (minutes) and M/MM (month) in the hint, and expects to find something properly formatted at that index in the string. To keep the complexity low, the "decoding-hint"-functionality is simple, and thus two rules are needed.

6.3.3 Complex rules

A number of more complex rules use the rules defined earlier to interpret relative time expressions such as *sent i måndags eftermiddag* (*late last Monday afternoon*). Figure 6.3 shows the rule that matches this very time expression.

```
<!-- 9-tiden -->
<rule class="NUM-tiden #1">
  <string regexp="[0-9]-tiden" ref="R1"/>
  <simpletime source="R1" decodinghint="H-tiden" min="0"/>
</rule>

<!-- 19-tiden -->
<rule class="NUM-tiden #2">
  <string regexp="[0-2][0-9]-tiden" ref="R1"/>
  <simpletime source="R1" decodinghint="HH-tiden" min="0"/>
</rule>
```

Figure 6.2: Two rules for detection of time expressions.

```
<!-- i måndags eftermiddag, i tisdags förmiddag, ... -->
<rule class="LAST DOWs POD">
  <string value="sent|tidigt" opt="yes"/>
  <string value="i"/>
  <string class="DOWs" ref="R1"/>
  <string class="POD" ref="R2"/>
  <time>
    <computation>
      <reftime reftime="doctime" op="set"/>
      <term source="R1^R2" op="findfirstprecedingorequal"/>
    </computation>
  </time>
</rule>
```

Figure 6.3: Complex rule for time expression detection.

```
<!-- All simple point-in-time-classes can be referred to by
     one name -->
<rule class="SIMPLE SINGLE POINT-IN-TIME">
  <string class="A DOW|A DOWsPOD|NUM-tiden #1|NUM-tiden #2|..."
          ref="R1"/>
  <simpletime source="R1"/>
</rule>

<!-- recursive rule for simple "point-in-time" time
     expressions -->
<!-- example: "en fredagseftermiddag klockan 17" -->
<rule class="CHAINED SIMPLE TIME EXPs">
  <string class="SIMPLE SINGLE POINT-IN-TIME" ref="R1"/>
  <string class="CHAINED SIMPLE TIME EXPs|
                 SIMPLE SINGLE POINT-IN-TIME" ref="R2"/>
  <simpletime source="R1^R2"/>
</rule>

<!-- top level rule for simple time expressions -->
<rule class="SIMPLE TIME">
  <string class="CHAINED SIMPLE TIME EXPs|
                 SIMPLE SINGLE POINT-IN-TIME" ref="R1"/>
  <simpletime source="R1"/>
</rule>
```

Figure 6.4: Top level time expression detection rules.

The idea encoded in the computation in Figure 6.3 is that the resulting time expression is first initialized with the document time, then a search is performed backwards from the document time for the first time expression matching the predicate $R1 \cap R2$.

6.3.4 Top level rules

All the declared rules are used by three rules at the "top level". Figure 6.4 shows them. The basic idea is to use all the other rules as primitives and to chain and merge their meanings.

The first rule in Figure 6.4 is shortened. It actually has 30 different rules to choose amongst, most of them as simple as the `A DOWsPOD` presented in Figure 6.1.

6.3.5 Building the rule database

Swedish time expressions are easy to detect in that time expressions often are non-ambigous. Thus, writing rules ought to be easy.

The task of figuring out what rules are needed was a bit complicated at first. But, the Carsim program was extended so that it could be run in batch mode, able to process several texts in each run. When run in batch mode, the TimeCore engine outputs an HTML-formatted log. Detected entities are

(DOCTIME=tid="t0" type="DATE" value="2002-11-29" debug="W48-5 doy=333")

en bilolycka med tre bilar inträffade på motorvägen förbi lund [TIMEX tid="t1" type="TIME" value="2002-11-29THH:MM" debug="W48-5 doy=333 PM pod=5"] på fredagseftermiddagen [/TIMEX] . minst en person skadades och trafiken påverkas [TIMEX tid="t2" type="DATE" value="2002-11-29" debug="W48-5 doy=333"] just nu [/TIMEX] rejält . enligt polisens första rapporter har ingen skadats allvarligt .

(In English: *A car accident involving three cars occurred at the highway past Lund [TIMEX ...]during the Friday afternoon[/TIMEX]. At least one person was hurt and the traffic is [TIMEX]currently[/TIMEX] heavily affected. According to the first reports from the police noone has been heavily injured.)*

Figure 6.5: Using verbose output to iteratively develop rules.

marked in color with lots of debug information. Figure 6.5 shows part of an example batch run log.

The rules were extended and debugged until almost all time expressions were found. A simple iterative process was used; the rules were changed, a batch run was made, the results were evaluated, and the rules were changed again.

Some 250 texts, some from the Nyheter corpus (the Carsim corpus), some from the SUC1A corpus (Källgren, 1992) were used, and it was actually quite easy to find and interpret all time expressions.

The document time

The document time deserves a special mentioning. It is the most important time expression in the document as almost all "underspecified" time expressions (such as *Monday*) are computed relative to the date of the document .

In the Nyheter corpus, the document time is not stored along with the text of its document, but it is available in a separate string. The formatting of the document time string is not standardized, generally the document time string was copied directly from the source web page, and hence the *source's* date formatting is used in the document time string.

As the time expression can take almost any form, the easiest way to interpret the document time is actually to use the entire time expression detection machinery. Thus, the time expression detector is first applied to the document time "text".

After the document time has been detected, the time expression detector is applied to the real text.

6.3.6 Size of the rule database

The time expression detection submodule has 18 enumerations (with a total of 113 enumerated elements), with 16 derivations from these (e.g., the finite form of the days of the week are derived from the indefinite form of these).

Texts	Detected	Partly detected	Total undetected	Undetected by design
R001-R050	134	7	29	26
R150-R200	77	3	16	13

Table 6.2: Performance for time expression detection.

70 rules supplement the enumerations and derivations. Most of them are really simple, some are complex, and a few are recursive.

6.4 Results

The performance was evaluated on some of the last texts of the Nyheter corpus (R150 - R199), but as the engine performed very well for those texts (the corpus is biased, the first reports are more complex), the performance was also evaluated it on the *first* 50 texts, texts that are a bit more complex but part of the training set.

6.4.1 Evaluation problems

The common measures of recall, precision and the F-measure are a bit unsuitable for the task of evaluation as partly detected time expressions really can't be classified as either correct or wrong.

Most parts of the Carsim program actually uses only the knowledge of the *existence* of a time expression, and hence the simple detection of only a part of the time expression is sufficient. It is therefore reasonable to count the partly detected entities as being correct, as most parts of the program will be satisified with the detection in those cases.

But, as they actually aren't correct, the measures are in the evaluation computed once with the partly detected entities classified as being correct, once with them counted as being erroneous.

6.4.2 Performance for time expression detection

Table 6.2 shows the results from the evaluation of time expression detection. In the table, "undetected by design" are the time expressions for which no attempt at detection was attempted, such as *summer of 1994* or *for five hours*.

Note that "undetected by design" is a subset of "total undetected".

If the time expressions that were "partly detected" are counted as being erroneous, the time expression detector achieves a recall of 95.4%, a precision of 82.4% and an F-measure of 88.4%.

If the partly detected are perceived as being correct in that at least one time expression was detected in the phrase in question, the time expression detector achieves a recall of 100%, a precision of 82.4% and an F-measure of 90.3%.

Texts	Completely	Partly detected	Erroneously detected
R000-R049	102	30	2
R150-R199	63	14	2

Table 6.3: Performance for interpretation of detected time expressions.

6.4.3 Performance for interpretation of the fully detected time expressions

Table 6.3 shows the results for the evaluation of the interpretation of fully detected time expressions. In the table, the "partly detected" time expressions are cases where the engine interprets only part of the time expression correctly.

If the partly detected time expressions are counted as being erroneous, the intepretator achieves a recall of 78.1%, a precision of 100% and an F-measure of 87.7%.

If only the erroneously interpreted time expressions are counted as erroneous, the recall is 98.1%, the precision again 100% and the F-measure achieved is 99.0%.

Chapter 7

Event detection

This chapter describes and evaluates the detection and interpretation of events. It starts with a declaration of what things are regarded as events in this context.

7.1 What is an event?

> *If it be said that 'strike, turn, run,' are verbs because they denote temporary or short-lasting events, i.e., actions, why then is 'fist' a noun? It also is a temporary event . . . It will be found that an "event" to US means "what our language classes as a verb" or something analogized therefrom.*
>
> *(Benjamin Lee Whorf, 1940)*

The above quote is taken from a paper by Whorf (1940), part of which is included in Appendix B. He argues that events, "things happening", can be "encoded" using verbs and/or nouns.

