C

Explorations of the Dynamic Environment

Explorations of the Dynamic Environment

Verkenningen van Dynamische Contexten
(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de Rector Magnificus, Prof. dr. J.A. van Ginkel ingevolge het besluit van het College van Decanen in het openbaar te verdedigen op donderdag 22 september 1994 des voormiddags te 10.30 uur

door

Cornelis Franciscus Martinus Vermeulen

geboren op 8 december 1966 te Zwolle

promotor: Prof. dr. D.J.N. van Eijck (Faculteit der Letteren)
co-promotor: Dr. A. Visser (Faculteit der Wijsbegeerte)

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Vermeulen, Cornelis Franciscus Martinus

Explorations of the dynamic environment/ Cornelis
Franciscus Martinus Vermeulen. - Utrecht: LEd; Utrecht:
Onderzoeksinstituut voor Taal en Spraak (OTS). - (OTS
dissertation series, ISSN 0929-0117)
Ook verschenen als handelseditie: Utrecht : LEd, 1994. -
Proefschrift Universiteit Utrecht. - Met lit. opg.
ISBN 90-5434-029-0
NUGI 941
Trefw.: semantiek.

Contents

Acknowledgements

In the Dutch university system the procedures automatically provide each 'promovendus' with a period of approximately three months between the completion of the manuscript of their thesis and the moment of its defense. To some this may look like a typical example of bureaucratic nonsense, but after four years of research in the Dutch logic-and-linguistics community I found that these three months hardly suffice to compile a more or less complete list of all the people whose support I wish to acknowledge. So I hope that all those whose name belongs in this list, but has been left out, will accept that this is certainly not because I have not valued their support: it is simply a result of this time limit which is inherent to 'the system'.

First of all I am most grateful to my promotor, Jan van Eijck, and my co-promotor, Albert Visser: they have put me on the right track in my explorations of the dynamic environment when I started in September 1990 and they have done their best to keep me from going astray ever since. Any AiO should already feel lucky to get the kind of support that I got from Jan and Albert, but in addition I was so lucky as to have Patrick Blackburn, Heleen Hoekstra, Tim Fernando, Marcus Kracht, Michael Moortgat and Henk Verkuyl around. Each of them have, at different times during the project, assisted me with useful advice, both concerning the subject matter of my research and about the wonderous ways of the academic world.

My PhD project was funded by the 'AiO Netwerk TLI', a national network for PhD students in Language, Logic and Information. I would like to thank Jacques van Leeuwen of the TLI-network for administrative support. I am very glad that the TLI-network have found the OTS (Research Institute for Language and Speech) of the Arts Department of Utrecht University prepared to give me a place to work and I would like to thank this institute—especially Sieb Nooteboom, Peter Coopmans and Martin Everaert—for providing excellent research facilities.

Together the two institutions have consituted a truely dynamic environment in which I was happy to discover many colleagues and friends. I would like to mention in particular (in order of appearance): Wilfried Meyer-Viol, Joost Zwarts, Dirk Heylen, Christ-Jan Doedens, Jules van Weerden, Herbert Ruessink, Renée Pohlmann, Joke Dorrepaal, Glyn Morrill, Maarten de Rijke, Willem Groeneveld, Erik Aarts, Teresa Solias, Marten Trautwein, Emiel Krahmer, Vera Stebletsova, Giovanna Cepparello, Claire Gardent, Koen Versmissen, Dirk Roorda, Petra de Wit, Anne-Marie Mineur, André Schenk, Martin van den Berg, Yde Venema, Kees Trautwein, Natasha Kurtonina, Jaap van der Does, Martijn Spaan, Willem-Olaf Huijsen, Herman Hendriks, Dorit Ben-Shalom and Annius Groenink.

Finally I want to thank Catholijn Jonker and Mieke Rats. Through our joint experiences at several ESSLLI summerschools we have become something of a mutual support group. Catholijn and Mieke have been most effective in getting me through some of the bad patches in the dynamic environment.

In the above list I have not included my family and personal friends: I hope that they will prove me right in thinking that their support during the past four years was not 'work-related' at all!

Matters of copyright

Chapter 1

First Steps in Dynamic Semantics

1.1 Dynamic semantics

Since the publication of Kamp (1981) and Heim (1983) we have seen the development of a new approach in formal semantics which we now call *dynamic semantics*. The contributions to this approach are large in number and diverse in character. This makes it hard to give a good demarcation of the area. Both Kamp (1981) and Heim (1983) were concerned with problems in the semantics of anaphora in texts. Therefore one view on dynamic semantics is that it is the area of formal semantics that is concerned with the interpretation of anaphora in texts along this line. This is not an unreasonable first attempt, but it seems that such a characterisation is too restrictive. For example, there is a whole body of work on presuppositions[1] that has obvious connections with dynamic semantics, but that is not captured by this characterisation.
There also is another way to look at dynamic semantics: all the developments in this field have in common that they use certain ideas and methods which are genuinely new to the area of natural language semantics. The introduction of these new ideas and techniques seems to be the more interesting contribution of dynamic semantics. Therefore it seems that these ideas about the semantics of natural language and the new

[1]Cf. Heim (1992), Van der Sandt (1989), Beaver (1994) and Van Eijck (1991b), for example.

techniques that these ideas have induced can serve as a more interesting criterion to determine what counts as dynamic semantics. In this thesis dynamic semantics will be approached from this angle. The ideas that are crucial to the development of dynamic semantics will be presented and the technical consequences of an approach to semantics along these lines will be illustrated in several situations. In particular we will see what a dynamic approach means for the semantics of variables (part I) and general principles will be proposed for the architecture of a semantics in a dynamic setting (chapter 4). Finally the ideas from dynamic semantics are applied to proof theory: this will lead to the idea of treating *proofs as texts* (chapter 5).

The decision to concentrate on the general ideas behind dynamic semantics on the one hand and the technical innovations that these ideas induce on the other hand, implies that the linguistic phenomena that have motivated dynamic semantics will not always be in the spotlight. Of course this does not mean that such applications are not the ultimate motivation for dynamic semantics, but it seems a good idea to abstract away from some of the linguistic details to try and find the best way of being dynamic in semantics.

In the remainder of this chapter we will first give a gentle introduction to dynamic semantics. For a more thorough introduction the reader is referred to the literature.[2] The dynamic interpretation of anaphora in texts will be sketched informally. After that a more precise account of the techniques used in *DRT* and *DPL* will be given. The relation between these forms of dynamic semantics will be discussed. At this point preliminary conclusions will be drawn about the nature of dynamic semantics: we will argue that a major goal of dynamic semantics is to provide a faithful picture of the step by step character of text interpretation. Finally some of the familiar results about *DRT* and *DPL* will be listed, both as a reminder for the reader and to fix some notations that will be used later on.

[2]Esp. Kamp (1981), Kamp and Reyle (1993), Heim (1983), Groenendijk and Stokhof (1991a).

1.2 A trip down memory lane

Although there are several authors who can claim to have anticipated developments in dynamic semantics — here I think of Seuren (1985), Karttunen (1976), but also Kaplan (1979), Stalnaker (1978) and even Montague (1970) — it is probably fair to say that the definitive start of dynamic semantics is made in Kamp (1981) and Heim (1983). It is their implementation of dynamic ideas about the semantics of discourse that gave rise to what we now call dynamic semantics. Two important characteristics of their approach are that:

- the semantic universe consists of *information structures*. These information structures do not only contain truth-conditional information, but also extra, *contextual* information;
- the interpretation of a text is given in terms of the *construction* of such an information structure.

Both Kamp and Heim[3] employ such a picture of discourse interpretation in a formalisation of the semantics of anaphora. Here I will first give an informal presentation of the interpretation of anaphora along such lines to get across the basic picture of dynamic interpretation. Later the formal details will be filled in in two different ways. The informal explanation will be based on the following two sample discourses:

- A dog howled. It was lonely.
- If a dog howls, then it is lonely.

The first discourse consists of two sentences, where an anaphoric expression in the second sentence, *it*, finds its antecedent, *a dog*, in the first sentence. The interpretation of this discourse leads to the construction of a *Discourse Representation Structure* (*DRS*) that represents the information conveyed by this example. Such a *DRS* consists of two components. The first component contains a set of *discourse referents*. Discourse referents represent the topics of the discourse. An expression such as *a dog*

[3]Kamp calls his information structures *Discourse Representation Structures*, Heim uses the metaphor of file cards that are updated with new information: we will consider these as different ways of presenting the same intuitive picture of the process of interpretation.

will typically be an indication of a new topic in the text. Therefore the interpretation of such an expression will lead to the introduction of a new discourse referent in the first component of the *DRS*.

In the second component truth-conditional information is stored about the discourse referents in the first component. For example, the *DRS* that we get by interpreting the first example will contain conditions on the discourse referent for *a dog* indicating that it is a *dog* and that it *howled*. Now, if we start with an empty *DRS*, $\langle\emptyset,\emptyset\rangle$, the interpretation of the first sentence will lead to the construction of the following *DRS*:

$$\langle\{r\},\ \{\textsc{dog}(r), \textsc{howl}(r)\}\rangle$$

This *DRS* is the starting point for the interpretation of the second sentence of the discourse. The second sentence contains the anaphor *it*. The interpretation of an anaphor consists of establishing a link with a discourse referent in the *DRS* under construction. In this case there is only one discourse referent available, namely r, so this will be the referent we link with. As a consequence of such a link the information that the second sentence gives will be interpreted as information about the discourse referent with which we have established a link. So the information in the second sentence will be interpreted as information about r. This gives the following *DRS*:

$$\langle\{r\},\ \{\textsc{dog}(r), \textsc{howl}(r), \textsc{lonely}(r)\}\rangle$$

Note that apart from adding truth-conditions to a *DRS* two kinds of operations occur in the interpretation of this example: first we sometimes have to add new discourse referents. This typically happens when we encounter indefinite descriptions such as *a dog*. Secondly we sometimes have to choose some discourse referent that already has been introduced in the *DRS* that we are constructing. This is what the interpretation of anaphors amounts to. The information that is given about the anaphor is then interpreted as information about the selected discourse referent, its *antecedent*. The information that is crucial for these links is stored in the first component of a *DRS*, whereas the second component just serves to collect the (truth) conditions on the referents from the first component.

In this set-up truth is introduced as a derived notion: it can be checked whether some discourse is true, by checking whether it is possible to match the discourse referents in the *DRS* with objects in the world such that all the conditions on the referents are satisfied by these objects. So this first discourse will be true precisely if we can find an object that is a dog, that barked and that was lonely. These are exactly the truth-conditions one wants for a discourse such as this.

In the interpretation of the second example we meet a new kind of operation on *DRS*s. This operation is triggered by the *if... then* construction in this sentence. *If... then* sentences are taken to express complex conditions on discourse referents. This complex condition is formed out of the *DRS*s that correspond to the *if*-part and the *then*-part of the sentence. The condition is represented as follows:

$$\langle\{r\},\ \{\textsc{dog}(r), \textsc{howl}(r)\}\rangle \Rightarrow \langle\emptyset,\ \{\textsc{lonely}(r)\}\rangle$$

Now in the interpretation of the second example, starting with the empty *DRS*, $\langle\emptyset,\emptyset\rangle$, we only have to add this condition. The result is:

$$\langle\emptyset,\ \{\ \langle\{r\},\ \{\textsc{dog}(r), \textsc{howl}(r)\}\rangle \Rightarrow \langle\emptyset,\ \{\textsc{lonely}(r)\}\rangle\ \}\ \rangle$$

So for the second discourse to be true, this complex condition has to be satisfied. This will be the case exactly if every way of matching the referents from the first *DRS* to satisfy the conditions of the first *DRS*, can be extended to a matching for the referents in the second *DRS* such that the conditions in the second *DRS* are also satisfied. Here this means that for any dog that howls, it must also be true that this dog is lonely. So the second example is true exactly if all howling dogs are lonely.

More details will be filled in in the next subsection, where two ways of making this informal account precise will be discussed. So far it should be clear that the two points that were made above about dynamic semantics indeed hold true. The explanation of the interpretation of texts that is given is in terms of actions on information structures. Both components of a *DRS* contain information that is essential for the interpretation of anaphors: the set of discourse referents gives the information which antecedents are available and the set of conditions is used to actually store information about the anaphor. Therefore it is essential for an explanation of anaphora along these lines that both kinds of information are represented in the semantics universe, for example in the guise of a *DRS*.

1.3 The formal residue

This informal picture of the workings of the dynamic interpretation of anaphora can be given a formal counterpart in different ways. This will be illustrated by the two interpretations of $\mathcal{L}$ that can be found below. $\mathcal{L}$ is the formal language that will be used as a representation language for texts here and throughout Part 1. Its relation to the usual representation languages of *DRT* and *DPL* is spelled out in section 1.5. The first interpretation of $\mathcal{L}$ that is discussed is due to Zeevat (1991a), who gives a very elegant formulation of *DRT*-semantics. Then an alternative formalisation along the lines of Groenendijk and Stokhof (1991a) will be presented. These two formulations nicely correspond to the two points that we made above about dynamic semantics: while in *DPL* the idea of *interpretation-as-construction* is modelled most elegantly, Zeevat-style *DRT* gives a perfect illustration of a semantic universe that consists of information structures with context information.
First we give the definition of $\mathcal{L}$.

Definition 1.3.1 ($\mathcal{L}$) *We assume that a fixed infinitely enumerable stock of variables VAR is given and that some set of n-ary predicate letters $\mathcal{P}^n$ is given for each $0 \leq n$. Then we define $\mathcal{L}$ as follows:*

1	$\bot \in \mathcal{L}$;	
2	$P(x_1, \ldots, x_n) \in \mathcal{L}$	*whenever* $x_1, \ldots, x_n \in VAR$ *and* $P \in \mathcal{P}^n$;
3	$\exists x \in \mathcal{L}$	*whenever* $x \in VAR$;
4	$\phi \cdot \psi \in \mathcal{L}$	*whenever* $\phi,\ \psi \in \mathcal{L}$;
5	$(\phi \rightarrow \psi) \in \mathcal{L}$	*whenever* $\phi,\ \psi \in \mathcal{L}$.

The language $\mathcal{L}$ is designed precisely to represent the actions that we encountered in our informal account above: $\exists x$ will be used to represent the act of introducing a discourse referent, the atomic formulas $\bot$ and $P(x_1, \ldots, x_n)$ are the atomic conditions and the implication sign stands for the construction of complex conditions as discussed above. So we now have the following formal analogues for our examples:

- $\exists x \cdot \text{DOG}(x) \cdot \text{HOWL}(x) \cdot \text{LONELY}(x)$
- $(\exists x \cdot \text{DOG}(x) \cdot \text{HOWL}(x) \rightarrow \text{LONELY}(x))$

Note that formulas in $\mathcal{L}$ are very similar to formulas of ordinary, static predicate logic. The atomic conditions are the same in both languages

and the connective $\cdot$ is analogous to the static connective $\wedge$. However, in our dynamic language $\exists x$ is a formula in its own right: it is simply the formula that tells us to introduce the referent x. Since in the definition of $\mathcal{L}$ $\exists x$ is treated as any other atomic formula, our definition has the shape of the definition of a propositional language, while at the same time the formulas look like the formulas of predicate logic. For now we simply give this as an observation, but we will get back to this point later.
In the Zeevat-style interpretation the formulas of $\mathcal{L}$ are interpreted in a universe of information structures, called *DRS*-interpretations.[4] The universe of *DRS*-interpretations is defined as follows.

Definition 1.3.2 *Let some fixed domain of interpretation DOM be given. Then* $\langle X, F\rangle$ *is a DRS-interpretation whenever* $X \subseteq VAR$ *is finite and* F *is a set of assignments VAR→DOM.*[5] *We consider the following two operations on DRS-interpretations:*

$$\begin{array}{lll}\langle X,F\rangle \oplus \langle X',F'\rangle & = & \langle X\cup X', F\cap F'\rangle\\ \langle X,F\rangle \Rightarrow \langle X',F'\rangle & = & \langle \emptyset, \{g\in ASS \mid \\ & & \forall f\in F\ g=_X f\ \exists h\in F' \textit{ such that } f=_{X'} h\}\rangle\end{array}$$

Here the notation $g =_X f$ indicates that f and g differ at most on the $x \in X$.
These *DRS*-interpretations are very well equipped to represent the information that is needed in the interpretation of anaphors along the lines sketched above: the first component of a *DRS*-interpretation will contain the discourse referents that have been introduced so far and the second component can be used to represent the conditions on these discourse referents.
Given an ordinary first order model $\mathcal{M} = \langle DOM, Int\rangle$ $\mathcal{L}$ can be interpreted in the universe of *DRS*-interpretations as follows:

Definition 1.3.3 (*DRT*-interpretation) *Let a model* $\mathcal{M} = \langle DOM, Int\rangle$ *be given. We define the DRT-interpretation of* $\mathcal{L}$ *as follows:*

[4]Here we give a formal definition of *DRS*-interpretations before defining *DRS*s formally. This omission will be made up for towards the end of this chapter.
[5]We will call the mappings *VAR→DOM ASS*, for assignments.

1 $[\![\bot]\!]_{drt} = \langle\emptyset,\emptyset\rangle$;
2 $[\![P(x_1,\ldots,x_n)]\!]_{drt} = \langle\emptyset,\{f|\ \langle f(x_1),\ldots,f(x_n)\rangle\ \in Int(P)\}\rangle$;
3 $[\![\exists x]\!]_{drt} = \langle\{x\},ASS\rangle$;
4 $[\![\phi\cdot\psi]\!]_{drt} = [\![\phi]\!]_{drt}\oplus[\![\psi]\!]_{drt}$;
5 $[\![(\phi\to\psi)]\!]_{drt} = [\![\phi]\!]_{drt}\Rightarrow[\![\psi]\!]_{drt}$

The atomic formulas $\bot$ and $P(x_1,\ldots,x_n)$ are interpreted as conditions: they only work on the second component of the *DRS*-interpretation, where the truth-conditional information is stored. The interpretations of these atomic formulas only allow those assignments that satisfy the conditions that the formulas express. Also $(\phi\to\psi)$ is interpreted as a condition: it follows from definition 1.3.2 that $(\phi\to\psi)$ only effects the second component of the *DRS*-interpretation. The effect of such a condition will be illustrated below, where we discuss the examples. $\exists x$ is the only atomic formula that effects the first component of the interpretation: we see that $\exists x$ introduces the discourse referent x in the first component. If we combine *DRS*-interpretations with $\oplus$, the discourse referents from the first components are collected and the truth-conditional information in the second components also is joined together: only those assignments remain that satisfy the conditions in both *DRS*-interpretations.
As was explained above, in the informal presentation, truth is a derived notion in dynamic semantics. We indicated how the truth of a text is checked by finding a *truthful* embedding, i.e. an embedding of the discourse referents that obeys all the truth-conditions on the referents. In the *DRS*-interpretations that we have defined above these truthful embeddings are stored in the second component. So a formula $\phi\in\mathcal{L}$ is true just in case the second component actually provides a truthful embedding for ϕ.

Definition 1.3.4 (*DRT*-truth) *Let $\phi\in\mathcal{L}$ be given and let $[\![\phi]\!]_{drt} = \langle X_\phi,F_\phi\rangle$. Then we call ϕ DRT-true iff $F_\phi\neq\emptyset$.*

In order to illustrate the working of this interpretation we will use the $\mathcal{L}$-representation of the examples given above.

- $\exists x\cdot\text{DOG}(x)\cdot\text{HOWL}(x)\cdot\text{LONELY}(x)$
- $(\exists x\cdot\text{DOG}(x)\cdot\text{HOWL}(x)\to\text{LONELY}(x))$

Now we can compute the *DRS*-interpretations of these examples. We will write $P^\star$ for $Int(P)$.

- $[\![\exists x \cdot \text{DOG}(x) \cdot \text{HOWL}(x) \cdot \text{LONELY}(x)]\!]_{drt} =$
  $\langle\{x\},\ \{f|\ f(x) \in \text{DOG}^\star \cap \text{HOWL}^\star \cap \text{LONELY}^\star\}\rangle$
- $[\![(\exists x \cdot \text{DOG}(x) \cdot \text{HOWL}(x) \rightarrow \text{LONELY}(x))]\!]_{drt} =$
  $\langle\emptyset,\ \{g|\ \forall f :\ g =_x f\ \&\ f(x) \in \text{DOG}^\star \cap \text{HOWL}^\star\ \rightarrow\ f(x) \in \text{LONELY}^\star\}\rangle$

For the two formulas to be *DRT*-true, the second component of their interpretations should be non-empty. We see that for the first example this means that $\text{DOG}^\star \cap \text{HOWL}^\star \cap \text{LONELY}^\star$ should be non-empty. So the first formula is *DRT*-true precisely if there is some dog that howled and was lonely. The second example will be *DRT*-true precisely if

$$\{g|\ \forall f :\ g =_x f\ \&\ f(x) \in \text{DOG}^\star{\cap}\text{HOWL}^\star\ \rightarrow f(x) \in \text{LONELY}^\star\} \neq \emptyset$$

If we look at this set more carefully we see that there really only are two options: either $\text{DOG}^\star \cap \text{HOWL}^\star \subseteq \text{LONELY}^\star$, and in that case

$$\{g|\forall f : g =_x f\ \&\ f(x) \in \text{DOG}^\star \cap \text{HOWL}^\star \rightarrow\ f(x) \in \text{LONELY}^\star\} = ASS.$$

Else $\text{DOG}^\star \cap \text{HOWL}^\star \not\subseteq \text{LONELY}^\star$. Then

$$\{g|\ \forall f :\ g =_x f\ \&\ f(x) \in \text{DOG}^\star \cap \text{HOWL}^\star\ \rightarrow\ f(x) \in \text{LONELY}^\star\} = \emptyset$$

So we see that the second formula is true precisely if $\text{DOG}^\star \cap \text{HOWL}^\star \subseteq \text{LONELY}^\star$, i.e. precisely if every howling dog is lonely. So we get the required truth-conditions for both examples.
It may seem that in this definition only one of the two main characteristics of the dynamic set-up is represented: the semantics universe consists of information structures, as promised, but it seems that the idea of interpretation as a construction on these objects is not available here. It is indeed true that the action metaphor is not very prominent in this formulation, but it still is available implicitly in the following way. With any *DRS*-interpretation $\langle X, F\rangle$ we can associate a canonical operation on *DRS*-interpretations, $f_{\langle X,F\rangle}$, that works as follows:

$$f_{\langle X,F\rangle}(\langle Y, G\rangle)\ =\ \langle Y, G\rangle \oplus \langle X, F\rangle$$

Intuitively the operation $f_{\langle X,F\rangle}$ represents the act of adding the information $\langle X,F\rangle$ to the information $\langle Y,G\rangle$, that we had already. In case $\langle X,F\rangle$ is the interpretation of some formula $\phi \in \mathcal{L}$, this function precisely computes the update with the information contained in ϕ. So for each $\phi \in \mathcal{L}$, $f_{[\![\phi]\!]_{drt}}$ provides an interpretation of ϕ as an action on *DRS*-interpretations. These two ways of looking at the interpretation of ϕ really amount to the same thing, since the interpretation of ϕ, $[\![\phi]\!]_{drt}$ can be obtained by updating the empty *DRS*, $\langle\emptyset,\emptyset\rangle$, with the information contained in ϕ:

$$f_{[\![\phi]\!]_{drt}}(\langle\emptyset,\emptyset\rangle) \;=\; [\![\phi]\!]_{drt}$$

As $\oplus$ is an associative operation (see definition 1.3.2), it follows that switching from $[\![\phi]\!]_{drt}$ to $f_{[\![\phi]\!]_{drt}}$ commutes with the $\oplus$ operation on *DRS*-interpretations, i.e. $f_{\langle X,F\rangle\oplus\langle Y,G\rangle} = f_{\langle X,F\rangle} \circ f_{\langle Y,G\rangle}$. Here $\circ$ stands for function composition. Since it has become custom to use postfix notation for function application in the literature on dynamic semantics, I will try to follow that custom from now on. This means that the notation for function composition has been adapted accordingly. So $f \circ g$ is computed by first applying f and then applying g. As a result it is no longer necessary to have different notations for function composition and relation composition.
Note that this way of making actions out of denotations can be applied in a lot of situations. This observation was also made by Van Benthem (1991), and is elaborated upon in Visser (1992a). Also see chapter 4.

There also is another natural and elegant way of interpreting $\mathcal{L}$, which is due to Groenendijk and Stokhof (1991a). In this second formulation each formula of $\mathcal{L}$ is interpreted as a relation on assignments. The idea is that each assignment tells us which values the variables have. An information state in general will consist of several assignments, each of which gives one possibility for the values of the variables. The interpretation of a formula will be an operation on such an information state. This operation can be computed *via* the atomic information states: starting in some assignment the interpretation of a formula can lead to several other assignments. The result of the interpretation of a formula on a set of assignments can then be obtained, simply by taking the union of all the possible outputs of the elements of the set. Therefore it suffices to specify the interpretation of a formula as a relation on atomic information states, i.e. assignments.
The definition works as follows:

Definition 1.3.5 (*DPL*-**interpretation**) *Let some model* $\mathcal{M} = \langle DOM, Int \rangle$ *be given. We define the DPL-interpretation of* $\mathcal{L}$ *as follows:*

1	$[\![\bot]\!]_{dpl}$	$= \emptyset;$
2	$[\![P(x_1, \ldots, x_n)]\!]_{dpl}$	$= \{\langle f, f \rangle \mid \langle (x_1)f, \ldots, (x_n)f \rangle \in Int(P)\};$
3	$[\![\exists x]\!]_{dpl}$	$= \{\langle f, g \rangle \mid f =_x g\};$
4	$[\![\phi \cdot \psi]\!]_{dpl}$	$= [\![\phi]\!]_{dpl} \circ [\![\psi]\!]_{dpl};$
5	$[\![(\phi \rightarrow \psi)]\!]_{dpl}$	$= \{\langle f, f \rangle \mid \forall g\ \langle f, g \rangle \in [\![\phi]\!]_{dpl} : \exists h\ \langle g, h \rangle \in [\![\psi]\!]_{dpl}\}.$

We see that the atomic formulas $\bot$ and $P(x_1, \ldots, x_n)$ work as *tests* or *conditions* on the atomic information states: in the interpretation of such a test it is checked whether the current assignment satisfies the condition. If this is so, then we can simply continue without altering anything. Assignments that do not satisfy the conditions do not survive the test. Also $(\phi \rightarrow \psi)$ has such a test-like behaviour. We will see how this works when we discuss the examples below. $\exists x$ is the only atomic formula that actually affects the assignment on which it operates: whenever we interpret $\exists x$ in some assignment f, we are allowed to choose a new value for the variable x. This new value is the value that will be tested by the conditions that follow. The interpretation of $\phi \cdot \psi$ is obtained by relation composition. So if we want to interpret $\phi \cdot \psi$, we simply interpret ϕ first and then ψ.
Again we introduce truth as a derived notion. We will say that some formula $\phi \in \mathcal{L}$ is *DPL*-true if its interpretation succeeds in all assignments.

Definition 1.3.6 (*DPL*-**truth**) *Let* $\phi \in \mathcal{L}$ *be given. Then we say that* ϕ *is DPL-true iff*

$$\forall f \in ASS\ \exists g \in ASS : \langle f, g \rangle \in [\![\phi]\!]_{dpl}$$

or, in other words,

$$dom([\![\phi]\!]_{dpl}) = ASS$$

So a formula $\phi \in \mathcal{L}$ is *DPL*-true if every input assignment $f \in ASS$ survives the interpretation of ϕ. Recall that in the *DRT* setting truth was defined in terms of truthful embeddings. We can compare this with *DPL*-truth if we consider the *DPL*-interpretation as the *construction* of such a truthful embedding: during the *DPL*-interpretation we will be allowed to choose new values each time we meet $\exists x$. This way we check whether it is possible to find values such that all the conditions in the formula are satisfied. This gives us an alternative outlook on *DPL*-truth: a formula is *DPL*-true if we can construct a truthful embedding g for the formula regardless of the input state f.[6] Let's see how this works out for the examples that we have seen above:

- $[\![\exists x \cdot \text{DOG}(x) \cdot \text{HOWL}(x) \cdot \text{LONELY}(x)]\!]_{dpl} =$
  $\{\langle f, g\rangle \mid f =_x g \ \&\ g(x) \in \text{DOG}^\star \cap \text{HOWL}^\star \cap \text{LONELY}^\star\}$
- $[\![(\exists x \cdot \text{DOG}(x) \cdot \text{HOWL}(x) \to \text{LONELY}(x))]\!]_{dpl} =$
  $\{\langle f, f\rangle \mid \forall h :\ f =_x h \ \&\ h(x) \in \text{DOG}^\star \cap \text{HOWL}^\star \ \to\ h(x) \in \text{LONELY}^\star\}$

We see that in the first example $\exists x \cdot \text{DOG}(x) \cdot \text{HOWL}(x) \cdot \text{LONELY}(x)$ is *DPL*-true if we can always choose a value for x in $\text{DOG}^\star \cap \text{HOWL}^\star \cap \text{LONELY}^\star$. This is the case precisely if there is a dog that howled and was lonely, as required. For the second example to be true we need that for any choice of $h(x) \in \text{DOG}^\star \cap \text{HOWL}^\star$, $h(x) \in \text{LONELY}^\star$ holds automatically. This is the case precisely if $\text{DOG}^\star \cap \text{HOWL}^\star \subseteq \text{LONELY}^\star$. So also the second example gets the required truth-conditions with the *DPL* notion of truth.
In this *DPL*-style interpretation of $\mathcal{L}$ the second main point of dynamic semantics is dominant: the formulas of $\mathcal{L}$ are interpreted as relations on basic information units, i.e. assignments, and this clearly gives a procedural flavour to the semantics. And it also seems possible to comply with the first characteristic: it is tempting to try and use *sets* of assignments as information structures. But we will see in chapter 2 that this does not work. Although sets of assignments are reasonable candidates for a sensible notion of information state, it can be shown that they do not combine very well with the transitions that the *DPL*-interpretation generates: we run into the so-called non-eliminativity problem. In this case the first characteristic of a dynamic semantics is somewhat lost: the *DPL*-interpretations do not lead to a suitable notion of information structure. In order to obtain a satisfactory notion of information structure from the *DPL*-interpretations we have to add information to the *DPL*-relations. In chapter 2 this point will be discussed in detail.

[6] We have to be careful with this analogy: crucial differences between the *DRT* and the *DPL* notion of truth arise if we interpret formulas with more than one occurrence of $\exists x$. See section 1.5 for more details.

Dynamic entailment

Above we have discussed two dynamic notions of interpretation and the two corresponding notions of truth. Here we complete the discussion be presenting the corresponding notions of entailment.

