Integrating time into **D**iscourse **R**epresentation **T**heory: A computational approach

Diego Mollá Aliod

September 14, 1992

This dissertation is basically a practical work where I try to implement the integration of time structures into Kamp's *Discourse Representation Theory* (*DRT*) by means of a computer program. By doing so, several theories have been put together, all them based on Kamp's *DRT* [7]. First of them, Johnson and Klein's paper [6], where a reformulation of *DRT* theory has been made in such a way that it can be implemented in a declarative programming language such as *PROLOG*. Secondly, another article written by Partee [10], where some basics of how to integrate time into *DRT* have been stated, and finally, an analysis made by Eberle and Kasper [5] where the temporal forms are described by means of a set of *temporal features*, which make temporal analysis be easier for a computer.

In the work written in these pages I will introduce part of each one of these theories, although not necessarily in this order, together with clues of how the integration can be made. The last chapters of the dissertation will be devoted to the final implementation and how this work can be improved during further research.

Diego Mollá Aliod

Final dissertation for the Msc *Speech and Language Processing*

Supervisor of the Msc: *S.D. Isard*

Supervisor of the dissertation: *R. Cann*

Department of Linguistics

The Universiy of Edinburgh

Edinburgh, September 14, 1992

Contents

Chapter 1

Time basics

In this chapter, and as an introduction to the following chapters, the main ideas of tense and aspect used along the dissertation will be presented. During this report, temporal reference will be handled by using a set of determined *temporal points* as defined by Eberle and Kasper (see, for example, [5]).

1.1 Events and States

There are two main types of temporal points: *events* and *states*. Events and states are basically the same, that is, they both are intervals of time. But there is a main difference bertween them: an event has a duration which appears limited during the discourse, whereas a state can have open edges. As an example, we will analize a few simple discourses, noting when a state or a referent is introduced, and how the references are intuitively made.

For example, consider the discourse:

(1) A man went to the station. His train arrived.

An event is introduced by means of the first sentence. This event represents the man's going to the station. It occurred somewhere in the past, thought when is not specified in this discourse. The second sentence introduces other event: the train arriving. This second event occurs somewhere in the past, but it is assumed to occur after the man's arrival to the station. And furthermore, the train arrival appears very close to the man's arrival in the discourse. This does not mean that the train would arrive after a short time. Rather, it means that the discourse treats both events as very closely related: first, the man arrives; then, the train does so, and nothing in the discourse happens in between. This relationship is described as that the train arrives *just after* the man does it.

Consider now the discourse:

(2) A man went to the station. The train was arriving.

In this other discourse, the second sentence states that the train was arriving somewhere in the past, but there are no limits in the duration of this interval of time: whether the train arrives before, after or while the man does is not specified. This is indicated by introducing a state. This situation is usually described as that the man's arrival occurs while the train arrives; in other words, the time interval defined by the event created by the man's arrival is included into the interval defined by the state created when the train was arriving.

1.2 Reference points

The use of a set of temporal points which can be referenced later on allows tense and aspect to be handled as a particular case of anaphoric reference. However, the paralelism between *anaphora resolution* and *temporal reference* is not so straightforward, as [10] pointed out.

The set itself of temporal points is very restricted. In fact, only three points will be treated: the speech time, the temporal perspective point and the reference point, following the works of Eberle and Kasper [5] and Kamp [8]. Although time resolution is (or seems to be) simpler than anaphoric reference, it also relies in discourse structure (introduction of flashback and embedded reported speeches, for example) which will be handled here in a very superficial way, if handled at all. This dissertation only attempts to formalise the basics of time resolution. Let me now focus on the temporal points I have introduced and explain why they have been chosen.

Speech time

The *speech time* **ST** indicates, as the name itself says, the point in time where the dialogue is uttered. Although this temporal point is very important, no complex theory is needed in order to handle it, at least in its simplest version: when the speech time of the whole discourse is a specific interval of time, with no internal structure. This assumption does not restrict the type of data to be accounted for drastically, except perhaps in cases where the discourse refers to events which overlap partly with the discourse speech time, like in sport reports.

Reference point

The *reference point* **Rpt** is always very closely related to the action described. Sometimes it denotes the time of the proper action, as it usually happens when an event is introduced. For example, the *Rpt* of the next sentence says when Peter got up:

(3) Peter got up at 5h 30.

At other times it denotes a time where the action is happening, when for example a state is introduced:

(4) Peter loved his job.

The *Rpt* of (4) indicates a time where Peter loved his job, but the action of Peter's loving his job may extend beyond that *Rpt*:

$$Rpt \subseteq \text{Peter's loving his job}$$

Finally, the *Rpt* may also denote a time where the result of the action holds, as in the use of a perfective form:

(5) The child has broken the glass.

From (4) and (3) we can see that the *Rpt* does not need to be specific; in (4) the *Rpt* is located somewhere in the past, while in (5) the *Rpt* and the *speech time* are the same.

If we look carefully at (5), we can see that, in a perfective form, what the *Rpt* indicates is not the event or state directly introduced by the main verb. In (5) specifically, the *Rpt* is neither when the glass is broken nor a point where the glass is being broken. Rather, the *Rpt* indicates a point in time where the state which results from the glass being broken by the child holds. That is, a perfective form introduces a complex structure where the action described by the main verb forms a part. In (5) we can also see that the glass is broken before the *Rpt*, while the result of its breaking (for example, the portions are spread over the floor) still holds during *Rpt*. But sometimes even the action itself may still hold at *Rpt*, as in:

(6) The Joneses have lived here for three years. *in [8] p61*

In (6), the use of the present perfect does not imply that the Joneses do not live there when the sentence is uttered. For example, somebody may say (6) to stress that the Joneses are well known in the area where *there* points, but the Joneses still live there.

The *Rpt* may shift in time, usually forwards, as in the discourse:

(7) Peter got up at 5h 30, prepared his lunch, started the trip and reached the highway before 7.

In (7), the *Rpt* goes from Peter's getting up to his reaching the highway at 7, following a temporal sequence.

Temporal perspective point

The *temporal perspective point* **TPpt** provides a temporal point which is treated in a special way. The need of this point arises from discourses where the temporal relations with a past reference point hold in a way which is very similar to other relations with the *ST*. Consider, for example, the last discourse:

(7) Peter got up at 5h 30, prepared his lunch, started the trip and reached the highway before 7.

In it, a chain of events occurs, but all of them do it before the speech point. Consider now:

(8) Peter arrived at the summit at 12. He had got up at 5h 30, prepared his lunch, started the trip and reached the highway before 7.

In this last case, the same chain of events is related, not with the *ST*, but rather with Peter's arrival to the summit, at 12. This time will be the *TPpt*.

The *TPpt* is also important when certain adverbs are used, like the case of the French word *maintenant*. Consider:

(9) Jean rentra chez lui content. Les dernieres jours avaient été penibles. Mais maintenant tout marchait bien. *from [5] p 21*

In this case, *maintenant* (now), which usually refers to the speech time, refers now to a past point. All this data can be explained if we introduce the concept of *TPpt*.

1.3 Time parameters

The tense and aspect of VPs indicate when the action described occurs, and may also indicate whether the action is finished and provide specific boundaries and even specific dates. This can be done by establishing relationships between the temporal points above mentioned and defining when they occur. Temporal points can be very specifically bounded or bounded by means of a rather vague expression, as the two following examples show, respectively:

(3) Peter got up at 5h 30.

(10) Peter was born in the 60s.

The location of temporal points is usually made by using locatives, as in the examples (3) and (10). However, no locatives are needed to establish the general relationships among the temporal points. This section explains how these relationships can hold.

The relationship between *ST* and *TPpt* is quite simple: either the *TPpt* coincides with the *ST* or it occurs before. The *TPpt* can never occur after the speech time – except in complex dialogues where, anyway, the speaker is not aware of this anomaly. This limitation rules out unsensible uses of *maintenant*. Thus, the next example is not good French because *maintenant* refers to a point in the future:

(11) Jean arrivera demain matin. Maintenant nous commencerons à repeter.
from [5] p 21

The relationship between *ST* and *TPpt* can be described by using the feature **+/- PAST**. Thus, if the *TPpt* occurs before the *ST* the feature *PAST* will be on.

There is also a relationship between *TPpt* and *Rpt*, but in this case it is more free: the *Rpt* may coincide with the *TPpt*, but it may also occur before or even after. The feature **TENSE** will cover this, having the values **past**, **pres** or **fut**. Some examples are, respectively:

(12) A farmer owned a donkey.

(13) A farmer owns a donkey.

(14) A farmer will own a donkey.

As an example of the use of this two features, consider the discourse (9):

(9) Jean rentra chez lui content. Les dernieres jours avaient été penibles. Mais maintenant tout marchait bien.
from [5] p 21

The first sentence of (9) instantiates the *Rpt* to a point somewhere before the *ST*, and the *TPpt* will coincide with the *ST*; it will have the feature values $-PAST$ and $TENSE = past$. In the third sentence, however, *maintenant* refers to a point in the past. This will mean that the *TPpt* is located before the *ST* (more exactly, the *TPpt* is *Jean's arrival home*), and the *Rpt* will coincide with the *TPpt*; the adequate feature values will be $+PAST$ and $TENSE = pres$.

Apart from relationships among temporal points, tense and aspect may describe other temporal features: *progressivity* and *perfectivity*. The *progressivity* feature **PROG** indicates whether the action describes a state, as in the example (13) or an event, as in the example (3). In contrast with other features, progressivity does not always depend only on the verbal form, but may also depend on the whole VP, as it can be seen in the two following examples:

(15) John thinks.

(16) John thinks about the problem.

The sentence (15) is fine, but (16) cannot be used unless reportive speech is being used. As I will explain in the subsection 1.4.2, this is so because (15) introduces a state, whereas (16) introduces an event.

Finally, *perfectivity* **PERF** can also be stated. A perfective expression does not need to indicate that the action is finished at *Rpt*, like the example (6) showed:

(6) The Joneses have lived here for three years. *in [8] p61*

Rather, perfectivity indicates the introduction of a state which occurs as a result of the event (in (6), the event is Joneses' living there for three years). Whether the event still holds is not specified. As a side effect, since a state is introduced, whenever $PERF$ is on $PROG$ will be also on.

1.4 Verb forms revisited

The previous sections described the temporal reference points and the relationships that may hold between them. A set of features was used as a mean to describe these relationships. In this section I will try to detail how the temporal points behave in a set of specific English verb forms, and how these verb forms can be broken down into their temporal features. Almost all the theory in this section appears with more detail in [8]. But before explaining this theory, a few points in the notation to be used may need to be stated.

1.4.1 Notation

The set of reference points, as it has been explained before, comprise at least the speech time ST, the reference point *Rpt* and the temporal perspective point $TPpt$. These reference points will point to real time intervals during the discourse. Two of them are actually defined before the discourse is analized:

- The *speech time*, which will be assumed to be a constant reference point, as it has been explained. This reference point will be linked with a time interval called **now**.
- The *reference point*, which will refer to a special temporal point at the beginning of the discourse. The actual time of the first reference point will be called r_0. This reference point is necessary if we want to treat the first sentence like the following ones. Although it has a name, r_0 actually represents an unspecified point which (possibly) occurs somewhere in the past.

Temporal and reference points hold relationships in between. These relationships will be stated with the following notation:

precedence $r_a < r_b$ means that r_a occurs before r_b and they do not overlap.

coincidence $r_a = r_b$ means that r_a and r_b are actually the same point.

just after $r_a \not> r_b$ means that r_b occurs after r_a and very close to it, but they do not overlap.

inclusion $r_a \subseteq r_b$ means that when r_a occurs r_b is happening, and when r_a finishes r_b has not finished yet.

Peter got up at 5h 30

$r_0 \not> e_1 \quad e_1 < now \quad e_1$:Peter gets up

prepared his lunch

$e_1 \not> e_2 \quad e_2 < now \quad e_2$:Peter prepares his lunch

started the trip

$e_2 \not> e_3 \quad e_3 < now \quad e_3$:Peter starts the trip

and reached the highway before 7

$e_3 < e_4 \quad e_4 < now \quad e_4$:Peter reaches the highway

Figure 1.1: *Simple temporal analysis of a discourse*

John watched through the window.