It can indeed be argued that a car is an event; the car-event "happens" from the time it is constructed until the time it is scrapped. Suffice to say, the threshold for what constitutes an event can be set quite arbitrarily.

In this context, all verbs and all verb groups and a select number of nouns are regarded as events. The nouns detected are nouns dealing with the accident proper, nouns such as "crash".

7.2 Detection of verb encoded events

The rules used for detecting verb phrases as well as stand-alone verbs were developed in the same way as the rules for time expression detection. Carsim was run in batch-mode, the detected events were logged in the HTML-file tagged with bright colors, the results were evaluated, the rules were refined and the batch job run again. Currently, the engine correctly detects most verb phrases as well as all stand-alone verbs.

It is worth noting that the engine creates only one `MAKEINSTANCE` per event. This is a simplification, but I could find no case in the Nyheter corpus where more than one `MAKEINSTANCE` would be appropriate.

7.2.1 Extraction of TimeML event features

Extracting the tense and aspect for events is not too difficult a task, they are mostly encoded morphologically in the verb(s). The task of extracting the TimeML class feature, though, is very difficult. But, the corpus in use (Nyheter), makes the annotation task easier:

1. The texts describe traffic accidents. Thus, almost all events that occur are *unintentional*. This avoids a whole class of TimeML difficulties.
2. The texts are factual and they generally describe what happened during the accident. In general, there are no speculations or hypothetical scenarios in the narrative. Thus, another class of TimeML class difficulties is avoided.
3. The texts makes very little use of the historic present tense and, surprisingly, very little use of the future tense. This also simplifies the mark-up task.

The approach finally used was to assign features on a per-rule basis. To examplify, one heuristic in use is that predicative clauses (copulas) normally are states (*I* am *alive*). This works surprisingly well, as presented in 7.4.2.

7.2.2 Extration of other features

In addition to the TimeML features, the morphological "structure" of the verb phrase is stored in the event objects. This allows the parts of the engine that orders events to discern between cases such as stand-alone verbs or predicative clauses, information that might be useful.

7.3 Detection of noun encoded events

The noun encoded events that are directly relevant for the purposes of the Carsim program are the only nouns detected as events. Thus, the event "threshold" was set at events such as *accident* and *collision*, events such as *queue* and *injury* were perceived to be above the threshold (the texts that used the word *queue* in direct connection to the accident were one or two out of 200, in the other texts the queues were results of the accident).

The noun events are detected using simple regular expressions that tests both the part of speech tag (assuring that the POS tagger thinks it's a noun) and the lemma of the noun, comparing it to the list of nouns it recognizes.

7.4 Results

When evaluating the performance of the event detection, two aspects are important. The first aspect is that of the detection itself, the second is that of the performance of the feature extraction, i.e. interpretation, for the events detected.

The detection of events is easy given the definition of events above (i.e. all verbs plus a select number of nouns are to be detected), and thus the fact that the results are very good for event *detection* shouldn't be too surprising.

Feature	$N_{correct}$	$N_{erroneous}$	Percent correct
Tense	179	1	99.4%
Aspect	161	19	89.4%
Class	150	30	83.3%

Table 7.1: Evaluatation results for feature detection for 180 events.

7.4.1 Detection

When evaluating the event detector on 40 texts from the Nyheter corpus (R150-R189), 584 events were detected. 3 events were overdetected, and 26 events were undetected. The event detector thus reaches a recall of 95.7%, a precision of 99.4% and an F-measure of 97.5% for the event *detection*.

The undetected events were generally on the form *the victims are safe and happy*. In this case, the engine currently only finds one event; *are safe*. It should either find the *are safe and happy* event or it should find two events (*are safe* and *are happy*) and link them with a "simultaneous" `TLINK`.

The overdetected events were the fault of the POS-tagger, that erroneously had classified words as verbs. An example from a text is *jag visste inte var jag var* (*I didn't know where I was*). When this phrase occurs in a subclause, the tagger believes that both *var*:s are verbs when in fact the first one is an adverb of place.

7.4.2 Interpretation of detected events

The performance of the interpretation of detected events was evaluated on 20 texts from the Nyheter corpus (R150-R169). In total, 180 events were detected, and the misclassifications made are presented in Table 7.1.

It is worth noting that I did the evaluation myself, and that the task involves a part of subjectivity. There were however, given my means, no other ways to do it.

Chapter 8

Event ordering

The final step in the TimeCore module is the creation of a temporal ordering of events. In TimeML, temporal orderings are encoded as `TLINKs`, and they should of course be non-conflicting.

The TimeCore module creates `TLINKs` mainly using decision trees. The trees are given features of sequences of events as well as some other measures believed to be useful, and are asked to decide the temporal relation holding between two of the events in question.

The trees are never applied to sequences of events crossing time expressions, as the time expressions might change the order of events significantly.

In addition to the trees, the module uses domain knowledge to create `TLINKs`. It is e.g. assumed that the texts describe only one accident, and thus all nouns ending with *olycka* (*accident*) are assumed to refer to the same entity and are linked with an "identity" TLINK. Carsim's "New Swedish" language module is aware of other identity relations, and the engine creates `TLINKs` for those relations as well.

The `TLINKs` not created by decision trees span arbitrarily large stretches of text, enabling ordering of events far apart, events possibly separated by time expressions.

8.1 The decision trees

Currently, there are five decision trees in total. Figure 8.1 shows the trees and Figure 8.2 shows the links they create. The first tree (DT1 in Figure 8.1) considers two adjacent[1] events and orders them. A second and a third tree (DT2 and DT3) order adjacent events considering features of the two events as well as features from the preceding and succeeding event, respectively. A fourth tree (DT4) orders two events separated by a third event, using features from all three events. The fifth tree (DT5) orders events separated by two other events, using features from all four events in question.

It is important to note that the trees are never applied across time expressions. This means that DT1 can be applied more often than the others as it only requires two events in sequence as opposed to e.g. DT5, which requires a sequence of four events.

[1] Adjacent in the narrative order of the text

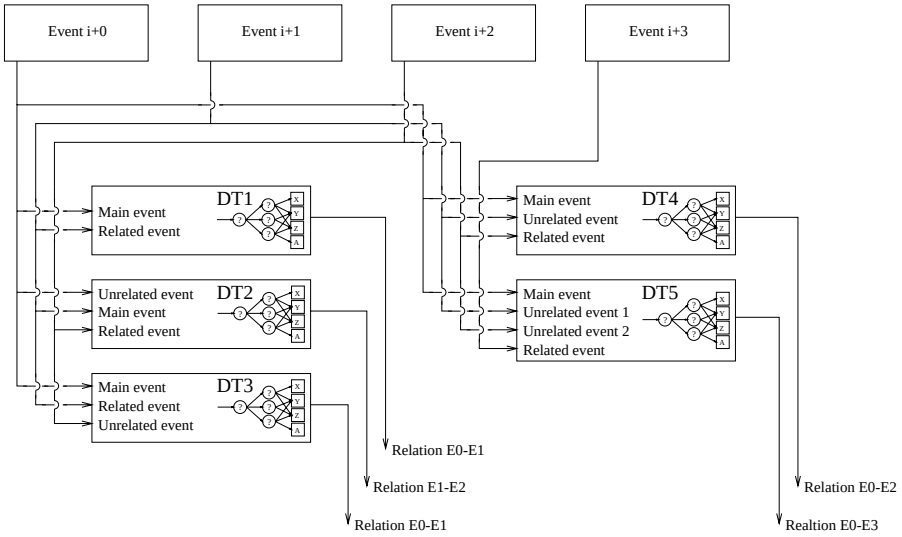


Figure 8.1: The five decision trees in use

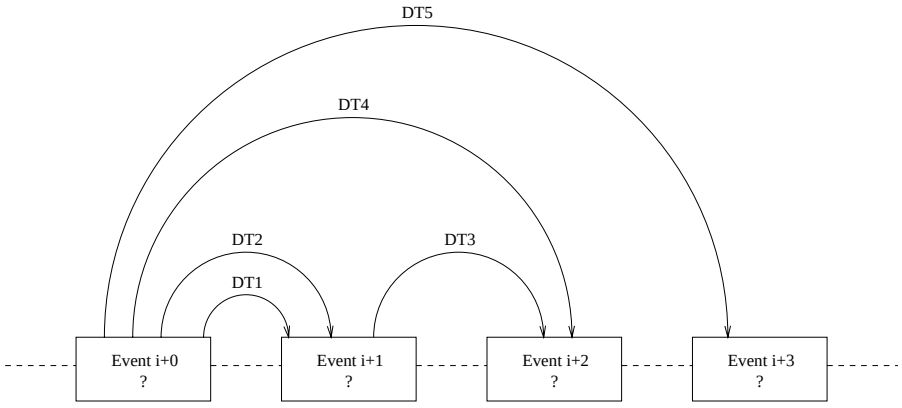


Figure 8.2: The links created by the five decision trees

The fact that DT1 can be applied in corner cases where DT2 and DT3 don't "fit" motivates having three different trees (DT1-DT3) that order adjacent events; the trees using input from three events ought to perform better, but they can't be applied as often as DT1.