Definition 1.3.7 *Let $\phi_1, \ldots, \phi_n, \psi \in \mathcal{L}$ be given. Call $[\![\phi_i]\!]_{drt} = \langle X_i, F_i \rangle$ and $[\![\psi]\!]_{drt} = \langle Y, G \rangle$. We define DRT-entailment, $\models_{DRT}$, and DPL-entailment, $\models_{DPL}$ as follows:*

$\phi_1, \ldots, \phi_n \models_{DRT} \psi$ *iff* $\forall f \in \bigcap\{F_i : 1 \leq i \leq n\}\ \exists g \in G : f =_Y g$

$\phi_1, \ldots, \phi_n \models_{DPL} \psi$ *iff* $range([\![\phi_1]\!]_{dpl} \circ \ldots \circ [\![\phi_n]\!]_{dpl}) \subseteq dom([\![\psi]\!]_{dpl})$

The dynamic notions of entailment simply follow the corresponding dynamic notions of implication that we have seen above. This means that 'anaphoric links' between the assumptions ϕ_i and the conclusion ψ will be effective. Therefore it is important to regard $\phi_1, \ldots, \phi_n$ in the definition of $\models_{DPL}$ as a *sequence* of assumptions, not simply as a *set*. This makes the *DPL*-notion of entailment sensitive to the so called structural rules such as permutation, monotonicity etc. For example, it is in general not harmless to permute two assumptions in the sequence $\phi_1, \ldots, \phi_n$: it can occur that $\phi_1, \phi_2 \models_{DPL} \psi$ while not $\phi_2, \phi_1 \models_{DPL} \psi$. Here $\phi_1 = \exists x$, $\phi_2 = \psi = P(x)$ can serve as an example.
The *DRT*-notion of entailment is not sensitive to the structure of the assumptions. This could lead one to assume that the *DRT*-notion will be easier to axiomatise. But this does not seem to be the case: for the *DRT*-notion of entailment the situation gets complicated by other facts. As an example we mention that *DRT*-entailment does not have the (strong) deduction theorem: we do not have

$$\phi_1, \ldots, \phi_n \models_{DRT} \psi \text{ iff } \phi_1, \ldots, \phi_{n-1} \models_{DRT} (\phi_n \rightarrow \psi)$$

which does hold for *DPL*-entailment. As a counter example for *DRT*-entailment consider $\phi_1 = \psi = P(x)$ and $\phi_2 = \exists x$. Then we find:

$\phi_1, \phi_2 \models_{DRT} \psi$

but not

$\phi_1 \models_{DRT} (\phi_2 \rightarrow \psi)$

1.4 The small unit principle

In the previous section we saw two ways of interpreting $\mathcal{L}$ and we checked that both styles of interpretation enable us to handle the anaphoric links in the examples. Crucial for both solutions is that they encode information about the available antecedents into the semantics. This makes it possible to establish a link with a suitable antecedent whenever we have to interpret an anaphoric expression. In the *DRT*-style semantics the information about the antecedents is made explicit in the first component of the *DRS*-interpretations. The first component precisely contains those discourse referents which are available as antecedent. In the *DPL*-style semantics the required information is available implicitly: the *DPL*-relations do not make explicit which variables are available as antecedent, but here relation composition provides a uniform way of establishing the required links implicitly. In this way we do not only pass on information about the truth or falsehood of the formula, but also the values that are assigned to the variables in the interpretation of the formula are passed on to the next formula automatically. These are two methods of incorporating the necessary non-truth-conditional information into the semantics.
Now we take a look again at the two features that we presented as general characteristics of dynamic semantics:

- dynamic semantics uses information structures to encode not only truth-conditional but also contextual information;
- in dynamic semantics interpretation is modeled as the process in which information structures are constructed.

In the previous section we certainly have seen both characteristics at work. The *DRS*-interpretations make the first feature more explicit and the use of relations in *DPL* underlines the second point. But we have already explained that implicitly also in *DPL* non-truth-conditional information is stored and we saw that in *DRT* functions can be defined to restore the construction metaphor of interpretation.
If we want to understand why these features should pop up in the semantics of anaphora, it is useful to think about the general picture of text interpretation. Here an important observation is that texts can be rather big. Of course there also are rather big sentences, but the length of sentences has never played a role in the set-up of sentence semantics.

Perhaps this can be understood from the fact that long sentences seem to be the exception rather than the rule. But the length of the expression is also simply a fact that has been conveniently suppressed in the static picture of sentence interpretation. Sentences are simply thought of as nice, complete syntactic units that are easy to work with.

The attention for discourse phenomena has brutally disturbed the idea that semantics works with such manageable units. Instead the attention for discourse phenomena has made us aware of the fact that the expressions we have to interpret are potentially big and usually unfinished pieces of natural language text. This may seem quite an innocent observation at first, but in fact it immediately gives rise to several constraints on models of text interpretation. In particular, because texts are big, it is less attractive to work with models of interpretation that work on the text as a whole. Instead the preferred picture of the interpretation of texts models interpretation as a *process*, in which we gradually work through the text, always working on one small part of the text at a time.

The demand that the model should provide such a picture of *text interpretation as a process*, automatically introduces a new kind of concern into formal semantics: we are not satisfied anymore with (compositionally)[7] producing the right truth-conditions for natural language expressions — as in sentence semantics — but now we also want to respect at least some of the features of the *process of interpretation* itself. This is one important aspect of dynamic semantics: the interest in the semantics of (anaphora in) texts has made us aware of the fact that a good semantics should not only worry about the *result* of the interpretation process, but should also take into account the relevant features of the interpretation process *qua* process. Apart from the fact that such a procedural flavour can help in giving a more satisfactory account of the phenomena — as we have seen in the case of anaphora — the idea of taking the interpretation process itself more seriously is also simply one of the challenges that is inherent in the scale enlargement that comes with the shift from sentence level to text level.

As was already said above, one aspect of the interpretation process that we consider to be particularly crucial, is the step by step character that the interpretation process has: it is not a good idea to give a model of the

[7]See also chapter 4 for discussion on the role of the compositionality principle in semantics.

interpretation of a large text that works on the text as a whole. Instead we should give a picture of text interpretation that works only on small parts of the text at a time. We call this the *small unit principle.*

The result of this requirement is that we will have to be careful when we try to model interactions between the meanings of different parts of a text. Because we only work in small units, we will in general not be able to work on different parts of a text at the same time. This means that if we want to model interactions between different parts of a text it does not help to simply make a syntactic rule that includes these different parts: the small unit principle tells us that this global construct will have to be analysed as the result of the interaction of small units anyhow. Therefore also in the semantics we will have to make the information about the interactions available locally in the text interpretations. This is the reason why non-truth-conditional information plays such a crucial role in dynamic semantics. It is precisely by adding non-truth-conditional information that we are able to represent the interaction between different parts of a texts without violating the small unit principle. We have seen this in the dynamic semantics of anaphora, where information about antecedents had to be represented in order to enable the interpretation of anaphors. We will see other examples of the incorporation of contextual information in the semantics in chapter 4.

But we can already see now that the consequence of the small unit principle will be that the distinction between the level of syntax and the level of semantics gets blurred. It is well known, and indeed one of the main observations in Montague's work (cf. Thomason (1974)), that syntactic information has a crucial role to play in formal models of the interpretation of natural language. The syntactic rules steer the compositional interpretation process. But in general syntactic constructions can be 'large' and we are only allowed to work in small units. Because of the small unit principle it will not always be possible to work on some syntactic construction as a whole: we will usually have to break it up into little pieces. But the syntactic information can be crucial for the interpretation. Therefore we will have to code some of it into the semantic representations. So it is inevitable that some of the structure that we like to think of as syntactic information pops up in the semantic representations, because of the small unit principle. Once it has popped up in the semantics, we will no longer have to represent it in the syntax. So we see a kind of exchange mechanism: we can make structure invisible in the syntax, breaking everything up into small units, but as a result we find the structure back in the semantics.[8] For example, we will be able to ignore the bracketing structure of expressions and simply parse them incrementally, *as long as* we are prepared to create some form of (bracketing) structure in the semantics to compensate for this. This point may seem highly abstract now, but will become clearer later, when it will be illustrated with a worked out example in chapter 4.

There are several other slogans besides the *small unit principle* that capture the aspects of text interpretation that we want to represent in dynamic semantics. Several of these will be discussed in chapter 4 in detail, but there is one slogan that we want to mention here already. This is the idea that *dynamic semantics takes the perspective of the hearer.* This suggests first of all that we try to model how we interpret a text *as we hear it.* This is in contrast with the situation of someone who *reads* a text. Whereas a reader can afford to leaf back and forth through the text, the position of a hearer nicely captures the on-the-spot nature of interpretation that we also try to capture with the small unit principle.
In addition the hearer perspective can also be contrasted with the perspective of the speaker. This contrast is less important for our purposes, but also here there are several interesting differences with the position of the hearer. For example, if we think of the use of indefinite expressions such as *a dog*, it seems natural in the hearer perspective to model this as the introduction of a new referent. But for the speaker the situation may be different: the speaker may use the indefinite expression to express his own lack of definite knowledge about the subject (which dog is it?), but it can also be the case that he really knows which dog he is talking about. In that case he could still use the indefinite, to indicate that this dog is a new topic from the perspective of his intended audience. By taking the perspective of the hearer instead of the speaker we avoid this kind of complication.

So we see that general considerations about the interpretation of texts can help us understand why the two features occur in dynamic semantics. We saw that the fact that dynamic semantics is concerned with texts rather

[8]Visser (1992c) calls this phenomenon the *flatness principle*, since the result of this move is that we see less structure in the syntax.

than sentences naturally leads to an interest in modelling interpretation *qua* process, which explains why procedural formulations are so popular in dynamic semantics. We have also argued that a model of interpretation as a process will automatically lead to the incorporation of non-truth-conditional information in the semantics. In what follows we will mainly be concerned with the role that this non-truth-conditional information plays. We will investigate the way in which it allows us to model the interactions between different parts of a text. In other words: we concentrate on the first characteristic of dynamic semantics. This is first of all a matter of choice: to us the small unit principle and its consequences for models of these interactions pose a more direct and a more interesting challenge than the general idea of using procedures in semantics. But there also is some less subjective motivation for the choice. First, we have seen that a procedural formulation can usually be recovered from an information structure based approach. Visser (1992a) discusses the precise conditions under which such a reformulation is available.[9] So in a sense we need not worry too much about the procedural metaphor as long as we conform to these conditions.
Secondly, it seems that the metaphor of *interpretation as construction* by itself is too vague to produce sufficiently precise questions that are of immediate interest for the semantics of natural language. At present there is in the literature an abundance of so called *dynamic logics*, in which general properties of procedures, actions and transitions can be studied.[10] The development of this area is certainly fascinating and it poses interesting challenges. And there is no doubt that some of these theories of actions can be of use in dynamic semantics. But at present it seems that the study of actions and procedures *in abstracto* does not lead to a better understanding of the procedures that actually are used in dynamic semantics and how we should use them. Therefore, before the results from these theories of actions can be made productive in dynamic semantics, we should make up our minds as to what kind of actions and procedures will occur in dynamic semantics. As was said above, it seems that the metaphor of interpretation as construction by itself does not do

[9]Also see chapter 4.

[10]Here we think of Van Benthem (1993), Van Benthem (1994), Venema (1993), De Rijke (1992), Blackburn and Venema (1993), Van Eijck and Visser (1994) and also—with another motivation—Andréka *et al.* (1993), but also of the tradition of process algebra (cf. Baeten and Weijland (1990) and references therein).

this for us. But conditions such as the *small unit principle* are precisely what can help us make up our mind here.

Warning:

In this section we have presented an informal argument for a certain way of doing semantics. Such informal arguments are quite useful: we have seen that thinking in these terms helps us understand the characteristics of discourse semantics and we will see that it will also help us to formulate other sensible constraints on the semantics of texts. In this way we can develop some kind of methodology and indicate boundaries as to what is and what is not allowed in dynamic semantics. In this thesis we use this kind of argument frequently. But, of course, the informal arguments are not decisive: the final test for our work will be in their application to hard linguistic data.

1.5 Facts and figures

So far we have seen two versions of the semantics of $\mathcal{L}$ that we have called *DRT*-style and *DPL*-style. Probably this suggestive terminology did not come as a surprise to the well informed reader, but here we will actually make the link of these styles of interpretation with *DRT* and *DPL* precise. Here we will give the usual presentations of *DRT* and *DPL* and make the relation with our presentation of the semantics of $\mathcal{L}$ precise. Then we will discuss the relation of $\mathcal{L}$ with predicate logic in its usual, static guise. We will see that for both notions of truth that we have discussed we can give a translation form $\mathcal{L}$ into static predicate logic that preserves truth. Finally we will discuss the relation between the *DRT* and *DPL* style semantics for $\mathcal{L}$ by constructing a relational presentation of the *DRT* semantics. This way we can see where the differences between *DPL* and *DRT* lie.

None of the results that we present here are really new: they can all be traced back to Groenendijk and Stokhof (1991a), Visser (1989), Zeevat (1991a) or Kamp (1981). A useful overview is Krahmer (1993).

DPL

The relation between *DPL*[11] and the *DPL*-style interpretation of $\mathcal{L}$ is extremely clear. The only difference is in the treatment of $\exists x$: in $\mathcal{L}$ we treat $\exists x$ as an atomic formula, in the standard presentation of *DPL* $\exists x$ is treated as a connective.
Let's make this precise by considering the standard presentation of *DPL* in detail. The syntax of *DPL* simply is the language of first order logic.

Definition 1.5.1 *We define the first order language over the alphabet given above as follows:*

$$\begin{array}{ll}
\bot \in \mathcal{L}_{pl}; & \\
P(x_1,\ldots,x_n) \in \mathcal{L}_{pl} & \textit{whenever } x_1,\ldots,x_n \in \mathit{VAR} \textit{ and } P \in \mathcal{P}^n; \\
\exists x\ \phi \in \mathcal{L}_{pl} & \textit{whenever } \phi \in \mathcal{L}_{dpl} \textit{ and } x \in \mathit{VAR}; \\
(\phi \wedge \psi) \in \mathcal{L}_{pl} & \textit{whenever } \phi,\ \psi \in \mathcal{L}_{dpl}; \\
(\phi \rightarrow \psi) \in \mathcal{L}_{pl} & \textit{whenever } \phi,\ \psi \in \mathcal{L}_{dpl}.
\end{array}$$

So in the syntax *DPL* just follows standard, static logic. The dynamics of *DPL* is located in the semantics.

Definition 1.5.2 (*DPL*) *We define the dynamic interpretation of* $\mathcal{L}_{pl}$ *as follows:*

$$\begin{array}{ll}
[\![\bot]\!]_{gs} & = \emptyset; \\
[\![P(x_1,\ldots,x_n)]\!]_{gs} & = \{\langle f,f\rangle : \langle f(x_1),\ldots,f(x_n)\rangle \in P^*\}; \\
[\![\exists x\phi]\!]_{gs} & = \{\langle f,g\rangle : \exists h\ f =_x h\ \&\ \langle h,g\rangle \in [\![\phi]\!]_{gs}\}; \\
[\![\phi \wedge \psi]\!]_{gs} & = [\![\phi]\!]_{gs} \circ [\![\psi]\!]_{gs}; \\
[\![(\phi \rightarrow \psi)]\!]_{gs} & = \{\langle f,f\rangle : \forall g\ \langle f,g\rangle \in [\![\phi]\!]_{gs}\ \exists h\ \langle g,h\rangle \in [\![\psi]\!]_{gs}\}.
\end{array}$$

As one can see the only difference with the *DPL*-style interpretation of $\mathcal{L}$ is in the clause for $\exists x\phi$. But we can see immediately that:

$$[\![\exists x\phi]\!]_{gs} = [\![\exists x \cdot \phi]\!]_{dpl}$$

Therefore it is clear that the following translations are meaning preserving.

Definition 1.5.3 ($\mathcal{L}$ **to** $\mathcal{L}_{pl}$) *We define a translation function* $(-)^\circ : \mathcal{L} \rightarrow \mathcal{L}_{pl}$ *as follows:*

[11] Here the appropriate reference is to Groenendijk and Stokhof (1991a). Also Barwise (1987) already contains some of the crucial ideas.

$$\begin{array}{ll}
(\bot)^\circ & = \bot; \\
(P(x_1,\ldots,x_n))^\circ & = P(x_1,\ldots,x_n); \\
(\exists x)^\circ & = \exists x\ \top; \\
(\phi \cdot \psi)^\circ & = (\phi)^\circ \wedge (\psi)^\circ; \\
((\phi \to \psi))^\circ & = ((\phi)^\circ \to (\psi)^\circ).
\end{array}$$

Here $\top$ *is an abbreviation for* $(\bot \to \bot)$.

The translation in the opposite direction is even easier to find. It is important to be aware of the fact that, although there are translations in both directions, these translations are not each other's inverse. This means that we use the notion of translation without implying that in general $((\phi)^\circ)_\circ = \phi$ or $((\phi)_\circ)^\circ = \phi$ will hold. Here this will be spoiled by the occurrences of $\exists x$.

Definition 1.5.4 ($\mathcal{L}_{pl}$ to $\mathcal{L}$) *The translation* $(-)_\circ : \mathcal{L}_{pl} \to \mathcal{L}$ *is defined as follows:*

$$\begin{array}{ll}
(\bot)_\circ & = \bot; \\
(P(x_1,\ldots,x_n))_\circ & = P(x_1,\ldots,x_n); \\
(\exists x\ \phi)_\circ & = \exists x \cdot (\phi)_\circ; \\
(\phi \wedge \psi)_\circ & = (\phi)_\circ \cdot (\psi)_\circ; \\
((\phi \to \psi))_\circ & = ((\phi)_\circ \to (\psi)_\circ).
\end{array}$$

Now it is easy to check that:

$$[\![\phi^\circ]\!]_{gs} = [\![\phi]\!]_{dpl} \text{ and } [\![\phi]\!]_{gs} = [\![\phi_\circ]\!]_{dpl}$$

DRT

The comparison of our *DRT*-style interpretation of $\mathcal{L}$ and traditional presentations of *DRT* will be made via a formulation of *DRT* that is based on Zeevat (1991a).[12] In this formulation of *DRT* the *Discourse Representation Structures*, *DRS*s for short, are constructed from basic *DRS*s with two operations, called *merger* and *sub*. *DRS*s are pairs of which the first component is a set of discourse referent and the second component a set of conditions on these discourse referents. Formally this works as follows:

[12]It seems fair to simply call this *DRT*, although in some of the details Zeevat (1991a) deviates slightly from both Kamp (1981) and Kamp and Reyle (1993).

Definition 1.5.5 (*DRT-syntax*) *We define $\mathcal{L}_{drt}$, the DRT-language, over the given alphabet as follows:*

$\langle\emptyset, \{\bot\}\rangle \in \mathcal{L}_{drt}$;	
$\langle\emptyset, \{P(x_1, \ldots, x_n)\}\rangle \in \mathcal{L}_{drt}$	*for any* $x_1, \ldots, x_n \in VAR$ *and* $P \in \mathcal{P}^n$;
$\langle\{x\}, \emptyset\rangle \in \mathcal{L}_{drt}$	*for any* $x \in VAR$;
$\phi \bullet \psi \in \mathcal{L}_{drt}$	*for any* $\phi, \psi \in \mathcal{L}_{drt}$;
$(\phi \Rightarrow \psi) \in \mathcal{L}_{drt}$	*for any* $\phi, \psi \in \mathcal{L}_{drt}$.

Here $\bullet$ and $\Rightarrow$ are defined as follows:

$$\langle X, C\rangle \bullet \langle X', C'\rangle = \langle X \cup X', C \cup C'\rangle;$$
$$\langle X, C\rangle \Rightarrow \langle X', C'\rangle = \langle\emptyset, (\langle X, C\rangle \Rightarrow \langle X', C'\rangle)\rangle$$

and called **merger** *and* **sub** *respectively. The formulas of $\mathcal{L}_{drt}$ are usually called discourse representation structures (DRSs).*

It can be checked that this definition is equivalent to the standard definition, which uses a simultaneous induction to define both the set of *DRS*s and *DRC*, the set of *DRS*-conditions.[13] In other words, we get the following proposition:

Proposition 1.5.6 (*DRT-syntax*) *We can obtain $\mathcal{L}_{drt}$ by the following simultaneous induction which defines both $\mathcal{L}_{drt}$ and the set of DRS-conditions, DRC:*

$\bot \in DRC$;	
$P(x_1, \ldots, x_n) \in DRC$	*for any* $x_1, \ldots, x_n \in VAR$ *and* $P \in \mathcal{P}^n$;
$(\phi \Rightarrow \psi) \in DRC$	*whenever* $\phi, \psi \in \mathcal{L}_{drt}$;
$\langle X, C\rangle \in \mathcal{L}_{drt}$	*whenever* $C \subseteq_{fin} DRC$ *and* $X \subseteq_{fin} VAR$.[14]

□

We see that these *DRS*s look just like the *DRS*-interpretations that we have used to interpret $\mathcal{L}$. The only difference is that here the second component consists of the (syntactic) conditions that the referents have to satisfy, whereas in the *DRS*-interpretations the second component contains the truthful embeddings. We can assign a *DRS*-interpretation to each *DRS* if we replace the syntactic conditions in a *DRS* with the embeddings that satisfy these conditions. To be precise:

[13] Also see Kamp (1981) for the traditional definition of a *DRS*.

[14] Here $\subseteq_{fin}$ stands for *is a finite subset of*.

Definition 1.5.7 *For each $\phi \in \mathcal{L}_{drt}$ we define its interpretation $[\![\phi]\!]_z$ as follows:*

$$\begin{array}{ll}
[\![\langle\emptyset, \{\bot\}\rangle]\!]_z & = \langle\emptyset, \emptyset\rangle; \\
[\![\langle\emptyset, \{P(x_1, \ldots, x_n)\}\rangle]\!]_z & = \langle\emptyset, \{f \in ASS : \langle f(x_1), \ldots, f(x_n)\rangle \in P^\star\}\rangle; \\
[\![\langle\{x\}, \emptyset\rangle]\!]_z & = \langle\{x\}, ASS\rangle; \\
[\![\phi \bullet \psi]\!]_z & = [\![\phi]\!]_z \oplus [\![\psi]\!]_z; \\
[\![(\phi \Rightarrow \psi)]\!]_z & = ([\![\phi]\!]_z \Rightarrow [\![\psi]\!]_z).
\end{array}$$

Here $\oplus$ and $\Rightarrow$ on the right hand side of the equations are the operations on *DRS*-interpretations that we defined in definition 1.3.2.
The *DRS*s in $\mathcal{L}_{drt}$ are of a different format then the formulas of $\mathcal{L}$: they are pairs of sets whereas the formulas of $\mathcal{L}$ are formulas in the usual sense. Still there is an obvious correspondence between the basic *DRS*s and the basic formulas of $\mathcal{L}$: $\langle\emptyset, \{\bot\}\rangle$ corresponds to $\bot$, $\langle\emptyset, \{P(x_1, \ldots, x_n)\}\rangle$ corresponds to $P(x_1, \ldots, x_n)$ and $\langle\{x\}, \emptyset\rangle$ corresponds to $\exists x$. This correspondence can be stretched to complex *DRS*s if we compare $\bullet$ to $\cdot$ and $\Rightarrow$ to $\rightarrow$. This way we can obtain meaning preserving translations from $\mathcal{L}$ to $\mathcal{L}_{drt}$ and vice versa. To be precise:

Definition 1.5.8 ($\mathcal{L}$ to $\mathcal{L}_{drt}$) *We define a translation function $(-)^\bullet : \mathcal{L} \rightarrow \mathcal{L}_{drt}$ as follows:*

$$\begin{array}{ll}
(\bot)^\bullet & = \langle\emptyset, \{\bot\}\rangle; \\
(P(x_1, \ldots, x_n))^\bullet & = \langle\emptyset, \{P(x_1, \ldots, x_n)\}\rangle; \\
(\exists x)^\bullet & = \langle\{x\}, \emptyset\rangle; \\
(\phi \cdot \psi)^\bullet & = (\phi)^\bullet \bullet (\psi)^\bullet; \\
((\phi \rightarrow \psi))^\bullet & = ((\phi)^\bullet \Rightarrow (\psi)^\bullet).
\end{array}$$

Definition 1.5.9 ($\mathcal{L}_{drt}$ to $\mathcal{L}$) *The translation $(-)_\bullet : \mathcal{L}_{drt} \cup DRC \rightarrow \mathcal{L}$ is defined as follows:*

$$\begin{array}{ll}
(\bot)_\bullet & = \bot; \\
(P(x_1, \ldots, x_n))_\bullet & = P(x_1, \ldots, x_n); \\
((\phi \Rightarrow \psi))_\bullet & = ((\phi)_\bullet \rightarrow (\psi)_\bullet); \\
(\langle\{x_1, \ldots, x_n\}, \{C_1, \ldots, C_m\}\rangle_\bullet & = \exists x_1 \cdot \ldots \cdot \exists x_n \cdot (C_1)_\bullet \cdot \ldots \cdot (C_m)_\bullet.
\end{array}$$

Note that the translation $(-)_\bullet : \mathcal{L}_{drt} \rightarrow \mathcal{L}$ is not defined as an induction on the construction of the *DRS*s from basic *DRS*s. This is impossible,

since different constructions of the same *DRS* would then result in different translations. Therefore we use the alternative characterisation given by the proposition.
Now it can be checked that these translations indeed are meaning preserving, i.e.:

$$[\![\phi^\bullet]\!]_z = [\![\phi]\!]_{drt} \text{ and } [\![\phi]\!]_z = [\![\phi_\bullet]\!]_{drt}$$

Predicate logic, static and dynamic

In this section we compare the dynamic semantics of $\mathcal{L}$ with the static semantics of standard predicate logic. The comparison will work via the notions of truth that these three formalisms induce. We will give a translation of $\mathcal{L}$ into $\mathcal{L}_{pl}$ that preserves *DRT*-truth and another translation that preserves *DPL*-truth. Also truth preserving translations from $\mathcal{L}_{pl}$ into $\mathcal{L}$ will be discussed.
There are several things that we can learn from these translations. First they make us aware of the similarities and differences between the use of quantifiers in dynamic and static logic. The translations show that, in spite of all sorts of differences, the two styles of dynamic semantics of $\mathcal{L}$ that we discuss, give $\mathcal{L}$ exactly the same expressive power as standard predicate logic. Here we use expressive power in the static sense, i.e. in terms of truth-conditions: if we are only interested in truth-conditions, we can simply choose whether we want to do this dynamically or statically, because of the existence of truth preserving translations. Note that this implies that we can use the deduction systems of static logic to discuss the dynamic notions of truth: if we want to know whether some formula $\phi \in \mathcal{L}$ is true in the *DRT* or *DPL* sense, then we first apply the appropriate translation and then check whether the result is a theorem in the static sense. Unfortunately the translations do not work in such a way that we can translate the deduction systems in static logic in a rule-by-rule fashion into a deduction system for the dynamic notions: it does not hold in general that

$$\phi \models_{DPL} \psi \text{ iff } (\phi)_{dpl} \models_{pl} (\psi)_{dpl}$$

or

$$\phi \models_{DRT} \psi \text{ iff } (\phi)_{drt} \models_{pl} (\psi)_{drt}$$

Here $\phi = \exists x \cdot P(x)$ and $\psi = P(x)$ can serve as an example.

Definition 1.5.10

For the formulas of $\mathcal{L}$ we define a DPL-based translation into $\mathcal{L}_{pl}$ as follows:

$$\begin{array}{lll@{\quad}r}
1 & (\bot)_{dpl} & = \bot; & \\
2 & (P(x_1,\ldots,x_n))_{dpl} & = P(x_1,\ldots,x_n); & \\
3 & (\exists x)_{dpl} & = \exists x \top; & \\
4 & (\phi \cdot \psi)_{dpl} & = \exists x(\psi)_{dpl} & \textit{in case } \phi = \exists x; \\
 & & = (\phi_1 \cdot (\phi_2 \cdot \psi))_{dpl} & \textit{in case } \phi = \phi_1 \cdot \phi_2; \\
 & & = (\phi)_{dpl} \wedge (\psi)_{dpl} & \textit{else;} \\
5 & (\phi \to \psi)_{dpl} & = \forall x(\psi)_{dpl} & \textit{in case } \phi = \exists x; \\
 & & = (\phi_1 \to (\phi_2 \to \psi))_{dpl} & \textit{in case } \phi = \phi_1 \wedge \phi_2; \\
 & & = ((\phi)_{dpl} \to (\psi)_{dpl}) & \textit{else.}
\end{array}$$

We define DRT-normal-forms as follows:

$$\begin{array}{lll@{\quad}r}
1 & (\bot)^{\diamond} & = \bot; & \\
2 & (P(x_1,\ldots,x_n))^{\diamond} & = P(x_1,\ldots,x_n); & \\
3 & (\exists x)^{\diamond} & = \exists x; & \\
4 & (\phi \cdot \psi)^{\diamond} & = \psi \cdot (\phi)^{\diamond} & \textit{in case } \psi = \exists x; \\
 & & = ((\phi \cdot \psi_1) \cdot \psi_2)^{\diamond} & \textit{in case } \psi = \psi_1 \cdot \psi_2; \\
 & & = \phi^{\diamond} \cdot \psi^{\diamond} & \textit{else;} \\
5 & (\phi \to \psi)^{\diamond} & = (\phi^{\diamond} \to \psi^{\diamond}) &
\end{array}$$

The operation $(-)^{\diamond}$ pulls occurrences of $\exists x$ to the front of the formula. The resulting formulas are the DRT-normal-forms. Now we can simply apply the DPL-based translation: the DRT-based translation of $\mathcal{L}$ is defined as:

$$(\phi)_{drt} = ((\phi)^{\diamond})_{dpl}$$

Note that we have based the *DRT* style translation of $\mathcal{L}$ on the *DPL*-based translations of so-called *DRT*-normal-forms. It can be checked that this is a safe move: for the *DRT*-normal forms the notions of *DPL*-truth and *DRT*-truth coincide.[15] It is also easy to see that the operation $(-)^{\diamond}$ preserves meaning: all that is different in the normal forms is the order of some of the conjuncts, to which the *DRT*-style interpretation is not sensitive in the first place. So we get:

[15]Also see the next subsection for more on the relation between the *DRT* and *DPL* semantics of $\mathcal{L}$.

$$[\![\phi]\!]_{drt} = [\![(\phi)^{\diamond}]\!]_{drt}$$

Now it just remains to be checked that the *DPL*-based translation indeed preserves truth.

Proposition 1.5.11
Let $\phi \in \mathcal{L}$ be given. Then: ϕ is DPL-true iff $(\phi)^{\diamond}$ is true statically.□

We omit the formal proof—which is simply an induction—but we do wish to point out the crucial facts about the *DPL* interpretation on which the proof relies. First note:

$$[\![(\phi \cdot \psi) \cdot \chi]\!]_{dpl} = [\![\phi \cdot (\psi \cdot \chi)]\!]_{dpl}$$
and
$$[\![(\phi \cdot \psi \to \chi)]\!]_{dpl} = [\![(\phi \to (\psi \to \chi))]\!]_{dpl}.$$

This implies, in particular that:

$$[\![(\exists x \cdot \phi) \cdot \psi]\!]_{dpl} = [\![\exists x \cdot (\phi \cdot \psi)]\!]_{dpl}$$
and
$$[\![((\exists x \cdot \phi) \to \chi)]\!]_{dpl} = [\![(\exists x \to (\phi \to \psi))]\!]_{dpl}.$$

For the forms on the right hand side the correspondence with static truth is easy: $\exists x \cdot (\phi)$ can safely be translated into $\exists x(\phi)_{dpl}$ and $(\exists x \to \phi)$ corresponds to $\forall x(\phi)_{dpl}$, as we have seen in the examples. As one can see in the definition, the *DPL* translation is based on the forms on the right hand side: in the translation there is an implicit reduction to these easy cases.

We can also give truth preserving translations in the other direction. This is easiest for *DPL*-truth. Basically we can simply translate $\wedge$ as $\cdot$, $\to$ as $\to$, $\exists x\phi$ as $\exists x \cdot \phi$ and $\forall x\phi$ as $(\exists x \to \phi)$. But a problem can arise if the following kind of configuration occurs in the $\mathcal{L}_{pl}$ formula:

$$(\exists x\ \phi) \ \wedge \ \psi$$

where x occurs freely in ψ. In such situations problems can arise, because:

$$[\![(\exists x \cdot \phi) \cdot \psi]\!]_{dpl} \;=\; [\![\exists x \cdot (\phi \cdot \psi)]\!]_{dpl}$$

as we have seen above, while *not*:

$$\exists x(\phi \,\wedge\, \psi) \leftrightarrow (\exists x\; \phi) \,\wedge\, \psi$$

in predicate logic. Therefore we make sure that such configurations do not occur by systematically switching the order of the conjuncts:

$$(\exists x\; \phi) \,\wedge\, \psi \;\rightsquigarrow\; \psi \wedge (\exists x\; \phi)$$

Let's call the formulas that we obtain after these switches 'safe for *DPL*-translation'.[16] It can be checked that any formula can be made safe for *DPL*-translation without altering the static truth value of the formula. Once this is done, we can apply the following translation procedure.