$r_0 \not> e_1 \quad e_1 < now \quad e_1$:John watches through the window

It was raining outside.

$e_1 \not> r_1 \quad r_1 \subseteq s_1 \quad r_1 < now \quad s_1$:it rains outside

He lit a cigarette

$r_1 \not> e_2 \quad e_2 < now \quad e_2$:John lights a cigarette

Figure 1.2: *Discourse with states. Note that the state introduced in the second sentence surrounds the new reference point at that moment (r_1).*

1.4.2 Verbal forms

Simple past and past progressive

The simple past is, perhaps, the easiest verbal form to handle. We will start with it therefore, and will look at it with a little more of detail.

A simple past introduces a reference point which is in the past. Furthermore, if a sequence of events is uttered, the corresponding sequence of reference points appears to be related in such a way that the later sentences refer to points in time which occurred more recently, as the example (7) showed quite clearly. The figure 1.1 shows how the discourse (7) can be analysed, if the temporal adverbs are not taken into account.

The action may describe a state instead of an event. In that case there is also a sequence of reference times, a reference time for each action, and the introduced state will appear surrounding the reference time of that action, as the second sentence in figure 1.2 shows. Note that the analysis performed earlier in section 1.1 handled the states in a different way; in that analysis no new reference time was created, and the state surrounded a reference point which was introduced earlier by other action. The analysis proposed here was already mentioned by Partee in [10], and it accounts for new data where the new state cannot surround what a previous action introduced, like in the discourse:

(17) Jameson entered the room, shut the door carefully, and switched off the light.
It was pitch dark around him, because the Venetian blinds were closed.

from [10], p.254

As a simple past can introduce a state or an event depending on the VP (compare *John watched through the window* and *John owned a donkey*), the feature $PROG$ will depend on the VP, and all of them will be:

$$-PAST \qquad TENSE = past \qquad ?PROG \qquad -PERF$$

The man arrived to the station.

$r_0 \not> e_1 \quad e_1 < now \quad e_1$:the man arrives to the station

=> He would wait for the train.

$e_1 < e_2 \quad e_2 \subseteq s_1 \quad s_1$:he waits for the train

=> He waited for the train.

$e_1 \not> e_2 \quad e_2 < now \quad e_2 \subseteq s_1 \quad s_1$:he waits for the train

Figure 1.3: *This figure shows the difference between the future past and the simple past; the action specified in the future past (first continuation) may happen in a real future, whereas in the simple past (second continuation) it should happen before the speech time. Besides, the action in the future past does not need to be very close to the previous one.*

The past progressive will be like the simple past, where the feature $PROG$ is always set to $+$. See an example of use of the past progressive in the figure 1.2.

Simple present and present progressive

The simple present is more difficult to analyse, since it may be used in reportive speech or for expressing habits. However, the only case which will be studied here is that which refers to real actions (no habits), and where the speech time does not vary (ruling out reportive speech too, but see section 4.2). It means that only VPs which introduce states which can surround the speech time will be treated, like in (13) or in present progressives. The features for the simple present are:

$$-PAST \qquad TENSE = pres \qquad ?PROG \qquad -PERF$$

where a verb form with $-PROG$ cannot be sensibly uttered, and the features for the present progressive are:

$$-PAST \qquad TENSE = pres \qquad +PROG \qquad -PERF$$

Future simple, future progressive

Having seen the past and present forms, it is easy to see that the future forms are like the previous ones, but where the value of $TENSE$ is fut. This interpretation rules out possible intentional analysis, which are not covered in this dissertation.

Future past, future past progressive

In these cases, the future is over a past reference time. This past reference time will actually be the $TPpt$. That is, $TPpt < Rpt$. The feature values are, for the simple and the progressive forms:

$$\begin{array}{llll} +PAST & TENSE = fut & ?PROG & -PERF \\ +PAST & TENSE = fut & +PROG & -PERF \end{array}$$

The figure 1.3 shows a simple analysis of two discourses, one with a future past form and the other with a simple past form.

FORM	PAST	TENSE	PROG	PERF
simple past	-	past	?	-
simple present	-	pres	?	-
future simple	-	fut	?	-
future past	+	fut	?	-
present perfect	-	pres	+	+
will + have	-	fut	+	+
s. past prog	-	past	+	-
s. pres. prog	-	pres	+	-
fut. s. prog	-	fut	+	-
fut. p. prog	+	fut	+	-
past perfect	-	past	+	+
would + have	+	fut	+	+

Table 1.1: *Verbal forms and their temporal features, in their simplest interpretation*

Perfective forms

The perfective forms will always set the $PERF$ feature to +, showing that the *Rpt* is in the state of affairs which occurs as a result of the described event. Thus, in the example (5) the *Rpt*, which in that case coincides with the ST, is inside the state which is the result of the glass being broken by the child:

$$r_0 \subseteq s_1 \quad e_1 \not\succ s_1 \quad e_1\text{:the child breaks the glass}$$

As a perfective form always introduce a state, the $PROG$ feature is always +. The table 1.1 summarizes all the features explained so far.

The past perfect can also be seen as the "past of a past" when a flashback is introduced, as (8) showed. In this case the past perfect behaves like the simple past form whose $TPpt$ is shifted to the past. However, this interpretation will not be treated here; see instead [8] for a brief account of it.

1.5 From features to temporal references

Up to now, a few ideas of Eberle, Kasper, Kamp and Partee have been introduced to make a theory of tense. However, no attempt of formalizing the relationship between features and temporal references has been made. In this section I will explain how every temporal feature can trigger a set of relationships between temporal points. First of all, I want to mention that the previous analysis of verbal forms is quite simplistic, in the sense that it does not take into account discourse structure or flashback mechanisms or locatives. Following the philosophy of not trying to cover everything at the first step, I am not going to treat these issues in this section, since their complexity is too big to be handled in a Msc dissertation. I hope that, given the robustness of the account here provided, its enhancement to cover all that will not be impossible.

The temporal features can be divided into two groups: those which deal with aspect ($PROG$ and $PERF$) and those which deal with tense ($PAST$ and $TENSE$).

VALUE	TRUTH-CONDITION
past	$Rpt < TPpt,\ OldRpt \not> Rpt$
pres	$TPpt = Rpt$
fut	$TPpt < Rpt$

Table 1.2: *The use of TENSE. Note the difference in past.*

1.5.1 Aspectual features

Progressivity

The fact that an action may introduce a state or an event is very important: if an event is introduced, this event becomes the new Rpt. The truth-conditions introduced are, then:

$$\text{new-event}(e),\ Rpt = e$$

If a state is introduced, however, as it was showed in the subsection 1.4.2, the state includes the new Rpt:

$$\text{new-state}(s),\ \text{new-ref}(r),\ r \subseteq s,\ Rpt = r$$

Perfectivity

If a verbal form is perfective, a state appears just after the action described by the main verb. This state was already introduced by means of the feature $PROG$, which will be always on if $PERF$ is on. The conditions introduced by $PERF$ are, then:

$$\text{new-action}(a),\ a \not> s$$

1.5.2 Temporal features

Past

This feature handles the relationship between the $TPpt$ and the ST (which is always *now*). If this feature is on, the $TPpt$ is prior to the speech time:

$$TPpt < now$$

and if $PAST$ is off, they both are the same:

$$TPpt = now$$

This feature is very closely related to flashbacking, or more general, to embedded time structures. When a flashback occurs, the feature is always $+PAST$, and the $TPpt$ is updated to the old Rpt (not to the current one). When finally the flashback finishes, the $TPpt$ will coincide again with the ST.

Tense

This last feature is quite straightforward: it marks the relationship between the $TPpt$ and the Rpt. The table 1.2 shows how this can be done.

When $TENSE$ has *past* value, the truth-conditions are slightly different: a link between the old *Rpt* and the new one is established. This accounts for the sequencing in discourses like (7). However, despite this new link has been introduced as a semantic condition, it might well be just a matter of pragmatics, since some discourses do not have this sequencing implied, like in:

(18) Everybody helped in the make-up of the party. Hellen prepared the sandwiches, Peter brought his CD player and Michael wrote the invitations.

The fact that only the forms which have $TENSE = past$ have the link between *Rpt*s is perhaps because the past forms are the preferred ones in narrating past events, and narrations usually describe chains of related events (see [2] for more details about this theory).

Finally, note that the condition introduced by fut works well even when verbs which introduce states are used. This is so because the *Rpt* is also updated, as it will be shown in the example proposed in the next section.

1.6 An example

In this section, I will try to explain a simple example of a full analysis of a discourse. Consider:

A man went to the station.
He would wait for the train.
He lit a cigarette.

The first sentence of this discourse introduces an event just after the previous *Rpt* (which will be called r_0). This new event occurs in the past:

$$r_0 \not> e_1,\ e_1 < now\ ,\ e_1\text{:a man goes to the station}$$

With our analysis with features, the values of the features for the first sentence are:

$$-PAST \qquad TENSE = past \qquad -PROG \qquad -PERF$$

The feature $-PAST$ sets $TPpt$ to now, the feature $-PROG$ sets *Rpt* to the new event e_1 and the feature $TENSE = past$ establishes the relationship between the *Rpt* and the $TPpt$: $e_1 < now$, and between the previous *Rpt* and the new one: $r_0 \not> e_1$.

In the second sentence, a state is introduced – the verb `wait` introduces this state. Also, a new referent is introduced which occurs after the man's arrival to the station:

$$e_1 < r_1,\ r_1 \subseteq s_1,\ s_1\text{:the man waits for the train}$$

With our analysis, the values of the features for the second sentence are:

$$+PAST \qquad TENSE = fut \qquad +PROG \qquad -PERF$$

The feature $+PAST$ signals the introduction of flashbacking: the previous reference times are saved and the new $TPpt$ is set to e_1, which is the old *Rpt*. Because of $+PAST$ again, the $TPpt$ is defined as appearing before the speech time: $e_1 < now$. This condition is redundant, since it was also introduced in the analysis of the first sentence. The feature $+PROG$ sets the *Rpt* to a new time interval $r1$ which will be linked with the new state s_1: $r_1 \subseteq s_1$. And finally, the feature $TENSE = fut$ links the $TPpt$ with the *Rpt*: $e_1 < r_1$.

In the last sentence an event is introduced just after the man's arrival, that is:

$$e_1 \not> e_2,\ e_2 < now\ ,\ e_2\text{:the man lits a cigarette}$$

The features for this last sentence are:

$$-PAST \qquad TENSE = past \qquad -PROG \qquad -PERF$$

In this case, since $-PAST$ holds again, the flashback is finished, restoring the previous reference points and making the old Rpt be again e_1. Again, as in the first sentence, the feature $-PAST$ sets the new $TPpt$ to now, the feature $-PROG$ sets the new Rpt to the new event e_2 and the feature $TENSE = past$ relates the Rpt and the $TPpt$: $e_2 < now$ and the old Rpt with the new one: $e_1 \not> e_2$.

Chapter 2

DRT theory and implementation

2.1 Basics of *DRT*

DRT stands for **D***iscourse* **R***epresentation* **T***heory*, introduced by Kamp [7]. This theory tries to explain the semantics of a discourse by building a structured intermediate semantic representation. This representation follows some special points of discourse structure, making it specially good for the treatment of intersentential anaphora and the so-called "donkey sentences."

2.1.1 Basic components

Before starting with *DRT* I would like to define what can be meant by *discourse*. A discourse will be a set of one or more sentences which are related in a coherent way. This intuitive definition needs to receive some constraints:

- The discourse always starts without any previous context; nothing in the environment is assumed to be taken in account.
- Pronouns will not be accompanyed by any deictic gestures.
- Only one person speaks during the whole discourse.
- The sentences are always declarative.