The motivation for having trees that order event spaced further apart (DT4, DT5) is that the resulting ordering can be more fine-grained.

8.1.1 Training the trees

Figure 8.3 shows the input data for the simplest tree, the tree that orders two adjacent events using features from just those two events (DT1). In addition to the features of the events, the trees are given some measures believed to be useful as well as an indication of what temporal signals were found between the events.

```
mainEventTense = {none, past, present, future, ND, UNKNOWN}
mainEventAspect = {progressive, perfective,
  perfective_progressive, none, ND, UNKNOWN}
mainEventStructure = {NOUN, VB_GR_COP_INF, VB_GR_COP_FIN,
  VB_GR_MOD_INF, VB_GR_MOD_FIN, VB_GR, VB_INF, VB_FIN,
  UNKNOWN}
relatedEventTense = {none, past, present, future, ND,
  UNKNOWN}
relatedEventAspect = {progressive, perfective,
  perfective_progressive, none, ND, UNKNOWN}
relatedEventStructure = {NOUN, VB_GR_COP_INF, VB_GR_COP_FIN,
  VB_GR_MOD_INF, VB_GR_MOD_FIN, VB_GR, VB_INF, VB_FIN,
  UNKNOWN}
temporalSignalInbetween = {none, before, after, later, when,
  still, several}
tokenDistance = {1, 2t3, 4t6, 7t10, gt10, UNKNOWN}
sentenceDistance = {0, 1, 2, 3, 4, gt4, UNKNOWN}
punctuationSignDistance = {0, 1, 2, 3, 4, 5, gt5, UNKNOWN}
```

Figure 8.3: The features used by decision tree 1. DT1 decides the relation between two events using features from only those events.

Note that the trees are given the morphological structure of the events instead of the TimeML class. The two contain more or less the same data, but as the number of morphological structures are more than the number of classes, the structure carries more information.

The tree returns a string in the set {`simultaneous`, `after`, `before`, `included_by`, `includes`, `none`, `UNKNOWN`} as the value for the relation between the main event and the related event. TimeML allows for more fine-grained ordering using relations such as `begun_by` and `i_after` (*immediately after*), but to keep the complexity low these were omitted. `none` and `UNKNOWN` are not part of TimeML, and they are used when no other classification is reasonable.

The other trees are given similar indata, although they also are given the features of the other events involved in the query.

To create the trees from this data, I used the freely available decision tree generator c4.5 by Quinlan (1992). Figure 8.4 shows part of c4.5's output for decision tree DT1.

The trees are applied to Swedish texts, but part of the tree for DT1 actually work for English too. Figure 8.5 shows how the tree is applied to the the English sentences "*He came. He ate*", and Figure 8.6 shows how the tree is applied to the sentences "*He came. He had eaten*".

The c4.5 program also outputs the pairs of numbers after the leaves of the decision trees. The first number is the "weight" of all queries reaching the leaf in question whereas the second is the weight of the queries that were erroneosly answered. These numbers do not correspond directly to the number of times the leaf is reached, but they are an indication of the accuracy of the leaf.

These numbers are used to compute a "score" for every branch. The score,

```
mainEventTense = past:
|   relatedEventTense = present: before (42.0/10.4)
|   relatedEventTense = future: before (0.0)
|   relatedEventTense = past:
|   |   relatedEventAspect = progressive: before (145.0/73.7)
|   |   relatedEventAspect = perfective: after (7.0/6.1)
|   |   relatedEventAspect = none: before (21.0/5.9)
|   |   relatedEventAspect = perfective_progressive:
|   |   |   sentenceDistance = 0: simultaneous (6.0/2.3)
|   |   |   sentenceDistance = 1: before (2.0/1.8)
|   |   |   sentenceDistance = 2: simultaneous (0.0)
|   |   |   sentenceDistance = 3: simultaneous (0.0)
|   |   |   sentenceDistance = 4: simultaneous (0.0)
|   |   |   sentenceDistance = gt4: simultaneous (0.0)
mainEventTense = present:
|   relatedEventTense = none: after (16.0/4.8)
|   relatedEventTense = past: after (37.0/13.5)
|   relatedEventTense = present: simultaneous (56.0/20.0)
|   relatedEventTense = future: simultaneous (0.0)
```

Figure 8.4: Part of c4.5's output for decision tree 1.

Text:	*He came. He ate.*	
Analysis:	came:	tense = past, aspect = progressive
	ate:	tense = past, aspect = progressive
Decision tree:	mainEventTense = past	
	relatedEventTense = past	
	relatedEventAspect = progressive =>	
	mainEvent **before** relatedEvent =>	
	came **before** *ate*	

Figure 8.5: Applying decision tree 1 to the sentences *He came. He ate.*

Text:	*He came. He had eaten.*	
Analysis:	came:	tense = past, aspect = progressive
	had eaten:	tense = past, aspect = perfective
Decision tree:	mainEventTense = past	
	relatedEventTense = past	
	relatedEventAspect = perfective =>	
	mainEvent **after** relatedEvent =>	
	came **after** *had eaten*	

Figure 8.6: Applying decision tree 1 to the sentences *He came. He had eaten.*

computed as $weight_{correct}/weight_{total}$, is used when temporal loops are to be resolved. Each generated `TLINK` will have a score of $score_{tree} * score_{answer\_branch}$, where $score_{tree}$ is $1-\{$c4.5's error estimate for the final tree$\}$. If the branch has a weight of 0.0, no queries reached that branch. The score is then set to the arbitrarily chosen value of 0.2.

The program uses these scores when resolving temporal loops. The `TLINKs` with the lowest score are removed iteratively until all conflicts are resolved.

8.1.2 Training and generation of decision trees

To train and generate the trees, c4.5 is used, and only two very simple "glue" scripts are needed to achieve machine learning capability for the TimeCore module.

Figure 8.7 shows how the trees are trained and how they are integrated with Carsim. The training process begins with a batch run of Carsim where all queries made to the trees are logged along with the features given in each query. A script merges the log with data from a database of temporal relations, creating an input file compliant with c4.5.

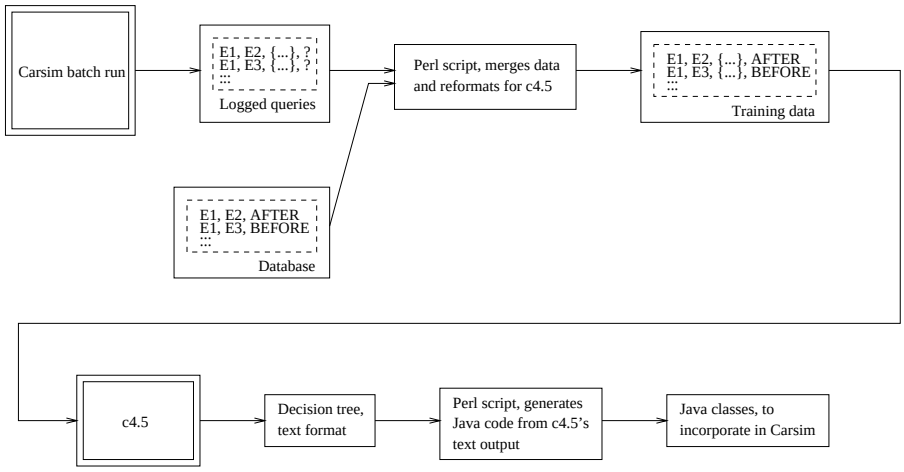


Figure 8.7: The tree generation process

c4.5 outputs a decision tree in a text format of its own (shown in 8.4), and that output is transformed into Java code by a second script. The generated Java code is copied back into the Carsim source code tree, after which Carsim is recompiled to use the new trees.