Definition 1.5.12 *Assume that all $\mathcal{L}_{pl}$-formulas on the left hand side are safe for* DPL*-translation. Now define:*

1	$(\bot)_{pl}$	$= \bot;$
2	$(P(x_1,\ldots,x_n))_{pl}$	$= P(x_1,\ldots,x_n);$
3	$(\phi \wedge \psi)_{pl}$	$= (\phi)_{pl} \cdot (\psi)_{pl};$
4	$(\phi \to \psi)_{pl}$	$= ((\phi)_{pl} \to (\psi)_{pl});$
5	$(\exists x\; \phi)_{pl}$	$= \exists x \cdot (\phi)_{pl};$
6	$(\forall x\; \phi)_{pl}$	$= (\exists x \to (\phi)_{pl}).$

Again the claim is that this translation preserves truth.

Proposition 1.5.13 *Let $\phi \in \mathcal{L}_{pl}$ be safe for* DPL*-translation. Then: ϕ is true statically iff $(\phi)_{pl}$ is DPL-true.*$\square$

The proof is an easy induction, which works precisely because we avoid the configurations $(\exists x\; \phi) \,\wedge\, \psi$.
In the case of *DRT*-truth there is one extra step to make: in the *DRT*-style interpretation of $\mathcal{L}$ it holds that:

$$[\![\exists x \cdot \phi \cdot \exists x \cdot \psi]\!]_{drt} \;=\; [\![\exists x \cdot \phi \cdot \psi]\!]_{drt}$$

[16]Another popular way to avoid problems is by renaming variables, but it is useful to see that this is not really necessary.

In the *DRT*-style interpretation the second occurrence of $\exists x$ is overruled by the first occurrence. This makes the translation slightly more difficult, since this is something that typically does not hold in static predicate logic. Therefore we have to rename the bound variables first in such a way that all bound variables become distinct from each other as well as from all free variables. Once we have done this we can be sure that the *DPL*-based translation will also preserve *DRT*-truth. For now the translations automatically end up in the so called *strict* fragment of $\mathcal{L}$ and we will see in the next subsection that in the strict fragment the *DPL* and *DRT* interpretation coincide.

The meaning of $\exists x$

In this subsection we investigate the differences between the two styles of semantics that we gave above. We will see that the crucial difference between the *DPL*-style and the *DRT*-style interpretation is in their treatment of $\exists x$ in conjunctions.
We will make the comparison by developing a relational version of the *DRT* semantics for $\mathcal{L}$. Of course this is only one way of comparing *DRT* and *DPL*. As an alternative we could, for example, try to make precise the relation between $range([\![\phi]\!]_{dpl})$ and the truthful embeddings of ϕ. As we discussed the notion of *DPL*-truth we already pointed out that one can think of the relations in the *DPL*-semantics as a way of constructing truthful embeddings. So we could try to compare the *DPL* and *DRT* interpretation of a formula $\phi \in \mathcal{L}$ by making this idea precise. For example, if $[\![\phi]\!]_{drt} = \langle X, F\rangle$ and $[\![\phi]\!]_{dpl} = R$ (for some $R \subseteq ASS \times ASS$), then we can look whether there is some systematic relation between F and $range(R) = \{f \in ASS : \exists\langle g, f\rangle \in R\}$. It is easy to see that these sets coincide for atomic ϕ, but are not in general identical: it can be checked that $\phi = P(x) \cdot \exists x$ is a counterexample.
Instead of working out the systematic differences along these lines we use a relational formulation of the *DRT*-semantics. This relational form of *DRT* can then be compared with the relational *DPL*-semantics. In the formulation we use several constructions. In these constructions we will need to know which of the variables in VAR are 'important' for a relation R: we will define for each R the sets $car(R)$ and $unt(R)$. In the current set-up — where we stay as close as possible to the standard way of using relations in dynamic semantics — we have to extract this kind of infor-

mation from the relations, as it is simply not available explicitly. Later on, in chapters 2 and 3, we will find it convenient to include information about the sets of important variables explicitly in the semantics.

Definition 1.5.14

- *For $R \subseteq ASS \times ASS$ we define $car(R)$, the carrier of R, by:*

$$car(R) = \{x : \exists \langle g, f\rangle \in R : f(x) \neq g(x)\}$$

- *We say that $R \subseteq ASS \times ASS$ is input-indifferent if:*

$$g \in range(R) \Rightarrow \langle f, g\rangle \in R \textit{ for all } f =_{car(R)} g$$

- *For $R \subseteq ASS \times ASS$, we define $unt(R)$, the set of variables untouched by R, as follows:*

$$x \in unt(R) \textit{ iff}$$
$$\forall \langle f, g\rangle \in R : f(x) = g(x) \;\&\; \forall d \in DOM : \langle f_{[x/d]}, g_{[x/d]}\rangle \in R$$

- *For a DRS-interpretation $\langle X, F\rangle$ we define the induced relation $R_{\langle X,F\rangle}$ as follows:*

$$R_{\langle X,F\rangle} = \{\langle g, f\rangle : g =_X f \;\&\; f \in F\}$$

- *For two relations S, $S' \subseteq ASS \times ASS$ we define $S \star S'$ as follows:*

$$S \star S' = R_{\langle car(S), range(S)\rangle \oplus \langle car(S'), range(S')\rangle}$$

Here the notation $f_{[x/d]}$ is used to denote the assignment that assigns d to x and is equal to f on all other variables.
We see that $car(R)$ is the set of variables that can get a new value in R. A relation R is called input indifferent if it does not care about the input values of the variables in its carrier set. On the intuitive level this means that the first thing that the process R does is to set the value of the $x \in car(R)$ to its liking. Only after that tests may be performed on the values that have been chosen. Therefore it does not matter what the input value of $x \in car(R)$ is. The variables in $unt(R)$ are the variables that R does not care about: if $x \in unt(R)$, then R does not change the value of x ($x \notin car(R)$) and it simply allows all $d \in DOM$ as value of x. We will use these two notions in our comparison of *DRT* and *DPL* below.
It can be shown that $R_{(.)}$ is almost an injective mapping from *DRS*-interpretations to relations: we find

$$R_{\langle X,F\rangle} = R_{\langle Y,G\rangle} \text{ iff } F = G \text{ and } (X = Y \text{ or } F = \emptyset)$$

Note that in the relation $R_{\langle X,F\rangle}$ that we assign to a *DRS*-interpretation $\langle X,F\rangle$, X is the carrier set unless F is empty. So from the relation $R_{\langle X,F\rangle}$ we can *almost* recover the information in the first component of the *DRS*. The only cases for which a problem arises are cases where $F = \emptyset$, i.e. in case of an inconsistency.
This constructions allows us to construct a relation from each $[\![\phi]\!]_{drt}$. For each ϕ we can simply define $R_\phi \subseteq ASS \times ASS$ as follows:

$$R_\phi = R_{[\![\phi]\!]_{drt}}$$

Again it can be checked that for atomic ϕ it holds that $R_\phi = [\![\phi]\!]_{dpl}$ and that this equality does not hold in general. We can make the discrepancy between R_ϕ and $[\![\phi]\!]_{dpl}$ precise by giving an inductive characterisation of R_ϕ.

Proposition 1.5.15

$$\begin{array}{ll} R_\perp & = \emptyset \\ R_{P(x_1,\ldots,x_n)} & = \{\langle f,f\rangle : \langle f(x_1),\ldots,f(x_n)\rangle \in P^\star\} \\ R_{\exists x} & = \{\langle f,g\rangle : f =_x g\} \\ R_{\phi\cdot\psi} & = R_\phi \star R_\psi \\ R_{(\phi\to\psi)} & = \{\langle f,f\rangle : \forall g : \langle f,g\rangle \in R_\phi \; \exists h : \langle g,h\rangle \in R_\psi\} \end{array}$$
□

We see that the difference between $[\![.]\!]_{drt}$ and $[\![.]\!]_{dpl}$ lies in the difference between $\star$ and $\circ$. Therefore it would be nice to have an easy way of deciding in advance exactly when $R \star S = R \circ S$ will hold. Unfortunately there is no handy set of necessary and sufficient conditions that characterises this situation. Instead we present an easy-to-work-with sufficient condition.
The crucial difference between $\star$ and $\circ$ can be illustrated with the formula $P(x) \cdot \exists x$. This is the simplest formula that gives rise to differences in the *DPL* and the *DRT* style interpretation: in the *DPL* relation, computed with $\circ$, we will first check whether $f(x) \in P^\star$ and then we assign a new value to x. This means that the output assignment g may not satisfy $g(x) \in P^\star$. In the *DRT* relation, computed with $\star$, we make sure that the output assignment g satisfies $g(x) \in P^\star$. Then we allow for all input assignments f that differ only on x from g. So now the input assignment may not satisfy $f(x) \in P^\star$. In fact we get precisely:

$$\langle f, g\rangle \in R_{P(x)\cdot\exists x} \text{ iff } \langle g, f\rangle \in [\![P(x)\cdot\exists x]\!]_{dpl}.$$

In the example we see that the first relation, $R_{P(x)}$, imposes a condition on the variable x while the second relation, $R_{\exists x}$, allows us to choose a value for this variable. This is precisely the sort of situation in which $\star$ and $\circ$ behave differently: $\star$ tells us to choose the value first and then check the condition, while $\circ$ tells us to check the condition first and then choose a new value for the variable. The result below basically says that $\star$ and $\circ$ coincide as long as we avoid such situations.
Formally we try capture this idea using the two auxiliary notions *car* and *unt*. Recall that the variables in the carrier $car(R)$ are the ones that gcan get a different value in R. Intuitively the variables in $unt(R)$ are the variables that R does not care about. So we need not worry about the variables in $unt(R)$ if we compute $R \star S$ or $R \circ S$: R does not care what S does to them or when it does it. In the example above we saw that $x \notin unt(R)$, while at the same time $x \in car(S)$. This means that S changes a value of a variable that R *does* care about. This is a crucial condition: we have to make sure that $car(S) \subseteq unt(R)$, otherwise things could go wrong.
There is one extra complication that we have to take care of: all relations of the form $R \star S$ are input indifferent, but not all relations of the form $R \circ S$ have this property. This means that we have to build in a restriction on R and S somehow to ensure that $R \circ S$ will also be input indifferent. It turns out that, in presence of the condition $car(S) \subseteq unt(R)$, the following works.

Proposition 1.5.16 *Let $R, S \subseteq ASS \times ASS$ be input indifferent relations such that $car(S) \subseteq unt(R)$. Then $R \star S = R \circ S$.*□

Note that we had to sneak in the extra requirement of input indifference to make the proposition work. However, for our purposes this is not a very high price to pay, since all relations associated with atomic formulas are input indifferent.
The proposition shows that for input indifferent relations $car(S) \subseteq unt(R)$ is a sufficient condition. It can be checked that it is not a necessary condition, for example by looking at $\exists x \cdot P(x) \cdot \exists x \cdot P(x)$. So in a sense the condition in the proposition is bit too strong.
Here we have given a condition that works in terms of the behaviour of the relations R and S. But for our purposes a condition that works directly on

the formulas is more convenient. Such a condition can be obtained quite easily: we already observed that all atomic formulas give rise to input indifferent relations. It can be checked that this also holds for the test relations $(\phi \to \psi)$. The fact that $x \in car(R_\psi)$ shows that $\exists x$ occurs as a conjunct of ψ and we can guarantee that $x \notin unt(R_\phi)$ by ensuring that x does not occur in ϕ. So the propositions tells us that as long as x does not occur in a condition before $\exists x$ in a conjunction, the two interpretations will coincide. Formulas that satisfy this condition have been called *strict* in the literature and the set of strict formulas is sometimes called the strict fragment of $\mathcal{L}$. So as a corollary of the proposition we get:

Corollary 1.5.17 *Let $\phi \in \mathcal{L}$ be strict. Then $R_\phi = [\![\phi]\!]_{dpl}$ will hold. As a result ϕ will be DRT-true iff ϕ is DPL-true.*$\Box$

Note that the syntactic notion of strictness is even stronger than the condition on the relations R and S that we gave above: there are non-strict formulas $\phi \cdot \psi$ that satisfy the condition of the proposition. Take for example $(P(x) \to P(x)) \cdot \exists x$.

On the road

Now we have made our first steps in dynamic semantics and we are on the road. We have presented the phenomenon 'dynamic semantics' informally and we have discussed two ways in which this phenomenon can be worked out formally. In the previous section we have linked up our presentation with what can be found in the literature.
In our presentation we have concentrated on *DPL* and *DRT* and thereby ignored several other interesting developments in dynamic semantics. For example, we have not discussed the Swedish dynamic tradition of Pagin and Westerståhl (1993). There are many things that can be said about the relation of their *Predicate Logic with Flexibly Binding Operators* (*PFO*) and the dynamic theories discussed here, but since we will not use any of these facts later on we have decided to leave *PFO* out all together.
This underlines the fact that we have not tried to give an objective overview of current developments in dynamic semantics. Instead the aim has been to present a thoroughly subjective view on what dynamics is

about. For a historically correct picture the reader can follow the references to the literature that we have included.
In the remaining part of this thesis we will build upon the picture of dynamics that we given here. In part I we will take a closer look at the role of variables in dynamic semantics: the achievement of both *DRT* and *DPL* has been the isolation of the semantic contribution of $\exists x$ in one small unit. We will analyse the consequences of the way in which this has been worked out formally. This will lead us to propose an improvement for the dynamic treatment of variables.
In part II we confront the *small unit principle* with the phenomenon of text structure. We will see how a step by step interpretation of structured expressions can be obtained. Finally we will apply our conception of dynamics to proof theory: we will discuss what a dynamic approach to proof theory could lead to.

Part I

Variables in Dynamic Semantics

Variables in Dynamic Semantics

The following chapters will be devoted to the role of variables in dynamic semantics. First we will consider a specific problem with variables in dynamic semantics in the next chapter, the so called *eliminativity* problem. The solution that we will develop for that problem will lead us into more general considerations about the treatment of variables in dynamic semantics. This will result in the definition of *referent systems* in chapter 4: we shall argue that referent systems are the proper tool for the dynamic treatment of variables.

Our attention for the treatment of variables in dynamic semantics has a very simple explanation: the treatment of variables is crucial for the semantics of anaphora. Some attempts have been made to represent anaphoric phenomena in a representation language without variables (cf. Purdy (1992a), Ben-Shalom (1994), Sánchez Valencia (1990), Böttner (1992) etc.), but in many approaches the variables represent the anaphoric links. Also in the dynamic approach to semantics, both *DRT* and *DPL* interpret anaphors using variables. Since the semantics of anaphora poses the problems that have motivated the development of dynamic semantics, the treatment of variables is of crucial importance to the dynamic enterprise.

Before we get to this we first wish to clarify the role that variables play in semantics of natural language in general. Our views on this issue are pretty standard as far as we can see. Still we have noticed that it quite often happens that misunderstandings about the role of variables give rise to serious confusion as to what is at stake in the semantics of anaphora.[17]

[17]For example, some people have had the impression that it was our aim to show that working with indexed formulas is unnecessary. It will become clear that this is

We hope to prevent such misunderstandings by repeating some well known truisms about variables and natural language semantics.

Variables and natural language

One inevitable observation about the relation between variables and natural languages is that natural languages do not contain variables. If we look at a text in a natural language we may find a lot of things, but we will not find the x-es and y-s that we use so abundantly in our formal machinery. Yet in the semantics of natural language it is very common to use formal languages that *do* contain variables. In these formal languages the variables are extremely important. There it are the variables that allow us to make the binding and scoping relations between the different items syntactically explicit. For example, only if two occurrences of a variable, x say, are within the scope of the same quantifier, we may conclude that the variable has the same 'meaning' in both positions.

We can see that this implies that a lot of facts about binding and scoping of natural language expressions can only be transmitted into the formal representation if we use the variables of the formal representation language correctly. Therefore variables are of crucial importance in the representation of the anaphoric links that we find in texts: if some anaphoric element is to be identified with some antecedent, then we had better made sure that the representations of both elements contain the same variable and are in the scope of the same quantifier. Then the connection between anaphor and antecedent will be interpreted as the semantic connection that the representation language establishes between the two variable occurrences in the representation.

From this we can conclude two things. First we see that the meaning that we give to anaphora in natural language simply *is* the meaning that we give to variables in the representation language. Since anaphoric links are represented by variables, the interpretation of anaphoric links is fixed by the semantics of the variables. So if we are working with formal representation containing variables, then giving a semantics of anaphora is the same thing as explaining how variables are interpreted. Secondly it means that for the interpretation of anaphora the question which variables arise in the representation of a text is crucial. If something goes wrong

not the case.

here, then the representation may not represent the anaphoric links in the text correctly and as a consequence the interpretation of the text gets corrupted. Therefore we will say something about the way in which variables come into the formal representation in the next subsection.

Where do variables come from?

In the compositional picture of the interpretation of natural language we usually distinguish two steps: the first step is the compositional translation of a natural language expression in a formal representation language. The second step consists of the compositional interpretation of this formal representation. Since variables are present in the formal representation and not in natural language expressions as we usually encounter them, they will have to be introduced into the picture somewhere before the second step. But where should they come from?
We could hope that they are generated automatically by the compositional translation procedure. Whether or not this is possible depends on the kinds of demands that we place on the formal representations that the translation procedure gives us. It depends in particular on what we expect the formal representation to tell us about the anaphoric links that are present in the natural language expression. For example, if we are allowed to *postpone* the problem of representing anaphoric links correctly until after the translation, then a compositional translation procedure might very well be possible. In such a situation we can simply ignore the problem of anaphora resolution during the translation. There are numerous ways to implement this delay strategy. We can, for example, use variables that are indexed with natural numbers: x_1, $x_2, \ldots$. In the representation of the basic constituents we use fixed but arbitrary variables and whenever we compose a representation for an expression from the representations of its constituents, we make sure that the variables arising from the different constituents are made distinct. We can do this, for example, by replacing each variable x_n in the left constituent with variable $x_{10 \cdot n+1}$ and replacing each variable in the right constituent with $x_{10 \cdot n}$.
Let's illustrate this with an example:

> A dog was sleeping in the room. A cat came in. It was spotted immediately.

If we apply our 'labeled-variable' translation procedure to this example, starting with variable x_0 in all basic representation, this would give rise to something like:

$$\exists x_{1111111} \cdot \text{DOG}(x_{1011111}) \cdot \text{SLEEPING-IN-THE-ROOM}(x_{101111}) \cdot \\ \exists x_{10111} \cdot \text{CAT}(x_{1011}) \cdot \text{CAME-IN}(x_{101}) \cdot \text{SPOT}(x_{10})$$

So in this approach the representations (in $\mathcal{L}$) can (probably) be built up compositionally and the variables are introduced by the compositional translation procedure.
This is all very well, but it is clear that the representation that we have created is not very useful. Before we can give it the intended interpretation we have to make sure that some variables x_n are identified with other variables x_m in such a way that the anaphoric relations that are implicit in the natural language expression get represented. In other words, this might be a good point to perform the process of anaphora resolution that we have postponed so far. Here this can be done by choosing some equivalence relation on the indices that tells us which indices are to be identified.[18]
Clearly such an approach gives a translation of the natural language expression into a formal representation language. However, the formal representation that we create does not yet reflect any of the anaphoric relations that are present in the natural language expression and this might be considered cheating. What we would really like is a compositional translation procedure in which the anaphoric relations get represented correctly. This means that the translation procedure has to perform the process of anaphora resolution — i.e. establishing the required connections between anaphors and antecedents — implicitly, as part of the larger task of representing the meaning of the formula.
It is clear that such a translation procedure would indeed be preferable over a procedure of the kind sketched above. But it is also clear that such a procedure is hard to get. For in order for such a translation to be compositional we would need a compositional procedure to trace all anaphoric links in natural language texts. And this is just too much. Even without going into the details of the problem of anaphora resolution[19],

[18]Note that there is a subtle difference between telling which variables are syntactically identical and telling which variables have identical values. It is the first thing that we are interested in here.

[19]Cf. Sidner (1983) and references therein for more detail.

we can see that it is very unlikely that all of anaphora resolution works compositionally. We can see this from the following example:

> The brick was thrown against the window. It broke.

If we regard this as one complete text, then the resolution process, however it may work, will have to produce the links between the anaphors and antecedents in this text. Presumably we would want a link between *It* and *the window.*[20] Note that this procedure cannot just rely on the syntax of the input text. For syntactically there seems to be no reason why *It* should be *the window* and not *The brick.* So the compositional resolution procedure would need as input not only the syntactic information, but also other information, presumably about the meaning of the words. But let's assume that extra information of this kind is available in the compositional translation procedure and that this anaphoric link actually is established in the representation. Then we will get into trouble with the following example:

> John was surprised by the strength of the window. The brick was thrown against the window. It broke.

Now the preferred anaphoric link is between *It* and *The brick.* In a compositional mechanism for anaphora resolution it will not be easy to account for such a switch. The problem can be explained as follows: either the first text *is* a component of this second text. Then it is clear that the representation of the first text will be a component of the representation of the second text. But we argued above that in this component *It* will be linked to *the window* instead of *The brick.* So there is a problem. Or else the first text is *not* a component of the second text. But it seems quite unlikely that any discourse analysis will predict this. For in the text the second and third sentence together play the role of an *explanation* or *elaboration* of the first sentence. It seems reasonable to expect that this kind of macro structure of texts will be respected by a good discourse grammar.

So we have to conclude that it would be unrealistic to expect a compositional representation procedure to *produce* all anaphoric links by itself.

[20]Perhaps one would like to argue for the ambiguity of such texts. This amounts to a strategy of postponing the resolution problem, at least to some extent. Such a strategy was mentioned above and will be discussed further below.

Since these anaphoric links are encoded in the choice of variables, this implies that the choice of variables cannot simply be part of the compositional translation procedure. This means that the choice of variables is either fixed *after* translation, as in the labeled-variable approach discussed above, or else the choice of variables is made *before* the compositional translation. The first option is not at all unreasonable: then the resolution procedure could simply work on the formal representation[21] rather than on the natural language expression itself. Still, the second option is preferred in most current approaches to natural language semantics: it is usually assumed that the input to the translation mechanism is an *indexed* natural language expression, i.e. a natural language expression in which variables have been added as indices to indicate the anaphoric links. This is also what we will assume in this thesis. So instead of the text above we will get the following kind of input:

> John was surprised by the strength of the window$_2$. The brick$_1$ was thrown against the window$_2$. It$_1$ broke.

Of course we have only sketched two extremes of a whole range of possibilities: we have discussed approaches in which all anaphora resolution is done in advance and approaches in which all the work is postponed until after the translation process. No doubt a realistic account of anaphora resolution has to be found somewhere in between these extremes. Probably there are a few reasonable side conditions on anaphoricity — such as gender and number agreement — which can savely be implemented in a compositional translation procedure.[22] Then the remaining ambiguities can be resolved by another mechanism.[23] But since we do not pretend to give a theory of anaphora resolution anyway, we can simply follow the conventional choice and assume that anaphora resolution has been taken care of in advance.

[21]Cf. Lewin (1994) for a worked out version of a similar approach.

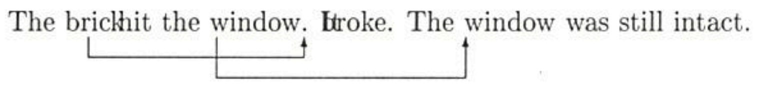
[22]Note that even gender agreement is not such a reliable side condition as it may seem at first sight. In particular problems arise if there is a discrepancy between syntactic gender and semantics gender. An example is the Dutch word for girl, which has syntactic gender neuter. It turns out that in these cases both the feminine and the neuter pronoun can be used for anaphoric reference.

[23]A. Visser suggests that perhaps it makes sense to expect a compositional translation to produce *resolution strategies*, instead of anaphoric links. The result of the compositional mechanism would then be a (partially resolved) representation plus a compositionally derived strategy for resolving the (remaining) anaphoric links.

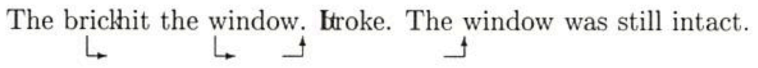
By now we are in a position to answer the question that was asked above: "Where do variables come from?" We simply assume that they are there already.

Anaphora resolution and indexing

Perhaps the discussion above has been a bit misleading about the relation between anaphora resolution and indexing procedures. From the previous section the reader might get the impression that we think that these two phrases stand for one and the same thing. This is not true, of course. What we can expect from the resolution procedure is that it will indicate which anaphoric links exist in the text. This need not be represented by variables as indices at all. We could also get something like this:


The brick hit the window. It broke. The window was still intact.

Here the arrows indicate anaphoric links.[24] It seems that this is the minimal information that the anaphora resolution process can be expected to give: it gives exactly the anaphoric links between the items and nothing else. As one can see, there are no indices in this picture and still all the information is made explicit. However, there is a problem with the picture as it stands: it cannot be fed into a compositional translator. This is because all the links that we see in the picture make it impossible to decompose the expression into components without losing the information that these links give.


The brick hit the window. It broke. The window was still intact.

We see that if we break up the text, and therefore also the links, into pieces information gets lost. In order to prevent this we have to make the information about the links available locally before we feed the text into the translator. This way each component contains by itself the information about its anaphoric intentions. This is what indexing does: it makes

[24] Note that the arrows point from antecedent to anaphor. They point in the direction in which the information flows.

the information about the anaphoric dependencies available *locally*.

```
The brick hit the window. It broke. The window was still intact.
       └→x        └→y  x┘↑          y┘↑
```

We see that indexing is in fact a mapping from the arrows in the picture to variable names: we start with a picture with only arrows, then we add a variable name to each arrow and then we can 'decompose' the arrows without loss of information. In order for such a mapping to encode the anaphoric dependencies in the right way it has to satisfy some special conditions. For example, the mapping should not assign the same variable name to two arrows that cross each other. We can see this in the example: if we call both arrows x, then there is confusion after the process of splitting up the arrows. It seems that this is not only a necessary but also a sufficient condition on indexings: an indexing of the arrows preserves all the information about the anaphoric connections, precisely if different names are assigned to crossing links. A slightly stronger condition that is certainly sufficient is injectivity: if the indexing function assigns different variables to *all* arrows, then surely no information can get lost in the decomposition process.
Of course there may be other ways of encoding the anaphoric dependencies locally. It is not obvious at all that such an encoding should involve variable names. It is also not necessary that the input for the compositional translation procedure contain the variable names: also another local encoding of anaphoric dependencies will do. But since variable names are going to be used in the formal representation language anyway, we might as well use *them* in our local encoding of the anaphoric information. If the formal representation language uses another mechanism to encode this kind of information, then we can use that other mechanism for the 'indexing'-procedure.

The interpretation of variables

So we see that in the semantics of anaphora there is a very big problem that cannot be solved compositionally: the anaphora resolution problem. Therefore we will not try to solve this problem in a compositional semantics. This does not answer the question where we should solve the resolution problem instead, let alone *how* we should solve it. It is clear that many different kinds of information interact in anaphora resolution: not only syntactic information, about number and gender of antecedent and anaphor for example, but also other information sources, such as the meanings of the words, will typically be involved. And even when all such features are taken into into consideration, there will probably be cases where genuine ambiguities remain.

The fact that one of the crucial problems in the semantics of anaphora, anaphora resolution, cannot be solved compositionally, does not imply that compositional semantics should stay away from the semantics of anaphora and instead a non-compositional interpretation procedure should be developed. This line of reasoning could make sense, if our interest in the interpretation of anaphora is crucially an interest in anaphora resolution. We will take the position that there are other aspects of the interpretation of anaphora which are also interesting and which *do* require a compositional treatment. In fact we will allow ourselves to work with indexed expressions, expressions of natural language to which indices have been added to indicate the anaphoric connections. So we start at a point at which the resolution problem has already been solved.

The indices that we start with will be transmitted into the formal representations. So all that remains to be done is to see to it that in the semantics of the formal representation language these anaphoric connections are indeed recognised as such. In other words, our major concern is to find a representation language and a semantics for it in which the representations of the components can be put together in such a way that the required anaphoric connections really will be made. In the introductory chapter we saw that this is not a trivial problem: if we choose the naive representations in the language of predicate logic we will get into trouble. This shows that the way that variables are treated in predicate logic does not allow us to represent the links between the different components in a satisfactory way. We have seen that dynamic formalisms such as *DRT* and *DPL* do a lot better and this is where their contribution to the semantics of anaphora lies. Their contribution crucially depends on how they treat their variables. In the next two chapters we will take a closer look at the way in which this actually works. In the end (chapter 3) this will result in an improved treatment of variables in dynamic semantics using *referent systems.*

Chapter 2

Preserving Information in Dynamic Semantics

2.1 Introduction

In this chapter we will discuss the so-called *eliminativity problem* of *DPL*. We will see that the eliminativity problem, which will be introduced shortly, is a consequence of the treatment of variables in *DPL*. Therefore our solution to the eliminativity problem will consist of a refinement of *DPL*'s variable management. We will show that the refinement does indeed lead to an eliminative semantics.
In the next chapter we will pick up the issue of the treatment of variables in dynamic semantics in general. There our main aim will be to give good overall picture of the role that we want variables to play in dynamic semantics and we will propose a way of formalising this picture using *referent systems*.

In the introduction we have presented the relational semantics for $\mathcal{L}$ due to Groenendijk and Stokhof (1991a). Their *Dynamic Predicate Logic (DPL)* is a synthesis of the insights of *Discourse Representation Theory* (*DRT*) and the elegant formalism of predicate logic. The value of such a synthesis is generally recognised, but at the same time some questions can been raised. Several properties of the formalism have been discovered that suggest serious problems for *DPL*. One of these problems, the *eliminativity problem*, is the topic of this chapter.

The eliminativity problem was raised for the first time in Groenendijk and Stokhof (1991b). There the relational semantics of DPL is compared with the update semantics of Veltman (1991) Veltman provides a dynamic theory of modalities. He specifies the meaning of a modal (propositional) language in terms of operations on information states: the meaning of a formula is its potential to update information states. This update potential is represented as a function from information states to information states. Groenendijk and Stokhof reformulate the semantics of DPL as an update semantics to make a comparison with Veltman's system possible. Their update formulation has the following (defining) property.

Definition 2.1.1
Let ASS be the set of assignments. Let $\sigma \in \wp(ASS)$ be an information state. Let $[\![\phi]\!]_{gs} \in ASS \times ASS$ be the interpretation of ϕ as a relation on assignments. Then $(\!|\phi|\!)_{gs}$, the interpretation of ϕ as an update function, is defined by the following property:

$$\sigma(\!|\phi|\!)_{gs} = \{g \in ASS : \exists f \in \sigma : \ f[\![\phi]\!]_{gs} g\}$$

It is this formulation of the semantics of DPL that is the basis for the comparison with Veltman's update semantics. The properties of the semantics are discussed in terms of properties of the update functions. The following observations are made:

Proposition 2.1.2

- *DPL-updates are* **distributive**, *i.e. we always have*[1]
  $\sigma(\!|\phi|\!)_{gs} = \bigcup\{\ \{f\}(\!|\phi|\!)_{gs} : f \in \sigma\}$.
- *DPL-updates are not in general* **eliminative**, *i.e. for some ϕ and σ*
  not: $\sigma(\!|\phi|\!)_{gs} \subseteq \sigma$.

[1]The notation that we use here is not the one used by Groenendijk and Stokhof or Veltman, but we hope that no confusion will arise. The subscripts v and gs are used to distinguish Veltman's update functions from the DPL-updates. We will use $[\![.]\!]$-brackets for relational interpretations and $(\!|.|\!)$- brackets for update interpretations throughout the chapter.