The semantics of a discourse will be then represented by **D***iscourse* **R***epresentation* **S***tructures*. Each *DRS* will consist of a pair of sets. The first one is the set of entities which can be referenced. The second set contains the restrictions the entities of the first set have. These restrictions can be propositions in the form of predicate logic or more complex structures, as it will be shown later. As a first example, consider the sentence:

(19) A farmer owned a donkey.

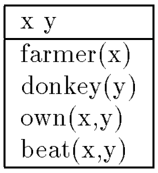
Its corresponding *DRS* will be:

x y
farmer(x) donkey(y) own(x,y)

which represents a discourse in which there are two entities, x and y which have the properties that x is a farmer, y is a donkey and x (the farmer) owns y (the donkey).

*DRS*s can represent more than one sentence:

(20) A farmer owned a donkey. He beat it.

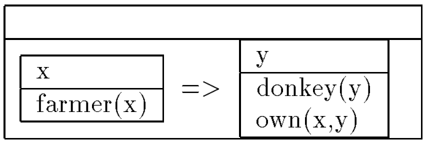


2.1.2 Introducing structures

A more complex sentence can generate a more complex *DRS*. The determiner *every* triggers the introduction of two subordinate *DRS*s, linked with the relation =>. Thus, the sentence

(21) Every farmer owned a donkey.

has the representation

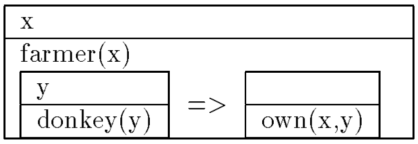


that can be paraphrased as "whenever there is an entity x which is a farmer, it is also true that there is a donkey y which is owned by x".

The determiner *every* is then much like an "if-then" sentence. In both cases there are two *DRS*s connected by an operator which can be seen as an implication. The *DRS* of the left side is the **antecedent** or **restrictor**, and the one of the right side is the **consequent** or **scope**.

Note that the following sentence will have a different meaning:

(22) A farmer owned every donkey.



When there are embedded *DRS*s we can speak of **subordination**. Generally, a *DRS* K_1 is subordinate to another *DRS* K_2 when either K_1 is embedded in K_2 or K_2 is the antecent of a conditional structure of which K_1 is the consequent, or there is a K_3 such that K_3 is subordinate to K_2 and K_1 is subordinate to K_3. Thus, in the previous example (21), the entities x and y are in two subordinate *DRS*s, being the *DRS* of y subordinate to that of x, but not vice-versa.

Subordination is important when anaphora is being taken in account, since the entities declared in a *DRS* can be accessed by any subordinate *DRS*, but not viceversa: an entity which is declared in a *DRS* cannot be accessed by any superordinate *DRS*. For example, (22) can be successfully followed in a way that (21) cannot:

(23) A farmer owned every donkey. He was happy.

```
x
farmer(x)
[ y | donkey(y) ] => [ | own(x,y) ]
happy(x)
```

There is an interpretation, however, to the discourse:

(24) Every farmer owned a donkey. He was happy.

In this case, the second sentence can be "inside" the previous *DRS*:

```
[ x | farmer(x) ] => [ y | donkey(y)
                           own(x,y)
                           happy(y) ]
```

This kind of constructions is treated in [11], but will not be treated in this dissertation.

2.2 Johnson and Klein's implementation

The project of this dissertation will use the ideas which emanated from Johnson and Klein's paper [6],[1] which described a declarative procedure of building a *DRS* from the sentence using an extension of PROLOG as the theorem-prover.

2.2.1 Flow of anaphoric information

Anaphoric information moves from left to right. This statement is easy to understand, since normally, before a pronoun can refer to an entity, this entity should have been introduced before.

Furthermore, in *DRT* the determiner plays a fundamental role in the composition of the main structure of the sentence semantics, as the previous sentences (19), (21) and (22) showed, here repeted together with their *DRS*s:

A farmer owned a donkey.

```
x y
farmer(x)
donkey(y)
own(x,y)
```

Every farmer owned a donkey.

```
[ x | farmer(x) ] => [ y | donkey(y)
                           own(x,y) ]
```

A farmer owned every donkey.

```
x
farmer(x)
[ y | donkey(y) ] => [ | own(x,y) ]
```

[1] Although the main ideas are from Johnson and Klein's paper, the practical framework and the use of GULP come from a paper from Covington [4].

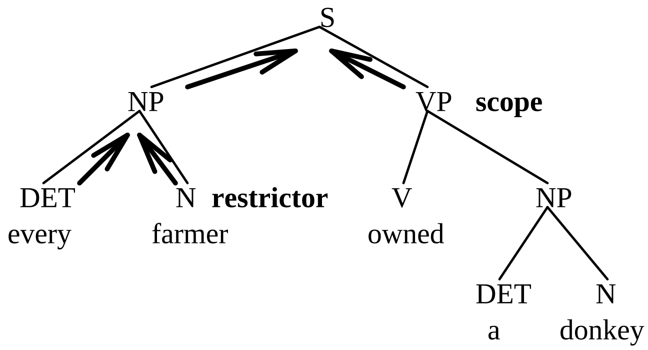


Figure 2.1: *A compositional way of building a* DRS *with a complex determiner.*

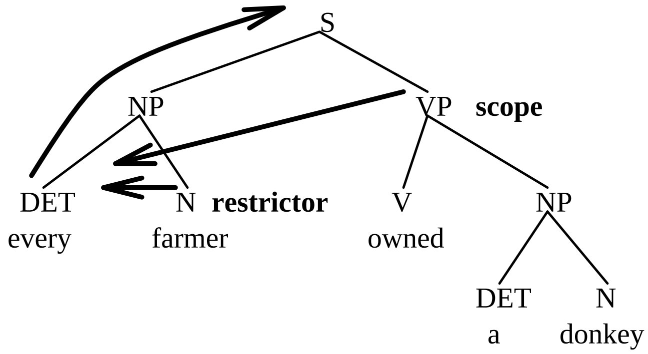


Figure 2.2: *A way of building a* DRS *with a complex determiner, where the semantics of the sentences are built in the main determiner.*

This complexity can be resolved in a compositional way. That is, the restrictor and the scope can send their information, and the upper levels can connect them and build the structure, as figure 2.1 shows with the example (21).

However, it seems more natural for the main structure to be built when the determiner be processed. The semantics of the whole sentence would then be that of the main determiner, as figure 2.2 shows. By doing this, the semantics are more closely related to the words which introduce them, and as a side-effect, the whole process is more distributed. Note that if the first approach was to be taken, the whole construction process ought to be done in the *S* node.

2.2.2 PROLOG and features: GULP

Before going deeper into how a declarational procedure of building a *DRS* from a discourse can be achieved, a short comment of the declarative language itself should be made. The use of a declarative programming language like PROLOG might weaken the property of declarativety unless we restrict ourselves to a subset of PROLOG commands, like the cut *!*, *assert* and possibly a few others. As we will see later, none of these commands are used in building the *DRT* (except to create new entities, where the assert and retract commands are used, and only to be able to create a new label each time), and the ordering itself of the sentences is only used to avoid loops and to provide better computational times. That is, the declarativeness of the procedure is

```
s(sem:(pred:X .. arg1:Y .. arg2:Z)) --> np(sem:Y .. case:nom),
                                        vp(sem:(pred:X .. arg2:Z)).
vp(sem:(pred:X1 .. arg2:Y1)) --> v(sem:X1),
 np(sem:Y1).
v(sem:'SEES')  --> [sees].
np(sem:'MAX')  --> [max].
np(sem:'BILL') --> [bill].
np(sem:'ME' .. case:acc) --> [me].
```

Figure 2.3: *A simple grammar in GULP.*

```
s(S) --> np(NP), vp(VP),
    { S  = sem:(pred:X .. arg1:Y .. arg2:Z),
      NP = sem:Y .. case:nom,
      VP = sem:(pred:X .. arg2:Z) }.
vp(VP) --> v(V), np(NP),
   { VP = sem:(pred:X1 .. arg2:Y1),
      V = sem:X1,
     NP = sem:Y1 }.
v(V)   --> [sees],{ V  = sem:'SEES' }.
np(NP) --> [max], { NP = sem:'MAX'  }.
np(NP) --> [bill],{ NP = sem:'BILL' }.
np(NP) --> [me],  { NP = sem:'ME' .. case:acc }.
```

Figure 2.4: *A form of declaring a simple grammar in GULP, by using equations.*

guaranteed.

As the use of many parameters will be a common rule, the parameters themselves will be arranged in a structural way. In order to express this on PROLOG, an extension of it is used: **GULP**. *GULP* stands for **G***raph* **U***nification* **L***ogic* **P***rogramming*, and has been developed at the *University of Georgia* [3]. The main feature of *GULP* is that it is a compiler which translates lists with attributes into normal PROLOG lists. The syntax of a list with attributes is of the form:

$$atr_1{:}atr_{11} \;..\; atr_2{:}(atr_{21} \;..\; atr_{22}) \;..\; atr_3$$

which can be interpreted in a more structured way:

$$\begin{array}{lll} atr_1 & atr_2 & \\ \quad atr_{11} & \quad atr_{21} & atr_3 \\ & \quad atr_{22} & \end{array}$$

If we use *GULP* notation in the argument structure, those arguments which are not used need not be declared. For example, the figure 2.3 shows how a simple grammar can be declared in *GULP*, and the figure 2.4 details how the same grammar can be declared in a different way, the one we will use in the final program.

```
rev(A,A) :- atom(A).
rev([],[]).
rev(In,Out):-
    In = [El|List],
    rev(El,Le),
    rev(List,Tsil),
    append(Tsil,[Le],Out).
```

Figure 2.5: *A function which reverses a tree.*

```
rev(Arg):- Arg = in:A .. out:A, atom(A).
rev(Arg):- Arg = in:[] .. out:[].
rev(Arg):-
    Arg = in:[El|List] .. out:Res,
    rev(in:El .. out:Le),
    rev(in:List .. out:Tsil),
    append(Tsil,[Le],Res).
```

Figure 2.6: *A function which reverses a tree, declared in a structured way by using GULP.*

2.2.3 Features and flow of information

The subsection 2.2.1 dealt with how we can build a *DRS* by using a proper flow of information in the parse tree. That can be encoded in *GULP*. The keypoint will be the use of a proper structure in the arguments. In plain PROLOG, it is possible to implement declaratively a function which returns a value: we only need to reserve one argument for the input, and other for the output. For example, in the figure 2.5 we can see the declaration in PROLOG of a function which reverses a tree, and in the figure 2.6 we can see how a more structured declaration can be used by means of *GULP*.

The same principle can be used for more complex flows or information. For example, the figure 2.7 shows the implementation of the interpretation of the semantics of the determiner *every* in such a way that the semantics of the scope and the restrictor are combined accordingly with figure 2.2. In that figure and in general along the whole dissertation, a *DRT* is codified by specifing the entities and the conditions in two different lists, as Covington did in [4]:

`drs(list-of-entities,list-of-conditions)`.