The process shown in Figure 8.7 is a bit simplified in order for it to be clearly understandable. The identifiers of the events may be different each run because of e.g. changes in the control file for the engine, causing an alternative set of events to be detected. Therefore, it is actually the *token number* of the event that is used to match queries and expected results when creating the input file for c4.5. The identifiers of the events are removed from the input data to c4.5, as they are of no use to c4.5, a step which is not shown in the figure.

Tree	Size	Errors_{final}	c4.5's error estimate
DT1	449	36.3%	44.2%
DT2	382	37.5%	46.1%
DT3	384	39.3%	46.0%
DT4	220	30.9%	47.5%
DT5	221	34.5%	46.2%

Table 8.1: Training set sizes and error rates for decision trees DT1-DT5.

8.1.3 Training set and performance

I built a training set where I annotated 27 texts from the Nyheter corpus with temporal relations. In cases where the relation between was difficult to classify, that relation was marked as "to be skipped", and the logged query between those events would be removed from the training set.

The final training sets for the trees also varies because of the number of matches made; DT1 is applied many more times than e.g. DT5.

Table 8.1 shows the final training set sizes, the final error rates for the trees as well as c4.5's error estimate for the final tree.

The reason that DT2 and DT3 have different training set sizes although they are applied exactly as many times is that some relations queries were removed from the training set, as described earlier.

The error rate presented in the table is quite high, but so is the difficulty of the task. The strategy relies on the redundancy of the trees and the assumption that the `TLINKs` with the higher scores are correct when they conflict with links with lower scores. The conflicting `TLINKs` with the lower scores are removed when temporal loops are resolved.

The results by Mani et al. (2003), when applying decision trees to a similar task, suggests that much better results can be achieved.

8.2 Resolving temporal loops

Figure 8.8 shows the 12 `TLINKs` that can be expected between a chain of four events. It is not unreasonable to expect some of these `TLINKs` to conflict, and therefore there is a need to remove some of them.

Figure 8.9 shows the pseudo code for the loop removal. Instead of *removing* `TLINKs`, the `TLINKs` are actually *added* to an initially empty set if their inclusion wouldn't introduce temporal conflicts. The `TLINKs` with the highest scores are added first, thus "removing" the conflicting `TLINKs` with the lowest score.

The idea behind this is Darwinistic; in the struggle for survival the strongest persist.

8.3 Results

8.3.1 Evaluation strategy

The evaluation of of the final ordering is non-trivial. This section discusses the strategy used for evaluation.

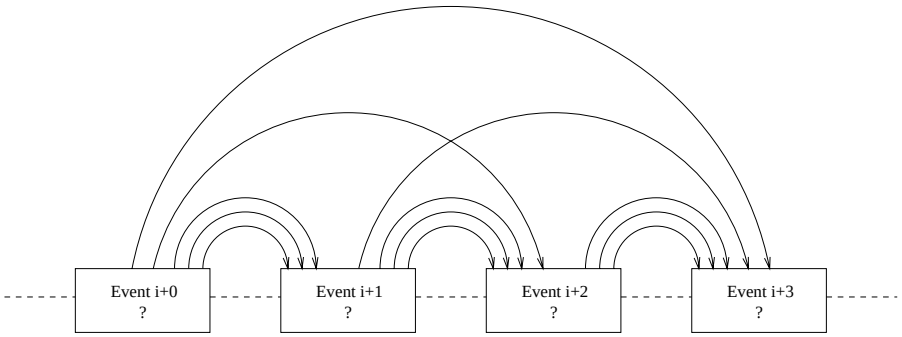


Figure 8.8: Between a stretch of four events, 12 `TLINKs` can be expected.

```
current_score = 100;
while current_score > 0 {
  next_score = -1;
  for all TLINKs do {
    if (TLINK.score == current_score) {
      if (introductionOfLinkWouldn'tCauseALoop) {
        addLink(TLINK);
      }
    }
    else if (TLINK.score < current_score &&
             TLINK.score > next_score) {
      next_score = TLINK.score;
    }
  }
  current_score = next_score;
}
```

Figure 8.9: Pseudo code for temporal loop resolving.

Figures 8.10 and 8.11 show texts R123 and R129 from the Nyheter corpus along with screenshots of the final event ordering as presented by the Carsim program. The figures are used in the discussion regarding the evaluation strategy, and an understanding of the way event chains are shown in the graphs is necessary.

Thus, before the discussion starts, I explain how the graphs work.

Explanation of the Carsim event order graph output

Figures 8.10 (page 43) and 8.11 (page 44) show the result of the parsing of texts R123 and R129. The graphs at the top of the figures are actual screenshots from a run of the Carsim program. The boxes in the graphs represent detected events. The number in front of the strings in the boxes are the *token number* of the string, with the numbering starting at 1 for the first token. When several strings share a box, they are (deemed) simultaneous events. The relations includes/included_by can also be shown in graphs, but no such relation is present in the two graphs, and thus their rendering need not be explained in

this context.

A line connecting two boxes means that the event in the first box preceeds the last box. The very first box (*Start*) is the start of the universe, the last box (*End*) represents the end of time.

Evaluation idea

Evaluating the final event ordering using the common measures of recall, precision and F-measure is difficult; there is, as I'll try to show, no clear path to follow.

Let's begin in Figure 8.10. In the figure, both @26:klämdes (was jammed) and @32:kom (came) are correctly ordered with respect to @14:körde (drove) and @47:var (were). But, they are ordered incorrectly in respect to each other.

Are they correct or are they wrong? Is maybe only one erroneous?

The approach I used when evaluating was to consider all possible event tuples and to count how many of these event tuples were ordered correctly respectively incorrectly in the graphs. Sometimes there was no defined order between the events, and then that relation was counted as missing. The measure of recall, to be used in the evaluation, requires that items not found are counted, and thus the undefined relations are counted as missing.

The missing relations are the actual problem. In Figure 8.11 (page 44), the absence of an ordering between events @3:fördes (were taken) and event @47:stängdes (was closed) is appropriate. However, in Figure 8.10 there ought to be an ordering between @2:trafikolycka (traffic accident) and @14:körde (drove), but that ordering is missing.

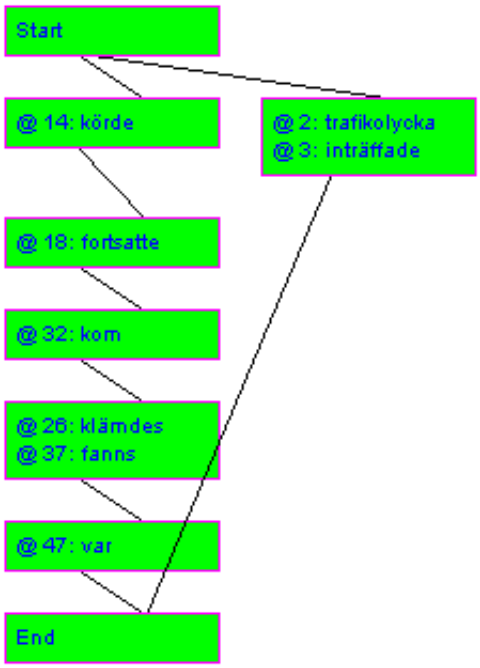
The method I chose for dealing with the missing links was to count them all as "missing"; neither correct nor wrong. Instead, the goal set for the program was to create an *acceptable* ordering between all events. One interpretation of the text in Figure 8.11 is that the road was closed (@47:stängdes) after the four persons were taken to the hospital (@3:fördes), that ordering would then count as being correct. As the reverse ordering also is acceptable, it too would have been counted as correct.

This method of evaluation is obviously flawed, but I could find no better way to deal with the missing links. The results, however, ought to give at least an indication of the merits of the decision tree approach.

In a personal communication, Inderjeet Mani pointed out that using an explicit `UNKNOWN` relation in the evaluation could solve some difficulties. This relation would be assigned to all ambiguous and missing relations as well as to those that really are unrelated, and thus the "missing relation" problem would be solved. Currently, no `TLINKs` are created from those relations that are classified as `UNKNOWN` by the trees, but actually creating `TLINKS` from these and keeping track of those links throughout the loop removal would improve the evaluation.

Formalizing the evaluation idea

When creating a full ordering between n events, $\frac{n(n-1)}{2}$ non-conflicting tuples ought to be created. But, as several alternative interpretations of the ordering between the events can be made, the goal should "only" be to create $\frac{n(n-1)}{2}$ *acceptable* tuples.