This is in contrast with the situation in Veltman's system, where we have:[2]

- Veltman's updates are not in general *distributive*, i.e. for some ϕ and σ

  not: $\sigma[\![\phi]\!]_v = \bigcup\{\ \{f\}[\![\phi]\!]_v : f \in \sigma\}$.
- Veltman's updates are *eliminative*, i.e. we always have

  $\sigma[\![\phi]\!]_v \subseteq \sigma$.

Groenendijk and Stokhof conclude that these are genuine differences between Veltman's update semantics and their own dynamic semantics, that will have to be taken into account when an attempt is made to incorporate a dynamic treatment of modalities along the lines of Veltman in DPL. They do not seem to be worried about the fact that their semantics behaves as it does: non-eliminatively and distributively.

We have a different view on the meaning of these observations. The property of distributivity allows us to compute the result of a function on a set *pointwise*. For our update functions this means that we can compute the result of an update on the whole information state by updating each of its elements. It is convenient that this holds for the operations in the DPL-semantics, but it cannot be expected to hold for all extensions. There seems to be no intuitively compelling reason why sentence meanings should be computable pointwise. For example, the dynamic modality that Veltman considers typically is a case where one would *not* expect this. (Also see section 2.5.)

For the eliminativity property things are somewhat different. In update semantics the ordering $\supseteq$ on information states corresponds to growth of information. The eliminative functions are the functions that are *monotone* along this order. Hence if an operation on information states is eliminative this means that the operation causes an *increase* of information. And if an operation is not eliminative this means that somehow we have lost some of the information that we used to have. Therefore eliminativity is a property that we always expect if we process information,

[2]We will not be very precise about the details of Veltman (1991)'s update semantics in this chapter. We just give the properties of his system to illustrate the differences that Groenendijk and Stokhof (1991b) have found. Veltman (1991)'s semantics does not work with sets of assignments as information states, but here this harmless misrepresentation of the facts that will facilitate the comparison.

as long as typically non-monotonic features are kept out of consideration. *DPL* is not concerned with typically non-monotonic forms of information processing: *DPL* is used for the representation of simple narrative sentences. Therefore we would not only expect eliminativity for Veltman's system, but also — and perhaps even more so — for *DPL*. Thereby for us the non-distributivity of Veltman's system simply is a fact of life, while the fact that *DPL*-updates are not monotone poses a problem that has to be solved.

This is so for the intuitive reason expressed above, but it is also essential for a different reason. If we accept the non-monotonicity of *PDL*, while intuitively by the interpretation of a *DPL*-formula information grows, then we seem to accept a situation in which information content cannot be represented in the *DPL*-formalism. This would be a very unfortunate situation, given the picture of dynamic semantics that we have seen in chapter 1. In fact, if *DPL*-semantics has nothing to say about information content, then it is no longer clear that the *DPL*-semantics has anything to say at all.

So we cannot afford to accept the non-monotonicity of *DPL*-semantics. Therefore we aim for a slight modification of the semantics (section 2.2), called *sequence semantics* for dynamic predicate logic, which will allow us to represent growth of information. We will show that in the modified semantics the spirit of the usual relational interpretation is preserved (section 2.2.3): from the modified semantics of a formula we can reconstruct its original *DPL*-interpretation. The modified semantics will give rise to an improved notion of information state (section 2.3) and will allow for a formulation as monotone update semantics (section 2.5).

The new notion of information state will be explored further (in section 2.4) and we will use it to shed a new light on familiar topics in dynamic semantics. First we discuss the notion of monotonicity that comes with the new notion of information state, then we attack some of the problems that the *DPL*-notion of inference gives rise to. Finally we will discuss a new option that becomes available in the sequence approach: down-dating (section 2.5.3).

Although originally Groenendijk and Stokhof did not seem very concerned about non-eliminativity *per se*, by now it has generally been recognised that an eliminative version of *DPL* is called for. Several cures for the eliminativity problem have been proposed in the literature, but only the solutions that we present in this thesis[3] obtain eliminativity 'without compromising'. Other attempts have had to compromise in several different ways: for example, Dekker (1993)'s proposal amounts to a restriction to the so called strict fragment of *DPL*, whereas Fernando (1991a) resorts to the use of *guarded* assignment statements instead of ordinary *DPL* assignment statements. Visser (1989) also presents a dynamic system in which eliminativity is obtained by changing the notion of assignment. We will show that such *solutions by modification* are not called for since it is possible to formulate *DPL* itself in an eliminative way. The preservation property that we prove in section 2.2.3 shows that our presentation really is a refinement, *not* a modification of the original semantics of Groenendijk and Stokhof (1991a). Thereby this thesis presents the only eliminative semantics[4] for *DPL* in which the formulas still mean what we think they mean.

2.2 Sequence semantics

2.2.1 The language of dynamic predicate logic

Throughout this chapter we will work with $\mathcal{L}$ as defined in the introduction. So we have as examples of DPL-formulas: $\exists x \cdot \exists y$, $P(x, y, z)$, $\exists x \cdot P(x) \cdot Q(x, x)$, $(\exists y \to P(y))$. We have conjunction $(\cdot)$ and implication $(\to)$ as logical connectives. Negation $(\neg)$ is defined as: $\neg(\phi) \equiv (\phi \to \perp)$ and universal quantification as $\forall x(\phi) \equiv (\exists x \to \phi)$.
As we have explained, we prefer to treat the existential quantifier as an atomic formula. One reason for this is methodological: since $\exists x$ has a clear interpretation as a relation in DPL, there is no reason not to regard it as a distinct syntactic unit. We prefer to have an independent representation on the syntactic level of everything that plays an independent role in the semantics. Another reason is that this way DPL-formulas look more like $DRSs$ which makes a comparison of DPL and DRT easier. To us the language DPL seems to be a medium in which both DPL and DRT could be studied.
Working in $\mathcal{L}$ also makes some proofs a bit easier. But we have seen in the introduction that the semantics of $\mathcal{L}$ corresponds naturally to the

[3]Here and in the next chapter

[4]I.e. the semantics here and the one discussed in the next chapter.

ordinary *DPL*-semantics. So our results will not depend on the choice of $\mathcal{L}$.

2.2.2 The refined relational semantics

The refined version of dynamic semantics that we present is very similar to the usual relational semantics. The inductive definition of the relations that are the interpretations of the formulas of DPL is almost identical that of Groenendijk and Stokhof (1991a). The refinement that we propose is obtained by using a richer notion of assignment. Instead of working with assignments that assign one value in the domain to each discourse referent, we use assignments that assign to each discourse marker a finite sequence of values. In the usual semantics we are forced to forget the current value of the variable x if we encounter the formula $\exists x$. If we consider, for example, the formula $P(x) \cdot \exists x \cdot Q(x) \cdot \exists x \cdot R(x)$, a pair of assignments $\langle f, g\rangle$ that is in the relational interpretation $[\![P(x) \cdot \exists x \cdot Q(x) \cdot \exists x \cdot R(x)]\!]_{gs}$ will only reveal a value $f(x)$ that is in the interpretation of P, and a value $g(x)$ that is in the interpretation of R, but we have no way of remembering a value in the domain that is in the interpretation of Q.
We think that this is where the usual semantics goes wrong. If we throw away values of some variable, then we lose track of the restrictions on the values of this variable. These restrictions on the value of a variable are the way in which information is represented in DPL-semantics. If we throw away this information, there is no way of recovering it. If we work with assignments that have sequences as values, we can see to it that the value that is in Q is remembered. For example, if we find a pair (k, h) in the refined relational interpretation $[\![P(x) \cdot \exists x \cdot Q(x) \cdot \exists x \cdot R(x)]\!]_{sass}$, such that $k(x) = \langle \ldots, p\rangle$ and $h(x) = \langle \ldots, p, q, r\rangle$: this tells us that $p \in I(P)$, $q \in I(Q)$ and $r \in I(R)$. This way no information is lost. We call these assignments that have sequences as values, sequence valued assignments.

Definition 2.2.1

1. *$SASS$, the set of sequence valued assignments, is the set that consists of all functions $f : VAR \to DOM^\star$.*

   (Here DOM is the domain of interpretation, and $DOM^\star =$ $\bigcup\{DOM^n : n \geq 0\}$.)

2. $\langle d_1, \ldots, d_n\rangle \in DOM^* \Rightarrow end(\langle d_1, \ldots, d_n\rangle) = d_n$
$end(\langle\rangle) = \langle\rangle$

3. $\langle d_1, \ldots, d_n\rangle \in DOM^* \Rightarrow pd(\langle d_1, \ldots, d_n\rangle) = \langle d_1, \ldots, d_{n-1}\rangle$
$pd(\langle\rangle) = \langle\rangle$

4. $\langle d_1, \ldots, d_n\rangle, \langle e_1, \ldots, e_m\rangle \in DOM^* \Rightarrow \langle d_1, \ldots, d_n\rangle * \langle e_1, \ldots, e_m\rangle$
$= \langle d_1, \ldots, d_n, e_1, \ldots, e_m\rangle$,
the concatenation of two sequences.

It should be noted that the last component of the sequence $f(x)$, $end(f(x))$, is the *current* value of x. We have $f(x) = pd(f(x)) * \langle end(f(x))\rangle$. If the empty sequence is assigned to x ($f(x) = \langle\rangle$), then we call $f(x)$ undefined and we say $end(f(x)) = \langle\rangle$. The fact that we allow the empty sequence as a value introduces some form of partiality into our semantics. Therefore we also call the elements of $SASS$ *partial*, sequence valued assignments. It is also possible to use partial assignments in the usual relational semantics. Instead of working with total assignments, Groenendijk and Stokhof could have used partial assignments $f : VAR \hookrightarrow DOM$. (Here the partiality is indicated by the $\hookrightarrow$.) Let's call the set of partial assignments $PASS$.[5]
In such an approach new questions concerning partiality have to be answered. For example, if we want to define truth for a formula ϕ in such an approach, we have two options:

> ϕ is true iff for any partial assignment f, there is a partial assignment g such that $f[\![\phi]\!]g$;

or

> ϕ is true iff for any partial assignment f *defined on the free variables of* ϕ, there is a partial assignment g such that $f[\![\phi]\!]g$.

If we always choose for the second kind of answer, this gives us a semantics virtually equivalent to the usual relational semantics. We do not give the details of this approach: it requires but a straightforward adaptation of the usual semantics. We will use the notation $[\![\phi]\!]_{pgs}$ to refer to this

[5]Note that $ASS \subseteq PASS \subseteq SASS$.

version of the relational semantics.[6] Following the first option will give us something quite different. It might be interesting to see what happens in that kind of semantics, but we will not do that here.

In general some information about the occurrence of variables in some formula ϕ is available in the interpretation of ϕ. This is interesting, since the occurrence of variables in the formal representation is closely connected to the availability of anaphora in natural language. Consider for example the ordinary formulation of the *DPL* semantics in terms of total assignments. From the interpretation of ϕ we might observe that the value of some variable x is restricted: i.e. there is a $d \in D$ such that for no $(f, g) \in [\![\phi]\!]_{gs}$ we have $f(x) = d$. Then we can conclude that x occurs in ϕ. And also if in the interpretation of ϕ the value of x can change — i.e. there are $(f, g) \in [\![\phi]\!]_{gs}$ such that $f(x) \neq g(x)$ — we can conclude that x occurs in ϕ. But the total assignments do not reveal *all* the information about the occurrence of variables.

For example, if $\phi = (P(x) \rightarrow P(x))$, then ϕ is a statement about x. We admit that the information that ϕ gives about x is trivial, but still ϕ is a statement about x and not about any other variable. This implies for example that a free x in a subsequent formula will be linked to this x. In the interpretation with total assignments we cannot see such things: $[\![\phi]\!]_{gs}$ simply contains all pairs (f, f). But if we use partial assignments then this information becomes available in the semantics: if $(f, f) \in [\![\phi]\!]_{pgs}$, then f will have to be defined on x, but not necessarily on any other variable. So now we can see from the interpretation of ϕ that x occurs in ϕ. In other words, if we use partial assignments the interpretation can tell us about which variables formulas make a statement.[7]

We could in fact try to define the set of variables that occur in a formula from its interpretation with partial assignments:

$$vars(\phi) \;=\; \bigcap\{dom(f) : \; f \in range([\![\phi]\!]_{pgs}\}$$

This would then help us keep track of the anaphoric possibilities: the variables that occur in $vars(\phi)$ correspond to the antecedents that are

[6]See for example Kamp and Reyle (1993), Dekker (1993) or Fernando (1991a) for a presentation of semantics with partial assignments. Kamp and Reyle (1993) use partial assignments for the semantics of *DRT*.

[7]Here the analogy between formulas and computer programs might be helpful: even if you want to run a trivial program, you have to make sure all variables that actually occur are properly declared.

available in the corresponding natural language expression. Such a definition works quite well in most cases, but it typically breaks down with inconsistent formulas: a formula such as $(P(x) \to (P(x) \to \bot))$ will have an empty *vars*-set, but still offers one antecedent, x.
Although the use of partial assignments in the semantics helps us to some extent to keep track of such issues, it certainly does not help to solve the eliminativity problem. Clearly the use of partial assignments is not enough to get a good representation of all the information a formula can contain: we still loose information if a variable is used twice.[8] In our set up with sequence valued assignments we will inherit the advantages of the partial assignments, since we will allow $f(x) = \langle\rangle$. But we will also be able to deal with multiple occurrences of variables.
Before we give this refined version of the relational semantics, where this information is present, we introduce an important technical notion: we define what it means for one (sequence valued) assignment to be a x-variant of another (sequence valued) assignment.

Definition 2.2.2 *Let $x \in VAR$, $V \in VAR^*$, $f, g \in SASS$ be given.*

1. *We say that g is a x-variant of f,*

   $f\langle\!\langle x\rangle\!\rangle g$ *iff*
   $\exists d \in DOM : g(x) = f(x) * \langle d\rangle \wedge \forall y : (y \neq x \to f(y) = g(y))$.

2. *We define the notion V-variant for sequences of variables V as follows:*

   $f\langle\!\langle\, \langle x\rangle\, \rangle\!\rangle g \Leftrightarrow f\langle\!\langle x\rangle\!\rangle g;$
   $f\langle\!\langle\, V * \langle x\rangle\, \rangle\!\rangle g \Leftrightarrow \exists h \in SASS : f\langle\!\langle\, V\, \rangle\!\rangle h \wedge h\langle\!\langle x\rangle\!\rangle g.$

The relation $\langle\!\langle x\rangle\!\rangle$ allows us to choose a new value for x, but we do *not* throw away the old value: we build up a sequence of values instead. Note that the x-variant relation is *not* symmetric. In fact we have $f\langle\!\langle x\rangle\!\rangle g \Rightarrow \neg\, g\langle\!\langle x\rangle\!\rangle f$. Another important property of these relations is that $\langle\!\langle V\rangle\!\rangle$ is

[8] In Dekker (1993) we see that even in the presence of partial functions a radical trick has to be introduced to obtain eliminativity: in his *EDPL* all formulas in which a variable occurs in more than one role (such as $\exists x \cdot P(x) \cdot \exists x \cdot Q(x)$) are regarded as meaningless.

the same as $\langle\langle V' \rangle\rangle$ whenever V' is a permutation of V. So we have, for example, $\langle\langle \, \langle x, y, x \rangle \, \rangle\rangle = \langle\langle \, \langle y, x, x \rangle \, \rangle\rangle$.
Now we will define the refined relational semantics.

Definition 2.2.3

1. *For each $\phi \in DPL$ we define $[\![\phi]\!]_{sass} \subseteq SASS \times SASS$, the meaning of ϕ as a relation on sequence valued assignments. Let f, g, h, k be sequence valued assignments, I an interpretation function for the predicates of DPL.*
   - $f[\![\bot]\!]_{sass}g \;\Leftrightarrow\; f \neq f$
   - $f[\![P(x_1, ., x_n)]\!]_{sass}g \;\Leftrightarrow$
     $f = g \;\wedge\; (end(f(x_1), ., end(f(x_n))) \in I(P)$
   - $f[\![\exists x]\!]_{sass}g \;\Leftrightarrow\; f\langle\langle x \rangle\rangle g$
   - $f[\![\phi \cdot \psi]\!]_{sass} \Leftrightarrow \exists h \; f[\![\phi]\!]_{sass}h[\![\psi]\!]_{sass}g$
   - $f[\![(\phi \rightarrow \psi)]\!]_{sass}g \;\Leftrightarrow\; f = g \wedge \forall h : \; f[\![\phi]\!]_{sass}h \Rightarrow \exists k : \; h[\![\psi]\!]_{sass}k$
2. *We define entailment as follows: let $\phi, \phi_1, \ldots, \phi_n, \psi \in DPL$. Then:*

   $\phi \models_{sass} \psi \;\Leftrightarrow\; \forall f, g : f[\![\phi]\!]_{sass}g \Rightarrow \exists h : g[\![\psi]\!]_{sass}h$

   and

   $\phi_1, \ldots, \phi_n \models_{sass} \psi \;\Leftrightarrow\; \phi_1 \cdot \; \ldots \; \cdot \phi_n \models_{sass} \psi.$

The definition is very similar to the original definition by Groenendijk and Stokhof. The main difference is the kind of assignments that is used. This has some interesting consequences. Note for example that in the definition of $[\![\phi \cdot \psi]\!]_{sass}$ the h such that $f[\![\phi]\!]_{sass}h[\![\psi]\!]_{sass}g$ is uniquely determined by f, g.
Now if we look at $[\![\phi]\!]_{sass}$, we can find almost all the information that is revealed by ϕ: if we look carefully at the variables on which pairs $\langle f, g \rangle$ that are in $[\![\phi]\!]_{sass}$ are defined, then we can find out which variables are free in ϕ, which variables occur quantified and how often a variable is quantified over. For example, if we consider $[\![P(x) \cdot \exists x \cdot Q(x)]\!]_{sass}$, we will see that for all pairs $\langle f, g \rangle$ that are in this relation, we have that both f and g are defined on x (i.e. $f(x) \neq \langle\rangle$),which means that x must occur in the formula; the fact that f will always be defined on x, means that there

is a *free* x in the formula. We will see that the length of the sequence $g(x)$ is at least two, which means that x occurs in the formula in two (possibly different) roles. Since x already occurs freely, we can infer from this that there must be exactly one occurrence of $\exists x$ in the formula. We will also see that for any other variable y, there is a pair $\langle f, g \rangle$ where $f(y) = g(y) = \langle\rangle$, indicating that y does not occur in our formula.
The exceptions to this story are still the inconsistent formulas. Therefore we could choose to make this information explicit in the relational semantics. If we add two components to our indices, we can build up the set of the free variables of a formula and the sequence of the quantified variables quite easily. Then $[\![.]\!]_{sass}$ would become a relation on triples (A, S, f) where A contains the free variables and S gives the information about the quantified variables and $f \in SASS$. We will not make this precise here, but the reader can see how it could be done in the static semantics that we will present later.

2.2.3 A comparison

In this section we compare the refined relational semantics with the usual relational semantics. We will show that for any ϕ the partial relational interpretation, $[\![\phi]\!]_{pgs}$, can be constructed out of the relational interpretation $[\![\phi]\!]_{sass}$. This way we also establish a relation between $[\![\phi]\!]_{sass}$ and $[\![\phi]\!]_{gs}$, since $[\![\phi]\!]_{gs}$ is just the restriction of $[\![\phi]\!]_{pgs}$ to total assignments.
We establish the relation between $[\![\phi]\!]_{sass}$ and $[\![\phi]\!]_{pgs}$ with a mapping Φ that is defined as follows:

Definition 2.2.4
We define $\Phi : \wp(SASS \times SASS) \to \wp(PASS \times PASS)$ *as follows:*
$\Phi(R) =$

$$\{(g, f) : \exists (k, h) \in R : \forall x \in VAR : f(x) = end(h(x)) \wedge g(x) = end(k(x))\}$$

As one can see, we construct a relation between partial functions out of a relation between sequence valued functions by restricting our attention to the current values of the sequence valued functions. The result of such a restriction is in general not a total function since we allow the value $\langle\rangle$ for sequence valued functions.[9]
Now our claim about Φ is the following:

[9]In this context we mean by $f(x) = end(\langle\rangle)$ that $f(x)$ is undefined.

Proposition 2.2.5 *For all $\phi \in DPL$ we have $\Phi([\![\phi]\!]_{sass}) = [\![\phi]\!]_{pgs}$.*

The proof of the proposition is an induction on the complexity of ϕ. It can be found in the appendix.
The relation that Φ establishes does *not* enable us for a given ϕ to construct $[\![\phi]\!]_{sass}$ from $[\![\phi]\!]_{pgs}$. As an example of this we consider the formula $\phi = \exists x \cdot P(x) \cdot \exists x \cdot Q(x)$. If we know $[\![\phi]\!]_{pgs}$, we only can see which values in the domain are such that they have property $I(Q)$ (by checking which values of x can occur as $f(x)$ in a pair $(g, f) \in [\![\phi]\!]_{pgs}$). If we want to have $[\![\phi]\!]_{sass}$, we also have to know which part of the domain has property $I(P)$.
It is possible, however, to define the relation $[\![.]\!]_{sass}$ in terms of the relation $[\![.]\!]_{pgs}$. In other words: if we know $[\![\phi]\!]_{pgs}$ for *all* ϕ, then we can give $[\![\phi]\!]_{sass}$ for all ϕ. As an example again consider $\exists x \cdot P(x) \cdot \exists x \cdot Q(x)$. Knowing $[\![\exists x \cdot P(x) \cdot \exists x \cdot Q(x)]\!]_{gs}$ does not suffice to find $[\![\exists x \cdot P(x) \cdot \exists x \cdot Q(x)]\!]_{sass}$. But if we know *both* $[\![\exists x \cdot P(x)]\!]_{gs}$ and $[\![\exists x \cdot Q(x)]\!]_{gs}$ then we do have enough information to construct $[\![\exists x \cdot P(x) \cdot \exists x \cdot Q(x)]\!]_{sass}$. In general, if we know how ϕ decomposes into conjuncts, then we can construct the sequence valued interpretation of ϕ from the sequence valued interpretation of the conjuncts. This weaker correspondence suffices to guarantee that the notions of validity for $[\![.]\!]_{sass}$, $[\![.]\!]_{pgs}$ and $[\![.]\!]_{gs}$ are equivalent. Therefore we have but one notion of validity, in spite of the subtle differences between the different semantics for DPL. First we give the definitions of entailment for $[\![.]\!]_{gs}$ and $[\![.]\!]_{pgs}$.

Definition 2.2.6
For ϕ, $\psi \in DPL$ we define:

1. $\phi \models_{gs} \psi \;\Leftrightarrow\; \forall f, g \in ASS: \; f[\![\phi]\!]_{gs}g \;\Rightarrow\; \exists h \in ASS: \; g[\![\psi]\!]_{gs}h;$

2. $\phi \models_{pgs} \psi \;\Leftrightarrow\; \forall f, g \in PASS:$

$$f[\![\phi]\!]_{pgs}g \text{ and } g \text{ is defined on the free variables of } \psi \;\Rightarrow\; \exists h \in PASS: \; g[\![\psi]\!]_{pgs}h.$$

All definitions of $\phi \models_x \psi$ are based on the same idea: if we are able to make a ϕ-step, then we are always able to make a ψ-step after that. Now our claim is that also formally all notions of valid inference coincide, i.e. we claim:

Corollary 2.2.7 *For any ϕ, ψ the following clauses are equivalent:*

1. $\phi \models_{sass} \psi$;
2. $\phi \models_{pgs} \psi$;
3. $\phi \models_{gs} \psi$.

The proof can be found in the appendix.
In the proof $2 \Rightarrow 1$ we need more than just $[\![\psi]\!]_{pgs}$to be able to construct the required sequence valued assignment h: we needed all the $[\![\psi_i]\!]_{pgs}$ (where the ψ_i are the formulas of which ψ is the conjunction). This confirms our remark above about the relation between $[\![\psi]\!]_{sass}$ and $[\![\psi]\!]_{pgs}$ not being an isomorphism.
By now we think that we have given sufficient proof of the fact that in our refinement of the relational semantics the Groenendijk and Stokhof semantics is preserved: we can reconstruct the original relations from our refined relational semantics and we have preserved the original notion of entailment.

2.3 Static semantics

2.3.1 Information structures

In this section we will introduce the notion of an information structure. These information structures will provide us with a suitable notion of information state. We will be able to interpret DPL-formulas as information structures and thus give a static semantics to DPL. We will show that this interpretation is compatible with the relational interpretation of the preceding section: we will be able to construct the relational interpretation from the static interpretation and we will see that the information structure that we associate with a formula is just the set of possible outputs of the relation $[\![.]\!]_{sass}$.
At first sight the fact that dynamic DPL allows for a static semantics may seem odd. Does this mean that dynamic semantics is static after all? It might be helpful to think about the situation in terms of *representation* or *coding*. The information structures that we will define can be used to *represent* (or *code*) the relations that we have defined above. Not all relations on $SASS$ will have such a representation, but it turns out that all

DPL-interpretations can be represented by some information structure. We already have a similar situation in the original formulation of the DPL-semantics. There conditions, such as $P(x)$, can be represented by a set, the set of assignments that satisfy the condition. Given this set the relational interpretation of the condition can be constructed. For example, $P(x)$ can be represented by the set $\mathbf{P} = \{f \in ASS : f(x) \in I(P)\}$. Then we can define $[\![P(x)]\!]_{gs} = \{\langle f, g \rangle : f = g \wedge g \in \mathbf{P}\}$. But for other formulas such a representation as a set is not available. With the new definition of the relational semantics and the new notion of information structure a similar reconstruction will be possible for all DPL-interpretations.
If we want to find a suitable notion of information state, then it is a good idea to ask: What is the information revealed by a formula? Surely, a formula gives information about the variables that occur in the formula. We represent this by considering all the values that such a discourse marker can take, i.e. our information structures will contain assignments of values to the variables that occur in the formula. The range of values of a discourse marker is restricted by the conditions that we find in the formula. So we will not find all assignments in the information structure, but only those that satisfy the restrictions that are expressed by the formula.
Some of these variables are introduced explicitly in the formula: they occur in the scope of a quantifier. For these variables a value gets *set* whenever we interpret the formula. But a formula also might say something about variables that it does not introduce itself: a formula might contain free variables. It would be unreasonable to treat these free variables on a par with the quantified variables. The formula does not *instruct* us to assign a value to them, it simply assumes that we have done this already: a formula *asks* for a value on its free variables. The information about the free variables that the formula gives specifies further which value we should have assigned to them. We include this information into our information structures by restricting the assignments according to these conditions. To make sure that we do not lose sight of the specific status that the restrictions on free variables have, we include in our structures components that tell us which variables occur freely and which variables are quantified over in the formula.
If the set of assignments that we have in our information structures really gives information about the variables of the formula, it has to have, in some sense, this set of variables as its "domain". It is not yet clearly defined what it means for an assignment to have that property. To make

the notion precise we give the following definitions.

Definition 2.3.1

1. *With each sequence $S \in VAR^*$ we associate a partial function $\alpha_S : I\!N \to VAR$ as follows:*

   $\alpha_{\langle x\rangle}(0) = x;$

   $\alpha_{\langle x\rangle}(n+1) = undefined\ for\ n \in I\!N;$

   $\alpha_{\langle x\rangle * S}(0) = x;$

   $\alpha_{\langle x\rangle * S}(n+1) = \alpha_S(n)\ for\ n \in I\!N.$

2. *We define the length of a sequence S as follows:*

   $$length(S) = max\{n \in I\!N : \alpha_S(n)\ is\ defined\} + 1.$$

3. *For each $S \in VAR^*$ we define what it means for $f \in SASS$ to be defined on S.*

   f is defined on $\langle x\rangle$ iff $length(f(x)) \geq 1$;

   f is defined on $S * \langle x\rangle$ iff f is defined on S and
   $length(f(x)) > min\{length(g(x)) : g$ is defined on $S\}$.

4. *For a set $V \subseteq VAR$ we write $seq(V)$ for the sequence consisting of of the components of V.*[10]

5. *We define the relation $\leq_l$ on assignments in $SASS$:*

   Let $f, g \in SASS$. We say $f \leq_l g$ iff $\forall x : \exists \sigma_x : \sigma_x * f(x) = g(x)$.

   If $f \leq_l g$, then we say that f is a left restriction of g, or that g is a left extension of f.

[10]Here we use some standard enumeration of VAR to order the elements of V in $seq(V)$. Such an enumeration is available since VAR is countable.

In definition 2.3.1.1 we associate a function defined on an initial segment of $I\!N$ with each sequence. The function α_S gives for $n \in I\!N$ the nth element of the sequence S. The relation of such a function to the sequence is the same as the relation of a characteristic function to a set: they are interchangeable and we only distinguish them notationally to avoid confusion.[11] Note how α_S can be used to give a concise definition of the length of a sequence.

The other clauses are concerned with making clear how we check whether a sequence S is in the domain of a sequence valued function f. Definition 2.3.1.5 makes precise a notion of the *extension* of a sequence valued assignment. If $f \leq_l g$, then $g(x)$ may be longer than $f(x)$, but the extra values of $g(x)$ are added to the left, so the current value of x does not change. We also call such an extension an irrelevant or *left* extension. If g extends f to the left, we have for each x that $f(x)$ and $g(x)$ agree on the relevant values, i.e. the last few values.[12]

As an example, consider a model where $I(P) = \{a\}$ and an assignment f with $f(x) = \langle a \rangle$. Then f satisfies the condition $P(x)$: the current value of x is in $I(P)$. If we extend f to the left to g such that $g(x) = \langle b, c, d, e, b, d, a \rangle$, then for the formula $P(x)$ this extension of the value of x is irrelevant. g is as good as f even though the new values on x are not in $I(P)$. The current value of x still is a, so g also satisfies the condition. We can cut of an irrelevant part of $g(x)$ to get for example $g'(x) = \langle b, d, a \rangle$. Usually we will only be interested in the values of a sequence valued assignment on the variables that actually occur in some formula. For example, in the case of $P(x)$ we are only interested in the current value of x, in the case of $P(x) \cdot \exists x \cdot Q(x)$ we want to know the last two values of x, etc.

Now we are ready for the definition of information structures.

Definition 2.3.2 *$INFO \subseteq \wp(VAR) \times VAR^{\star} \times \wp(SASS)$ such that $(A, S, F) \in INFO$ iff*

[11] Sometimes finite sequences are simply defined to *be* functions defined on initial segments of $I\!N$.

[12] In the appendix we give a lemma that says exactly what we mean when we say that left extensions are irrelevant.

1 $f \in F \Rightarrow f$ *is defined on* $seq(A) * S$
2 $f \in F \wedge f \leq_l g \Rightarrow g \in F$
3 $f \in F \wedge g \leq_l f \wedge g$
is defined on $seq(A) * S \Rightarrow g \in F$
(Here A is for asking, *S is for* setting *and F is for* function.*)*

In an information structure (A, S, F) the assignments in F contain information about A and S. This means first of all that the $f \in F$ have to be defined on both A and S: $seq(A) * S$ is the minimal domain for the $f \in F$ (condition 1). It also means that the values outside this minimal domain should not matter. Hence the $f \in F$ should at least be defined on A and S, but any extension outside this minimal domain should be allowed. Furthermore the values outside the minimal domain should not be restricted: value restrictions code information and we only want information about A and S, not about other variables. Conditions 2 and 3 have this effect: given an $f \in F$ defined on the minimal domain, 2 says that *all* its (left-)extensions also are in F and 3 says that if $g \in F$, then also all restrictions of g are in F, as long as they are defined on $seq(A) * S$. Later on we will see that the sequence S can be used to code up the exact order of all the quantifiers in a formula. These are the variables for which the formula *sets* a value. The set A can then be used to tell us which variables occur freely in the formula. A formula cannot by itself *set* these variables to the right value: it has to take over the value that the context provides. One could say that the formula — and therefore also the information structure — *asks* the context for a value for such a variable.
Given an information structure (A, S, F) it is possible to reconstruct $seq(A) * S$ from F Up to permutation. we just have to find the minimal domain of the $f \in F$. Therefore it is possible, given F and S to find A and given F and A to find S.[13] This suggests that we only need two and not three components in an information structure. But that is not the case: it is impossible to define the domain conditions on the F in an information structure without knowing mentioning all three components. It *is* true that — once the definition is given — the last two of the three components suffice to find the first.
Our notion of information structure is motivated mainly by considering formulas in *DPL*: it is natural that an information state contains some

[13]Again, up to permutation.

set of assignments but the role of s and A can only be motivated by looking at examples from *DPL*. Therefore it might seem rather artificial. To give some more motivation we want to point out the similarity to discourse representation structures (*DRSs*). A *DRS* also consists of a component of discourse markers and a set of assignments that is to express the restrictions on these discourse markers that we find in discourse. The set A that we have in our information structures can be compared to the set of *anchored* discourse markers. In *DRT* these anchors are mainly used for deictic expressions. This is also something that we can use the A-variables for.