The declaration of figure 2.7 needs further complications, as we need to keep a list of super-

```
det(DET) --> [every],
    { Det =   res:(in:drs([],[]) .. out:Res) ..
            scope:(in:drs([],[]) .. out:Scope) ..
              sem:(in:drs(Ent,Cond) ..
                  out:drs(Ent,[Res>Scope | Cond]))
    }.
```

Figure 2.7: *A simple declaration of the semantics of* every.

```
det(DET) --> [every],
    { Det = sem:in:A,
      Det = res:(in:[drs([],[])|A] .. out:B),
      Det = scope:(in:[drs([],[])|B] ..
                    out:[Scop,Res,drs(Ent,Cond)|Super],
      Det = sem:out:[drs(Ent,[Res>Scope|Cond])|Super]
    }.
```

Figure 2.8: *Declaration of the semantics of every, taking also superordinated* DRS*s in account*

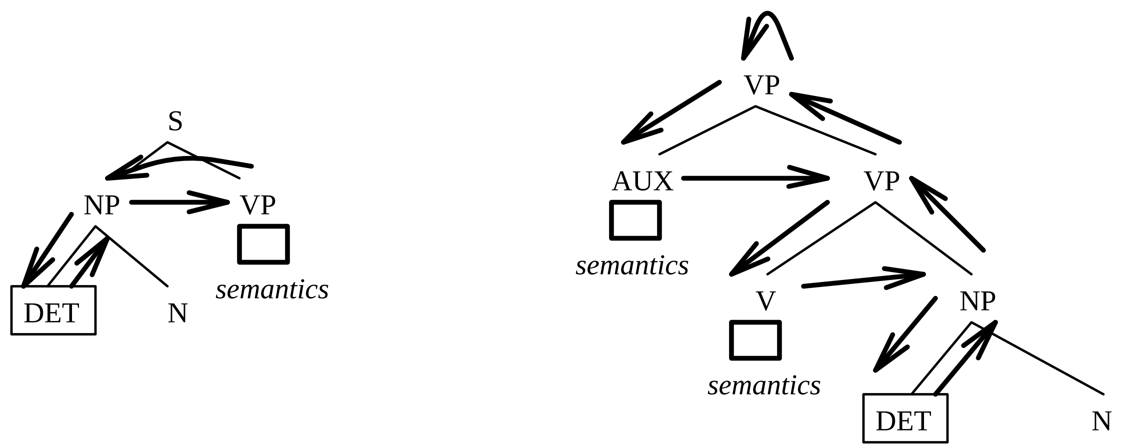


Figure 2.9: *The flow of scope information. A thin box indicates the node which introduces the first* **in** *attribute and collects the* **out** *result. A thick box under a node indicates that the information is modified in that node. The label* **semantics** *indicates that the information is passed to the attribute* **semantics** *and is modified there. See an example in the text.*

ordinated *DRS*s in order to account for anaphora, and pass the proper lists to the restrictor and the scope, as the figure 2.8 shows.

But the information must arrive to the corresponding scope and back to the determiner. The figure 2.9 shows this flow of information for the scope, for the cases of the determiner being in the main NP of the sentence and of it being inside the VP, and the appendix A shows a more complete set of trees with the flow of information as it was used in the final implementation, including time (which will be treated in the next chapter). This flow should be implemented in the declaration of every node. For example, if we want to declare the required links for dealing with scope information in the S node, we should do it as it appears in figure 2.10. For the inner VP node, the declaration is as simple as in S, but there are more connections. A procedure *link* is declared which simplifies this task, giving a declaration (for handling only scope connections) in the form which appears in figure 2.11.

```
s(S) --> {
           NP = scope:B, VP = sem:B,
           VP = scope:(in:C .. out:C)
         }, np(NP), vp(VP).
```

Figure 2.10: *How to declare the connections which pass the scope information in the S node.*

```
link(B1,B2,B3):-
    B1 = in:Z1,   B2 = in:Z1,
    B2 = out:Z2,  B3 = in:Z2,
    B3 = out:Z3,  B1 = out:Z3.
vp(VP) --> { NP = scope:B1, VP = scope:B2, V = sem:B3,
             link(B1,B2,B3)
           }, v(V), np(NP).
```

Figure 2.11: *How to declare the connections which pass the scope information in the VP node.*

Chapter 3

The implementation

3.1 Time in *DRT*

Time intervals will be declared in the same way as the entities which can be accessible in normal anaphoric reference. There will be two special time intervals that will always be declared: the *speech time* (which is the constant *now*) and the first *Rpt*. (This first *Rpt* will be called r_0; the existence of r_0, as it was noted elsewere, makes the first sentence be treated like the following ones.)

The action described by the verb will be bounded by the event or state introduced. This means that the action will be true during and only during that time interval. In order to make this relevant in the *DRS*, the action will be labelled with the time interval. For example, the following discourse

Peter watched through the window.
It was raining outside.
He lit a cigarette.

will have this *DRS*:

r_0 e_1 r_1 s_1 e_2 *now* x y z
peter(x)
window(y)
$r_0 \not\succ e_1$ $e_1 < now$
e_1:watch-through(x,y)
$e_1 \not\succ r_1$ $r_1 \subseteq s_1$ $r_1 < now$
s_1:rain-outside()
cigarette(z)
$r_1 \not\succ e_2$ $e_2 < now$
e_2:light(x,z)

3.2 Syntax

The syntax of the sentences to be analysed is very simple, as the dissertation was centered only in the verbal forms. The syntax declaration, in *PROLOG*-like notation, appears in figure 3.1.

```
discourse --> s, [.], discourse.
discourse --> [].
s --> np, vp.
np --> det, n.
np --> pronoun.
vp --> v, np.
vp --> aux, vp.
```

Figure 3.1: *The grammar rules of the portion of discourse to be accounted for.*

x y now r_0 r_1 s_1
farmer(x)
donkey(y)
$r_0 \not> r_1$ $r_1 \subseteq s_1$
s_1:own(x,y)

```
drs([x,y,now,r0,r1,s1],
    [
        farmer(x),
        donkey(y),
        r0 /> r1,
        r1 in s1,
        s1@own(x,y)
    ]
   )
```

Figure 3.2: *The* DRS *of the sentence* A farmer owned a donkey *and its representation into PROLOG lists.*

3.3 Data structures

In this section I will explain how all the information is coded into *PROLOG*. But before doing so, I would like to note that some time operations have a different form when translated into *PROLOG*. The modifications are:

Operation	*PROLOG* notation	example
$<$	`<`	`r1 < now`
$\not>$	`/>`	`r0 /> r1`
$\subseteq$	`in`	`r1 in s1`
:	`@`	`s1@own(x,y)`

DRT structure

The *DRT* structure will represent the two sets by means of two separate lists:

```
drs([ent1,ent2,...],[cond1,cond2,...])
```

Figure 3.2 shows an example.

Time structure

The time structure will be a list where the first element contains the current temporal points, and the following elements a set of points which occurred sometime in the past. This system allows the treatment of discourse structure, although in this dissertation it will be reduced to the possibility of only one level of flashback.

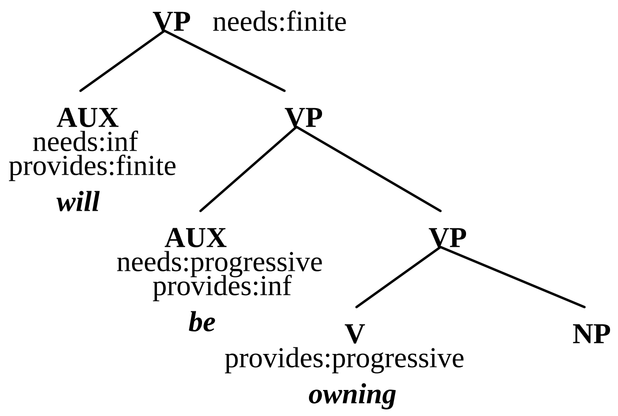


Figure 3.3: *The use of the features* needs *and* provides *for maintaining verb coherence.*

The set of temporal points to be treated in the list are:

Name	Name in $PROLOG$	Explanation
Rpt	`RP`	Reference point
TPpt	`TP`	Temporal perspective point

As the set of temporal points might rise in number in further implementations, it is declared in $PROLOG$ by means of an open list:

```
[[RP,TP|X]|Y]
```

thus permitting the program to be updated more easily.

3.4 Verbal forms

The set of verbal foms used in the dissertation is quite big. This feature makes the problem of verb form coherence quite tricky, since many combinations of verb forms are not permitted, such as "will owning", or sentences like "A farmer own a donkey," which are allowed in the grammar rules of figure 3.1. In order to maintain the coherence in complex verbs, two attributes are used: *needs* and *provides*, following the work exposed in Beaver's dissertation [1].

The feature *needs* will indicate which verbal continuation is needed, and *provides* will say the type of verbal form which is introduced by the current word. Figure 3.3 shows how these features can be used to recognise only correct verb forms.

The whole set of verb forms is:

will own would own	will have owned would have owned	is owning was owning
has owned had owned	will be owning would be owning	owns owned

3.5 Features

In this section I will enumerate all the features which are used in the final *GULP* program, together with a brief explanation of their use.

syn Information used to keep the correct anaphorical references.

> **index** Entity to be referred to.
>
> **arg1** First argument of a transitive verb.[1]
>
> **arg2** Second argument of a transitive verb.

sem Information about *DRS*s and time.

> **in** Input *DRS*.
>
> **out** Modified *DRS*.
>
> **time** Information about time.
>
> > **in** Incoming information about time.
> > **out** Result after (possibly) modifying the *in* information.

scope *DRS* and time which are handled in the scope.

> **in** Input *DRS*.
>
> **out** Modified *DRS*.
>
> **time** Information about time.
>
> > **in** Incoming information about time.
> > **out** Result after (possibly) modifying the *in* information.

res *DRS*s handled in the restrictor.

> **in** Input *DRS*.
>
> **out** Modified *DRS*.

needs What the verbal form needs to make a grammatical sentence.

provides What the verbal form provides to make a grammatical sentence.

Note that not every feature will make sense in every node of the syntactic tree. For example, the features *needs* and *provides* are not used either in the NP or DET or N nodes.

The attribute *scope:time*, as is explained above, collects the temporal information that is modified in the scope. That is, *scope:time* behaves in a way which is very similar to *scope*. However, their function is not so similar. Whereas *scope* basically adds elements to a list without consulting what was there before, *scope:time* transforms one temporal structure into another depending on how it was before. Furthermore, this transformation is very simple: when $PROG$ is checked the *Rpt* is updated, and when $PAST$ is checked it is the *TPpt* who is updated (independently of whether there is flashbacking or not). As both operations are independent, there is no need of chaining *in*s with *out*s; despite, all the AUX and V share the *scope:time* of the common embedded NP. This has been done by linking the *scope:time* of the NP with the *sem:time* of the corresponding main or auxiliary verb, as figure 3.4 shows. The use of a declarative language like PROLOG and of shared time data for the nodes in the VP in the way it has been implemented here allow the nodes to process these data independently, making irrelevant which features are checked before. That is, as if the features were being checked in parallel.

[1] In order to keep simplicity, all the verbs used are transitive.

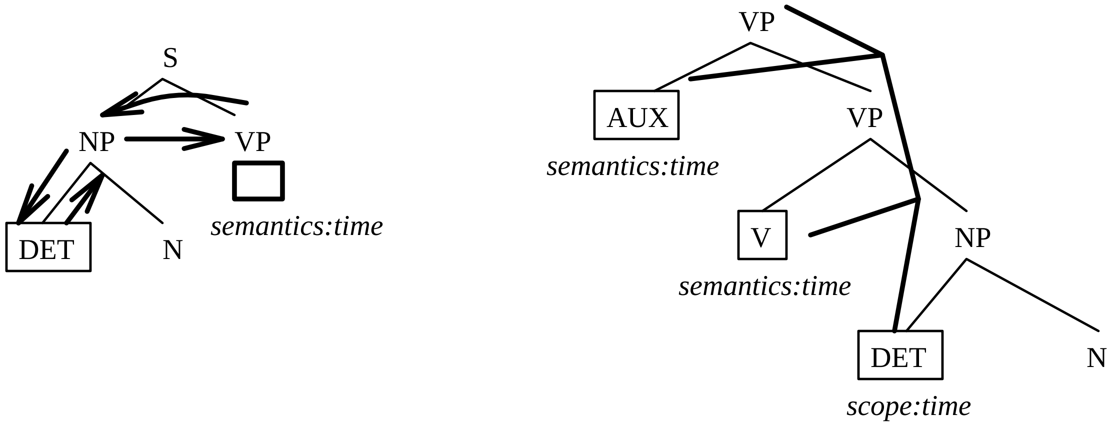


Figure 3.4: *Temporal information in the field "scope:time." Note that the variables in "scope:time" in the whole VP are those of the embedded NP.*

form	features	form	features
will	$PAST = -, TENSE = fut$	ptperf	$PROG = +, PERF = +$
would	$PAST = +, TENSE = fut$	progr	$PROG = +, PERF = -$
has	$PAST = -, TENSE = pres$	finite verb	$PROG = ?, PERF = -$
had	$PAST = -, TENSE = past$	finite aux	*none*
is	$PAST = -, TENSE = pres$	present s.	$-\quad PRES \quad ? \quad -$
was	$PAST = -, TENSE = past$	past simple	$-\quad PAST \quad ? \quad -$

Table 3.1: *Verb forms and features.*

3.6 From verb forms to features

Each verb form has a set of features associated with it. In chapter 1 the features $PAST$, $TENSE$, $PROG$ and $PERF$ were studied. We saw that every verb form has a very precise value for every feature. In the verbs studied, where a simple flashbacking was studied and no locatives were considered, for every verb form there was only one possible value for every feature. This is not completely true, however, as Eberle and Kamp showed [5] [8]. Although keeping this in mind, this dissertation deals only with the analysis of chapter 1, and the implementation will take only this data into account, for simplicity.