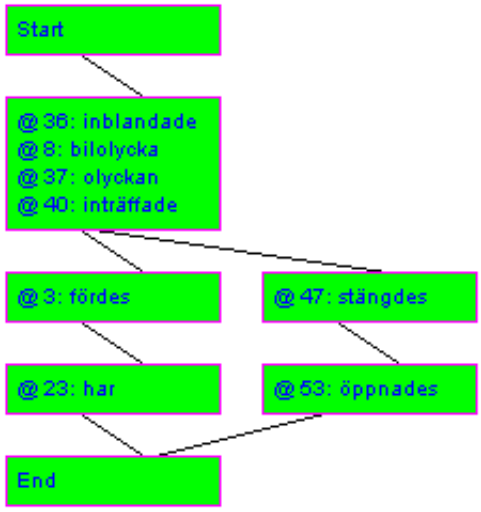


R123 Swedish En **trafikolycka**$_2$ **inträffade**$_3$ i snöovädret vid Fårö kyrka i går förmiddag. En bil **körde**$_{14}$ av vägen och **fortsatte**$_{18}$ in i ett träd varpå en person **klämdes**$_{26}$ fast. Räddningstjänsten och ambulans **kom**$_{32}$ på plats. Det **fanns**$_{37}$ under gårdagskvällen inga uppgifter på hur pass allvarliga personskadorna **var**$_{47}$.

English A **trafic.accident**$_2$ **occured**$_3$ in the.snow.bad.weather by Fårö church yesterday forenoon. A car **drove**$_{14}$ off the.road and **continued**$_{18}$ in into a tree after.which a person **was.jammed**$_{26}$ stuck. The.rescue.service and ambulance **came**$_{32}$ to the.site. There **were**$_{37}$ during yesterday.evening no reports regarding how serious the.person.injuries **were**$_{47}$.

Note. The translation to English is done word-for-word as the order and indices of the tokens are important.

Figure 8.10: Text R123 in Swedish and English, along with its event chain graph.



R129 Swedish Fyra personer **fördes**$_3$ till sjukhus efter en **bilolycka**$_8$ på riksväg 66 vid Erikslund i Västerås vid tiotiden på söndagsförmiddagen. Enligt polisen **har**$_{23}$ ingen av dem livshotande skador. Två personbilar och en lastbil var **inblandade**$_{36}$ (1) **olyckan**$_{37}$, som **inträffade**$_{40}$ på Riksväg under E18 (2). Vägen **stängdes**$_{47}$ av från olyckplatsen söderut men **öppnades**$_{53}$ igen efter ett par timmar.

English Four persons **were.taken**$_3$ to hospital after a **car.accident**$_8$ on national.highway 66 by Erikslund in Västerås at the.ten.time on Sunday.forenoon. According [to] the.police **have**$_{23}$ none of them life.threatening injuries. Two person.cars and a truck were **involved**$_{36}$ (1) **the.accident**$_{37}$, which **occurred**$_{40}$ on national.highway under E18 (2). The.road **was.closed**$_{47}$ off from the.accident.site southwards but **was.opened**$_{53}$ again after a couple [of] hours.

Note. The translation to English is done word-for-word as the order and indices of the tokens are important. Also note that the necessary preposition *i* (*in*) is missing in (1) and that the subclause (2) is entirely ungrammatical. This is because the texts have been taken directly from the net without change.

Figure 8.11: Text R129 in Swedish and English, along with its event chain graph.

Text	N_{tuples}	N_{found}	$N_{missing}$	$N_{correct}$	$N_{erroneous}$
R120	55	37	18	29	8
R123	28	16	12	15	1
R126	55	45	10	29	10
R129	28	16	12	9	7
R150	55	20	35	14	6
R155	36	16	20	11	5
R160	45	29	16	12	17

Text	Precision	Recall	F-measure
R120	0.78%	0.53%	0.63%
R123	0.94%	0.54%	0.68%
R126	0.64%	0.53%	0.58%
R129	0.56%	0.32%	0.41%
R150	0.70%	0.25%	0.37%
R155	0.69%	0.31%	0.42%
R160	0.41%	0.27%	0.32%
Mean	0.49%	0.67%	0.39%

Table 8.2: Evaluation results for the naïve algorithm.

With this thinking, precision is $N_{acceptable}/N_{all\_generated}$, and recall is $N_{acceptable}/\frac{n(n-1)}{2}$. The F-measure is defined as normally. Unknown relations are counted as missing although some of them might rightfully be so.

8.3.2 Evaluation of final ordering

The baseline with which the engine was compared is a simple algorithm that assumes that everything is correctly ordered but doesn't dare make any guesses across time expressions. Guessing that would have created a poorer-performing algorithm.

Both algorithms (naïve and decision tree) were evaluated on the same seven texts, two of which were presented in Figures 8.10 and 8.11. Table 8.2 and Table 8.3 show the results for the naïve algorithm and the decision tree based algorithm, respectively.

The numbers vary much, as the texts in general are short, and thus one error more or less has a large effect. The mean values for precision, recall and F-measure, however, are consistently better for the decision tree based algorithm.

8.3.3 What about the rest of the world?

Inter-annotator agreement is known to be problematic in the context of temporal mark up. In one pilot study, Setzer and Gaizauskas (2001) amongst other results report a precision of 0.68 on average for inter-annotator agreement for classification of temporal relations. They used the same set of temporal relations that I used (i.e., a subset of TimeML), and they also used newswire texts, so their measure of precision for inter-annotator agreement is not too far off in this context.

Text	N_{tuples}	N_{found}	$N_{missing}$	$N_{correct}$	$N_{erroneous}$
R120	55	47	8	41	6
R123	28	16	12	13	3
R126	55	42	13	26	16
R129	28	24	4	24	0
R150	55	17	38	14	3
R155	36	24	12	16	8
R160	45	29	16	15	14

Text	Precision	Recall	F-measure
R120	0.87%	0.75%	0.80%
R123	0.81%	0.46%	0.59%
R126	0.62%	0.47%	0.54%
R129	0.86%	0.86%	0.86%
R150	0.82%	0.25%	0.39%
R155	0.67%	0.44%	0.53%
R160	0.52%	0.33%	0.41%
Mean	0.59%	0.67%	0.51%

Table 8.3: Evaluation results for the algorithm that uses decision trees.

The building of the training set for the decision trees as well as the evaluation of the resulting graphs were both done by me alone. Had someone else evaluated the performance of the final ordering, the results would most probably have been somewhat different.

8.4 Conclusion

From my experiments, the decision trees perform better than the naïve algorithm in most cases, but the numbers are somewhat inconclusive as the evaluation method needs to be improved.

Chapter 9

Carsim integration

The final task was to integrate the TimeCore module with the Carsim program. The event ordering found by the TimeCore module was to be used to order the events in the Carsim program.

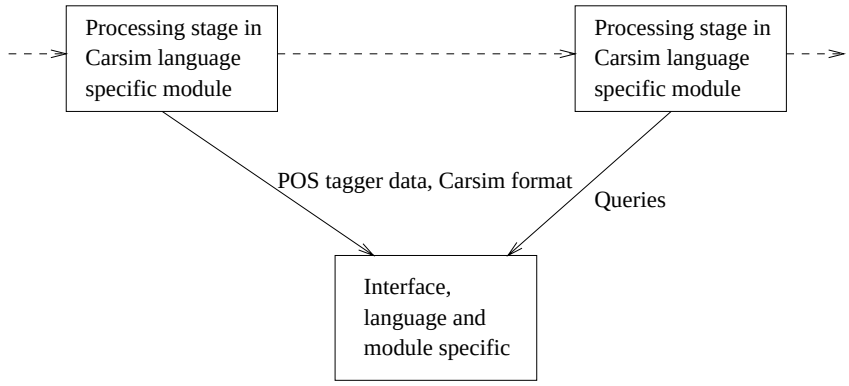


Figure 9.1: Integration with Carsim.

The connection between the Carsim program and the TimeCore module is shown in Figure 9.1. The interface class was developed along with the TimeCore module, and thus only the last step, the handling of queries, required "extra" work.

The queries are used by the Carsim program when it wants to order its list of events. It queries for the temporal order between pairs of Carsim events, and it reorder its list of events according to the answers it gets.

The only problem that arose was that the TimeCore module and the Carsim program detected different events.