For further motivation for the presence of this third component in our information structures we compare our situation with the practice of computer programming. In a program we are not allowed to work with undefined variable names. Nevertheless names can occur in a program that are not declared in the program itself. Usually these names are called constants, but this does not mean that they have never been defined. Instead of being defined in the program, a constant can be said to be defined in the program environment. Such a situation can also occur in discourse theory, where we may want to study a discourse fragment and not the whole of discourse. We then assume some *discourse environment* in which the free variables are defined. Free variables could stand for proper names, for example. We simply assume that proper names have been introduced properly. We would not want to be forced to introduce them in every bit of discourse in which we want to use them; they are declared in the discourse environment.

It is also possible that we are simply not *able* to link the free variables properly to the context. Such a situation arises, for example, if one overhears part of a conversation. Suppose there is a discussion going on between two people about a third person. To these two people it probably is clear who this third person is. Therefore he (or she) will probably not be represented as a free variable in their representation of the discourse. But if we miss the introduction of this discourse marker, overhearing only a part of the conversation, we will be forced to represent this person as a free variable. We will assign all the properties that are assigned to this person to our free variable. But our understanding of the discourse is incomplete until we find out who it was that they were talking about; we only have a partial understanding of that piece of discourse, because our discourse environment is not rich enough. The component A represents this kind of partiality of information.

2.3.2 Static interpretation

Now we will define the interpretation of DPL-formulas as information structures. We will assign an element of *INFO* to each formula ϕ, that we will call (A_ϕ, S_ϕ, F_ϕ).

Definition 2.3.3

1. *For each DPL-formula ϕ, we define the static interpretation of ϕ, $[\phi]$, inductively as follows:*

$$\begin{array}{ll}
[\bot] & = (\emptyset, \langle\rangle, \emptyset) \\
[P(x)] & = (\{x\}, \langle\rangle, \{f : end(f(x)) \in I(P)\}) \\
[\exists x] & = (\emptyset, \langle x\rangle, \{f : \exists g : g\langle\!\langle x\rangle\!\rangle f\}) \\
[\phi \cdot \psi] & = (A_\phi \cup (A_\psi \backslash range(S_\phi)), S_\phi * S_\psi, \\
 & \{f \in F_\psi : \exists g \in F_\phi : g\langle\!\langle S_\psi\rangle\!\rangle f\}) \\
[(\phi \rightarrow \psi)] & = (A_\phi \cup (A_\psi \backslash range(S_\phi)), \langle\rangle, \\
 & \{f : \forall g \in F_\phi : f\langle\!\langle S_\phi\rangle\!\rangle g \rightarrow \exists h \in F_\psi g\langle\!\langle S_\psi\rangle\!\rangle h\})
\end{array}$$

2. *We define entailment as follows:*

$$\phi \models_{stat} \psi \;\Leftrightarrow \forall g \in F_\phi \; \exists h \in F_\psi : g\langle\!\langle S_\psi\rangle\!\rangle h$$

The first component of $[\phi]$ is the set of the free variables in ϕ.[14] In the second component of $[\phi]$ we build up the sequence of the quantified variables. That allows us to keep track of the order in which variables are (re)introduced.[15]

Example:

$$\begin{array}{lcl}
[P(z) \cdot \exists x] & = & (\{z\}, \langle x\rangle, \ldots) \\
[Q(x,y) \cdot \exists x \cdot R(y,x)] & = & (\{x,y\}, \langle x\rangle, \ldots) \\
[P(z) \cdot \exists x \cdot Q(x,y) \cdot \exists x \cdot R(y,x)] & = & (\{z\} \cup (\{x,y\} \backslash \{x\}), \langle x\rangle * \langle x\rangle, \ldots) \\
 & = & (\{z,y\}, \langle x,x\rangle, \ldots)
\end{array}$$

[14] We have compared the variables in A with anchors, but while free variables can get bound in larger contexts, anchors will always remain anchors. So it would be better to consider the A-variables as *temporary* anchors.

[15] Note that here $[\exists x \cdot P(x) \cdot \exists y \cdot P(y)] \neq [\exists y \cdot P(y) \cdot \exists x \cdot P(x)]$. If this becomes a problem, we can try to use multisets instead of sequences in the second component of information structures. Then these two information structures become equal.

Here 3 is the conjunction of 1 and 2. We see that the x that is free in 2, is not free in the conjunction; it is bound by the $\exists x$ in 1. Therefore is has to be removed from the set of free variables. Note that a variable can be free only once, while it can occur many times in the sequence of the quantified variables.
The assignments in the third component will all be defined on the free and the quantified variables. It is also easy to check that the third components of the interpretations satisfy the other conditions in the definition of $INFO$. This means that indeed $[\phi] \in INFO$ for any $\phi \in DPL$.
There is an obvious correspondence between this static semantics and the refined relational semantics of the preceding section:

Proposition 2.3.4 *Let* $[\phi] = (A_\phi, S_\phi, F_\phi)$. *Then:*

1. $f[\![\phi]\!]_{sass} g \;\Leftrightarrow\; g \in F_\phi \wedge f\langle\!\langle S_\phi \rangle\!\rangle g.$

2. $\phi \models_{stat} \psi \;\Leftrightarrow \phi \models_{sass} \psi.$

□

(The proof is omitted.) If we had used a version of $[\![.]\!]_{sass}$ that has three components (cf. p. 57), this would mean that: $[\phi] = range([\![\phi]\!]_{sass})$ (or $\uparrow[\![\phi]\!]_{sass}$ in the notation of Groenendijk and Stokhof (1991b)). The proposition has the following corollary:

Corollary 2.3.5 *For any* ϕ, ψ *the following clauses are equivalent:*

1. $\phi \models_{[[} \psi;$

2. $\phi \models_{[} \psi;$

3. $\phi \models_{pgs} \psi;$

4. $\phi \models_{gs} \psi;$

Proposition 2.3.4 and corollary 2.3.5 guarantee that our interpretation of formulas in $INFO$ preserves the spirit of dynamic semantics. This means that a relational formulation is not essential for the dynamic semantics of DPL. This may seem surprising at first, but maybe not if we recall that already in DRT we have a kind of static semantics that covers almost the same evidence as DPL. Also recall our discussion in 2.3.1 where it is pointed out that for certain extensions of dynamic semantics a relational formulation could be essential (cf. section 2.5).

At this point we basically have done what we set out to do: we have given a natural refinement of the original *DPL*-semantics in order to solve the eliminativity problem. We have also checked that this semantics indeed preserves the *DPL*-interpretations. The only thing that remains to be done is checking that this semantics is indeed eliminative. This will be done shortly, in subsection 2.4.3.
The rest of the chapter is devoted to the discussion of some other issues concerning information content and *DPL* semantics. This way we get a chance to see the improved notion of information state at work.

2.4 Topics in dynamic semantics

2.4.1 The general point

Now that we have an improved notion of information, we want to discuss some topics in dynamic semantics in terms of it. One of the issues we will discuss is monotonicity. Remember that we said (section 2.2.2) that probably the defect of the original formulation of DPL was that we were sometimes forced to forget the value of a variable. If we consider $[\![P(x) \cdot \exists x \cdot Q(x) \cdot \exists x \cdot R(x)]\!]_{gs}$, we will only find the p such that $p \in I(P)$ and the $r \in I(R)$, but no $q \in I(Q)$. This means that if there is any $q \in I(Q)$, we would have $[\![P(x) \cdot \exists x \cdot R(x)]\!]_{gs} = [\![P(x) \cdot \exists x \cdot Q(x) \cdot \exists x \cdot R(x)]\!]_{gs}$. This makes it impossible to have an intuitively acceptable notion of information content based on these relations. Indeed, it makes it impossible to answer questions about information by looking at the semantics.
In our set up we have created a new kind of information states. Now it will be possible to define an ordering of the semantic objects, information structures, that corresponds to our basic intuitions about the informativity of the formulas. We will define this ordering and show how it works. This is the ordering along which we will check the monotonicity of our semantics. Then we will try to use our ordering for the study of dynamic inference. This will cause some problems that suggest that our ordering is not "the unique right one", at least not for all purposes.

2.4.2 Ordering information structures

We will introduce an ordering of information structures, that will be motivated by the intuition that *more discourse contains more information.* We will discuss the basic properties of this ordering. There are basically two ways in which we can give more information in discourse. First we can say more about the objects that we already were talking about. We would then add restrictions on discourse markers, and as a consequence eliminate some assignments. The other way in which we can add information, is by introducing a new object in discourse.
It is difficult to find a piece of discourse that does just this. One could think of the indefinite article 'a',[16] but larger discourse fragments that only *introduce* and do not *restrict* are hard to find. Usually as soon as we introduce some object we say something about it as well. But maybe we can illustrate the two kinds of information be contrasting the following two sentences.

Example:

1. There is a unicorn.

2. There is a man.

The first sentence clearly is informative in both senses. An object is introduced for us to talk about and a very interesting claim about the object is made. From this sentence we can infer that unicorns exist. Probably this is what the speaker wants to say with this sentence. From the second sentence we can infer that men exist. But probably this will not be what the speaker is trying to tell us in this sentence. Here this claim is merely a side effect. The main goal of the speaker is to introduce an topic that he wants to talk about. The first claim about it is already made — that it is a man — but, surely, more will follow. Such introductory acts are essential in the chain of information exchange.
In other words, although both sentences are informative in both ways, the second sentence *mainly* serves to introduce the object for further discussion. The first sentence in addition makes a remarkable claim about the

[16]Even for 'a' the situation is not so simple. Sometimes 'a' does more then just introduce an object. Think, for example, of generic uses of the indefinite article.

object it introduces. Other, more artificial examples can be found in in mathematics. In proving a theorem, in arithmetic for example, a phrase such as

Let n be given.

is often used. Clearly the purpose of this phrase is just to make it possible to talk about some number.
In DPL-syntax these two ways of giving information correspond, roughly, to two kinds of formulas: the first kind of information is typically represented by DPL-conditions, and the second kind goes together with the DPL-quantifier. We should be careful with this correspondence: when we find a formula $\exists x$ in DPL, we are not sure that this x will really stand for a new object, it might just be another name for an old object (cf. section 2.4.4). Nevertheless, it is the formula $\exists x$ that makes it possible to talk about new objects and therefore it seems reasonable to say that $\exists x$ contains positive information (in this sense). In our semantics the two ways of giving information can be represented as follows:

Definition 2.4.1
We define an ordering $\geq$ on $INFO$. Let (A, S, F), $(A', S', F') \in INFO$ be given. Then $(A, S, F) \geq (A', S', F')$ iff

1. $S' = S * S''$ *for some* S'';
2. $A' \subseteq A$;
3. $\forall f' \in F' : \exists f \in F : f\langle\langle S''\rangle\rangle f'$.

We have defined $\geq$ in such a way that the smaller information structures are more informative. In this definition 1 represents (part of) the second idea about increase of information: the more informative state defines all the discourse markers that are defined in the larger state and maybe some more.
If we have a situation in which $S'' = \langle\rangle$, the third clause reads as $F' \subseteq F$. This is the easiest way to see that the first idea is also reflected by the definition: if no more discourse markers are introduced, giving more information simply means eliminating some assignments. What 3 actually says is that every assignment in F' should be an extension (to the right, so *not* irrelevant!) of an assignment in F. This corresponds to

a more refined notion of the elimination of assignments: we do not want to say that an assignment is eliminated if it has "grown". An assignment is not eliminated if it gets a larger domain: it can only be said to be eliminated if it does not reoccur at all, not even with a larger domain. If we want to consider assignments as possibilities, it is this refined notion of elimination that corresponds to the idea of eliminating a possibility. The sequence valued assignments are like possible histories of information. Just as history will not have to be revised because of developments in the future, assignments are not eliminated if they become "longer". Now we can see that 3 corresponds to this notion of elimination of possibilities: every $f' \in F'$ is an extension of an $f \in F$, but some assignments $f \in F$ might not reoccur in F', they are eliminated. Note that 2 usually follows from 1 and 3; the only exception is $F = F' = \emptyset$. (We have $(A, S, \emptyset) \in INFO$ for all A, S.) It would be interesting to understand what this exception means. It seems to say that when a contradiction arises, i.e. $F = \emptyset$, we have to make a choice: either we say that all contradictions contain the same information or some give different information from others. Here we have chosen the last option, not for any intuitive reason but because it makes the formalism easier to handle.
Before we show what this way of ordering information structures means for our DPL-interpretations, we give some abstract properties of the ordering.

Proposition 2.4.2 $\geq$ *is a partial order, i.e.*

1. $(A, S, F) \geq (A', S', F') \wedge (A', S', F') \geq (A'', S'', F'')$
   $\Rightarrow (A, S, F) \geq (A'', S'', F'')$
2. $(A, S, F) \geq (A', S', F') \wedge (A', S', F') \geq (A, S, F)$
   $\Leftrightarrow (A', S', F') = (A, S, F)$

Proof:

2 '$\Leftarrow$' is obvious. '$\Rightarrow$': Clearly the antecedent implies $S = S'$ and $A = A'$. Therefore we find both $F \subseteq F'$ and $F' \subseteq F$, i.e. $F = F'$.

1 $S'' = S' * S^\star$ for some $S^\star$, and $S' = S * S^\circ$ for some S°. So $S'' = S * S^\circ * S^\star$. Also, if $f'' \in F''$, we have $f' \in F'$ such that $f' \langle\langle S^\star \rangle\rangle f''$, and for this f' we have $f \in F$ such that $f \langle\langle S^\circ \rangle\rangle F'$, and therefore $f \langle\langle S^\circ * S^\star \rangle\rangle f''$.□

2.4.3 Monotonicity

In this section we show that our ordering of information structures works fine for the easiest way of giving more information in discourse: we show that larger discourse fragments are more (or: not less) informative in our sequence based interpretation. We call this property of our semantics monotonicity. As was explained in the introduction it is our view that any good semantics for ordinary, narrative discourse should have this property. It is simply true that we do not lose information if we continue our story.[17] And it is this kind of discourse that we want to represent in DPL. So we want the following property for our semantics:

Proposition 2.4.3 *Let $\phi, \psi \in DPL$ be given. Then we have $[\phi] \geq [\phi \cdot \psi]$.*

Proof:
$[\phi.\psi] = (A_\phi \cup (A_\psi \backslash range(S_\phi)), S_\phi * S_\psi, \{f \in F_\psi : \exists g \in F_\phi : g\langle\!\langle S_\psi \rangle\!\rangle f\})$,
$[\phi] = (A_\phi, S_\phi, F_\phi)$. Now the proposition holds by definition of $\geq$.
$\square$

At this point we can see in which way our semantics is an improvement of the usual relational semantics for DPL. By using sequence valued assignments, we have enriched the semantics in a natural way. As a result we get the possibility of a systematic discussion of information in DPL-semantics. We now have semantic objects that are rich enough to make such an approach possible. The definition of information structures and our ordering of these structures are examples of these new possibilities. They have enabled us to see that DPL-semantics is monotone. It is not claimed here that these two notions are all we will ever need, but they show what kinds of things are possible in this richer environment, using sequence valued assignments.

2.4.4 Inference

Dynamic inference in general

Inference is a notoriously difficult topic in dynamic semantics. Different branches of dynamic semantics have given rise to different notions of in-

[17]Of course, this does not mean that we *never* forget information, but forgetting is not the result of ordinary narrative discourse.

ference and some have even produced more than one. The DPL-notion of inference is defined as follows:

$$\phi \models \psi \Leftrightarrow \forall f, g : f[\![\phi]\!]_{gs} g \rightarrow \exists h : g[\![\psi]\!]_{gs} h.$$

We have seen that this notion of inference is preserved in all the reformulations of DPL-semantics that we have considered.
Update semantics has produced its own notions of inference. One of them reads as follows[18]:

$$\phi \models_v \psi \Leftrightarrow \forall \sigma : \sigma(\![\phi]\!)_v \subseteq \sigma(\![\phi.\psi]\!)_v$$

Both notions of inference make sense in the context in which they arise. So it seems that dynamic semantics in general does not have *one* notion of inference. Instead we find a whole spread of inference relations. Of course, this gives rise to the question what the common features of these relations are. Or — to put it ironically — is there any relation that is not a dynamic inference relation?
This turns out to be a surprisingly difficult question. More about this issue can be found in Van Benthem (1991). But here we do not aim at solving this general problem. Instead we restrict ourselves to the DPL-notion of inference. Note that the fact that this notion of inference has survived the reformulations in this chapter, is evidence that it is indeed at the core of DPL, even if in the more general picture it is just one of a number of candidates.
We are especially interested in the way in which the inference relation fits into our algebra of information structures. We know that for all sorts of formal systems there is a nice fit between the algebraic semantics and the inference relation. For propositional logic $\models$ coincides with the ordering in Boolean (or, in the intuitionistic case, Heyting) algebra. For predicate logic and modal (propositional) logic it is the inclusion relation on satisfaction sets. For linear logic the situation is less straightforward: here $\models$ can be defined in terms of $\otimes$, the tensor product of linear logic.[19]
Now what kind of relation holds between the DPL-inference relation and the ordering that we have defined on $INFO$?

[18]Notation as in introduction: v stands for Veltman.

[19]Cf. Troelstra (1992). A similar situation exists in Pratt's action logic (Pratt (1991)).

Inference and information

The idea that there might be a connection between the inference relation of DPL and the ordering on $INFO$, is not just inspired by the fact that this is so for other formal systems. There also is a clear intuition that inference and information are related concepts. It seems that if we can *infer* ψ from ϕ, this must be because ϕ gives all the *information* that we need to conclude that ψ. And, conversely, if ψ contains no more *information* than ψ, then, surely, we should be able to *infer* ψ from ϕ.
In this chapter we have been concerned with the information contained in DPL-formulas from a different perspective. We have been trying to model the idea that "more discourse contains more information". This has given rise to a notion of information structure. Of course it is our hope that this notion of information will also allow us to say something about inference.
There are different relations between $\models$ and $\leq$ that we could discuss. We will start with the most obvious one and we will consider some other options as we proceed. The first guess is that the inference relation simply *is* the $\leq$-relation on information structures:

$$\phi \models \psi \;\Leftrightarrow\; [\phi] \leq [\psi].$$

One example which supports this is:

$$\exists x.P(x) \models P(x).$$

But unfortunately the first counterexample is not far away. If we turn around the $\models$-sign in the above example we get a case where $\phi \models \psi$ but *not* $[\phi] \leq [\psi]$.
A somewhat weaker relation, which is still possible in view of these examples, is:

$$\phi \models \psi \;\Leftarrow\; [\phi] \leq [\psi].$$

For, it is not the case that $[P(x)] \leq [\exists x.P(x)]$, so the counterexample that we had, is not a counterexample for this weaker correspondence. This weaker correspondence would suggest that our ordering on $INFO$ is too strong: if $[\phi] \leq [\psi]$, then $\phi \models \psi$, but even if *not* $[\phi] \leq [\psi]$, $\phi \models \psi$ can still hold.
But here there also is an easy counterexample: we can have ϕ and ψ such that $[\phi] \leq [\psi]$ but *not* $\phi \models \psi$, as the following example shows.

$[P(x) \cdot \exists x \cdot \neg(P(x))] \leq [P(x)]$, but not: $P(x) \cdot \exists x \cdot \neg(P(x)) \models P(x)$.

That the first relation holds is true in view of the monotonicity result that we established in the previous section: more discourse contains more information. But it is also clear that *not* $P(x) \cdot \exists x \cdot \neg(P(x)) \models P(x)$, since this would imply that the same instance of x would both have property P and not have property P.
We can see what goes wrong in this example: in $\phi \models \psi$ the binding between the antecedent ϕ and the consequent ψ makes that we are talking about the same x both having and not having property P. But the x in the antecedent that has property P is not the same as the x that has property *not* P.
We could try to prevent this kind of anaphoric confusion as follows. Instead of comparing $[\phi]$ and $[\psi]$, we compare $[\phi]$ and $[\phi.\psi]$. If we do this, we can take into account the bindings between ϕ and ψ already if we are looking at $\leq$. So this should help to prevent the unpleasant surprises that these bindings, that are essential for the $\models$-relation, cause in cases as the above. Therefore our next guess is that:

$$\phi \models \psi \;\Leftrightarrow\; [\phi] \leq [\phi \cdot \psi].$$

Note that this guess reflects the same intuition about the relation between information and inference: if $\phi \models \psi$, then what we learn from $\phi.\psi$ is no more than what we learn from ϕ.[20] In other words: given ϕ, ψ contains no new information. Also note that for most of the systems that we mentioned above the two guesses coincide formally. For example, for propositional logic and Boolean algebras we have that for two propositions P_1 and P_2, $[P_1] \leq [P_2]$ iff $[P_1 \wedge P_2] \geq [P_1]$. It might be the case that in our situation this formulation with the conjunction is simply more suitable, since in DPL anaphoric bindings are so important.
In the new situation the counterexample that we had no longer works. For now $[\phi \cdot \psi] = [P(x) \cdot \exists x \cdot \neg(P(x)) \cdot P(x)]$ has to be compared with $[\phi] = [P(x) \cdot \exists x \cdot \neg(P(x))]$. We see that not $[\phi] \leq [\psi]$. Therefore we would not expect $\phi \models \psi$ in the first place. But again there is a counterexample. The counterexample is embarrassingly simple — and in fact it also defeats our earlier guesses — but also very instructive:

[20] Also note the similarity with Veltman's notion of entailment.

$\exists x \cdot P(x) \models \exists y \cdot P(y)$.[21]

Here we see that *not* $[\phi] \leq [\phi \cdot \psi]$, simply because $\phi \cdot \psi$ says something about more variables than ϕ alone. Since the introduction of new variables counts as an increase of information according to $\leq$, we do not find $[\exists x \cdot P(x) \exists y \cdot P(y)] \geq [\exists x \cdot P(x)]$.
Here we see the main problem for the comparison of $\models$ and $\leq$. $\leq$ is based on the idea that as a rule new variables are introduced to give new information. But the variables that are introduced in the consequent of $\models$ typically are *not* introduced for this purpose. They are there to make claims about old information.[22] If we say $\exists x \cdot P(x) \models \exists y \cdot P(y)$, than we do not mean that some unknown object y has the property P, but we claim that when we know $\exists x \cdot P(x)$, we already know an object with property P.
It seems that we are at a dead end: in order to model the idea that more discourse contains more information, we had to count the introduction of variables as informative acts. This was an important motivation for our definition of $\leq$. We have seen that in the context of inference we typically consider discourse that is not supposed to contain new information, but nevertheless can contain new variables. These are conflicting requirements on the ordering of information.

Multi-dimensional information algebras

The conflict that we have seen seems irreparable. In DPL information is given by conditions on variables. These conditions can be represented as restrictions of the values of the variables. Any sensible notion of information state for DPL should show the variables that the information is about as well as the restrictions on these variables that embody the information. So the problem that we have sketched will arise for any sensible notion of information structure that one might come up with. Always the same question will arise: do more variables mean more information or not? And always the answer will be both yes and no.
Therefore our conclusion must be that there is more than one way to look at the information of a DPL-formula. These different ways give rise to

[21]Note that also $\exists x \cdot P(x) \models \exists x \cdot P(x)$ is a counterexample. So the choice of the variable in the consequent is not important.
[22]The same holds for the variables in the consequent of an implication.

different orderings of the information structures. We have already seen two perspectives on information that give rise to two different orderings of information structure. One is the perspective where we consider ϕ and $\phi \cdot \psi$ and ask ourselves which of the two we would prefer to hear. Of course we would choose $\phi \cdot \psi$, since *it contains more information* than ϕ alone.

In the other perspective we imagine that we are in a situation that we have heard ϕ and we wonder whether we still need to hear ψ. Of course we only want hear ψ, if given ϕ *it contains new information.* This situation, in which we consider ϕ and ψ in a specific order and not as unordered alternatives, gives rise to a different ordering of information structures. We call this the *diachronic* information order. This is the ordering that should correspond directly to $\models$. The situation where we can choose between ϕ now or $\phi.\psi$ now gives rise to a *synchronic* ordering of information structures. This is the situation that we have considered in section 2.4.2.[23]

We can illustrate the difference between these two ways of looking at information with an example from *DPL*. Consider the formula $\phi \equiv \neg P(x) \cdot \exists x \cdot P(x)$. This is an example of a *DPL*-formula that does not entail itself. Therefore it will behave funnily in a diachronic information ordering. But in a synchronic information ordering it will not behave funnily: synchronically each formula is of course as informative as itself.

We can think of these different orderings as the dimensions of the information algebra. So the conclusion is that we have to work in a multi-dimensional information algebra. Of course this cannot be the final word about information orderings. Just as with the study of dynamic inference the fact that there are several sensible information orderings gives rise to further questions. We would like to know what kind of relations count as information orderings, i.e. how many dimensions there are in our information algebra. Is there any relation on information structures that is not an information ordering?

At the moment it seems to us that there is a feature that all ordering relations should have in common. Let's assume that our information structures contain both a set of variables[24] — the variables that the in-

[23]The terminology is taken from Visser (1992a), who applies the distinction in a slightly different context.

[24]Strictly speaking we should say "a set of variable instances", for we will have to distinguish different occurrences of variables (just as we do in sequence semantics), according to the different roles one variable can play.

formation is about — and a set of functions embodying the restrictions on the values of the variables. This is not only true for our information structures but also — as we have argued — for any reasonable alternative. Comparing the information contained in these information structures always amounts to comparing the restrictions on variables that we find in the different information structures. If for every variable in one structure, σ say, we can find a variable in the other structure, σ', that is at least as severely restricted, then we are inclined to say that σ' contains *more* information than σ.

This could be tested with a mapping from the variables in σ' to the variables in σ. If we can find a mapping such that the restrictions on the images of the variables are at least as severe as the restrictions in σ, then we would say that σ' is more informative than σ. The severity of the restrictions can, of course, be compared by looking at the assignments. We intend to develop this general idea about the ordering of information states elsewhere. Here we just check how the two information orderings that we have seen relate to this general idea.

We find that both the synchronic and the diachronic information ordering embody this idea. But both notions have some extra conditions on the variables that we are allowed to compare, conditions on the mapping from σ' to σ as it were. If we test whether $(A, S, F) \leq (A', S', F')$, we check whether for any $f' \in F'$ there is an $f \in F$ that has the same values for the variables in S' as f itself. Here this comparison of variables in different information structures is effectuated by the condition $f\langle\!\langle S'' \rangle\!\rangle f'$ (where S'' is such that $S = S' * S''$). This condition tells us which variable occurrences have to be compared. For example, if $(A, S, F) = (\emptyset, \langle x, x\rangle, F)$ and $(A', S', F') = (\emptyset, \langle x\rangle, F')$, then we will compare the last value of $f'(x)$, not with the last value of $f(x)$, but with the value before last: $f(x) = f'(x) * \langle d\rangle$ for some d.

For $\models$ the relation with our general information ordering is less straightforward. The relation between entailment and the general ordering can be made precise, but the technical details would take us beyond the scope of this chapter. Suffice it to make the connection intuitively clear.

In checking intuitively that something like $\exists x.P(x).\exists y.Q(y).R(y) \models \exists z.Q(z)$ holds, we try to find (for z) in the antecedent a variable that satisfies at least the condition Q. Here this variable is y. There are no restrictions on the variable that we can choose: any variable will do as long as it satisfies Q.

If we have a free variable in the conclusion (as in $\exists x \cdot P(x) \models P(x)$), then the situation is different. This time we do not have a free choice at all. A free variable in the conclusion can be bound by a variable in the antecedent. If this is the case, then this is the variable we should choose. So in the example we can only compare the x in the conclusion with the x in the antecedent. If the variable is not bound by the antecedent (as in $\models (P(x) \rightarrow P(x))$), then the conclusion has to hold for all values of x. So we can see that also for $\models$ we have to compare variables, taking into account some restrictions.
The conclusion of this section is that there is no straightforward algebraic relation between $\leq$ and $\models$. In this respect DPL-semantics is less well behaved then the formal systems we mentioned above. But the orderings on information structures that $\leq$ and $\models$ lead to, seem to be instances of one general scheme for the comparison of information. This general information order will get more attention elsewhere.

2.5 Update semantics

2.5.1 Update semantics

In the preceding sections we have given interpretation of DPL in terms of assignments that have sequences as values. We have checked that our semantics is faithful to the original DPL-semantics (in section 2.3) and we have seen that we can give both a relational and a static formulation of our semantics. In this section we formulate sequence semantics as update semantics. We will see that such a formulation is available. Then we will discuss the issue of eliminativity again. We have addressed this issue already in terms of the static semantics (section 2.4.3), but if we formulate the notion of monotonicity in update style, this will make the comparison with the original discussion by Groenendijk and Stokhof (1991b) more straightforward.
First we define an operation on information states, that we call the *merger*.

Definition 2.5.1 *We define the merger of information structures,* $\bullet$: $INFO \rightarrow INFO$, *as follows:*
$(A, S, F) \bullet (A', S', F') =$
$$(A \cup (A' \backslash (range(\alpha_S)), S * S', \{f \in F' : \exists g \in F : g\langle\langle S'\rangle\rangle f\}.$$

We use this operation to define the interpretations of formulas as update functions on information structures.

Definition 2.5.2 *For a DPL-formula ϕ we define the update function $(\!\lbrack\phi\rbrack\!) : INFO \rightarrow INFO$ as follows:*[25]

$$(A, S, F)(\!\lbrack\phi\rbrack\!) \;=\; (A, S, F) \bullet [\phi]$$

Here we have defined $(\!\lbrack\phi\rbrack\!)$ in terms of $[\phi]$, but it is an easy exercise to show that we can also define $(\!\lbrack\phi\rbrack\!)$ directly in such a way that the defining property holds. Note that the update functions make it possible to build up the static interpretation: for example, if $(A, S, F) = [\psi]$, then $(A, S, F,)(\!\lbrack\phi\rbrack\!) = [\phi \cdot \psi]$. This explains the notation for the merger as $\bullet$: it is the semantic analogue of $'\cdot'$. Now we can find the interpretation of a conjunction by updating the state of no information, $(\emptyset, \langle\rangle, SASS)$, step by step. In other words:

Proposition 2.5.3 *Let $\phi_0, \ldots, \phi_n \in DPL$ be given. Then we have:*

$$[\phi_0 \cdot \;\ldots\; \cdot \phi_n] \;=\; (\ldots((\emptyset, \langle\rangle, SASS)(\!\lbrack\phi_0\rbrack\!))\ldots)(\!\lbrack\phi_n\rbrack\!).$$

We can also give a definition of inference in terms of update functions.

Definition 2.5.4
Let $(\emptyset, \langle\rangle, SASS)(\!\lbrack\phi\rbrack\!) \;=\; (A, S, F)$ and $(\emptyset, \langle\rangle, SASS)(\!\lbrack\psi\rbrack\!) \;=\; (A', S', F')$. Then we define:

$$\phi \models_{up} \psi \;\Leftrightarrow\; \forall f \in F : \exists g \in F' : f\langle\!\langle S'\rangle\!\rangle g.$$

Because of the close relationship with the static interpretation that we have established in proposition 2.5.3, it can be checked easily that this notion of inference coincides with the one(s) discussed before.