The analysis performed in chapter 1 permits a very simple association word-feature for compound verbs, in such a way that the feature analysis results to be distributed along the verb form. The whole set of verb forms is, together with their syntactic analysis and features, shown in fig 3.5. The features can be transferred to the leaves of the syntactic tree, as it can be seen in the same figure. This permits the association word-feature which appears in table 3.1.

I repeat that this association may be artificial and only useful for computer performance, and has no theoretical robustness at all. Bearing this in mind, note that

- The auxiliary verbs have associated $PAST$ and $TENSE$ features.

- The auxiliary infinite verbs have no temporal information at all.

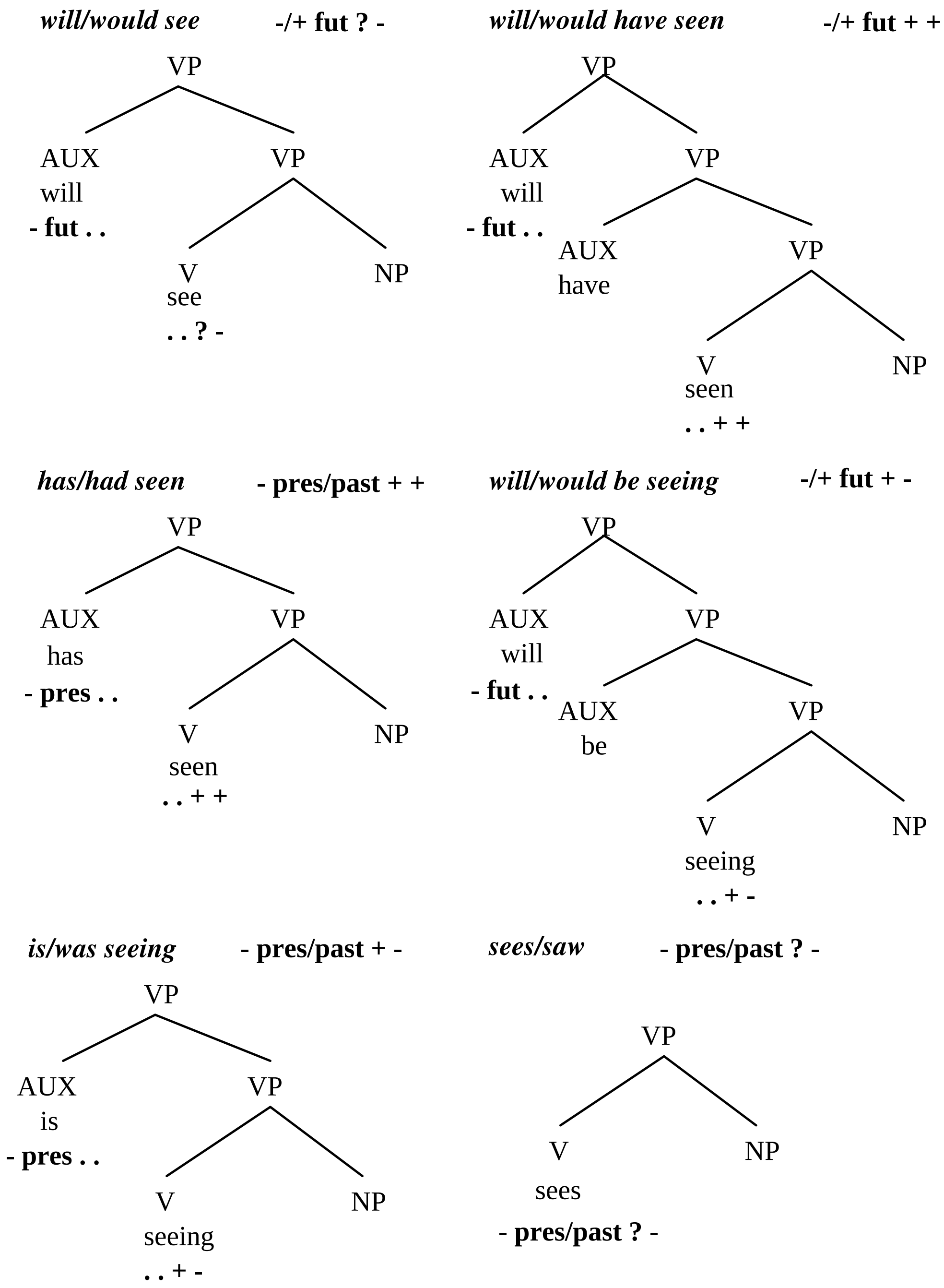


Figure 3.5: *The set of forms together with their syntactic analysis and features.*

- The main verbs of the compound forms only cover the *PROG* and *PERF* features.

And, as was mentioned in chapter 1, the aspectual information (*PROG*, *PERF*) introduces the new referents and establishes the relationships among them, whereas the temporal information (*PAST*, *TENSE*) relates the *TPpt*, the *ST*, the *Rpt* and the old *Rpt* among them.

3.7 Flashback

The flashback mechanism in this dissertation is very simple. The feature *PAST* controls flashbacking; when it is off, there is no flashbacking at all. When it is on, a level of flashbacking exists. In other words, the first $+PAST$ introduces flashbacking, and the first $-PAST$ finishes it.

When flashback occurs, the current time structure is pushed into a stack, and the new *TPpt* coincides with the *Rpt* of the pushed time structure. As flashbacking is controlled by the feature *PAST*, pushing or popping is done in the node which checks this feature during the parsing process. The other nodes will use the time structure which this node outputs to the shared variable attribute *sem:out* in a way which suggests parallel processing.

3.8 An example

As a final part of this chapter, an example of sentence analysis, *DRS* construction and flashback treatment will be explained. The discourse is:

A man went to the station.
He would wait for the train.
He lit a cigarette.

This discourse has been analyzed in chapter 1. Here, we will see how the program of this dissertation will handle it. In order to follow the explanations of this section, it is advisory to see the actual sentences of the program in the Appendix B, especially those concerning with the verb forms. Note that the vocabulary of this discourse is not covered by the program, but this enhancement can easily be done. Another feature which has not been implemented is the treatment of definite *NP*s ("the train"), but this is outside the scope of this dissertation. We shall focus only in the part relating to verb analysis.

The first sentence,

A man went to the station,

introduces an event by means of a verb in its simple past form. The node V will contain the predicate

```
past_event(V,go)
```

which will have, as input data

```
V = sem:time:in:[[r0|[now|_]]]   ..   sem:in:[drs([x1,x2],[gender(x1,m),
                                                      gender(x2,n),
                                                      man(x1),
                                                      station(x2)
                                                     ] ) ]
```

where `x1` represents the man and `x2` represents the station. These two references will have been made in the corresponding *NP* nodes.

The old reference point `Orp` will be instantiated to `r0`, and the time structure which results after being processed by this node is

```
V = sem:time:out:[[e1,now|_]]
```

which means that *TPpt* equals to *now* (as a consequence of $-PAST$), and that the new *Rpt* becomes the new event e_1 (as a consequence of $-PROG$). The resulting *DRS* will be:

```
V = sem:out:[drs([e1,x1,x2],[r0/>e1,e1<now,e1@go(x1,x2),gender(x1,m),...])]
```

The next sentence is more difficult:

He would wait for the train.

In this case, a compound verb is used. Each one of the two words of the verb will share the same time structure. They both will have, as input, the output of the last sentence:

```
sem:time:in:[[e1,now|_]]
```

The word *would* will introduce flashback (because of $+PAST$), giving the following structure:

```
sem:time:out:[[RP,e1|_],[e1,now|_]]
```

This structure will be shared by both verb words. The form *wait*, which is in the infinitive form, will instantiate the `RP`, giving the final time structure:

```
sem:time:out:[[e2,e1|_],[e1,now|_]]
```

The *DRS* costruction is made in a different way. First, the word *wait* receives:

```
sem:in:[drs([x3,e1,x1,x2],[gender(x3,n), train(x3), ...... ])]
```

and will give as a result:

```
sem:out:[drs([s1,e2,x3,e1,x1,x2],
             [e2 in s1, s1@wait(x1,x3),gender(x3,n), train(x3), ...... ])]
```

This result will be passed to *would* in `sem:in`. This verb will update it to the final *DRS* up to this sentence:

```
sem:out:[drs(...., [e1<e2, e1<now, ....])]
```

And finally, the third sentence:

He lit a cigarette.

This last sentence will be analyzed like the first one. But note that the second sentence had introduced a level of flashbacking. This third sentence must pop it, since $-PAST$ holds, giving a time structure:

```
sem:time:in:[[e2,e1|_],[e1,now|_]]
        out:[[e3,now|_]]
```

and the *DRS*:

```
sem:in:[drs([x4,s1,e2,x3,e1,x1,x2],[gender(x4,n),cigarette(x4),.....])]
   out:[drs([e3,x4,s1,e2,x3,e1,x1,x2],
            [e1/>e3,e3@light(x1,x4),gender(x4,n),cigarette(x4),
             e1<e2,e2<now,e2 in s1, s1@wait(x1,x3),
             gender(x3,n),train(x3),
             r0/>e1,e1<now,e1@go(x1,x2),
             gender(x1,m),gender(x2,n),man(x1),station(x2)])]
```

Note that the order in which the conditions appear is generally reversed to that in the utterances. This feature does not change the truth-conditions, but makes the result less readible. To overcome this, a predicate called `display_drs` has been defined which, among other things, reverses the lists of the *DRS*.

Chapter 4

Conclusions and further research

From this dissertation a few conclusions can be made, and a lot of ideas for further research can be obtained. In this chapter I will try to make a brief review of all them.

4.1 Conclusions

In this dissertation, an attempt to put very related theories even closer has been made, all this with the help of programming skills. The conclusions which can be made from this work are very simple. The two more general ones is that *DRT* can be straightforwardly improved to account for temporal information too, as several researchers have pointed out (for example, Barbara Partee in [10]), and that *DRT* theory can be implemented with a computer program. But there are other conclusions and achievements shown through this dissertation:

- The ideas of flow of information through an analysis tree presented by Johnson and Klein [6] have been exploited extensively in this dissertation. The conclusions to be drawn from it is that Johnson and Klein's algorithm has been proven to be powerful enough to be improved to account for time.

- Eberle and Kasper's analysis of temporal features is very useful for the integration of time into *DRT* and it can be implemented in a computer program, as the actual implementation has shown.

- Flashbacking has been modelled with a stack of temporal structures and controlled with the *PAST* feature. This system has proven to be very useful, although discourse structure and context may be needed too.

- The idea that a state surrounds not the previous *Rpt* but a new one allows an analysis of a wider range of data and simplifies the analysis of future forms.

4.2 Further research

This dissertation did not intend to cover all the areas relating to time. It has been done, rather, as an attempt to elaborate a part of a framework which could be used for analysing temporal aspects of sentences in a very explicit way, so that it may be used in a computer. The second goal (that it can be used as a computer program) is not the least important, since this ability allows the theory to be easily tested, and it opens the door even wider for human discourse processing by computers, which is one of my main goals.

As this is a startpoint, there are many subareas which still need to be considered. Some of them have been mentioned in some paragraphs throughout this report. In the remaining of this section I will introduce them again, together with a short comment of each.