9.1 Event matching

Among other methods, the Carsim program uses word spotting to find events. To examplify; Carsim thinks that "fire" (as in "caught fire") is an event. In this

case, the TimeCore engine thinks that "caught" is an event, and there is a need to match the events of Carsim with the events of the TimeCore module.

This is done via a simple back-off mechanism. The Carsim program queries for the event of $token_{fire}$, the TimeCore engine first looks at that token, finds no event, then looks at $token_{index-1}$, then $token_{index+1}$, then -2, +2 etc. until it finds an event that it detected.

This works quite well, when evaluating the performance of this on 40 texts (R150-R189), 49 matches were correct and 4 erroneous (one of the erroneous matches was caused by a POS tagger error). This gives a match correctness of 92.4%, which is acceptable.

9.2 An example run

To show how the TimeCore module is used within the Carsim context, this section cursorily presents how the text R009 is processed by the Carsim program.

Figure 9.2 shows the text in Swedish and English and the resulting event chain graph. Figure 9.3 shows the TimeML-output the graph is based on. In the figure, all `MAKEINSTANCE` tags have been removed so that the output fits on one page. All the `MAKEINSTANCE` tags are on the following form:

`<MAKEINSTANCE eiid="eiX" eventID="eX"/>`, $0 \leq X \leq 11$.

The ordering created by the TimeCore module is used by Carsim's "New Swedish" language processing module to order the events in its template for the animation.

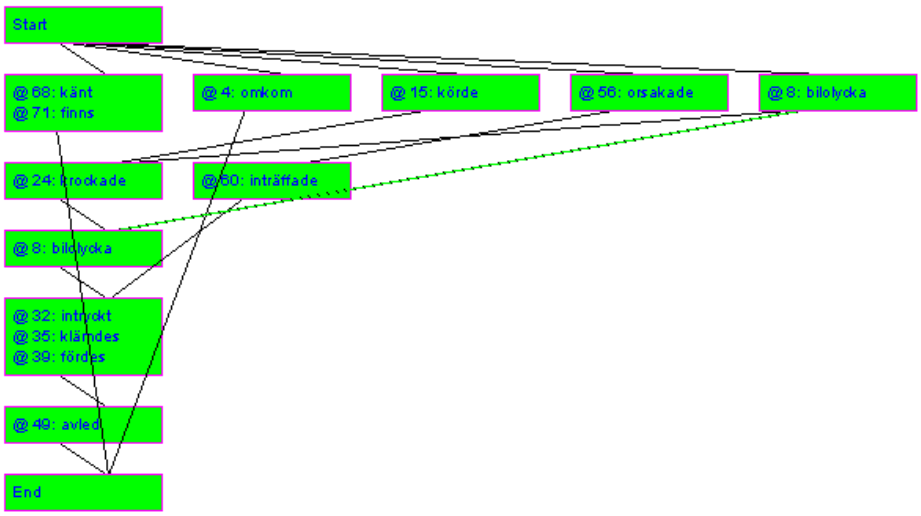
The template for text R009 is shown in Figure 9.4. First, the scene is described, in this case a rural road between Ystad and Simrishamn. Next, the participants objects are listed, in this case a car and a tree. The final set of elements are the events.

The first event is the non-listed starting position. Next, the car leaves the road. `StrictlyAfter` that an impact occurs, where the actor is a car (`RoadObject3`), and the victim a tree (`RoadObject3`).

Figure 9.5 shows three screenshots from the resulting animation. The first screenshot shows the starting position, the second shows the car leaving the road and the third shows the car colliding with the tree.

9.3 Summary

To summarize, integrating the TimeCore module with the Carsim program didn't pose any difficult problems and it didn't require much work.



R009 Swedish En 20-årig man **omkom**$_4$ i dag vid en **bilolycka**$_8$ mellan Ystad och Simrishamn. Mannen **körde**$_{15}$ av vägen vid gamla skolan i Hannas och **krockade**$_{24}$ med ett träd. Bilen blev kraftigt **intryckt**$_{32}$ och föraren **klämdes**$_{35}$ fast. Mannen **fördes**$_{39}$ i ambulans till Ystads lasarett, där han senare **avled**$_{49}$ av sina skador. Vad som **orsakade**$_{56}$ avåkningen, som **inträffade**$_{60}$ vid tiotiden på förmiddagen, är inte **känt**$_{68}$. Det **finns**$_{71}$ inte heller några vittnen till händelsen.

English A 20-year.old man **died**$_4$ today in a **car.accident**$_8$ between Ystad and Simrishamn. The.man **drove**$_{15}$ off the.road at the.old the.school in Hannas and **crashed**$_{24}$ with a tree. The.car became heavily **impressed**$_{32}$ and the.driver **was.jammed**$_{35}$ stuck. The.man **was.taken**$_{39}$ in ambulance to Ystads hospital, where he later **deceased**$_{49}$ from his injuries. What that **caused**$_{56}$ the.drive.off, which **occurred**$_{60}$ at the.ten.time in the.forenoon, is not **known**$_{68}$. There **are**$_{71}$ not either any witnesses to the.accident.

Note. The translation to English is done word-for-word as the order and indices of the tokens are important. The dashed line in the event chain graph indicate "identity"; the crash (@24) happens during the car accident (@8). Also note that the head of predicatives are marked as events, thus e.g. @68:known is tagged as an event instead of @66:is.

Figure 9.2: Text R009 in Swedish and English, along with its event chain graph.

```
<?xml version="1.0"?><!DOCTYPE TimeML SYSTEM "TimeML.dtd">
<TimeML>

En 20-årig man <EVENT eid="e2" class="OCCURRENCE" tense="PAST"
aspect="PROGRESSIVE"> omkom </EVENT> <TIMEX3 tid="t1" type="DATE"
value="2001-10-26"> i dag </TIMEX3> vid en <EVENT eid="e11"
class="OCCURRENCE" tense="NONE" aspect="NONE"> bilolycka </EVENT>
mellan Ystad och Simrishamn .  Mannen <EVENT eid="e3" class="OCCURRENCE"
tense="PAST" aspect="PROGRESSIVE"> körde </EVENT> av vägen vid gamla
skolan i Hannas och <EVENT eid="e4" class="OCCURRENCE" tense="PAST"
aspect="PROGRESSIVE"> krockade </EVENT> med ett träd .  Bilen blev
kraftigt <EVENT eid="e0" class="STATE" tense="PAST" aspect="PROGRESSIVE">
intryckt </EVENT> och föraren <EVENT eid="e5" class="OCCURRENCE"
tense="PAST" aspect="PROGRESSIVE"> klämdes </EVENT> fast .  Mannen <EVENT
eid="e6" class="OCCURRENCE" tense="PAST" aspect="PROGRESSIVE"> fördes
</EVENT> i ambulans till Ystads lasarett , där han <SIGNAL sid="s0">
senare </SIGNAL> <EVENT eid="e7" class="OCCURRENCE" tense="PAST"
aspect="PROGRESSIVE"> avled </EVENT> av sina skador .  Vad som <EVENT
eid="e8" class="OCCURRENCE" tense="PAST" aspect="PROGRESSIVE"> orsakade
</EVENT> avåkningen , som <EVENT eid="e9" class="OCCURRENCE" tense="PAST"
aspect="PROGRESSIVE"> inträffade </EVENT> <TIMEX3 tid="t2" type="TIME"
value="10:00"> vid tiotiden på förmiddagen </TIMEX3> , är <SIGNAL
sid="s1"> inte </SIGNAL> <EVENT eid="e1" class="STATE" tense="PRESENT"
aspect="PERFECTIVE_PROGRESSIVE"> känt </EVENT> .  Det <EVENT eid="e10"
class="OCCURRENCE" tense="PRESENT" aspect="PROGRESSIVE"> finns </EVENT>
<SIGNAL sid="s2"> inte heller </SIGNAL> några vittnen till händelsen .

<SLINK subordinatedEventInstance="ei1" signalID="s1" relType="NEGATIVE"
polarity="false"/>
<SLINK subordinatedEventInstance="ei10" signalID="s2" relType="NEGATIVE"
polarity="false"/>
<TLINK eventInstanceID="ei10" relatedToTime="t0" relType="SIMULTANEOUS"/>
<TLINK eventInstanceID="ei2" relatedToTime="t1" relType="SIMULTANEOUS"/>
<TLINK eventInstanceID="ei11" relatedToTime="t1" relType="IAFTER"/>
<TLINK eventInstanceID="ei9" relatedToTime="t2" relType="INCLUDES"/>
<TLINK eventInstanceID="ei11" relatedToEvent="ei4" relType="INCLUDES"/>
<TLINK eventInstanceID="ei3" relatedToEvent="ei4" relType="BEFORE"/>
<TLINK eventInstanceID="ei11" relatedToEvent="ei0" relType="BEFORE"/>
<TLINK eventInstanceID="ei3" relatedToEvent="ei5" relType="BEFORE"/>
<TLINK eventInstanceID="ei4" relatedToEvent="ei5" relType="BEFORE"/>
<TLINK eventInstanceID="ei4" relatedToEvent="ei6" relType="BEFORE"/>
<TLINK eventInstanceID="ei5" relatedToEvent="ei6"
relType="SIMULTANEOUS"/>
<TLINK eventInstanceID="ei0" relatedToEvent="ei5"
relType="SIMULTANEOUS"/>
<TLINK eventInstanceID="ei0" relatedToEvent="ei7" relType="BEFORE"/>
<TLINK eventInstanceID="ei5" relatedToEvent="ei8" relType="AFTER"/>
<TLINK eventInstanceID="ei6" relatedToEvent="ei7" relType="BEFORE"/>
<TLINK eventInstanceID="ei6" relatedToEvent="ei9" relType="AFTER"/>
<TLINK eventInstanceID="ei8" relatedToEvent="ei9" relType="BEFORE"/>
<TLINK eventInstanceID="ei1" relatedToEvent="ei10"
relType="SIMULTANEOUS"/>

</TimeML>
```