Corollary 2.5.5 $\phi \models_{up} \psi \;\Leftrightarrow\; \phi \models \psi$.

Now we can formulate the monotonicity property in terms of update functions.

[25]Recall that we use postfix notation for update functions.

Proposition 2.5.6 *Let $\phi \in DPL$ be given. Then $(\!|\phi|\!)$ is monotone decreasing, i.e.:*

$$(A, S, F) \geq (A, S, F)(\!|\phi|\!) \text{ for all } (A, S, F) \in INFO.$$

The proof of the proposition again relies on the correspondence between the static and the update interpretation. Because of this correspondence the result simply follows from the monotonicity result for the static interpretation (section 2.4.3).
We see that for the improved notion of information state the *DPL*-updates are monotone, or — in the terminology of Groenendijk and Stokhof (1991b) — eliminative. Now that we have discovered this improved notion of monotonicity, we can check what this property amounts to in terms of the relational semantics, i.e. we can reformulate $\geq$ in terms of the original relational formulation of *DPL*-semantics. We find, by a careful reconstruction, the following reformulation of monotonicity:

Proposition 2.5.7 *Let $\sigma \in ASS$, $\phi, \psi \in DPL$ be given. Then:*[26]

Monotonicity $\forall f \in \sigma(\!|\phi \cdot \psi|\!)_{gs} \exists g \in \sigma(\!|\phi|\!)_{gs}$ *such that* $g\langle\!\langle range(\alpha_{S_\psi})\rangle\!\rangle f$.

Truth property $\sigma(\!|\phi \cdot \psi|\!)_{gs} \neq \emptyset \Rightarrow \sigma(\!|\phi|\!)_{gs} \neq \emptyset$.

Hence the truth of $\phi \cdot \psi$ implies the truth of ϕ.

Proof: [Monotonicity] From the monotonicity property for our update semantics we learn that $\sigma \bullet [\phi] \bullet [\psi] \leq \sigma \bullet [\phi]$. For $\sigma = (\emptyset, \langle\rangle, SASS)$, this means that all f' in the static interpretation of $\phi.\psi$ are extensions of some g' in the static interpretation of ϕ. Because of the relation between the sequence semantics and the original relational semantics we see that this means that for any $f \in range([\![\phi.\psi]\!]_{gs})$ there is a $g \in range([\![\phi]\!]_{gs})$ that differs from f only on the variables occurring in ψ. This proves the proposition for $\sigma = (\emptyset, \langle\rangle, SASS)$. The proof of the general case is completely analogous.
[Truth property] Follows immediately from the Monotonicity property.□

[26]Here we abuse the notation $\langle\!\langle . \rangle\!\rangle$: we use it for a set instead of a sequence. Of course we mean the usual notion of resetting a function here, where $f\langle\!\langle X\rangle\!\rangle g$ allows us to reset all the values of the variables in the set X.

Of course, this does not look anything like the eliminativity constraint that Groenendijk and Stokhof considered. Since they were not careful in the choice of their notion of information state, their notion of eliminativity is inappropriate as well: it does not correspond to the notion of information growth. Since for Veltman's system this is exactly what the eliminativity property is about, the resulting comparison of DPL and Veltman's update semantics is confused. Now we are in a position to clear up the confusion and we find the monotonicity property that we should expect for DPL.

2.5.2 Might

Veltman does not introduce update semantics just as a nice way to present dynamic semantics. He has some substantial applications in mind (see Veltman (1991)) in the dynamic semantics of modalities. The simplest system that Veltman applies the techniques of update semantics to is propositional logic with an operator $\Diamond$, *might*. The update semantics enables us to give a dynamic interpretation to *might*. What we mean by this is shown by the following example:

Example:

1. It might be raining. ... It is not raining.
2. It is not raining. ... It might be raining.

The first sentence says that we first think that it might be raining and later find out that it is not raining. This is all right. But in 2 we still think that it might rain after we have found out that it is not raining. That is nonsense. This example shows that a dynamic treatment of *might* is needed. Only a dynamic *might* could explain why the first sentence seems acceptable, and the second not. Veltman has succeeded in giving an elegant semantics for propositional logic with $\Diamond$ that deals with this phenomenon.
In this section we will see whether an easy extension of our system with a dynamic kind of modality is available. We will show that it is possible to define a dynamic might operator in our system. The semantics of this operator, $\Diamond$, cannot be given "pointwise". We mean that it is not possible

to compute the effect of $\Diamond\phi$ in some complex state, by first computing its effect on the atomic substates and then simply adding the results to find the effect on the complex state. This is in contrast with what we have seen so far: for our DPL-updates we have:

Proposition 2.5.8 *Update functions for DPL-formulas are "pointwise":* $(A, S, F)(\!\llbracket\phi\rrbracket\!) = (A', S', \bigcup\{G_f : f \in F\})$, *where for all* $f \in F$ $(A, S, \{f\})(\!\llbracket\phi\rrbracket\!) = (A', S', G_f)$.

The fact that such a result cannot be obtained for the semantics of $\Diamond$ is not a defect of our semantics: it is an essential property of the meaning of $\Diamond$. $\Diamond\phi$ induces a test on our current state of information: the test succeeds if ϕ is compatible with our information. Then it leaves the state of information unchanged. If ϕ is incompatible with what we already know, the test fails. Then the result $\Diamond\phi$ is total confusion: the information of $\Diamond\phi$ gives a contradiction.
If we try to perform such a test bit by bit, we will (possibly) throw away some information, since it is incompatible with ϕ, while we leave other bits of information intact. Then, if we add up the resulting bits of information, we could only retrieve some of the information that we started with. This is in contradiction with the test character of $\Diamond\phi$. Hence a pointwise approach to the semantics of *might* does not stand a chance.
We can define the concept of acceptability or compatibility that is associated with $\Diamond$, as follows:

Definition 2.5.9
ϕ is acceptable in (A, S, F) iff $\exists f \in F : \exists g \in F_\phi : f \langle\!\langle S_\phi \rangle\!\rangle g$.

Remember that we defined :

$$\phi \textit{ is valid in } (A, S, F) \textit{ iff } \forall f \in F : \exists g \in F_\phi : f \langle\!\langle S_\phi \rangle\!\rangle g.$$

So, while validity says that ϕ should hold in all possible cases, acceptability says that ϕ should hold in at least one possible case. In this context we can think of the set F as the set of possibilities (or possible information histories).
Now we can explain the meaning of $\Diamond\phi$ as follows: checking whether $\Diamond\phi$ holds in a situation (A, S, F) means checking whether ϕ is acceptable in (A, S, F). So we define:

Definition 2.5.10 *For each ϕ we define $[\![\Diamond\phi]\!]$ as follows:*[27]

$$(A,S,F)[\![\Diamond\phi]\!] \;=\; (A,S,F) \text{ if } \phi \text{ is acceptable in } (A,S,F);$$

$$(A,S,F)[\![\Diamond\phi]\!] \;=\; (A,S,\emptyset) \text{ else.}$$

It can happen that for some formula ϕ, $(\Diamond\phi).\neg(\phi)$ is acceptable, while $\neg(\phi).(\Diamond\phi)$ is not acceptable, just as in the example above. Consider, for example, the formula $P(x)$. We find that, in a model where there is some $p \in I(P)$, while not $I(P) = DOM$:

$$(\emptyset, \langle x\rangle, \{f \in SASS : \mathit{length}(f(x)) \geq 1\})[\![\Diamond P(x)]\!][\![\neg P(x)]\!] \;=$$

$$(\emptyset, \langle x\rangle, \{f \in SASS : \mathit{length}(f(x)) \geq 1\})[\![\neg P(x)]\!] \;=$$

$$(\emptyset, \langle x\rangle, \{f \in SASS : \mathit{length}(f(x)) \geq 1 \;\wedge\; \mathit{not}: \mathit{end}(f(x)) \in I(P)\}),$$

but also:

$$(\emptyset, \langle x\rangle, \{f \in SASS : \mathit{length}(f(x)) \geq 1\})[\![\neg(P(x))]\!][\![\Diamond P(x)]\!] \;=$$

$$(\emptyset, \langle x\rangle, \{f \in SASS : \mathit{length}(f(x)) \geq 1 \wedge$$
$$\mathit{not}: \mathit{end}(f(x)) \in I(P)\})[\![\Diamond P(x)]\!] =$$

$$(\emptyset, \langle x\rangle, \emptyset).$$

This confirms that what we have defined is a *dynamic* might operator. However, the non-commutativity of $[\![\phi]\!]$, which is the clue to its dynamic character, holds only for a restricted class of formulas: the non-commutativity can only hold if ϕ contains free variables. This can be checked as follows:

Suppose that ϕ contains no free variables, i.e. $A_\phi = \emptyset$. We know that

$$\phi \text{ is acceptable in } (A,S,F) \text{ iff } \exists f \in F\ \exists g \in F_\phi :\ f\langle\!\langle S_\phi\rangle\!\rangle g.$$

But if $A_\phi = \emptyset$ then this is the case exactly if:

$$\forall f \in F\ \exists g \in F_\phi :\ f\langle\!\langle S_\phi\rangle\!\rangle g.$$

[27]Note that $[\![\Diamond\phi]\!]$ is not representable as an information structure!

(This follows from the irrelevance lemma that we proof in the appendix. The remark there about free variables is important here.) But this is the definition of validity. So formulas without free variables are acceptable iff they are valid. And if ϕ is valid, then $\neg(\phi)$ is not acceptable, so $(A,S,F)(\![\neg(\phi)]\!) = (A,S,\emptyset)$. But then we find that in case ϕ does not contain free variables

$$\begin{aligned}
&\phi \text{ is acceptable in } (A,S,F) \Leftrightarrow \\
&(A,S,F)(\![\Diamond\phi]\!) \;=\; (A,S,F) \Rightarrow \\
&(A,S,F)(\![\Diamond\phi]\!)(\![\neg(\phi)\,]\!) \;=\; (A,S,F)(\![\neg(\phi)]\!) \;=\; (A,S,\emptyset).
\end{aligned}$$

Hence:

$$(\![\Diamond\phi \cdot \neg(\phi)]\!) = (\![\neg(\phi) \cdot \Diamond\phi]\!).$$

This restricts the applicability of our $\Diamond$ as a dynamic might operator. We can understand the restriction from the technical point of view, but it does not seem to make sense intuitively. If something *might* exist, then usually it does not follow that it *does* exist. Still the fact remains that $\Diamond$ gives us a consistency test for DPL as an operation on information structures. And for a non-trivial fragment of the language this consistency test has a dynamic character.

2.5.3 Down dating

In the section on monotonicity we explained why the semantics of DPL should be monotone: DPL is to be the language of ordinary discourse in which more and more information is revealed by the speaker and gathered by the hearer. We also said that for some other situations it might be handy to have a language and a semantics of forgetting or down-dating, as we will call it. In this section we extend DPL with atomic formulas $x\mathsf{E}$ — read as "x exit" — for any variable x, that will be interpreted as an instruction to forget the current value of x. We will see that down-dating helps us to formulate old ideas more elegantly.
The interpretation of $x\mathsf{E}$ is essentially relational: $x\mathsf{E}$ does not give information, it is purely an action.[28] We define the relational interpretation of $x\mathsf{E}$ as follows:

[28]This means that it can not be represented by some information structure.

Definition 2.5.11
$[\![x\mathsf{E}]\!]_{sass} = \{\langle f,g\rangle : pd(f(x)) = g(x) \wedge (y \neq x \rightarrow g(y) = f(y))\}$.[29]

In this extension of the language we can have local variables: for example in $\exists x \cdot \phi \cdot x\mathsf{E}$. We can also give an update formulation of the meaning of a down-date.

Definition 2.5.12 $(A,S,F)(\![x\mathsf{E}]\!) = (A',S',\{g : \exists f \in F : g\langle\!\langle x\rangle\!\rangle f\})$, *where S' is obtained from S by removing the last x. If there is no occurrence of x in S, $S' = S$ and we remove x from A: $A' = A\backslash\{x\}$. If x does not occur in A either: $A' = A$.*

Note that $(\![x\mathsf{E}]\!)$ works best if there is at least one x available in (A,S,F). If there is no x, then $(A,S,F)(\![x\mathsf{E}]\!) = (A,S,F)$.
Now it is possible to establish another relation between $[\![\phi]\!]_{sass}$ and $[\![\phi]\!]_{gs}$. As one can see:

$$[\![\exists x]\!]_{pgs} = [\![x\mathsf{E} \cdot \exists x]\!]_{sass} \cap PASS \times PASS$$

and

$$[\![\exists x]\!]_{gs} = [\![x\mathsf{E} \cdot \exists x]\!]_{sass} \cap ASS \times ASS.$$

This is no surprise: the original interpretation of $\exists x$ told us to *replace* a value of x. In the refined semantics this can be established by two separate actions: first throw away the old value of x with $x\mathsf{E}$, then add the new value with $\exists x$. Thereby we are able to translate ordinary dynamic predicate logic into the enriched language by replacing all quantifiers $\exists x$ by $x\mathsf{E} \cdot \exists x$. If we call this translation $\circ$, we find:

Proposition 2.5.13 *Let $f,g \in PASS$. Then: $f[\![\phi^\circ]\!]_{sass}g \Leftrightarrow f[\![\phi]\!]_{pgs}g$.*

Corollary 2.5.14 *Let $f,g \in ASS$. Then: $f[\![\phi]\!]_{gs}g \Leftrightarrow f[\![\phi^\circ]\!]_{sass}g$.*

(The proof is omitted.) The restriction that $f,g \in PASS$ is added because $[\![.]\!]_{pgs}$ is defined on $PASS$. The corollary follows immediately from the proposition, since $[\![.]\!]_{gs}$ is the restriction of $[\![.]\!]_{pgs}$ to total assignments. It is also possible to use down-dating to give an elegant definition of dynamic validity, $\models$. First we introduce some notation:

[29] Again $pd(\langle\rangle) = \langle\rangle$.

Notation: We will write $\downarrow\langle x\rangle g$ for the assignment f such that $g[\![x\mathsf{E}]\!]_{sass}f$. For $\downarrow\langle x_n\rangle(\dots(\downarrow\langle x_1\rangle(g))\dots)$ we write $\downarrow\langle x_1,\dots,x_n\rangle(g)$ and if G is a set of partial assignments, we write $\downarrow\langle x_1,\dots,x_n\rangle(G)$ for $\{\downarrow\langle x_1,\dots,x_n\rangle(g) : g \in G\}$.

We define:

Definition 2.5.15

1. *$\ll$ is a relation on states defined by: $(A,S,F) \ll (A',S',F') \Leftrightarrow F \subseteq \downarrow S'(F')$.*
2. *For any state (A,S,F), $\downarrow(A,S,F)$, the projection (or domain) of (A,S,F), is defined by: $\downarrow(A,S,F) = (A,\langle\rangle,\downarrow S(F))$.*

So to find $\downarrow(A,S,F)$, we simply forget the values of the variables in S. This way the functions in $\downarrow S(F)$ are exactly the ones that have an $\langle\!\langle S\rangle\!\rangle$-extension in F. We see that for any ϕ, $\downarrow[\phi]$ is the input state, or domain, for ϕ: ϕ asks for values on the variables in the first component of $\downarrow[\phi]$ and accepts only those assignments that are in the third component of $\downarrow[\phi]$. $\downarrow S(F_\phi)$ is for $[\phi]$, what $dom([\![\phi]\!]_{gs})$ is for $[\![\phi]\!]_{gs}$.
It can be checked that $\ll$ corresponds exactly to dynamic entailment. I.e. we claim:

Proposition 2.5.16 *For any $\phi,\psi : [\phi] \ll [\psi] \Leftrightarrow \phi \models \psi$.*

Proof: By comparing the definition of $\ll$ and, for example, $\models_{stat}$, using the fact that $f\langle\!\langle S\rangle\!\rangle g \Leftrightarrow \downarrow S(g) = f$.
□

This result seems rather strong, but in fact it is already known for the original *DPL*-semantics that

$$\phi \models \psi \;\Leftrightarrow\; range([\![\phi]\!]_{gs}) \subseteq dom([\![\psi]\!]_{gs}).$$

Here we see that the same relation holds, but now it is possible to define domains in terms of a more primitive notion: the down-date operator. In the extended language it is even possible to give for each formula an *expression* that gives the domain of the formula: we simply add the right amount of $x\mathsf{E}$'s for each variable. If we call this expression for the domain of ϕ, $\downarrow\phi$, we get for example:

Example:

$$\downarrow P(x) \; = \; P(x);$$

$$\downarrow (\exists x \cdot R(x,y)) \; = \; \exists x \cdot R(x,y) \cdot x\mathsf{E}$$

$$\downarrow (\exists x \cdot R(x,y) \cdot \exists x \cdot P(x)) \; = \; \exists x \cdot R(x,y) \cdot \exists x \cdot P(x) \cdot x\mathsf{E} \cdot x\mathsf{E}$$

The fact that we are able to model down-dating in sequence semantics, shows how powerful sequence semantics really is. In the applications that we have given, we have shown that some familiar notions in the semantics can be redefined elegantly in terms of the down-date operator. It remains to be seen how down-dating can be used in the study of phenomena that are genuinely non-monotonic. Perhaps $x\mathsf{E}$ could be used to model some cases of belief revision, but that will have to wait.

2.5.4 Finite variable fragments

Above we pointed out that $x\mathsf{E} \cdot \exists x$ is the proper analogue of $\exists x$ in Groenendijk and Stokhof's presentation of *DPL*. This shows that our extended language allows us to distinguish two aspects of *DPL*'s $\exists x$: whereas $[\![\exists x]\!]_{gs}$ at one and the same time throws away the old value of x and introduces a new one, we have made this into two distinct steps here.
We can also compare our extended language with static predicate logic in this respect. In static predicate logic the quantifier is a sentential operator — $\exists x(-)$ —, where the '('-bracket indicates the moment at which a new value for x may be set and the ')'-bracket throws away this value. So also in static predicate logic the two actions can be located at distinct points in the formula, as in the extended language, but unlike ordinary *DPL*. In this respect static predicate logic is more like our extended language with

$\exists x($ as the analogue of $\exists x$

and

$)$ as the analogue of $x\mathsf{E}$.

However, the extended language is more flexible than static logic: in static logic $\exists x($ and $)$ always travel in pairs, whereas $\exists x$ and $x\mathsf{E}$ can float around freely in the extended language. This gives us an extra bit of freedom. In particular it allows us to 'mix scopes': in the extended language we can close the scope of some quantifier while a quantifier that is nested deeper in the formula remains active. We can have, for example:

$$\exists x \cdot \phi \cdot \exists y \cdot \psi \cdot x\mathsf{E} \cdot \chi \cdot y\mathsf{E}$$

where x is thrown away before y, although y was introduced later than x! This situation has no analogon in static predicate logic:

$$\exists x(\ \phi\ \wedge\ \exists y(\ \psi\ \wedge\ \ldots$$

At first sight this may seem a useless sort of trick, but we can see that it can pay of by looking at the so-called finite variable fragments. In the search for decidable (efficient, 'attractive', etc.), but reasonably expressive fragments of predicate logic special attention has been paid to these finite variable fragments. For each $n \in I\!N$ we can look at the formulas of predicate logic that use at most n variables. The intuition is that the number of variables influences the amount of memory that is used. Restrictions on the number of variables should then give us 'easier' formulas, computationally speaking, since they need less memory space.[30]
Also for our extended *DPL* language the notion of a finite variable fragment makes perfect sense. Static predicate logic can be translated into *DPL* (cf. section 1.5) and this can again be translated into the extended language, as indicated above. Hence (some of) the computational atrocities of predicate logic transfer to our extended language. In particular it inherits the undecidability of static predicate logic. So also here it makes sense to look for expressive, yet efficient fragments of the language by restricting the number of variables that can occur in a formula.
Although it is clear that also in the extended language restrictions on the number of variables will have consequences (just try to express $\exists x \cdot \exists y \cdot P(x, y)$ with one variable only), we will show that here the consequences of such restrictions are less severe than in the case of static predicate logic.

Let's first fix some notation:

$\mathcal{L}^n_{pl}$ is the set of sentences from $\mathcal{L}_{pl}$, the language of static predicate logic, that use at most n variables (free or bound)

[30]Cf. Van Benthem (1993) and references therein.

$\mathcal{L}_e^n$ is the set of sentences from $\mathcal{L}_e$, the extended language that includes $x\mathsf{E}$ as an atomic proposition, that use at most n variables (free or bound)

It is clear from the translation of $\mathcal{L}_{pl}$ into $\mathcal{L}$ that we have seen in section 1.5 that $\mathcal{L}_e^n$ is at least as expressive as $\mathcal{L}_{pl}^n$. To be precise:

> for all $\phi \in \mathcal{L}_{pl}^n$ there is a $\phi^\star \in \mathcal{L}_e^n$ such that: ϕ is statically true iff $\phi^\star$ is dynamically true.

But the converse does not hold in general. We consider $n = 2$ as an example. There are several well-known examples of properties that cannot be expressed in $\mathcal{L}_{pl}^2$. We consider the 'square property' as an example:

$$\exists x \exists y \exists z \exists u \; (x < y \;\wedge\; y < z \;\wedge\; z < u \;\wedge\; u < x)$$

This formula expresses the property that the interpretation of $<$ contains a 'square' $\Box xyzu$. In static predicate logic we need three variables to express this property, but in $\mathcal{L}_e$ we can do it with two variables, as follows:

$$\exists x \cdot \exists y \cdot x < y \cdot \exists x \cdot y < x \cdot \exists y \cdot x < y \cdot x\mathsf{E} \cdot y < x$$

Here it is crucial that we can 'close of' with $x\mathsf{E}$ while leaving the latest y intact.
If we consider languages with $=$, we can do a more famous example: relation composition. In static predicate logic it is impossible to express the fact that $R \subseteq S \circ U$, i.e.

$$\forall x \forall y (xRy \;\rightarrow\; \exists z (xSz \;\wedge\; zUy))$$

using only two variables. But there is an $\mathcal{L}_e^2$-formula that says exactly this:

$$(\exists x \cdot \exists y \cdot xRy \;\rightarrow\; \exists y \cdot xSy \cdot \exists x \cdot x = y \cdot y\mathsf{E} \cdot xUy)$$

These examples show that $\mathcal{L}_e^2$ is strictly more expressive than $\mathcal{L}_{pl}^2$.
Note that we need $=$ to express relational composition. Therefore it may be expected that $\mathcal{L}_e^2$ without equality still gives a fairly significant restriction of the expressive power. But our first experiments with $\mathcal{L}_e^2$ with $=$ suggest that in the presence of $=$ anything goes.
The fact that we can express relation composition in the extended language with $=$ immediately gives us the following result:

Proposition 2.5.17 *Any first order definable operation on binary predicates that can be written in static predicate logic with = using only three variables, can be expressed in $\mathcal{L}_e^2$ with =.* $\Box$

This is an immediate consequence of a result by Tarski[31]: he proves that any first order definable operation on relations defined with three variables (and =) can be written in relation algebra with only $\cap$, $\cup$, $\smile$, *id* and $\circ$.[32] Since we can express all these operations in two variables in the extended language, the proposition follows. This shows that the way we use variables in the extended *DPL*-language gives us a lot more expressive power.
This discussion suggests that the intuitive connection between the number of variables and the expressivity may have to be reconsidered. It seems that it is not the number of variables simpliciter that counts, but also the way we handle these variables. In this respect the variable management in our extended language $\mathcal{L}_e$ is significantly more flexible than what static predicate logic gives us.

2.6 Conclusion

The main conclusion of this chapter is that it is possible to give a formalisation of the ideas of Groenendijk and Stokhof (1991a) in which the information content of a formula can be represented correctly. This means that interesting questions about information can be discussed in *DPL*-semantics such as the growth of information that is the effect of the interpretation of *DPL*-formulas.
We have obtained the improved representation by using sequence valued assignments. The use of these assignment inspires a suitable notion of information structure. We have shown that different ways of looking at information in *DPL* lead to different orderings on the information structures.
Perhaps it is useful to point out here that as we are distinguishing different perspectives on information in dynamic semantics, different questions regarding information growth come into play. The non-eliminativity

[31] Cf. Tarski (1941), Maddux (1983).

[32] Here $\smile$ stands for *converse*, the unary operation on binary relations that exchanges input and output: i.e. $\langle x, y\rangle \in R$ iff $\langle y, x\rangle \in R^{\smile}$.

problem is concerned with the preservation of information content as the updating process proceeds. This is a concern with information growth in the 'synchronic dimension' while moving along 'diachronically'. But also other concerns about the preservation of information arise in a multi-dimensional setting. For example, we could aim at keeping all possibilities for 'diachronic interaction' open while accumulating information in the 'synchronic dimension'. Here this would amount to preserving all options for anaphoric linking as one is accumulating information. Clearly this is not a sensible requirement for the full *DPL*-language (with its intended meaning), but there could be situations in which such a requirement makes sense. The advantage of our discussion here is that we are now in a position in which we can separate these two questions: we have seen that the requirement of preservation of truth-conditional information is independent from the question for the availability of antecedents and we have developed techniques that make this formally precise. We will follow up on this observation in the next chapter, where we discuss the different roles that variables play in the dynamic accumulation of information.
At some point in the future we hope to improve our understanding of the ways in which information can be compared in dynamic semantics in general. We think that the ideas that we have developed about comparing information will be of use there.
We were also able to define a down-date operator in our semantics. This operator is an instruction to forget the value of a variable. We have looked at the relation of down-dating with the original semantics for DPL and with the DPL-notion of entailment. Maybe down dating can also be used to model genuinely non-monotonic phenomena, such as belief revision, but this falls outside the scope of this thesis.

Appendix

In this appendix we present a proof of proposition 2.2.5 and corollary 2.2.7. In the proofs we will use the following lemma:

Lemma(Irrelevance):

1. $k[\![\phi]\!]_{sass}h \Rightarrow k'[\![\phi]\!]_{sass}h'$ for all k', h' such that for all x $k'(x) = \sigma_x * k(x)$ and $h'(x) = \sigma_x * h(x)$ (for some sequence σ_x).

2. $k[\![\phi]\!]_{sass}h \Rightarrow k''[\![\phi]\!]_{sass}h''$ for all k'', h'' such that for all x $k(x) = \sigma_x * k''(x)$ and $h(x) = \sigma_x * h''(x)$ (for some sequence σ_x) and $k''(x)$ extends $end(k(x))$.

The lemma says that if (k, h) is in the relation $[\![\phi]\!]_{sass}$, only those values of k and h are relevant that occur after $end(k(x))$. This corresponds to the fact that we always add the current value of a variable at the end. ($end(k(x))$ itself is relevant if x is free in ϕ.) We will not prove the lemma: the proof is an easy induction.
Now we will prove proposition 2.2.5:

Proposition 2.6.1 *For all $\phi \in DPL$ we have $\Phi([\![\phi]\!]_{sass}) = [\![\phi]\!]_{pgs}$.*

Remember that Φ is defined by:

Definition 2.6.2 *We define $\Phi : \wp(SASS \times SASS) \to \wp(PASS \times PASS)$ as follows:*
$\Phi(R) =$
$\{\langle g, f\rangle : \exists\langle k, h\rangle \in R : \forall x \in VAR : f(x) = end(h(x)) \wedge g(x) = end(k(x))\}$.

Proof (of 2.2.5): We have to prove that:
1 $\langle g, f\rangle \in \Phi([\![\phi]\!]_{sass}) \Rightarrow g[\![\phi]\!]_{pgs}f$
2 $g[\![\phi]\!]_{pgs}f \Rightarrow \langle g, f\rangle \in \Phi([\![\phi]\!]_{pgs})$.
We prove this by a simultaneous induction on the complexity of ϕ. We will need 1 as induction hypothesis for the $\to$-clause of 2 and vice versa.

1. $P(x)$ Suppose $\langle g, f\rangle \in \Phi([\![P(x)]\!]_{sass})$. Then there are h, k such that $k = h$ and $end(h(x)) \in I(P)$ and $\forall y : g(y) = end(k(y))$ and $f(y) = end(h(y))$. Hence $f = g$ and $end(h(x)) = f(x) \in I(P)$, i.e. $g[\![P(x)]\!]_{pgs}f$.

   $\exists x$ Suppose $\langle g, f\rangle \in \Phi([\![\exists x]\!]_{sass})$. Then there are h, k $k[\langle x\rangle]h$ and $\forall y : f(y) = end(h(y))$ and $g(y) = end(k(y))$. Obviously $g[\![\exists x]\!]_{pgs}f$.

   $\psi \cdot \chi$ Suppose $\langle g, f\rangle \in \Phi([\![\psi \cdot \chi]\!]_{sass})$. Then there are h, k, l such that: $k[V_\psi * V_\chi]h$ and $h \in F_\chi$ and $l[V_\chi]h$ and $l \in F_\psi$ and $\forall y : g(y) = end(k(y))$ and $f(y) = end(h(y))$. Define m such that $m(y) = end(l(y))$. Then $\langle m, f\rangle \in \Phi([\chi])$. Also, since $k[V_\psi]l$, $\langle g, m\rangle \in \Phi([\psi])$. Hence (ind. hyp.): $g[\![\psi \cdot \chi]\!]_{pgs}f$.

$(\psi \to \chi)$ Suppose $\langle g, f\rangle \in \Phi([\![(\psi \to \chi)]\!]_{sass})$. Then there are h, k such that $k[\![(\psi \to \chi)]\!]_{sass}h$ and $\forall y : g(y) = end(k(y))$ and $f(y) = end(h(y))$. Then $f = g$, $k = h$. Now let $f[\![\psi]\!]_{pgs}m$. Then 2 gives us $l[\![\psi]\!]_{sass}n$. Now the lemma gives us n' such that $h[\![\psi]\!]_{sass}n'$. By assumption this means that there is a n'' $n'[\![\chi]\!]_{sass}n''$. By the definition of Φ, this gives a p such that $\langle m, p\rangle \in \Phi([\![\chi]\!]_{sass})$. Now the induction hypothesis for 1 guarantees $m[\![\chi]\!]_{pgs}p$. Hence $g[\![(\psi \to \chi)]\!]_{pgs}f$.

2. $P(x)$ Suppose $g[\![P(x)]\!]_{pgs}f$. Then $g = f$ and $end(f(x)) \in I(P)$. So we can choose $h = k = f$ to prove that $\langle g, f\rangle \in \Phi([\![P(x)]\!]_{sass})$.

$\exists x$ Suppose $g[\![\exists x]\!]_{pgs}f$. Then $\forall y : g(y) = f(y)$ and $y = x \wedge f(x)$ is defined. Choose h such that $h(x) = g(x) * \langle f(x)\rangle$ and $h(y) = f(y)$ for all other y, and choose k such that $k(x) = pd(h(x))$ and $k(y) = h(y)$ for all other y. These h, k guarantee that $\langle g, f\rangle \in \Phi([\![\exists x]\!]_{sass})$.

$(\psi \to \chi)$ Suppose $g[\![(\psi \to \chi)]\!]_{pgs}f$. Then $f = g$ and $\forall m :\ g[\![\psi]\!]_{pgs}m$ $\exists n : m[\![\chi]\!]_{pgs}n$. Let $k = f = g = h$. We prove that $k[\![(\psi \to \chi)]\!]_{sass}h$: suppose $k[\![\psi]\!]_{sass}l$. This gives $\langle g, p\rangle \in \Phi([\![\psi]\!]_{sass})$ for p such that $p(y) = end(l(y))$ for all y. Hence, by 1, $g[\![\psi]\!]_{pgs}p$. But then, by assumption, there must be a q such that $p[\![\chi]\!]_{pgs}q$. By induction hypothesis for 2 we get h', l' such that $l'[\![\chi]\!]_{sass}h'$ and $\forall y : end(h'(y)) = q(y)$ and $end(l'(y)) = p(y) = end(l(y))$. Using the lemma we find that there is a h'' such that $l[\![\chi]\!]_{sass}h''$.