Locatives

Adverbs of location behave in a way which is similar to that of verb tense. They also refer to the temporal points discussed in chapter 1, and normally they add precision to what the verbs say, being then complementary to them. For example, one of the first examples of chapter 1 was:

(3) Peter got up at 5h 30

In this example, the verb *got up* says that the event occurred somewhere in the past, and the locative *at 5h 30* goes further by saying more explicitely when it happened.

The very fact that a locative is usually more precise than a verb form adds more complexity to locative analysis. Furthermore, as what they do is complement what the verb forms said, the analysis of the verb forms themselves becomes more complex, definitely more than the analysis reviewed in this dissertation, as it can be seen in works like that of Kamp in [8]. This further complication in verb form analysis and the integration of the locatives is, then, a convenient way of complementing the work done in this dissertation.

Discourse structure

Discourse structure is, perhaps, the toughest of the possible ways of improving this dissertation.

A few modifications of *DRT* theory might be needed to take in account more complex discourse structures, like for example the one proposed by [11].

There is also the problem that sometimes the relationship *old Rpt/new Rpt* is not that the new one occurs *just after* the old one. Sometimes, depending on the context, it occurs before, or is inside the old *Rpt*. This may imply a revision of part of the theory here stated (concretelly the interpretation of the temporal feature *tense*, as it has been pointed out at the end of chapter 1). A work which may be useful in this integration is the one done by Caenepeel, Isard and Klein in [2].

Quantification

Quantification poses an interesting problem to temporal reference. Consider the following sentence:

(25) A farmer beat every donkey. He laughed loudly.

When did the farmer laugh? There are two main answers to this question:

- After he beat all the donkeys, a farmer laughed loudly.
- After he beat each one of the donkeys, a farmer laughed loudly.

Wich one of them is the correct answer? The solution may depend on the context, or on the discourse structure, and perhaps a modification of the theory exposed in this dissertation may be needed.

Temporal quantification

Temporal quantification, like the use of *whenever* or $when - always$ in sentences like

(26) when it rains, the street is always wet

might be seen as forming part of *quantification* or *locatives*, but there are also problems that occur only to temporal quantification which suggest that the *DRT* should have more changes, as Ogihara pointed out in [9].

Present forms

Other of the main limitations of this dissertation is the poor treatment of the *speech time ST*, which makes the analysis of the present forms incomplete. For example, we can see that the *ST* plays an important role in reportive speech, since the events happen while the reporter is telling them. If we update the *ST* in each utterance in such a way that a chain of points each one occurring *just after* the previous one is made, we can model how time passes and therefore reportive speech can be accounted for. In other words, reportive speech is different only because the *ST* varies in each utterance.

Note that if this is so, an interesting paralelism between present and past forms will appear: in past forms, there is a chain of events in the past, each one occurring just after the previously related one, which makes the *Rpt* shift forward in time; in present forms, the same holds except that the last event occurs at the current speech time. Interesting as well, the analysis of flashbacking (at least at the level presented in this report) reinforces this paralelism, with the only difference that the *TPpt* shifts to the past. This will happen even with the introduction of future forms. Compare, for example:

Peter arrived at the station.	*Peter arrives at the station.*
He would wait for the train.	*He will wait for the train.*
He lit a cigarette.	*He lights a cigarette.*

In the second discourse, a behaviour appears which is very similar to the flashback mechanism of the first discourse.

Aspect

Not everything concerning with aspect has been treated in this dissertation. There are still other areas which need attention too. Consider for example the case of putting limits to states. The main difference between a state and an event is that an event is bounded, where a state may remain open. For example, the sentence

It was raining.

does not imply that it has stopped raining. This has been modelled by saying that *it was raining* introduces a state and part of it occurs somewhere in the past.

However, the use for example of the future may present problems. If someone says

Peter will love her.

s/he almost certainly is saying that Peter does not love her yet. That is, the state introduced occurs entirely after the speech time. However, this implication cannot be made with the analysis of this dissertation. A new time feature might be needed to account for this.

Bibliography

[1] David Beaver. Discourse anaphora: extending a declarative analysis with particular reference to modal subordination and related phenomena. Msc thesis, Department of AI, the University of Edinburgh, Edinburgh, 1989.

[2] M. Caenepeel, S. Isard, and E. Klein. Towards a cognitive account of temporal reference in narrative discourse. End of ESRC award report, Department of Cognitive Science, the University of Edinburgh, Edinburgh, March 1992.

[3] Michael A. Covington. GULP 2.0: an extension of Prolog for unification-based grammar. Technical Report AI-1989-01, AI Programs, the University of Georgia, Athens, January 1989. Available via FTP from aisun1.ai.uga.edu (128.192.12.9).

[4] Michael A. Covington and Nora Schmitz. An implementation of Discourse Representation Theory. Technical Report 01-0023, AI Programs, the University of Georgia, Athens, 1989. Available via FTP from aisun1.ai.uga.edu (128.192.12.9).

[5] Kurt Eberle and Walter Kasper. Tense, aspect and temporal structure in French. In Hans Kamp, editor, *Tense and Aspect in English and French*, pages 4–40. DYANA deliverable R2.3.B, Stuttgart, January 1991.

[6] Mark Johnson and Edward Klein. Discourse anaphora and parsing. Technical Report 86-63, Stanford University, October 1986.

[7] Hans Kamp. A theory of truth and semantic representation. In Janssen Groenendijk and Stokhof, editors, *Formal Methods in the Study of Language*, pages 277–322. Mathematical Centre Tracts, Amsterdam, 1981.

[8] Hans Kamp. The perfect and other tenses in French and English. In Hans Kamp, editor, *Tense and Aspect in English and French*, pages 41–64. DYANA deliverable R2.3.B, Stuttgart, January 1991.

[9] Toshiyuki Ogihara. Frequency adverbs, temporal adverbial clauses, and DRT. In Hans Kamp, editor, *Tense and Aspect in English and French*, pages 65–74. DYANA deliverable R2.3.B, Stuttgart, January 1991.

[10] Barbara H. Partee. Nominal and temporal anaphora. *Linguistics and Philosophy*, 7:243–286, 1984.

[11] Craige Roberts. Modal subordination and pronominal anaphora in discourse. Technical Report CSLI 88-127, Standford Centre for the Study of Language and Information, Standford University, 1988.

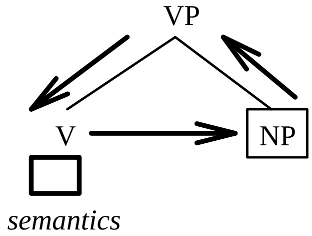
Appendix A

Flow of information

In this appendix the flow of information is shown through the syntactic tree. A node labelled with a thin box indicates the node which introduces the first *in* attribute and collects the *out* result; it might be seen as the "calling node." A thick box under a node indicates that that node (and possible related tree) modifies the information as it passes through it. If the box is labelled, the label is the attribute to where the information is passed (and modified). For example, if the *GULP* program contains the instructions:

```
vp(VP) --> { NP = scope:A1,  VP = scope:A2,  V = sem:A3,
             link(A1,A2,A3),
               .........
            }, v(V), np(NP).
v(V) --> [beat],
         { V = sem:in:IN,
             .........
           V = sem:out:OUT
         }.
```

the flow of information is of the form



The case of the tree of figure A.6, where syntactic information is shown, is slightly different: in that case, a labelled box indicates where the attribute is instantiated, the thick lines show links between nodes, and the labels say the name of the attributes to be linked together.