Figure 9.3: Abbreviated TimeML-mark up of text R009.

Template

Template					
Scene	Scene	Location	mellan Ystad och Simrishamn		
		RoadConfiguration	StraightRoad	LeftAttrs	SideAttrs
				RightAttrs	SideAttrs
				Id	RoadConfig4
		Environment	rural		
Objects	Car	Id	RoadObject3		
		IntroducedAs	En 20-årig man		
		Positions			
		Directions			
	Tree	Id	RoadObject4		
		IntroducedAs	ett träd		
		Positions			
Events	LeaveRoad	Id	Event3		
		Actor	(RoadObject3)		
		Positions			
		Directions			
		Times			
	Impact	Id	Event4		
		Actor	(RoadObject3)		
		Victim	(RoadObject4)		
		Positions			
		Directions			
		Times	StrictlyAfter	RelativeTo	(Event3)

Figure 9.4: Carsim New Swedish template for text R009.

The first screenshot shows the start position, the second, the car leaving the road, and the third screenshot shows the car colliding with the tree.

Figure 9.5: Screenshots from animation of text R009.

Chapter 10

Future work

In this chapter, I present some approaches to improve the performance of the engine.

10.1 Detecting more features

Currently, the interpretation of events is quite shallow. The decision trees might benefit from knowing more about the event, thus adding detection of more features might be an idea worth exploring.

10.1.1 WordNet

One approach to finding more features for events is to find a Swedish WordNet (Miller, 1995) work-alike. It might be that the hypernym chains for different words allows the engine to make distinctions between different kinds of events. Although it is difficult to disambiguate between different meanings of a word, a rough classification could still be forwarded to the trees. The trees might learn how to use the "noisy" input data to order events.

This approach was actually attempted, but aborted. The most common nouns and verbs in the SUC1A and Nyheter corpora were extracted and automatically translated to English using an online-web dictionary. The hypernyms of the English words were generated using WordNet 2.0, and the Swedish words were then stored along with the hypernyms of their English counterparts. I stopped the experiment when the Swedish word *antal* (*number* as in *cardinality*) was classified as an event as one of the hypernyms of the English word *number* is "social event".

Using a Swedish WordNet or filtering the English hypernyms in a better way might work, though, so it is an approach worth exploring.

10.2 Extending the decision trees' training set

Of course extending the training set is a good idea. Currently, the building of the training set is quite demanding. Writing a tool for such annotation would be an excellent idea.

10.3 More trees

The training set available today is suitable for relations between events adjacent to each other as well as for events spaced one or two events apart.

From this training set, currently five trees are generated. It would be trivial to generate more trees from the same training data. One such tree might order adjacent events using features from both the previous and the successive event.

The argument for still having the trees that consider only fewer events is that they can be used in "corner cases", where a tree that considers more events wouldn't be applicable.

10.4 Another machine learning approach?

My belief is that machine learning approaches are very suitable for the task of event ordering. Decision trees, however, need not be the best tool. The use of e.g. support vector machines (SVM) (Vapnik, 1998) might give better results, and it is an approach worth exploring.

10.5 A good evaluation method for the final ordering

The method I used to evaluate the final ordering is obviously flawed. Improving it or developing an alternative evaluation method is a very good idea.

10.6 A tool for evaluation

Currently, it is very demanding to evaluate the resulting event chain graph. If an annotation or evaluation tool were written, that tool could ease the evaluation task considerably. It would also be easier to experiment and compare different approaches to the question of event ordering.

10.7 Another method for resolving temporal conflicts

The method in use for resolving temporal loops need not be the best one. If an evaluation tool were written, it would be easy to experiment with and evaluate other algorithms for loop removal.

10.8 TimeML

Using TimeML for Swedish along with a somewhat simple matcher was at times difficult.

A TimeML annotation guidline for Swedish would be a good thing to have. Some Swedish constructs were difficult to map to the examples given for English in the annotation guidelines. If such a guideline were to be written, the tagging might be improved, as the rules then would be clear(er).

10.8.1 Extending TimeML

TimeML itself is somewhat cumbersome to use. Signals like "before" are to be tagged as:

```
<SIGNAL sid="1">before</SIGNAL>
```

The signal has no features beyond its identifier. The value of the signal is to be encoded using `ALINKs`, `SLINKs` and `TLINKs`, but if the tagger is unable to find such links, the meaning of the signal gets lost.

A tagger that can't find any good links would lose that information if it uses pure TimeML, and such information, even though unconnected to entities, is very useful for e.g. decision trees such as those used in the TimeCore module.

In my opinion, it would be a good idea to add features to the TimeML signals, as they can be useful in different contexts even if they aren't linked to any entities.

Chapter 11

Summary & conclusion

I have developed a method for detecting time expressions, events and for ordering these events temporally. The method has been implemented and integrated in the Carsim program.

The time expression detection and interpretation as well as the TimeML mark-up parts of the module perform well. The module ought to be easy to separate from the Carsim framework, and it ought to be reusable in other contexts.

The central task, though, the creation of a full ordering between the events, leaves lots of room for improvement. If the training set were to be extended or the accuracy of the decision trees were to be improved by some other means, the overall performance of the engine could improve dramatically.

Appendix A

Terms and concepts

A.1 Terms used in this report

Corpus (*korpus*) A collection of texts. Today these are generally computer readable.

Decision tree (*beslutsträd*) Decision trees are a type of classifier that can be trained automatically.

Hypernym (*hypernym*) A word that is more generic than another word; "animal" is a hypernym of "cat".

Hyponym (*hyponym*) A word that is less generic than another word; "cat" is a hyponym of "animal".

Natural language *naturligt språk* A language used by humans, as opposed to a programming language.

Part-of-speech tagger A part-of-speech tagger (POS-tagger) tags the words of a text with part-of-speech tags. Figure A.1 shows the output of the POS tagger Granska (Carlberger and Kann, 1999) when applied to a small text, a text that could have been taken from a online Swedish news paper in August 2004.

A.2 Measures

Some measures are common when evaluating performance of algorithms in computational linguistics. The definitions make use of Figure A.2, where the sets A, B and C are shown.

Recall is a measure of how many of the relevant entities that were found. It is defined as *Relevant items retrieved / All relevant items* $= \frac{B}{A+B}$. A recall of 100% can be achieved by finding all items in the dataset.

Precision is a measure of many of the retrieved entities that are relevant. It is defined as *Relevant items retrieved / All retrieved items* $= \frac{B}{B+C}$. A precision of 100% can be achieved when finding just one item in the dataset.

Idag morse sken solen. Det hade den inte gjort sedan i maj.
This morning the sun shone. It hadn't done that since May.

Word	POS-tag	Lemma	English
Idag	ab	idag	Today
morse	nn	morse	morning
sken	vb.prt.akt	skina	shone
solen	nn.utr.sin.def.nom	sol	the sun
.	mad	.	.
Det	pn.neu.sin.def.sub/obj	det	That
hade	vb.prt.akt.aux	ha	had
den	pn.utr.sin.def.sub/obj	den	it
inte	ab	inte	not
gjort	vb.sup.akt	göra	done
sedan	ab	sedan	since
i	pp	i	in
maj	nn.utr.sin.ind.nom.dat	maj	May
.	mad	.	.