$\psi \cdot \chi$ Suppose $g[\![\psi \cdot \chi]\!]_{pgs}f$. It suffices to consider the cases in which χ is not of the form $\chi' \cdot \chi''$.

a $\chi \equiv P(x)$: Then $g[\![\psi]\!]_{pgs}f$ and $f(x) \in I(P)$. By induction hypothesis for 2 we get h, k such that $k[\![\psi]\!]_{sass}h$ and $\forall y : f(y) = end(h(y)) \wedge g(y) = end(k(y))$. But then $end(h(x)) \in I(P)$. So these h, k also do the job for $\psi \cdot P(x)$.

b $\chi \equiv \exists x$: Then there is a f' such that f' does not differ from f on variables other than x and $g[\![\psi]\!]_{pgs}f'$. By the induction hypothesis for 2, there are k, h' such that $k[\![\psi]\!]_{sass}h'$ and $\forall y : end(h'(y)) = f'(y) \wedge end(k(y)) = g(y)$. Now we define h such that $h(x) = h'(x) * \langle f(x)\rangle$ and $h(y) = h'(y)$ for all other variables. Then h, k do the job for $\psi \cdot \exists x$.

c $\chi \equiv (\rho \to \tau)$: Then $g[\![\psi]\!]_{pgs}f$ and $f[\![(\rho \to \tau)]\!]_{pgs}f$. By the induction hypothesis for 2 there are k, h, h' such that $k[\![\psi]\!]_{sass}h$ and $h'[\![(\rho \to \tau)]\!]_{sass}h'$, where for all x $end(h(x)) = end(h'(x)) = f(x)$. By the lemma we find that also $h[\![(\rho \to \tau)]\!]_{sass}h$. But then $\langle g, f\rangle \in \Phi([\![\phi]\!]_{sass})$.

□

Note that the proof of the proposition is not difficult. It is just hard work. The same holds for the proof of corollary 2.2.7. This time we will provide less details.

Corollary 2.6.3 (2.2.7) *For any ϕ, ψ the following clauses are equivalent:*

1. $\phi \models_{sass} \psi$;
2. $\phi \models_{pgs} \psi$;
3. $\phi \models_{gs} \psi$.

Proof (corollary 2.2.7): We sketch the proof of $1 \Leftrightarrow 2$ and leave $2 \Leftrightarrow 3$ to the reader.

$1 \Rightarrow 2$ Assume $\phi \models_{sass} \psi$. Let $f, g \in PASS$ be given such that $f[\![\phi]\!]_{pgs}g$. We have to find $h \in PASS$ such that $g[\![\psi]\!]_{pgs}h$. Now, since $PASS \subseteq SASS$, it follows from the assumption that we find a $h' \in SASS$ such that $g[\![\psi]\!]_{sass}h'$. But then Φ gives a $h \in PASS$ such that $g\Phi([\![\psi]\!]_{sass})h$, i.e. $g[\![\psi]\!]_{pgs}h$.

$2 \Rightarrow 1$ Assume $\phi \models_{pgs} \psi$. Let $f, g \in SASS$ be given such that $f[\![\phi]\!]_{sass}g$. We have to find $h \in SASS$ such that $g[\![\psi]\!]_{sass}h$. Φ gives for f, g $f', g' \in PASS$ such that $f'[\![\phi]\!]_{pgs}g'$. By the assumption this gives an $h' \in PASS$ such that $g'[\![\psi]\!]_{pgs}h'$. If we decompose ψ into atoms $\psi_0, \ldots, \psi_n$, then we can see all the changes of values that we need to build the $h \in SASS$ such that $g[\![\psi]\!]_{sass}h$.

□

Chapter 3

Merging Without Mystery

In the introduction to this part we have discussed the relevance of the semantics of variables for dynamic theories of meaning. After that we have looked at one particular problem that arises in a particular theory of dynamic semantics, the non-eliminativity problem of *DPL*, and we showed that in fact this problem was closely connected with the treatment of variables in *DPL*. So now we have two good reasons for looking at the notion of variable in dynamic semantics: first of all the motivation from anaphora and secondly the technical motivation of the previous chapter. Therefore in this chapter we reconsider the concept 'variable' in the light of the general objectives of dynamic semantics. We will see that this leads in fact to a shift from the Fregean notion of a variable, that pervades traditional logic to a notion of variable which is more reminiscent of the outlook on variables that we find in computer science. For the purpose of dynamic semantics it turns out to be extremely useful to distinguish for each variable three things: the name of the variable, the variable *qua* storage facility, and the value of the variable. The three level picture of variables allows us to distinguish between the different roles that variables play in dynamic semantics. This picture will be formalised by the notion of a *referent system*. Then we go on to show how a dynamic semantics based on referent systems works.[1]

[1]Although the eliminativity problem is not the topic of this chapter, it should be pointed out that also the use of referent systems allows us to define the *DPL* semantics in an eliminative way.

3.1 Introduction

We have seen that in a dynamic semantics along the lines of Kamp (1981) and Heim (1983) we obtain a semantics of anaphora that deviates from the narrow path of traditional, static logic. Crucial for the way in which Kamp and Heim interpret anaphora is the attention in the formalism not only for the *results* of the interpretation process, but also for aspects of the process itself.
The idea underlying both *DRT* and *FCS* is that, when we have to interpret an anaphoric expression, we have to establish a connection with an antecedent. In both formalisms the potential antecedents can be found in a representation of the result of the interpretation process so far. Interpreting an anaphor now amounts to establishing the link with the correct antecedent. In the introduction we saw such an account in the particularly elegant formulation due to Zeevat.
We have also seen the relational formulation of this idea, namely in the *Dynamic Predicate Logic* (*DPL*) of Groenendijk and Stokhof (1991a). There the procedural aspect of semantics becomes the main point of attention. Now each sentence has as its meaning a relation between an assignment that is provided by the context (for example by the interpretation of previous text) and the assignments that can be obtained from the input assignment by interpreting the sentence. In this approach an interpretation can set a variable at a certain value in its output state. This output state is the input state for the interpretation of the next sentence. Thus the interpretation of one sentence can pass on a value to the interpretation of the next sentence. This is the way in which *DPL* represents the act of linking antecedent and anaphor.
The success and the elegance of Groenendijk and Stokhof's formulation and also the development of another interesting formalism with a procedural flavour, namely Veltman (1991)'s *Update Semantics* (*US*),[2] has led to an interest in the logic of procedures per se. For if the basic objects in our semantic universe are going to be *actions* (either represented as relations (*DPL*) or as functions (*US*)), then we should try to discover which are the general principles governing the behaviour of actions. This interest lead to various logical investigations centered around *DPL*.[3] Thus

[2]Of course Veltman's system is not designed for the treatment of anaphora. It gives a dynamic treatment of modalities. The details of his system need not concern us here.
[3]Here we think of the Hoare Logic approach of van Eijck (Van Eijck (1991c), Van Ei-

interest in the logic of actions is one important development in dynamic semantics.
But we have seen that in a Zeevat style formulation the dynamics comes in in another way. Zeevat's *DRS*-interpretations are not actions. Instead he uses semantic objects which are context-content pairs. In this set-up the context component allows us to understand the interaction with the context, the content component collects the information content. The interaction between context and content is captured by *the merger*, an operation on the pairs that tells us how to compose meanings. Thus all the dynamic effects of *DRT* get an elegant representation in an algebraic framework, without representing meanings as procedures.
This formulation of *DRT* leads to a new look on the old idea that the implementation of the context-content distinction is very useful for formal semantic in general. It enables us to distinguish different kinds of semantic contributions and to study their interaction in an elegant way. Actually we already find this idea in *Universal Grammar* Montague (1970)'s, and later in the work of Kaplan (1979) and Stalnaker (1978). But Zeevat's presentation leads to a new outlook on it and has inspired Visser (1992c) to develop general techniques for a semantics with context-content pairs.
In this chapter we will to reformulate Zeevat's treatment in a manner that permits both the ideas underlying *DRT* and the ideas underlying *DPL* to be captured. As a context-component we will use machinery that allows us to describe the interaction of variables in dynamic semantics in a particularly elegant way, introducing the notion of a *referent system* (section 3.3). Thus the referent systems will model the linking behaviour of variables, i.e. they will be the crucial ingredient for the semantics of anaphora. The motivating ideas behind our definition of referent systems lead to a natural notion of the *merger* of referent systems. Some abstract properties of the algebra of referent systems are presented in section 3.4. Then we show how they can be used in dynamic semantics (section 3.5). We will discuss the relation of our semantics with both *DRT* and *DPL* (section 3.6). It will be argued that the use of referent systems is philosophically preferable to the treatment of variables in both systems. We will also argue that a clear implementation of the distinction between the

jck and De Vries (1992b)), modal logical approaches, especially using two-dimensional modal logic (Blackburn and Venema (1993)), and relation algebra approaches (De Rijke 1992). Similar considerations have driven the development of *Arrow Logic* by Van Benthem (1994).

different roles that variables play in semantics — as part of the context component on the one hand, but also as a carrier of information content on the other hand — is the obvious step to make in a dynamic setting. A clear implementation of this distinction will also make the system more robust: it will allow us to make adjustments in one of these areas (e.g. the context component) without disturbing the balance in the other compartment (i.e. the content component). Finally there is some general discussion on the use of variables in the semantics of anaphora (section 3.7).
But before we do all this we need to look at Zeevat's proposal in some detail.

3.2 Contexts and contents

In Zeevat's formulation of *DRT* we find a discourse representation language in which discourse fragments are represented as pairs.[4] These pairs are the *discourse representation structures* (*DRSs*). The first component of a *DRS* consists of a number of *discourse markers*. They are the topics that have been introduced by the corresponding discourse fragment. In the other component conditions on these discourse markers are stored. Thus for a piece of discourse such as

A dog barked. It was lonely.

we can expect a representation such as

$\langle\{r\}, \{\ \mathit{dog}(r),\ \mathit{bark}(r),\ \mathit{lonely}(r)\ \}\rangle.$

In the first component of the representation we find the discourse markers introduced in the example. In fact there is just one such marker, the dog. In the second component we see the conditions on this marker that the piece of discourse expresses.
In the semantics these representations are interpreted as pairs: the set of discourse markers is simply copied and in the second component of the semantic objects we store all assignments of values to these markers that satisfy the conditions in the second component of the *DRS* (cf 1.5). So the interpretation of the *DRS* given above is:

[4]For more details, also on the relation with Kamp's formulation, cf Zeevat (1991a) and the introductory chapter of this thesis.

$$\langle\{r\}, \{f : MARK \to DOM : \; f(r) \in \mathbf{dog} \cap \mathbf{bark} \cap \mathbf{lonely}\}\rangle.$$

Here D is the domain of objects in (our model of) the real world. $MARK$ is the set of discourse markers.[5] The second component represents the actual testing of the information content of the DRS in the real world. The role of the first component in the interpretation of anaphora becomes clear as soon as we see how the interpretation of a large text is composed from the interpretations of its parts. The DRS-interpretation we see above can be described as the result of such a composition process.

$$\begin{array}{lcl}
\langle\{r\},\ \{dog(r),\ bark(r)\}\rangle & \rightsquigarrow & \langle\{r\},\ \{f :\ f(r) \in \mathbf{dog} \cap \mathbf{bark}\}\rangle \\
\langle\emptyset,\ \{lonely(r)\ \}\rangle & \rightsquigarrow & \langle\emptyset,\ \{\ f :\ f(r) \in \mathbf{lonely}\}\rangle \\
\langle\{r\}, \{\ dog(r), bark(r), & & \\
\qquad\qquad lonely(r)\}\rangle & \rightsquigarrow & \langle\{r\}, \{f : f(r) \in \mathbf{dog} \cap\ \mathbf{bark} \cap \mathbf{lonely}\}\rangle
\end{array}$$

We see that if we put two first DRSs on the left hand side together, we get the DRS on the third line and the condition in the second DRS, *lonely*, is linked to the marker r that is introduced in the first DRS. In the semantics the process of linking is handled by the first components of the DRS-interpretations. This is what the first component is for.
Zeevat simply takes set union as the operation on the marker sets. Thereby the markers from the different components are simply put together. As a consequence we can also simply put together the conditions on these markers. The corresponding operation on the sets of assignments is intersection. So we find:

$$(V,\ F) \bullet (W,\ G) \quad = \quad (V \cup W,\ F \cap G).$$

Zeevat uses assignments f that are defined on all the markers in VAR. We can slightly facilitate the discussion if instead we use assignments f that are only defined on the markers that actually occur in the corresponding

[5]In this chapter we have a slightly different notation for sets of variables and variable names: whereas the rest of the chapter simply uses VAR for 'the variables of the language', in this section we use MAR for the (countably infinite) set of discourse markers. Later we will then use NOM for the set of variable names. The difference in notation is used to underline the view on variables that we are developing in this chapter.

DRS. This way we get pairs (V, F) such that for each $f, f' \in F$ $dom(f) = dom(f')$ and $V \subseteq dom(f)$. Note that we do not demand that $dom(f) = V$, as will be clear from the second *DRS* in the example above: there $f(r)$ is defined, but $r \notin V$. Now the *DRS*-interpretation will look like:

$$(\{r\},\ \{\ f:\ \{r\} \to D\ :\ f(r) \in \mathbf{dog} \cap \mathbf{bark} \cap \mathbf{lonely}\ \}).$$

Now also our definition of the merger will look slightly different, since we are no longer using total assignments. When we simply lump together the conditions that we find in different *DRS*s, we now have to glue together assignments from F and G which may have different domains. Let's write $f \oplus g$ for the function that has domain $dom(f) \cup dom(g)$ and assigns $f(r)$ to r when $r \in dom(f)$ and $g(r)$ when $r \in dom(g)$. Note that $f \oplus g$ is only defined if f and g agree on $dom(f) \cap dom(g)$. $\oplus$ simply glues f to g, provided this can be done. The operation $\oplus$ on sets of assignments F and G can then be defined in the obvious way:

$$F \oplus G \quad = \quad \{f \oplus g :\ f \oplus g \text{ is defined } \ \&\ f \in F \ \&\ g \in G\}.$$

Now we get:

$$(V,\ F) \bullet (W,\ G) \quad = \quad (V \cup W,\ F \oplus G).$$

as the analogon of Zeevat's merger.
We see that if we use this definition of the merger, the link between the marker that is lonely and the marker that is a barking dog, is established in the required way. This was the result of our decision to simply lump together the two sets of markers. If we make other choices for the operation on the marker sets we can steer the linking process in the semantics in another way. In fact the definition of the merger that we gave above generalises to the following definition *schema*:

$$(V,\ F) \bullet_{\star} (W,\ G) \quad = \quad (V \star W,\ F^{\star} \oplus G^{\star}).$$

We see that different choices for the operation $\star$ on the marker sets, lead to different operations $\bullet_{\star}$ on the *DRS*-interpretations.[6] The idea is that

[6]The notation $F^{\star}$ suggests that $(-)^{\star}$ simply is a function on sets of assignments, but we will see that in general it can also depend on other information, in particular V, W, G.

the choice of $\star$ will represents our 'theory' about the behaviour of discourse markers. The Zeevat 'theory' is plain and simple: markers should simply be lumped together. But other 'theories' are possible. Once we have chosen our 'theory of markers', everything is fixed: the behaviour of the discourse markers will automatically lead to a transformation of the domains of the assignments. Thus it will induce a corresponding transformation on the sets of assignments — which we write as $(-)^\star$ — and once these transformations are performed we can simply add these sets in the usual way.

As an example, consider the following alternatives for the definition of the merger:

$$\begin{array}{lcl}(V,\ F)\bullet_1(W,\ G) & = & (V\sqcup W,\ F^{\sqcup}\oplus G^{\sqcup})\\ (V,\ F)\bullet_2(W,\ G) & = & (V\sqcup_{F,G}W,\ F^{\sqcup_{F,G}}\oplus G^{\sqcup}_{F,G})\end{array}$$

Both $\sqcup$ and $\sqcup_{F,G}$ will stand for some form of disjoint union, thus implementing a linking strategy that keeps referents that occur both in V and W distinct. But, as we will see shortly, $\sqcup$ implements this idea less carefully than $\sqcup_{F,G}$. The details are as follows:

$$\begin{array}{lcl}V\sqcup W & = & (V\times\{1\})\cup(W\times\{2\})\\ V\sqcup_{F,G}W & = & (V\backslash W)\cup(W\backslash V)\cup X\\ & & \qquad\text{such that } X\cap V=X\cap W=\emptyset,\ |X|=2\cdot|V\cap W|\\ & & \qquad\qquad\text{and } X\cap dom(h)=\emptyset \text{ for all } h\in F\cup G\end{array}$$

We see that $\sqcup$ is disjoint union in its most insensitive guise: in $V\sqcup W$ we simply make two new sets $V\times\{1\}$ and $W\times\{2\}$ which have the same size as V and W respectively and of which we can be sure that they are disjoint. In the definition of $V\sqcup_{F,G}W$ we are more careful: here we only disturb elements from V and W if this is really necessary and we make sure that the new elements X that we choose are not used in the domains of the assignments in F and G.[7]

Both $\sqcup$ and $\sqcup_{F,G}$ give rise to transformations on the marker sets V and W. Thus they transform the domains of the assignments in F and G. This

[7]Recall that when $f\in F$ $V\subseteq dom(f)$, but not necessarily $V=dom(f)$. In fact these examples show that it would be convenient to have $dom(F)\backslash V$ available explicitly in the *DRS*-interpretations, but here we stay faithful to the traditional discussion. Later on, in our referent systems these variables will be made available explicitly in the semantics.

induces transformations on the assignments in the obvious way, resulting in the sets $F^{\sqcup}$, $F^{\sqcup_{F,G}}$, $G^{\sqcup}$, $F^{\sqcup_{F,G}}$. For example:

$$\begin{aligned}
&f' \in F^{\sqcup} \text{ iff} \\
&\qquad \exists f \in F: \ f'((v,1)) = f(v) \text{ for } v \in V \text{ and } f'(x) = f(x) \text{ for } x \notin V \\
&g' \in G^{\sqcup} \text{ iff} \\
&\qquad \exists g \in G: \ g'((w,2)) = g(w) \text{ for } w \in W \text{ and } g'(x) = g(x) \text{ for } x \notin W
\end{aligned}$$

We can illustrate the differences between the three definitions of the merger with an example:

$$\begin{aligned}
(V,F) &= (\{r\}, \{f : \{r\} \to DOM : \ f(r) \in \mathbf{dog}) \\
(W,G) &= (\emptyset, \{f : \{r\} \to DOM : \ f(r) \in \mathbf{bark}) \\
(X,H) &= (\{r\}, \{f : \{r\} \to DOM : \ f(r) \in \mathbf{bark})
\end{aligned}$$

Now we see that in $(V,F) \bullet (X,H)$ the condition on r in (X,H) is linked to the condition on r in (V,F), but this does not hold for $\bullet_1$ or $\bullet_2$. Whereas $\bullet$ regards the two conditions as conditions on the same r, the operations $\bullet_i$ do not make this identification: $\bullet_i$ keeps the two occurrences of r distinct and as a consequence it also keeps the information about these occurrences separate. This is the dividing line between $\bullet$ and $\bullet_i$.
In $(V,F) \bullet (W,G)$, the condition on r expressed by G is added to the condition on r that is expressed by F. $\bullet$ does not distinguish between the $r \in V$ and the $r \in dom(g)$, $g \in G$. This way a link is established and this is in fact the kind of link that corresponds to the anaphoric link in the example. We see that the same connection is established by the careful version of disjoint union, $\sqcup_{F,G}$: since $\{r\} \cap \emptyset = \emptyset$, we find that

$$(V,F) \bullet_2 (W,G) = (\{r\}, \{f : \ f(r) \in \mathbf{dog} \cap \mathbf{bark}\}).$$

So $\bullet_2$ also establishes the anaphoric link, as required. However, if we use $\bullet_1$ instead, things go wrong. Now we get:

$$(V,F) \bullet_1 (W,G) = (\{(r,1)\}, \{f : \ f(r) \in \mathbf{bark} \ \& \ f((r,1)) \in \mathbf{dog}\}).$$

Because $\bullet_1$ automatically replaces r by $(r,1)$, we do not only keep the markers from V and W distinct, but we also disturb the relations between V and $dom(g)$ (for $g \in G$). Here this has the effect that we do not get an anaphoric link where we would have expected one, but we can also get the reverse effect. For example, consider

$$(W', G') = (\emptyset, \{f : \{(r,1)\} \to D : \ f((r,1)) \in \mathbf{bark}).$$

If we now compute $(V, F) \bullet_1 (W', G')$, r and $(r, 1)$ will get confused and the conditions on these referents will be joined together.
We see here that it is indeed the first component that steers the linking process. By varying the operation $\star$ on the marker sets, we can implement all kinds of ideas about anaphoric linking. But once the first components have decided which referent goes where, the sets of assignments have to follow these decisions. Thus the two tasks that are involved in the interpretation of anaphora are nicely separated in this format: the first components establish the anaphoric links between different pieces of discourse and the second components compute the joint information content (and truth-conditions) according to these links.

3.3 Referent systems

3.3.1 Variables

What Zeevat's formulation makes clear is that in the semantics of anaphora, we need to distinguish carefully between control features and information content. Here *control* amounts to linking the variables in the appropriate way, analogous to the linking of anaphors to antecedents in natural language. Therefore it is a good idea to conduct a more serious investigation into the notion of variable and the role it plays in semantics.
Logic traditionally uses the 'Fregean' notion of variable. In such an approach it is not correct to think of variables as having a denotation. To say that a variable has a reference of some sort, would merely provoke the question what it is that a variable denotes. Probably the answer would be something like: variables denote arbitrary objects. Attempts to make sense of this answer would, according to Frege[8], lead us to problems that we do not even want to think about: certainly he showed that this was not necessary in standard predicate logic.
However, Frege's objections against other concepts of variables may have been too hasty. For example, in the work of Fine (1985) it is shown that it is possible to have a sensible theory of variables as arbitrary objects, something that Frege strongly rejects. Moreover in computer science, for

[8]In Frege (1979): Logische Mängel in der Mathematik.

example in denotational semantics (cf. for example Schmidt (1988)), a notion of variable is used that is very different from the Fregean notion of variable, but nevertheless makes perfect sense. In the computer science notion of variable a distinction is made between the syntactic variable or variable name, and the real variable, which is usually thought of as some location (in the memory of a computer) where information can be stored. Thus it becomes possible to distinguish three things for each variable: its name, the variable itself and the information that is stored in the variable. It can be suspected that some such notion of variable will be particularly useful for us, who are trying to distinguish between the control features that have to be represented in dynamic semantics, and the information content. With this notion of variable we can think of variables as storage facilities and their names as the mechanism by which we manipulate the variables and thereby indirectly the information that we have stored in them. So for us the interesting control feature is the way we manipulate the variables via their names.
In what follows we will give a formal treatment of such a notion of variable. We will then use the new techniques to do dynamic semantics. We will see how in the new formulation the referent systems are used to describe the control features essential for the explanation of anaphora.

3.3.2 Referent systems

In the formal definition of the referent systems we will use a fixed stock of names, $NOM = \{x_1, x_2, \ldots\}$. We will usually write x, y, z, v, w for elements of NOM. We also need some set theoretic representation for the variables, or referents as we shall be calling them.[9] Remember that referents in our set-up are just locations in memory that we happen to have reserved to store a particular piece of information. We do not want to include in our model any assumptions about the nature or the structure of memory. We regard memory as an unstructured substance that has no properties that are of interest to us: memory gets all its interesting properties by our actions on it. One such action is the declaration of a variable, whereby we reserve some arbitrary part of memory for the storage of information. This action actually *creates* the variable: before

[9]In fact we use the words 'variable', 'referent' and 'discourse marker' interchangeably. Note, however, that when we talk about the Fregean notion of variable, we are *not* talking about referents.

we performed the action there was nothing. After the declaration there is a variable. We will assume that at each point it is possible to perform such a creation.
An advantage of this way of looking at variables over the use of a fixed stock of variables, declared in advance, is that we will now always have just a finite amount of these storage facilities. Since there is, in principle, no limit to the amount of variables one can use, working with a fixed stock would imply working with an infinite amount of variables from the start. In our approach we will always just have a finite set of variables in our model, the ones we actually use.
Also note that the use of a fixed stock introduces a moment of choice in the way of dealing with variables: whenever we need a variable, we have to decide which one to use. This is a kind of complication that we choose to avoid, both for reasons of technical convenience an because this issue does not fit our intuitions concerning variables.[10] The complication can be avoided as described above, by creating new variables 'on the spot'.
The lack of structure and properties of memory is inherited by the variables. They are simply arbitrary parts of memory. This is another reason why it would be unfortunate if the variables in our model would all come from a fixed set. This would inevitably give them properties that real variables do not have.
Hence, if we want to find a set theoretic representation of some set of referents, it seems that no set is good enough. Or, if we look on the bright side, any set is equally good. When we are at a point where we have created some referents and we want to represent this situation set theoretically, then there is no *natural* choice for the set of referents: any set of the right size will do equally well (or equally bad). Therefore we can use any set as a set of referents, as long as we keep in mind that nothing forced us to choose this set instead of another one (with the same cardinality).
This leads to the following definition of *referent systems*.

Definition 3.3.1 *A triple* (I, R, E) *is a referent system iff:*

1. R *is a finite set, the referents;*

2. I *is a partial injection from* NOM *to* R*, the import function;*

[10]Note that such a 'moment of choice' does apply on the level of variable names, as was argued in the introduction to this part.

3. *E is a partial injection from R to NOM, the export function.*

   (From now on we will use postfix notation for function application and function composition. We will omit brackets whenever this is convenient.)

At the core of a referent system (I, R, E) we find the referents R. To this we have added an import function and an export function. They allow us to manipulate the referents.[11] Recall that referent systems will be used later to model the way we manipulate variables in dynamic semantics. In section 3.3.4 we will see examples of this, her we just give the general picture. The idea is that the import function tells us which referents are picked up from the context and under which name. So if $xI = r$, then this means that the referent r is not created in this referent system: it was created before and is now picked up from the context under the name x. Hence r will be the referent that was called x already.
The export function E tells us which referents in R can be picked up in what is to follow and under which name. Hence, if $rE = x$, then x is the current name of this referent. This means that the next referent system can import r under the name x. In such a situation the first referent system exports what the next referent system wants to import: a transaction can take place. Thereby a referent can interact with the context to its left via the import function and with the context to its right via the export function. Thus we create a model in which the (chronological) order of our actions on variables is represented.
Every referent system comes with a dual. It can be obtained, as it were, by reading the original referent system from right to left. So the dual of (I, R, E) is (E^{-1}, R, I^{-1}). From this duality we can learn that E^{-1} will have to behave like a proper import function or, dually, that I^{-1} has to behave like a proper export function. We will use this duality of referent systems whenever we can.
Remember that the choice of the set R is arbitrary. R is just a finite set of the right cardinality, but any other set of the same cardinality would have done equally well for a set theoretic representation of the same referent system. Therefore we should not use set theoretic identity as the identity criterion for referent systems. We will introduce a notion of isomorphism

[11] Below (proposition 3.4.7) we will see that in important cases alternative, inductive definitions of interesting classes of referent systems are available.

for our representations of referent systems that will provide us with the sort of identity that makes sense for referent systems.

Definition 3.3.2 *Let two referent systems (I, R, E) and (I', R', E') be given. A homomorphism $\Phi : (I, R, E) \to (I', R', E')$ is an injection $R \to R'$ such that:*

1. *$dom(I) \subseteq dom(I')$ & $xI\Phi = xI'$ (for all x);*
2. *$dom(E) \subseteq dom(\Phi E')$ & $rE = r\Phi E'$ (for all r).*

   (Note that this is not *the dual of the previous clause.)*

By Φ we can recognise the referents of (I, R, E) in (I', R', E') in such a way that the names that a referent has in (I, R, E) are also present in (I', R', E'). Note that the conditions (1) and (2) can be rewritten as:

$I\Phi \subseteq I'$

and

$E \subseteq \Phi E'$.

In a picture:

$$\begin{tikzcd}
& R \arrow[dd, "\Phi"] \arrow[dr, "E"] & \\
NOM \arrow[ur, "I"] \arrow[dr, "I'"'] & & NOM \\
& R' \arrow[ur, "E'"'] &
\end{tikzcd}$$

Now we can make the following observation: referent systems and homomorphisms form a category. Hereby the familiar notion of isomorphism from category theory becomes available:[12] two referent systems (I, R, E) and (I', R', E') are isomorphic iff there are homomorphisms $\Phi : (I, R, E) \to (I', R', E')$ and $\Psi : (I', R', E') \to (I, R, E)$ such that $\Phi\Psi = id_{(I,R,E)}$ and $\Psi\Phi = id_{(I',R',E')}$. The following proposition gives us easy way of recognising isomorphic referent systems.

[12]Recall that we use postfix notation.

Proposition 3.3.3 *There is a bijection Φ between R and R' such that $I\Phi = I'$ and $E = \Phi E'$ iff (I, R, E) and (I', R', E') are isomorphic.*

Proof:
Clearly such a bijection Φ induces a homomorphism $(I, R, E) \to (I', R', E')$ and Φ^{-1} induces a homomorphism $(I', R', E') \to (I, R, E)$ such that $\Phi\Phi^{-1} = id_{(I,R,E)}$ and $\Phi^{-1}\Phi = id_{(I',R',E')}$.
On the other hand if Φ and Ψ are homomorphisms such that $\Phi\Psi = id_{(I,R,E)}$ and $\Psi\Phi = id_{(I',R',E')}$, then we know from the definition of homomorphism that

1. that Φ is a bijection $R \to R'$ (with inverse Ψ);

2. $I\Phi \subseteq I'$;

3. $I'\Psi \subseteq I$;

From (2) we get $dom(I) \subseteq dom(I')$. From (3) we get $dom(I') \subseteq dom(I)$. This means $I\Phi = I'$. Similarly we can prove $E = \Phi E'$.$\Box$

The proposition shows that the notion of isomorphism that we have defined is indeed the one we were looking for: two triples are isomorphic if the only difference between them is the choice of the set of referents. As we have explained, two such triples represent the same referent system. Therefore we will no longer distinguish them. At this point our only use for the notion of homomorphism of referent systems is that it allows us to make the notion of equivalence of referent systems precise. The reader who feels uncomfortable with these notions can rest assured: in the rest of the chapter he can simple use proposition 2.3 as a definition of the equivalence of referent systems. But we suspect that at some point we will have some other use for it.[13]

[13]We expect this for the following reason: in dynamic semantics we should not only be concerned with the question whether some sentence is a part of some text, but also which part it is exactly. So we are not interested in some 'part of' relation $\leq$, but we need to keep track of the precise location. This kind of distinction can easily be made in category theory. Therefore it seems that at some point the use of categories in text semantics will be inevitable. Also see Visser (1992c) for more details on the use of categories in dynamic semantics.

3.3.3 The merger

In the previous subsection we have explained what the import and export function are for. The export function tells which referents are passed on by the system and under which name, the import function tells which referents are needed by the system by giving their names. This is how the communication between variables in different parts of a text will be modelled (cf. section 3.3.4 for examples). With this explanation in mind, we define the merger of two referent systems as follows: (We use a dot "•" for the merger because that reminds us of the way sentences in discourse are merged into a text by the full stop.)

Definition 3.3.4 *Let two referent systems (I, R, E) and (I', R', E') be given. We define the merger of the two referent systems, $(I'', R'', E'') = (I, R, E) \bullet (I', R', E')$, as follows:*

1. $R'' = (R \oplus R')/\sim$

   where $\oplus$ stands for disjoint union and $\sim$ is the smallest equivalence relation such that for $r \in R$, $r' \in R'$ we have:

   $r \sim r'$ *iff* $rEI' = r'$.

   (We will write r for $r \in R$ as well as for the image of r in $R \oplus R'$ and also for the equivalence class of r in $(R \oplus R')/\sim$ if no confusion can arise.)