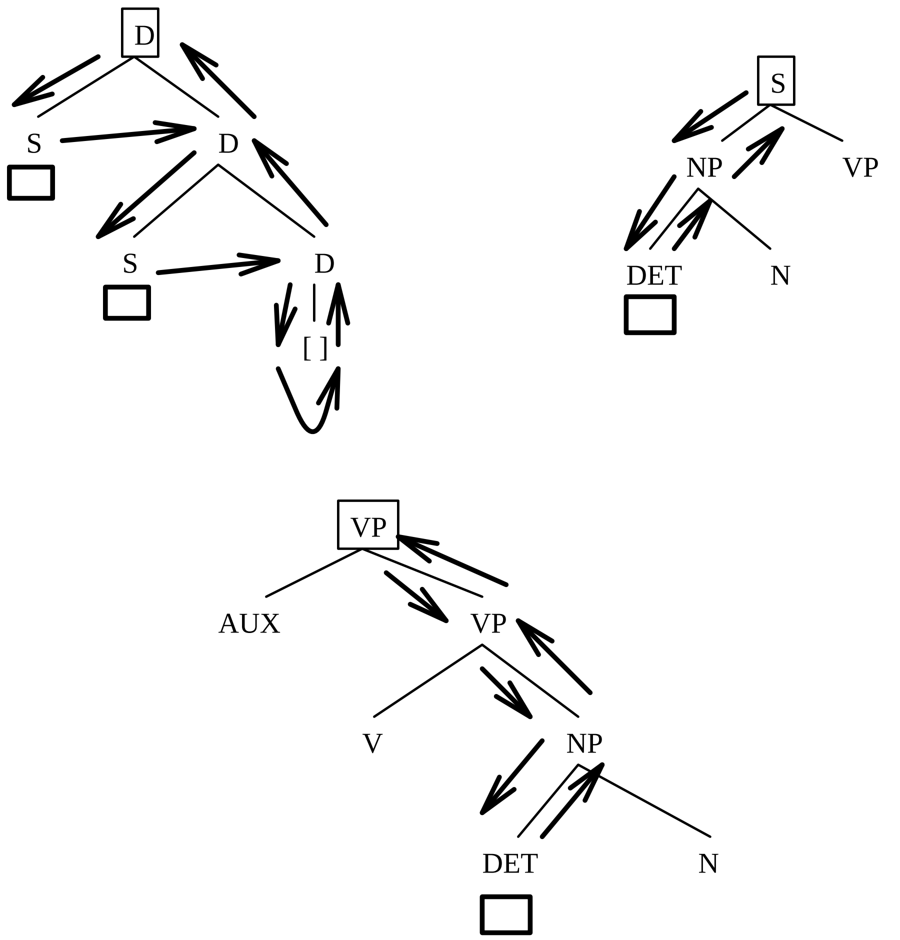


Figure A.1: *Semantic information.*

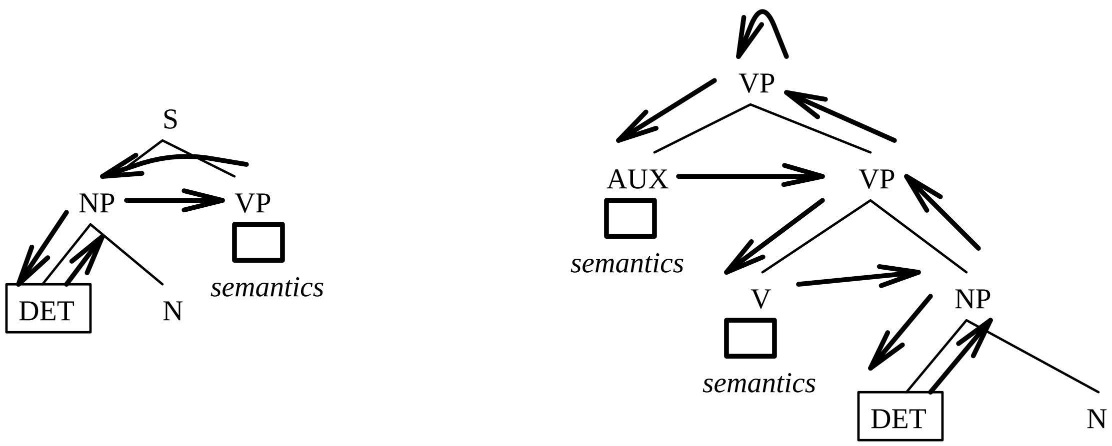


Figure A.2: *Scope information.*

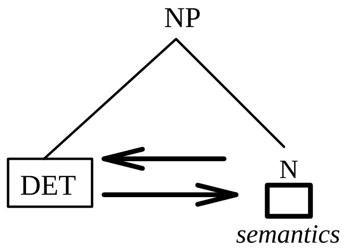


Figure A.3: *Restrictor information.*

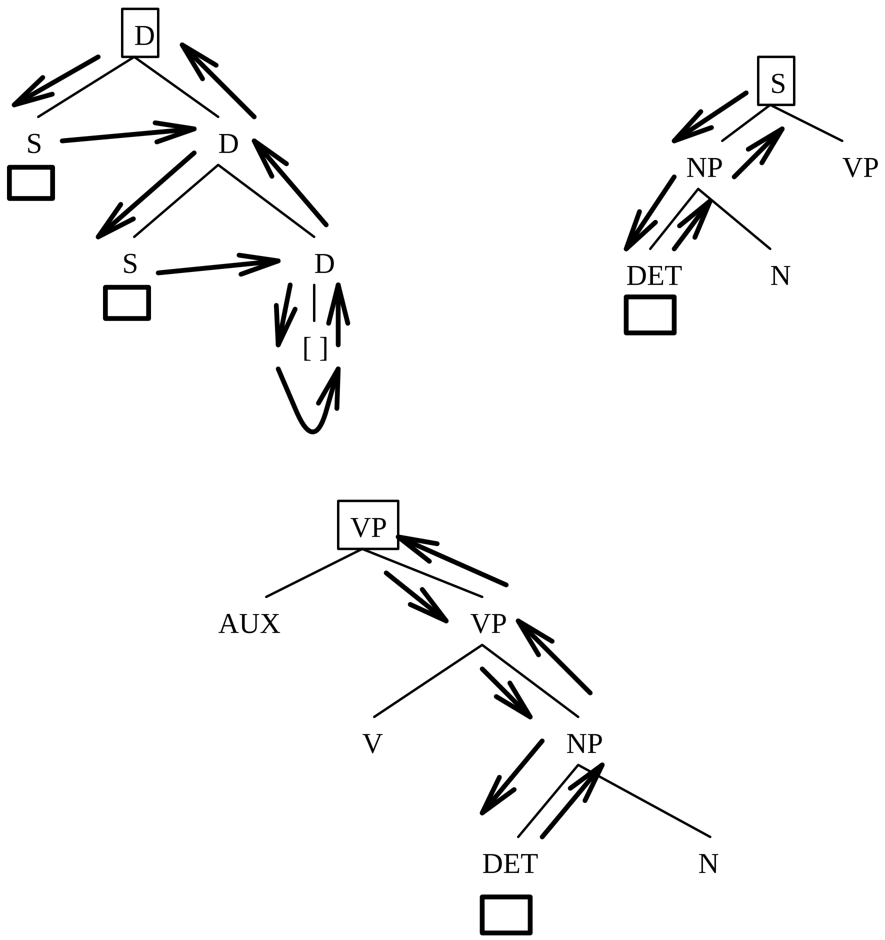


Figure A.4: *Temporal information in the field "sem:time."*

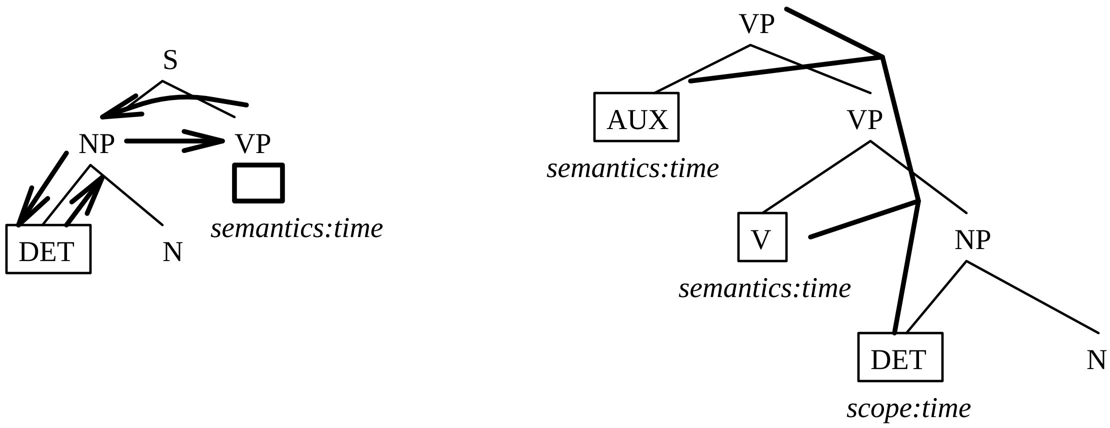


Figure A.5: *Temporal information in the field "scope:time." Note that the variables in "scope:time" in the whole VP are those of the embedded NP.*

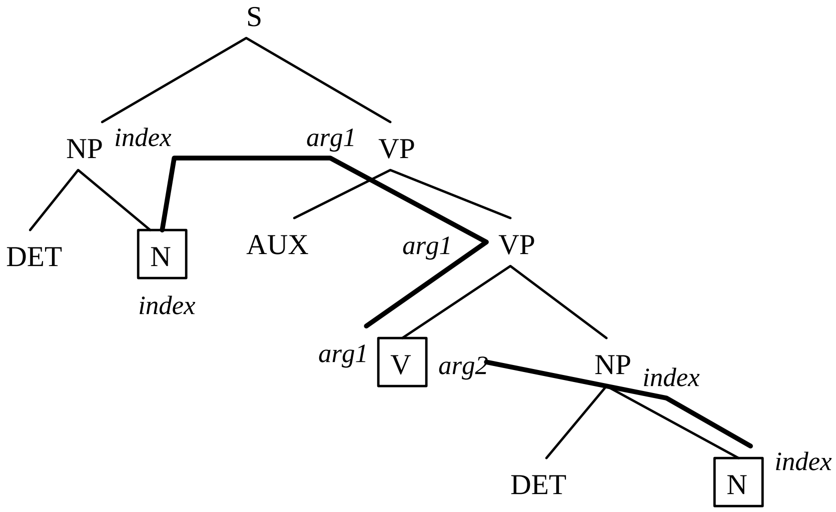


Figure A.6: *Syntactic information: parameters "index", "arg1" and "arg2."*

Appendix B

The program

The final program has been implemented in a UNIX machine using Edinburgh PROLOG 1.5 and GULP 2.0 (which has been adapted conforming to the syntax of Edinburgh PROLOG). Special care has been made in such a way that the program can be easily modified, thought perhaps the listing is not well documented.

A set of tests is included in the program (called `test1`, `test2`, etc.). For building new tests the predicate `try` must be used. For example, the declaration of `test1` is:

```
test1 :- try([a,man,beat,a,donkey,'.']).
```

The syntax of the data is very simple. In PROLOG-like notation:

```
discourse --> s, [.], discourse.
discourse --> [].
s --> np, vp.
np --> det, n.
np --> pronoun.
vp --> v, np.
vp --> aux, vp.
```