Granska tags verbs with *vb.[?]+*, nouns with *nn[?]** et cetera. Their lemmas, basic form, are also returned.

Figure A.1: POS tagged text.

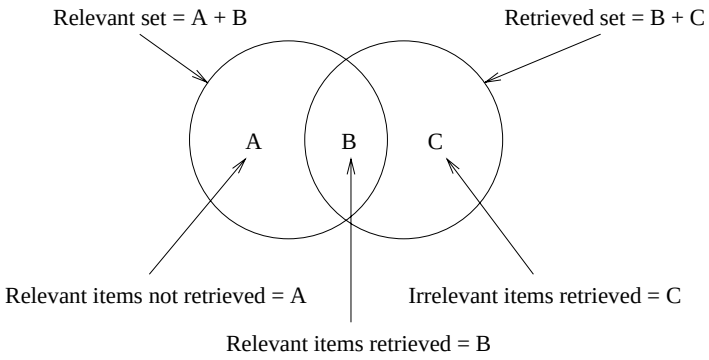


Figure A.2: The measures of precision and recall.

F-measure is the harmonic means of precision (P) and recall (R), normally defined as $2 * \frac{P*R}{P+R}$. The $F - measure$ is useful as precision and recall by themselves can be a bit hard to interpret.

Appendix B

Whorf; Are events only verbs?

A good argument for the viewpoint that nouns also can be events is the statement "an event is an event". But the argumentation can be taken further.

Benjamin Lee Whorf discusses the issue of "event encoding" in Whorf (1940) where he compares event encoding in different language families, and his argumentation is quite interesting:

> *Let us consider a few examples. In English we divide most of our words into two classes, which have different grammatical and logical properties. Class 1 we call nouns, e.g., 'house, man'; class 2, verbs, e.g., 'hit, run'. Many words of one class can act secondarily as of the other class, e.g., 'a hit, a run,' or 'to man (the boat),' but, on the primary level, the division between the classes is absolute. Our language thus gives us a bipolar division of nature. But nature herself is not thus polarized. If it be said that 'strike, turn, run,' are verbs because they denote temporary or short-lasting events, i.e., actions, why then is 'fist' a noun? It also is a temporary event. Why are 'lightning, spark, wave, eddy, pulsation, flame, storm, phase, cycle, spasm, noise, emotion' nouns? They are temporary events. If 'man' and 'house' are nouns because they are long-lasting and stable events, i.e., things, what then are 'keep, adhere, extend, project, continue, persist, grow, dwell,' and so on doing among the verbs? If it be objected that 'possess, adhere' are verbs because they are stable relationships rather than stable percepts, why then should 'equilibrium, pressure, current, peace, group, nation, society, tribe, sister,' or any kinship term be among the nouns? It will be found that an "event" to US means "what our language classes as a verb" or something analogized therefrom. And it will be found that it is not possible to define 'event, thing, object, relationship,' and so on, from nature, but that to define them always involves a circuitous return to the grammatical categories of the definer's language.*

(In the paper Whorf goes on arguing that the indian language Hopi is better structured than most other languages in that short-lasting events are verbs and long-lasting events are nouns.)

Bibliography

[Allen, 1984] Allen, J. F. (1984). Towards a General Theory of Action and Time. *Artificial Intelligence*, 23(2):123–154.

[Carlberger and Kann, 1999] Carlberger, J. and Kann, V. (1999). Implementing an efficient part-of-speech tagger. *Software Practice and Experience*, 29:815–832.

[Dowty, 1986] Dowty, D. R. (1986). The effects of Aspectual Class on the Temporal Structure of Discourse: Semantics or Pragmatics? *Linguistics and Philosophy*, 9:37–61.

[Hitzeman et al., 1995] Hitzeman, J., Moens, M. N., and Grover, C. (1995). Algorithms for Analyzing the Temporal Structure of Discourse. In *Proceedings of the Annual Meeting of the European Chapter of the Association of Computational Linguistics (EACL'95)*, pages 253–260, Dublin, Ireland.

[ISO8601, 2000] ISO8601 (2000). ISO8601:2000(E): Data elements and interchange formats – Information interchange – Representation of dates and times. Available, possibly illegaly, at *http://lists.ebxml.org/archives/ebxml-core/200104/pdf00005.pdf*.

[Johansson et al., 2004a] Johansson, R., Williams, D., Berglund, A., and Nugues, P. (2004a). Carsim: A System to Visualize Written Road Accident Reports as Animated 3D Scenes. In Hirst, G. and Nirenburg, S., editors, *Proceedings of the Second Workshop on Text Meaning and Interpretation, 42nd Annual Meeting of the Association of Computational Linguistics*, pages 57–64, Barcelona, Spain. Association for Computational Linguistics.

[Johansson et al., 2004b] Johansson, R., Williams, D., and Nugues, P. (2004b). Converting Texts of Road Accidents into 3D Scenes. In de Mántaras, R. L. and Saitta, L., editors, *ECAI2004, Proceedings of the 16th European Conference on Artificial Intelligence*, pages 1035–1036, Valencia, Spain. IOS Press, Amsterdam.

[Källgren, 1992] Källgren, G. (1992). SUC - The Stockholm-Umeå Corpus Project: Corpus-Based Research on models for processing unrestricted Swedish Text.

[Lascarides and Asher, 1993] Lascarides, A. and Asher, N. (1993). Temporal Interpretation, Discourse Relations, and Common Sense Entailment. *Linguistics & Philosophy*, 16(5):437–493.

[Mani, 2003] Mani, I. (2003). Recent Developments in Temporal Information Extraction. In Nicolov, N. and Mitkov, R., editors, *Proceedings of RANLP'03*. John Benjamins.

[Mani et al., 2003] Mani, I., Schiffman, B., and Zhang, J. (2003). Inferring Temporal Ordering of Events in News. In *Human Language Technology Conference (HLT'03)*, Edmonton, Canada. Available at *http://complingone.georgetown.edu/~linguist/papers/hlt03-short.pdf*.

[Miller, 1995] Miller, G. (1995). WordNet: A Lexical Database for English. *CACM*, 38(11):39–41.

[Ng and Cardie, 2002] Ng, V. and Cardie, C. (2002). Improving Machine Learning Approaches to Coreference Resolution. In *Proceeding of the 40th annual Meeting of the Association for Computational Linguistics*, pages 104–111, Philadelphia. Association for Computational Linguistics.

[Pustejovsky et al.,] Pustejovsky, J., Ingria, R., Saurí, R., Castaño, J., Littman, J., Gaizauskas, R., Setzer, A., Katz, G., and Mani, I. The Specification Language TimeML, opthowpublished = , month = January, year = 2004, note = Available at *http://complingone.georgetown.edu/~linguist/papers/TimeML.pdf*, optannote = .

[Pustejovsky et al., 2002] Pustejovsky, J., Saurí, R., Setzer, A., Gaizauskas, R., and Ingria, B. (2002). TimeML Annotation Guidelines.

[Reichenbach, 1947] Reichenbach, H. (1947). *Elements of Symbolic Logic*. Academic Press, New York.

[Setzer and Gaizauskas, 2001] Setzer, A. and Gaizauskas, R. (2001). A Pilot Study on Annotatin Temporal Relations in Text. In *ACL 2001, Workshop on Temporal and Spatial Information Processing*, pages 73–80, Toulouse, France.

[TimeBank, 2002] TimeBank (2002). TimeBank corpus. Available via *http://www.cs.brandeis.edu/ jamesp/arda/time/timebank.html*.

[TimeML DTD, 2002] TimeML DTD (2002). DTD for TimeML 1.0. Available at *http://www.cs.brandeis.edu/~jamesp/arda/time/documentation/TimeML.dtd*.

[TimeML Specification,] TimeML Specification. Specification for TimeML 1.0, opthowpublished = , month = July, year = 2002, note = Available at *http://www.cs.brandeis.edu/~jamesp/arda/time/documentation/TimeML-Draft3.0.9.html*, optannote = .

[Vapnik, 1998] Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley, New York.

[Webber, 1988] Webber, B. (1988). Tense as Discourse Anaphor. *Computational Linguistics*, 14(2):61–73.

[Whorf, 1940] Whorf, B. L. (1940). Science and linguistics. *Technological Review*, 42(6):229–231, 247–248.