And the vocabulary has been reduced to two nouns, `farmer` and `donkey` and three verbs, `beat`, `own` and `kick`. Although the vocabulary size is very restricted, it can be easily incremented.

```
% Discourse Representation Theory
%       Implementation of DRT with anaphora and time;
%       no quantification used yet

% Features used so far:
%       sem:    in
%               out
%               time:
%                    in
%                    out
%       syn:    arg1
%               arg2
%               index
%       scope:  in
%               out
%               time:
%                    in
%                    out
%       res:    in
%               out
%       needs
%       provides

%       time structure: [[RefPt,TPpt,OldRpt|Rest]|Stacked]



:- op(100,xfx,[/>,@,in]). % operators of time

% Link(A1, A2, A3)
%       Links A1, A2 and A3:  =>A1 out - out A3->
%                               in           in
%                                 \         /
%                                  in A2 out
%                                     |
%                                     V

link(A1,A2,A3):-
        A1 = in:Z1,     A2 = in:Z1,
        A2 = out:Z2,    A3 = in:Z2,
        A3 = out:Z3,    A1 = out:Z3.

% new_entity(X) new_state(X) new_event(X) new_ref(X)
%       instantiate X to a different entity, state, event or reference time
%       every time it is called, thereby generating
%       unique indices.
```

```
new_entity(X) :-
        retract(unique_ent(N)),
        !,
        NewN is N+1,
        name(NewN,List),
        name(x,[C]),
        name(X,[C|List]),
        asserta(unique_ent(NewN)).

unique_ent(0).

new_state(S) :-
        retract(unique_state(N)),
        !,
        NewN is N+1,
        name(NewN,List),
        name(s,[C]),
        name(S,[C|List]),
        asserta(unique_state(NewN)).

unique_state(0).

new_event(E) :-
        retract(unique_ev(N)),
        !,
        NewN is N+1,
        name(NewN,List),
        name(e,[C]),
        name(E,[C|List]),
        asserta(unique_ev(NewN)).

unique_ev(0).

new_ref(R) :-
        retract(unique_ref(N)),
        !,
        NewN is N+1,
        name(NewN,List),
        name(r,[C]),
        name(R,[C|List]),
        asserta(unique_ref(NewN)).

unique_ref(0).
```

```
%
% reverse(List,Result)
%   Fast list reversal with stacks.
%   NOTE: If the tail of the list is uninstantiated, this procedure
%   will instantiate it to nil. So don't use this procedure on
%   open lists that need to remain open.
%

reverse(List,Result) :-
     nonvar(List),
     reverse_aux(List,[],Result).

reverse_aux([],Result,Result).

reverse_aux([H|T],Stack,Result) :-
     reverse_aux(T,[H|Stack],Result).


% Display_drs:
%       Displays drt structures


display_drs(X) :- display_drs_indented(X,0).

display_drs_indented(X,N) :-
     var(X),
     !,
     write(X),nl.

display_drs_indented(X>Y,N) :-
     !,
     tab(N), write('IF:'), nl,
     NN is N+2,
     display_drs_indented(X,NN),
     tab(N), write('THEN:'), nl,
     display_drs_indented(Y,NN).

display_drs_indented(neg(X),N) :-
     !,
     tab(N), write('NOT:'), nl,
     NN is N+2,
     display_drs_indented(X,NN).

display_drs_indented(query(X),N) :-
     !,
     tab(N), write('QUERY:'), nl,
     NN is N+2,
     display_drs_indented(X,NN).
```

```
display_drs_indented(drs(X,Y),N) :-
     !,
     reverse(X,RX),
     tab(N), write(RX), nl,
     reverse(Y,RY),
     display_drs_indented(RY,N).

display_drs_indented([H|T],N) :-
     !,
     display_drs_indented(H,N),
     display_drs_indented(T,N).

display_drs_indented([],_) :- !.

display_drs_indented(Cond,N) :-
     tab(N), write(Cond), nl.

% Nouns
%       Each noun generates a unique index and inserts it,
%       along with a condition, into the DRS that is passed
%       to it.

n(N) --> [man],
             {  new_entity(C),
                N = syn:index:C ..
                    sem:  (in:[drs(U,Con)|Super] ..
                           out:[drs([C|U],[man(C),gender(C,m)|Con])|Super])
             }.

n(N) --> [donkey],
             {  new_entity(C),
                N = syn:index:C ..
                    sem:  (in:[drs(U,Con)|Super] ..
                           out:[drs([C|U],[donkey(C),gender(C,n)|Con])|Super])
             }.
```

```
% Verbs
%       Each verb is linked to the indices of its arguments through
%       syntactic features. Using these indices, it adds the
%       appropriate predicate to the semantics.
%               Time is also handled.

past_event(V,Name):- /* Features: -PAST past -PROG -PERF */
          new_event(RP),
          ( V = sem:time:in:[[Orp|_]]; V = sem:time:in:[_,[Orp|_]]),
          V = sem:time:out:[[RP,now|_]] ..
              syn:  (arg1:Arg1 .. arg2:Arg2),
          Pred =.. [Name,Arg1,Arg2],
          V =   sem:(in:[drs(U,Con)|Super] ..
                     out:[drs([RP|U],[Orp/>RP,RP<now,RP@Pred|Con])
                          |Super] ) ..
                provides:finite.

past_state(V,Name):- /* Features: -PAST past +PROG -PERF */
          new_state(S), new_ref(RP),
          ( V = sem:time:in:[[Orp|_]]; V = sem:time:in:[_,[Orp|_]]),
          V = sem:time:out:[[RP,now|_]] ..
              syn:  (arg1:Arg1 .. arg2:Arg2),
          Pred =.. [Name,Arg1,Arg2],
          V =   sem:(in:[drs(U,Con)|Super] ..
                     out:[drs([S,RP|U],[Orp/>RP,RP in S,RP<now,S@Pred|Con])
                          |Super] ) ..
                provides:finite.

present_state(V,Name):- /* Features: -PAST pres +PROG -PERF */
          new_state(S),
          V = sem:time:out:[[now,now|_]] ..
              syn:  (arg1:Arg1 .. arg2:Arg2),
          Pred =.. [Name,Arg1,Arg2],
          V =   sem:(in:[drs(U,Con)|Super] ..
                     out:[drs([S|U],[now in S,S@Pred|Con])|Super] ) ..
                provides:finite.

ptperf(V,Name):- /* Features: +PROG +PERF */
          new_event(E), new_state(S), new_ref(RP),
          V = sem:time:out:[[RP|_]|_] ..
              syn:  (arg1:Arg1 .. arg2:Arg2),
          Pred =.. [Name,Arg1,Arg2],
          V =   sem:(in:[drs(U,Con)|Super] ..
                     out:[drs([S,E,RP|U],[RP in S,E/>S,E@Pred|Con])|Super] ) ..
                provides:ptperf.
```

```
inf_event(V,Name):- /* Features: -PROG -PERF */
          new_event(RP),
          V = sem:time:out:[[RP|_]|_] ..
              syn:  (arg1:Arg1 .. arg2:Arg2),
          Pred =.. [Name,Arg1,Arg2],
          V =   sem:(in:[drs(U,Con)|Super] ..
                     out:[drs([RP|U],[RP@Pred|Con])|Super] ) ..
                provides:inf.

inf_state(V,Name):- /* Features: +PROG -PERF */
          new_state(S), new_ref(RP),
          V =   sem:time:out:[[RP|_]|_],
          V =   syn:  (arg1:Arg1 .. arg2:Arg2),
          Pred =.. [Name,Arg1,Arg2],
          V =   sem:(in:[drs(U,Con)|Super] ..
                     out:[drs([S,RP|U],[RP in S, S@Pred|Con])|Super] ) ..
                provides:inf.

progress(V,Name):- /* Features: +PROG -PERF */
          new_state(S), new_ref(RP),
          V =   sem:time:out:[[RP|_]|_] ..
                syn:  (arg1:Arg1 .. arg2:Arg2),
          Pred =.. [Name,Arg1,Arg2],
          V =   sem:(in:[drs(U,Con)|Super] ..
                     out:[drs([S,RP|U],[RP in S, S@Pred|Con])|Super] ) ..
                provides:progressive.

v(V) --> [kicked],
        {  past_event(V,kick)  }.

v(V) --> [kicked],
        {  ptperf(V,kick)  }.

v(V) --> [kick],
        {  inf_event(V,kick)  }.

v(V) --> [kicking],
        {  progress(V,kick)  }.

v(V) --> [beat],
        {  past_event(V,beat)  }.

v(V) --> [beaten],
        {  ptperf(V,beat)  }.

v(V) --> [beat],
        {  inf_event(V,beat)  }.

v(V) --> [beating],
        {  progress(V,beat)  }.
```

```
v(V) --> [owned],
       {  past_state(V,own)  }.

v(V) --> [owns],
       {  present_state(V,own)  }.

v(V) --> [owned],
       {  ptperf(V,own)  }.

v(V) --> [own],
       {  inf_state(V,own)  }.

v(V) --> [owning],
       {  progress(V,own)  }.

aux(AUX) --> [had], /* Features: -PAST past */
      { (AUX = sem:time:in:[[Orp|_]]; AUX = sem:time:in:[_,[Orp|_]]),
        AUX = sem:time:out:[[RP,now|_]] ..
              sem:(in:[drs(U,Con)|Super] ..
                   out:[drs(U,[Orp/>RP,RP<now|Con])|Super]) ..
              needs:ptperf ..
              provides:finite }.

aux(AUX) --> [have], /* Features: none */
     {  AUX = sem:(in:A .. out:A) ..
              needs:ptperf ..
              provides:inf  }.

aux(AUX) --> [will], /* Features: -PAST fut */
     {  AUX = sem:time:out:[[RP,now|_]] ..
              sem:(in:[drs(U,Con)|Super] ..
                   out:[drs(U,[now<RP|Con])|Super]) ..
              needs:inf ..
              provides:finite }.

aux(AUX) --> [would], /* Features: +PAST fut */
     {  (AUX = sem:time:(in:[[_,PP|_],X] .. out:[[RP,PP|_],X]);
         AUX =sem:time:(in:[[PP|Y]] .. out:[[RP,PP|_],[PP|Y]])),
        AUX = sem:(in:[drs(U,Con)|Super] ..
                   out:[drs(U,[PP<RP,PP<now|Con])|Super]) ..
              needs:inf ..
              provides:finite }.

aux(AUX) --> [was], /* Features: -PAST pres */
     {  AUX = sem:(time:out:[[RP,now|_]] ..
                   in:[drs(U,Con)|Super] ..
                   out:[drs(U,[RP<now|Con])|Super]) ..
              needs:progressive ..
              provides:finite  }.
```

```
aux(AUX) --> [be], /* Features: none */
     {  AUX = sem:(in:A .. out:A) ..
              needs:progressive ..
              provides:inf  }.

% Determiners
%       Determiners tie together the semantics of their scope
%       and restrictor. This simple determiner, 'a', simply
%       passes semantic material to its restrictor and then
%       to its scope. A more complex determiner such as 'every' would
%       pass an empty list to its scope and restrictor,
%       collect whatever semantic material they add, and then
%       arrange it into an if-then structure.

det(Det) --> [a],
            {   Det = sem:A1 .. res:A2 .. scope:A3,
                link(A1,A2,A3),
                Det = scope:time:B .. sem:time:B }.

% Noun Phrase
%       Passes semantic material to the determiner, which will
%       specify the logical structure. It also links the scope and
%       restrictor, and the syntactic event.

np(NP) --> {    NP = sem:A,     Det = sem:A,
                Det = res:B,    N = sem:B,
                Det = scope:C,  NP = scope:C,
                NP = syn:D,     N = syn:D  },  det(Det), n(N).

% Anaphoric pronouns
%       Pronouns must look for the reference in the list of
%       active drs and check gender. As no determiners are used,
%       they pass semantic information to the scope and collect
%       the results.

np(NP) --> ([he];[him]),
          {     NP = sem:in:DrsList,
                member(drs(U,Con),DrsList),
                member(Index,U),
                member(gender(Index2,m),Con), Index == Index2,
                NP = syn:index:Index ..
                     scope:(in:DrsList ..
                            out:DrsOut ..
                            time:T) ..
                     sem:(out:DrsOut ..
                          time:T)  }.
```

```
np(NP) --> ([she];[her]),
          {     NP = sem:in:DrsList,
                member(drs(U,Con),DrsList),
                member(Index,U),
                member(gender(Index2,f),Con), Index == Index2,
                NP = syn:index:Index ..
                     scope:(in:DrsList ..
                            out:DrsOut ..
                            time:T) ..
                     sem:(out:DrsOut ..
                          time:T)  }.

np(NP) --> ([it]),
          {     NP = sem:in:DrsList,
                member(drs(U,Con),DrsList),
                member(Index,U),
                member(gender(Index2,n),Con), Index == Index2,
                NP = syn:index:Index ..
                     scope:(in:DrsList ..
                            out:DrsOut ..
                            time:T) ..
                     sem:(out:DrsOut ..
                          time:T)  }.

% Verb Phrase
%       Passes the semantic material to the embedded NP
%        (the direct object).
%       Passes the time material from the NP to the verb and then
%        collects the result to the VP.
%       Passes semantic material from the VP to the verb  and then
%        to the scope of the NP.
%       Passes the syntactic markers to the verb.

vp(VP) -->  {   VP = sem:A,      NP = sem:A,
                NP = scope:B1,   VP = scope:B2,   V = sem:B3,
                link(B1,B2,B3),
                NP = scope:time:T, V = sem:time:T, VP = scope:time:T,
                V = syn:arg2:C,  NP = syn:index:C,
                VP = syn:arg1:D, V = syn:arg1:D,
                V = provides:X,  VP = provides:X },  v(V), np(NP).
```

```
% Verb phrase with auxiliaries
%       The VP with auxiliar verb is like the one with normal verb,
%       except that the syntactic information does not go to the AUX.

vp(VP) -->
            {   VP = sem:A,       VP2 = sem:A,
                VP2 = scope:B1,   VP= scope:B2, AUX = sem:B3,
                link(B1,B2,B3),
                VP2 = scope:time:T, AUX = sem:time:T, VP =scope:time:T,
                VP = syn:C,       VP2 = syn:C,
                AUX = provides:X, VP = provides:X,
                AUX = needs:Y,    VP2 = provides:Y
            }, aux(AUX), vp(VP2).

% Sentence
%       Passes semantic material to the subject NP.
%       Passes VP semantics to the subject NP as its scope.
%       Passes time material to the VP.

s(S) -->  {     S = sem:A,       NP = sem:A,
                NP = scope:B,    VP = sem:B,
                VP = scope:in:C, VP = scope:out:C,
                VP = syn:arg1:D, NP = syn:index:D,
                VP = provides:finite  },  np(NP),vp(VP).

% Discourse
%       Passes semantic  and time material to the current sentence and then to
%       the next stage of the discourse.

discourse(D1) -->
           {    D1 = sem:V1, S = sem:V2, D2 = sem:V3,
                link(V1,V2,V3),
                D1 = sem:time:T1, S = sem:time:T2, D2 = sem:time:T3,
                link(T1,T2,T3)   }, s(S),['.'],{!},discourse(D2).

discourse(D) --> [],
           {    D = sem:(in:A .. out:A),
                D = sem:time:(in:B .. out:B)  }.

% Procedure to parse and display a sentence.

try(String) :-  write(String), nl,
                Features = sem:in:[drs([r0,now],[])] ..
                           sem:time:in:[[r0,now|_]],
                phrase(discourse(Features),String),
                Features = sem:out:[SemOut|_],  /* extract what was built */
                Features = sem:time:out:Time,
                write('Reference times: '), write(Time), nl,
                display_drs(SemOut).
```

```
% Example sentences

test1  :- try([a,man,beat,a,donkey,'.']).
test2  :- try([a,man,beat,a,donkey,'.',he,owned,it,'.']).
test3  :- try([a,man,owned,a,donkey,'.',he,beat,it,'.']).
test4  :- try([a,man,had,beaten,a,donkey,'.',it,had,kicked,him,'.']).
test5  :- try([a,man,will,be,beating,a,donkey,'.']).
test6  :- try([a,man,will,beat,a,donkey,'.']).
test7  :- try([a,man,would,beat,a,donkey,'.']).
test8  :- try([a,man,was,beating,a,donkey,'.']).
test9  :- try([a,man,will,have,beaten,a,donkey,'.']).
test10 :- try([a,man,would,have,beaten,a,donkey,'.']).
test11 :- try([a,man,owned,a,donkey,'.',it,would,kick,him,'.',he,beat,it,'.']).
test12 :- try([a,man,beat,a,donkey,'.',he,would,own,it,'.',it,kicked,him,'.']).
test13 :- try([a,man,would,beat,a,donkey,'.',he,owned,it,'.']).
test14 :- try([a,man,owns,a,donkey,'.',he,beat,it,'.']).
test15 :- try([a,man,beat,a,donkey,'.',he,owns,it,'.']).
test16 :- try([a,man,owned,a,donkey,'.',he,had,beaten,it,'.',
               it,kicked,him,'.']).
```