2. $xI'' = xI$ *if* xI *is defined;*

   $xI'' = xI'$ *if* xI' *is defined and* xI *is not defined and for no* $r \in R: \ r \sim xI'$*;*

   xI'' *is undefined otherwise.*

3. E'' *is defined dually, i.e.*

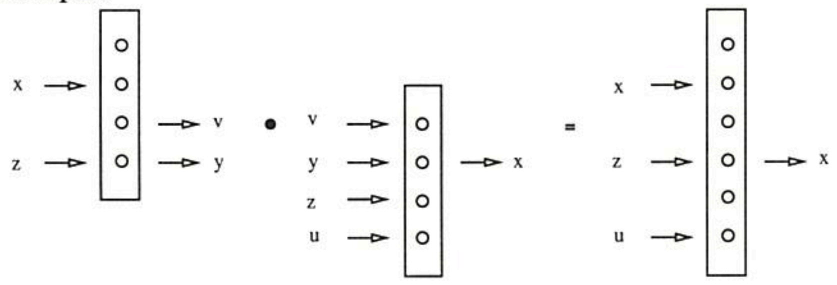
   $rE'' = rE'$ *if* $r \in R'$ *and* rE' *is defined;*

   $rE'' = rE$ *if* $r \in R$ *and* rE *is defined and for no* $r' \in R'$ $r \sim r'$ *and for no* $r' \in R'$ $r'E' = rE$*;*

   rE'' *is undefined otherwise.*

Before we discuss this definition, we will show in a picture how the merger works. In the picture the sets of referents are represented as vertical blocks. Their import names, if any, are found on the left hand side, their export names on the right hand side. We have tried to make this example such that you can see for each of the import clauses of (2) how they work. By duality a good example for the export clauses can then be obtained by turning the page upside down.

Example:



Note that it is the import of z by the first referent system that is preserved in the merger of the two systems. Also note that the variable called v has now become invisible: it has become a local variable.

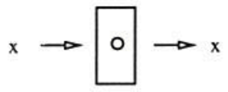
Now if we consider the definition, we see that clause (1) contains no surprises: two referents get identified iff a transaction can take place, i.e. iff $rEI' = r'$. In other words, two referents are identified iff they have the same name at the time of a merger. The new import and export functions are defined in clauses (2) and (3). To understand these definitions it is important to keep in mind that the left to right order is to correspond to the chronological order. Of course the merger of the two referent systems still exports all the referents the second referent system exports (the first part of clause 3). And if no transaction has taken place, then the export behaviour of the first referent system should be taken over by the merger as well. For, there will still be the possibility to export the referents that the first referent system provides—unless the second referent system already provides them.

A difficulty arises when the two referent systems both supply a referent with the same name, i.e., for some $r \in R$, $r' \in R'$ $rE = r'E'$. This will

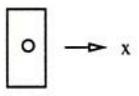
only cause problems, of course, if the second referent system does not import the referent r. For when this referent will be imported by the second referent system, the merger of the two systems will only try to export one referent, r'. But if $rE = r'E'$ and for no $r'' \in R:\ \ rEI' = r''$, then there will be a serious competition between the two systems for the export of a referent with name rE. It seems natural to assume that in this case the second referent system will win the competition since it will be 'closer' to the candidates that may want to import a referent called x. Hence $rE'' = rE'$ whenever rE' is defined (first option in 3), while we can only have $rE'' = rE$ if not $rE \in range(E')$. We have this priority rule for export functions, because the order of the referent systems is to reflect the chronological order. Hence, if some referent with name x is imported later on, then the importing referent system will not wait for the first referent system to make a transaction: it will just import the referent that the second system — that is closer — has to offer.
As we have said, I'' is defined dually. This is the first time that we can see that duality can save a lot of work. But because it is the first time we have given the details anyhow. By duality we know that I''^{-1} will be the export function of $(E'^{-1}, R, I'^{-1}) \bullet (E^{-1}, R, I^{-1})$. This means that it will behave like I and like I' (if there are no transactions) and that in case of a clash (i.e. $x \in dom(I) \cap dom(I')$) I will have priority over I'. As one can see, this is what is (2) gives us.
Note that we are only interested in referent systems up to isomorphism: we do not care about the particular set R that is used to represent the referents. Therefore it should be checked that the definition of the merger preserves isomorphism of referent systems. Given the characterisation in proposition 3.3.3 this is not difficult.

3.3.4 Referent systems in semantics

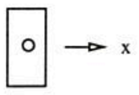
We want to use the referent systems for dynamic semantics: with each formula we will associate a referent system that tells us all about the variables that occur in the formula. We will do this in detail later, but we can already give some examples. A formula $\phi(x)$ with a free variable x will give rise to a referent system with a referent, r say, such that $xI = r$ and $rE = x$. The referent and its name are simply passed on. The referent system of $P(x)$, for example, will be: $(\{\langle x, r\rangle\}, \{r\}, \{\langle r, x\rangle\})$ or, in a picture:[14]



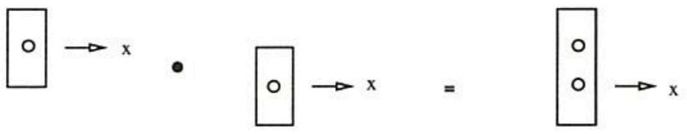
Quantified variables are exported, but they do not have to be imported. So $\exists x$ gets the referent system



Thereby $\exists x$ only allows for interaction with the context to the right, while $P(x)$ can interact both ways. By merging the two referent systems in the right order we get the referent system for $\exists x.P(x)$:



The priority rule for the export function will be used to handle repeated quantification: the formula $\exists x.P(x).\exists x Q(x)$ gets the referent system:



This is what we predicted: the first referent is still there, but it no longer has a name.

[14]In pictures we prefer to represent referents as (featureless) os. In the text we will still use letters r, r' etc.

3.4 Properties of referent systems

The purpose of this section is to study the formal properties of referent systems. First we will define some special referent systems. They are special for two reasons: first because they are probably the easiest ones you can think of and second because they are in a way the basic referent systems.

Definition 3.4.1 (Special referent systems) *For each finite set $V \in NOM$ we define:*

1. $Z = (\emptyset, \emptyset, \emptyset)$;
2. $O[V] = (\emptyset, V, \emptyset)$;
3. $I[V] = (id_V, V, \emptyset)$;
4. $E[V] = (\emptyset, V, id_V)$;
5. $T[V] = (id_V, V, id_V)$.

When $V = \{v\}$, we will omit the brackets.

Here we have chosen the names as (representations of) referents. Of course this is not essential (compare section 3.3.1): it is just a very convenient choice. Note also that we use capitals I, E here, that are also used for import and export functions. This is because here I and E also stand for 'import' and 'export'. Furthermore, T stands for transport and Z for zero.

We can use these special referent systems to show how calculations with referent systems work. Some easy results (and their duals) are in the following proposition.

Proposition 3.4.2 *Let $V, W \subseteq NOM$ be given.*

- *(The clauses on the left hand side are the special case where $V = W$.)*

(i)	$E[V] \bullet I[V] = O[V]$	$E[W] \bullet I[V] =$ $E[W \backslash V] \bullet O[W \cap V] \bullet I[V \backslash W]$
(ii)	$T[V] \bullet T[V] = T[V]$	$T[W] \bullet T[V] = T[W \cup V]$
(iii)	$E[V] \bullet E[V] = O[V] \bullet E[V]$	$E[W] \bullet E[V] =$ $O[W \cap V] \bullet E[W \cup V]$
(iv)	$I[V] \bullet I[V] = I[V] \bullet O[V]$	$I[V] \bullet I[W] =$ $T[V \cup W] \bullet O[V \cap W]$
(v)	$E[V] \bullet T[V] = E[V]$	$E[W] \bullet T[V] = E[W] \bullet T[V \backslash W]$
(vi)	$T[V] \bullet I[V] = I[V]$	$T[V] \bullet I[W] = T[V \backslash W] \bullet I[W]$

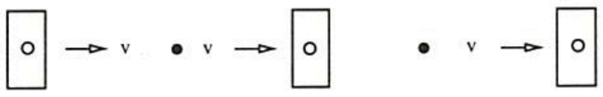
In $T[V] \bullet E[V]$, $I[V] \bullet T[V]$ and $I[V] \bullet E[V]$ no transactions are possible.

- *(Non-associativity)*

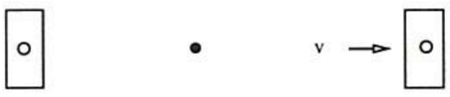
  $(E[V] \bullet E[V]) \bullet I[V] \neq E[V] \bullet (E[V] \bullet I[V])$ $(V \neq \emptyset)$;

  $E[V] \bullet (I[V] \bullet I[V]) \neq (E[V] \bullet I[V]) \bullet I[V]$ $(V \neq \emptyset)$.

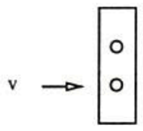
Proof:
Omitted. We will just give a picture of an example of a situation where the merger is not associative. We take $V = \{v\}$.



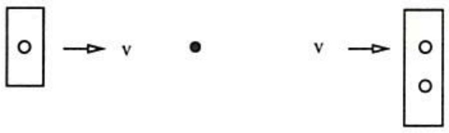
Either this reduces first to:



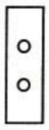
and then to:



or it reduces first to:



and then to:



□

The results of the proposition are easy, so we have omitted the proofs. On the other hand, they are representative for the way referent systems work, so we will discuss them in a little more detail. Note that the statements in the first column are just instances of the statements in the second column. In (i) we see that we can create local variables explicitly: first we export a variable, then it is imported. Note that the statements in (i) and (ii) are self-dual. In (iii) we find another way to create local variables: we already have variables called V and then we declare new variables called V. Now, by our priority rules, the old variables lose their names and become local. Clause (iv) is the dual of (iii).
The second item shows that the merger is not an associative operation. From a technical point of view this is an unpleasant property.[15] It will

[15]There are also methodological reasons for preferring an associative operation in text semantics. We will not go into that here: this will have to wait until part 2 of the

complicate the calculations with referent systems. It should be noted, however, that the example of non-associativity that we have here is rather subtle: it involves both a transaction between referent systems and a clash of names. Therefore we can still hope that there are natural classes of referent systems in which this does not occur and in which the merger is an associative operation. Let us consider a few of those natural classes.

Definition 3.4.3

1. *A referent system* (I, R, E) *is partially persistent (PP) iff* $IE \subseteq id_{NOM}$;
2. *A referent system* (I, R, E) *is extending iff* $dom(I) \subseteq range(E)$;
3. *A referent system* (I, R, E) *is faithful iff* $dom(I) = range(E)$.

These classes of referent systems are all defined by a condition on the way they handle the names of referents. This is a natural kind of condition to consider for referent systems, since this is what referent systems were introduced for. For example, a restriction on the number of referents in R would make much less sense. The first condition says that if a referent r is imported with a name, x say (i.e. $xI = r$), then, if the referent is also exported, its export name is still x (if $rE\downarrow$, then $rE = x$). Note that a referent can lose its name in a PP system, it is only a change of name that is not allowed.
The extending referent systems are such that the names that are imported are also exported. It is possible that there are also other export names or that some old name now stands for another referent, but the old names still have a reference.
Faithful referent systems have a fixed set of names. Again the reference of the names may change, but not the presence of the names.
These three classes of referent systems have the following closure property:

Proposition 3.4.4

1. *The PP referent systems are closed under merger.*
2. *The extending referent systems are closed under merger.*

thesis. (Also see Vermeulen (1993a), Visser (1992c).)

3. *The faithful referent systems are closed under merger.*

□

It is easy to see that the counterexample against associativity that we have, is built up from partially persistent referent systems. Therefore the merger is not an associative operation on partially persistent referent systems. With some effort it can be shown that merger is associative within the other two classes. We will not prove this now, because it is an easy consequence of a more general result:

Proposition 3.4.5 *Let referent systems σ, τ and ρ be given. Then:*

$$(\sigma \bullet \tau) \bullet \rho \;=\; \sigma \bullet (\tau \bullet \rho) \textit{ iff}$$
$$range(E_\sigma) \cap dom(I_\tau) \cap dom(I_\rho) \subseteq range(E_\tau) \cup dom(I_\sigma) \textit{ and}$$
$$dom(I_\rho) \cap range(E_\tau) \cap range(E_\sigma) \subseteq range(E_\rho) \cup dom(I_\tau).$$

This result is proved and discussed in the appendix 3.7. It gives rise to the following corollary:

Corollary 3.4.6

1. *The merger is not associative in the class of partially persistent referent systems;*

2. *The merger is associative in the class of extending referent systems;*

3. *The merger is associative in the class of faithful referent systems.*

□

The referent systems that will actually occur in our semantics are all partially persistent and extending. So we will get an associative merger in our semantics.
We conclude this section with an observation:

Proposition 3.4.7

- $Z = I[\emptyset] = E[\emptyset] = T[\emptyset] = O[\emptyset]$.

- *The class of partially persistent referent systems can be generated from referent systems of the form Z, $I[v]$, $E[v]$ and $T[v]$.*
- *The class of partially persistent and extending referent systems can be generated from referent systems of the form Z, $O[v]$, $E[v]$ and $T[v]$.*

□

This proposition shows that a lot of referent system can be seen as the result of some very basic actions, such as importing or exporting one referent. This gives us the possibility to check the properties of the most interesting classes of referent systems inductively.

3.5 Dynamic semantics with referent systems

In this section we show how we can use referent systems for dynamic semantics. In fact we will use referent systems in the semantics of $\mathcal{L}_{DPL}$, our language for dynamic predicate logic.[16] The referent systems will be used to interpret the variables of this language: the syntactic variables of $\mathcal{L}_{DPL}$ are the names in NOM.
We will call the interpretations of $\mathcal{L}_{DPL}$ *discourse structures* (*DS*s). They will consist of a referent system and a set of assignments of values to the referents in the system. We use assignments to referents, not to their names, because the information that we find in $\mathcal{L}_{DPL}$ is not information about the syntactic variables but about the 'real' variables that have the syntactic variables as names.
The formal definition is given below. As usual we will assume that the domain of interpretation of our model, D, is given. The notation $f|_X$ is used for the restriction of the function f to domain X.

Definition 3.5.1 (Discourse Structures)

1. *A discourse structure δ is a pair $(\sigma_\delta, F_\delta)$, where $\sigma_\delta = (I_\delta, R_\delta, E_\delta)$ is a referent system and F_δ is a set of assignments from R_δ to D;*

[16]Recall that in this language we can define abbreviations such as $\forall x(\phi) \equiv (\exists x \to \phi)$ and $\neg(\phi) \equiv (\phi \to \bot)$.

2. *A homomorphism of discourse structures* $\Phi : \delta \to \varepsilon$ *consists of a homomorphism of referent systems* $\Phi_1 : \sigma_\delta \to \sigma_\varepsilon$ *and a function* $\Phi_2 : F_\varepsilon \to F_\delta$ *such that for all* $f \in F_\varepsilon : \quad f\Phi_2 = \Phi_1 \circ f$.

   (Thereby $\forall f \in F_\varepsilon \exists g \in F_\delta : \quad g = \Phi_1 \circ f$.*)*

   We say: Φ_2 *gives the restriction of* f *to* σ_δ *according to* Φ_1.

3. *If* δ *and* ε *are discourse structures, then the merger* $\gamma = \delta \bullet \varepsilon$ *where* $\sigma_\gamma = \sigma_\delta \bullet \sigma_\varepsilon$ *and*

   $F_\gamma = \{f : R_{\delta\bullet\varepsilon} \to D : f|_{R_\delta} \in F_\delta$ *and* $f|_{R_\varepsilon} \in F_\varepsilon\}$.

$$\begin{tikzcd}
R_\delta \arrow[rr, "\Phi"] \arrow[dr, "f\Phi_2"'] & & R_\varepsilon \arrow[dl, "f"] \\
& D &
\end{tikzcd}$$

Again we are only interested in the referent systems up to isomorphism. The notion of homomorphism of discourse structures shows how we can extend this notion to discourse structures. For the homomorphisms of discourse structures Φ we introduce a notation convention: we will simply use Φ instead of Φ_i if no confusion can arise.
In the discourse structures we have sets of total functions: they are total on the set of referents of the structure. If we merge two discourse structures, we glue these total assignment together, if possible. The result is a total assignment on the new referent set.
The definition of homomorphism, as we have given it, requires some explanation. The idea is that there is a homomorphism from δ to ε, iff ε contains more information than δ. Therefore it is natural that we require that ε has 'more' referents and that ε has better access to the referents. This is guaranteed by the presence of a homomorphism of referent systems. It is also natural that the more informative *DS* should allow less assignments. This is what the mapping Φ_2 takes care of: it guarantees that no new (configurations of) values are allowed.

Now we can interpret $\mathcal{L}_{DPL}$. In fact we have already seen most of the crucial examples in section 2, where we defined referent systems. Here we will extend the apparatus with the construction of the referent system for implications. Once this is done we can straightforwardly add the second component of the interpretation, the set of assignments.

Definition 3.5.2

1. *For two referent systems σ, τ we define $(\sigma \to \tau)$ as follows:*
   $(\sigma \to \tau) = (I, R, E)$,
   where $I = I_{\sigma\bullet\tau}$, $R = range(I)$ and $E = I^{-1}$.
2. *For two DSs δ, ε we define $(\delta \to \varepsilon)$ as follows:*
   $(\delta \to \varepsilon) \; = \; ((\sigma_\delta \to \sigma_\varepsilon), \; F)$
   where $f \in F$ iff $f : R_{(\sigma_\delta \to \sigma_\varepsilon)} \to D : \; \forall g \in F_\delta :$
   $f|_{dom(g)} \leq g \to \exists h \in F_{\delta\bullet\varepsilon} : \quad f \leq h \;\; \& \;\; g \leq h.$

We will discuss this definition shortly, but first we give the interpretation of $\mathcal{L}_{DPL}$. (Here the notation X^Y is used for all the functions from Y to X and $\mathbf{P}$ is the extension of P.)

Definition 3.5.3 *Each formula of $\mathcal{L}_{DPL}$ is interpreted as a discourse structure according to the following clauses:*

- ∘ $[\![\bot]\!]_{ds} = \; (Z, \emptyset)$;
- ∘ $[\![\exists x]\!]_{ds} = \; (E[x], D^{\{x\}})$;
- ∘ $[\![P(x_1, ., x_n)]\!]_{ds} = (T[x_1, ., x_n], \{f \in D^{\{x_1,.,x_n\}} : \langle x_1 f, ., x_n f\rangle \in \mathbf{P}\}$;
- ∘ $[\![\phi.\psi]\!]_{ds} = [\![\phi]\!]_{ds} \bullet [\![\psi]\!]_{ds}$;
- ∘ $[\![(\phi \to \psi)]\!]_{ds} = [\![\phi]\!]_{ds} \to [\![\psi]\!]_{ds}$.

A first remark about this definition is that all the referent systems that we find in it are partially persistent and extending. From this it follows that the merger is an associative operation on $\mathcal{L}_{DPL}$ interpretations.
We can see that the sets of assignments for simple formulas are just the sets that were to be expected: the formula $\exists x$ does not restrict the values

of the assignments and an atomic formula, $P(x)$ say, gives rise to the obvious restriction that the value on the referent of x should have property P. For the conjunction of formulas, $\phi.\psi$, we have to glue together the assignments of $[\![\phi]\!]_{ds}$ and $[\![\psi]\!]_{ds}$. This will only be possible for some assignments. For example, if we try to glue together assignments from $[\![P(x)]\!]_{ds}$ with assignments from $[\![\neg P(x)]\!]_{ds}$, we will see that this cannot be done: any assignment from $[\![P(x)]\!]_{ds}$ will assign to the referent called x a value in $\mathbf{P}$, while the assignments in $[\![\neg P(x)]\!]_{ds}$ will assign values outside $\mathbf{P}$ to the referent called x. By the definition of the merger, these referents called x are to be identified in $[\![P(x).\neg P(x)]\!]_{ds}$. Hence no assignments will survive the glueing procedure. As a result we get: $[\![P(x).\neg(P(x))]\!]_{ds}= (T[x], \emptyset)$.
The clause for implication requires some explanation. We see that in the referent system of an implication, $(\phi \to \psi)$, we find the referents that are imported in $\phi.\psi$. So we get, for example:

for $(P(x) \to Q(x))$, a singleton set, $\{x\}$ say;

for $(\exists x.P(x) \to Q(x))$, the empty set;

for $(P(x) \to \exists x.Q(x))$, a singleton set containing just the first referent called x.

In general we get the referents that correspond to the free variables in $(\phi \to \psi)$ (with the *DPL* notion of binding and freeness in mind). These 'free referents' are simply transported through the referent system: they have the same im- and export name.
The assignments that are allowed in $(\phi \to \psi)$ are the assignments of which any assignment that satisfies ϕ can be extended to an assignment that satisfies $\phi.\psi$. This is just the usual construction from *DRT* and *DPL*. There are some subtleties involved that have to do with the domains of the assignments that we have to consider.

First we have to restrict a function f that is defined on $R_{\sigma_\delta \to \sigma_\epsilon}$ to R_δ. This is necessary for implications like $(P(x) \to Q(y))$, where there are free referents that do not occur in the antecedent.

Then any extension of the resulting assignment that satisfies ϕ should be extendible to an assignment that satisfies $\phi.\psi$.

> But there is a further requirement on this assignment: it has to agree with the original f on the free variables of ψ. That is why we also have to demand $f \leq h$ in the definition.

This can be illustrated with the formula $(\exists x.P(x) \rightarrow Q(x,y))$: f will be defined on $\{y\}$. First this f will be restricted to the empty function. Now every extension of the empty function that assigns to x a value in $\mathbf{P}$, needs an extension h such that $(xh, yh) \in \mathbf{Q}$. But we are only interested in extensions h that assign to y the value yf.

The definition of $[\![(\phi \rightarrow \psi)]\!]_{ds}$ can be motivated further: we can compare it with the notion of entailment that we will develop for discourse structures. We will see that the implication that we have defined is just the implication that goes with the notion of entailment for *DSs*.
In the definition of entailment we will touch upon the issue of partiality. So this is a good opportunity to make some general remarks about partiality and referent systems. Since a *DS* represents the information expressed by some discourse fragment, it will, in general, only give partial information about the underlying model. For not every discourse fragment gives complete information about the whole world. This is the first kind of partiality we have to distinguish.
We have seen already that this does not give rise to the use of partial functions in our *DSs*: all assignments in a *DS* are defined on all the referents in the *DS*. The partiality is expressed in other ways: first by the limited (finite) number of referents that we have in one *DS*. This reflects the fact that we only have information about a limited number of objects. Then there is the fact that we work with a set of assignments, not with one assignment. By allowing ourselves to consider several values for one referent we reflect the partiality of our information about this referent: we do not always know exactly which entity the referent is a stand in for, we just know some conditions that limit the range of possibilities.
This kind of partiality is important, of course. But it is not—or should not be—typical for dynamic semantics.
However, there also is a kind of partiality that is directly related to the dynamics. It has to do with the context components of the *DS*s. In a *DS*, $((I, R, E),\ F)$ say, not only F gives information about the referents in R. I and E also reveal some relevant facts. If a referent is exported, $rE = x$ say, then this means that more information about r can be given

in what is to follow. If a DS imports some referent, $xI = r$ say, then this means that this DS will depend on the context—some other DS—to get the referent r.
This is important information about r. In particular if a referent is *imported* this reveals a strong kind of partiality of our information: in the context component of the DS it is not really known where r comes from. The context component is not saturated. This kind of partiality, *partiality-as-context-dependency*, is typical of dynamic semantics. It corresponds more or less to the distinction of free and bound variables in traditional logic.
With this kind of partiality in mind we define entailment as follows:

Definition 3.5.4

1. *Let two referent systems σ and σ' be given. Then:*

   $\sigma \models \sigma'$ *iff* $dom(I_{\sigma'}) \subseteq range(E_\sigma)$

2. *Let two DSs δ and δ' be given. Then: $\delta \models \delta'$ iff*

   $\sigma_\delta \models \sigma_{\delta'}$

   and

   $\forall f \in F_\delta : f \in F_\delta \; \exists f' \in F_{\delta'} : \;\; I_{\delta'} f' \;\subseteq\; E^{-1} f.$

3. *For two formulas ϕ and ψ we define entailment as follows:*

   $\phi \models \psi$ *iff* $[\![\phi]\!]_{ds} \models [\![\psi]\!]_{ds}$.

4. *A formula ϕ is valid iff* $(Z, \{\emptyset\}) \models [\![\phi]\!]_{ds}$.

(Recall that $Z = (\emptyset, \emptyset, \emptyset)$.)
In clause (1) $\sigma \models \sigma'$ can be read as: σ provides a suitable context for σ': the context set-up by σ supports σ'. In general in these clauses one can read the $\models$ sign as 'supports'. Here we are talking about truth-conditional and contextual support at the same time.
Of course we want $\delta \models \delta$ to express that the information in δ' follows from the information in δ. But, as we have seen, δ' does not only give information about $R_{\delta'}$, it also requires information about the origin of the referents in $range(I_{\delta'})$. This means that if $\delta \models \delta'$, then these referents have to be supplied by δ. In other words, we want $\sigma_\delta \models \sigma_{\delta'}$. But we also want all the assignments in F_δ to have a corresponding assignment in

$F_{\delta'}$ to guarantee that δ' allows more values for the referents than δ. For otherwise there is more information about those referents in δ' than in δ. This is what clause (2) says.
Now we can look at the definition of $[\![(\phi \to \psi)]\!]_{ds}$ again. Our observations about partiality do not apply here. A formula $(\phi \to \psi)$ can occur in a context in which the imported referents of ψ have been introduced properly, even if this is not done by the formula ϕ.
Think for example of

$$\exists x.(\exists y.P(y) \to Q(x,y)).$$

We have argued that $[\![\exists y.P(y)]\!]_{ds} \models [\![Q(x,y)]\!]_{ds}$ cannot hold, because the referent of x is not supplied by $[\![\exists y.P(y)]\!]_{ds}$. Similarly, the formula $(\exists y.P(y) \to Q(x,y))$ cannot be valid. But this does not mean that the formula $(\exists y.P(y) \to Q(x,y))$ does not have a meaning. It just means that the context component of its meaning will have to import a referent called x. In this situation the best relation between inference and implication that we could hope for is:[17]

$$[\![\phi]\!]_{ds} \models [\![\psi]\!]_{ds} \text{ iff } F_{[\![(\phi\to\psi)]\!]} \neq \emptyset \quad \& \quad I_{[\![(\phi\to\psi)]\!]} = I_{[\![\phi]\!]}.$$

This says that truth-conditionally—i.e. in terms of assignment sets—both situations are equivalent, but for entailment we have the extra requirement that the context component of ϕ supports the context component of ψ.
It can be checked that this is indeed what we get. This gives another justification of our definition of implication: it is just the internalisation of the notion of validity for *DSs*.
Note that our concern with the partiality of the context is not standard in *DPL* or *DRT*. Above we have argued that this kind of partiality is typically a concern of a dynamic semantics, but nevertheless it is usually ignored in presentations of *DRT* and *DPL*. We will see in the next section that our interpretation of implications corresponds to the familiar notions in *DRT* and *DPL*. This will, at the same time, make clear how our notion of entailment relates to what we find in those formalisms: they agree up to partiality.

[17]Here we use the notation $[\![\phi]\!]_{ds} = (I_{[\![\phi]\!]}, R_{[\![\phi]\!]}, E_{[\![\phi]\!]})$.

3.6 DS, DRS and DPL

In this section we compare the *DS* semantics, which uses referent systems, with the relational *DPL* semantics and the representational *DRT* semantics. *DPL* likes to see its variables as just syntactic entities, in the Fregean sense. On the other hand the notion of discourse marker that we find in *DRT* seems to be closer to the notion of a storage facility than to the strictly syntactic notion of variable.
Therefore we would expect to find that we get something very much like *DRT* if we ignore the names of the referents in our referent systems, while ignoring the referents and concentrating on the names instead should give us something very close to *DPL*. We will see that this is indeed what we find.

3.6.1 DRT

Technically the comparison with *DPL* will be straightforward, since the language we use is the *DPL* language. *DRS*s cannot be compared with *DPL* formulas as easily: *DRSs* are just a different kind of things. They live on an intermediate level between syntax and semantics: in a *DRS* we find both the discourse markers, which also belong to the semantic domain, and the conditions on these markers that live on the level of syntax. Hence for the comparison it is convenient to introduce a similar intermediate level in between the *DPL* formulas and *DS* interpretations. We will call the intermediaries *DPSs*, *Dynamic Predicate Structures.* The *DPSs* can be defined inductively as follows:[18]

Definition 3.6.1 *We define DPSs and DPS-conditions by a simultaneous induction:*

1. *an atomic DPL-formula is a DPS-condition;*
2. *if δ and δ' are DPSs, then $(\delta \rightarrow \delta')$ is a DPS-condition;*
3. *for each referent system σ and set of DPS-conditions C, (σ, C) is a DPS.*

Now we can give for each *DPL* formula a *DPS* that represents it.

[18] This definition closely follows the definition of a *DRS* in Zeevat (1991a).

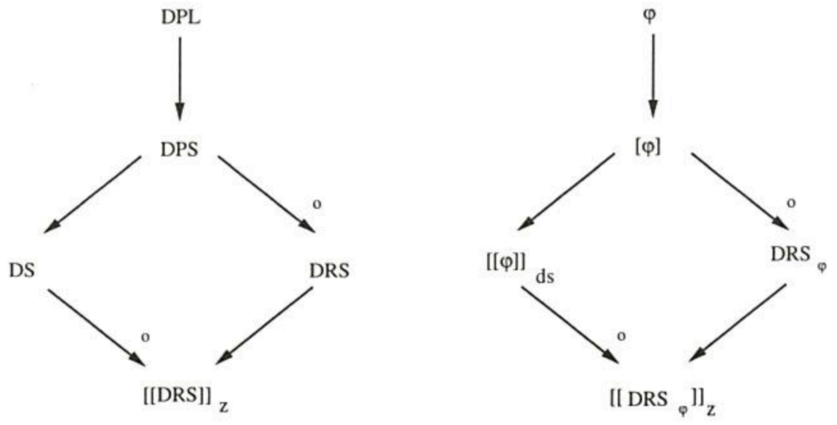
Definition 3.6.2 *For each DPL formula ϕ, we define $[\phi] = (\sigma_\phi, C_\phi)$, the DPS of ϕ, as follows:*

$$\begin{array}{ll}
[\bot] & = (Z, \emptyset) \\
[\exists x] & = (E[x], DOM^{\{x\}}); \\
[P(x_1, \ldots, x_n)] & = (T[\{x_1, \ldots, x_n\}], \{P(x_1, \ldots, x_n)\}); \\
[(\phi \to \psi)] & = ((\sigma_\phi \to \sigma\psi), \{([\phi] \to [\psi])\}); \\
[\phi.\psi] & = (\sigma_\phi \bullet \sigma_\psi,\ C_\phi \bullet C_\psi).
\end{array}$$

Here $C_\phi \bullet C_\psi$ indicates that the names of the variables as they occur in C_ϕ and C_ψ should follow the identifications and dis-identifications of $\sigma_\phi \bullet \sigma_\psi$. For example, $[\exists x.P(x)] = (E[x], \{P(x)\})$ and $[\exists x.Q(x)] = (E[x], \{Q(x)\})$. But if we merge the referent systems the two referents x will not be identified. We have to make this visible in the conditions:

$$[\exists x.P(x).\exists x.Q(x)] = (O[x'] \bullet E[x], \{P(x'), Q(x)\}).$$

The interpretation of the *DPSs* as *DSs* is obvious: we simply replace the set of conditions C by the set of assignments (on the referents) that satisfy these conditions.
Now we have a representational level in our interpretation with referent systems and we are able to compare the referent systems approach with *DRT*.



$$\begin{tikzcd}
 & \text{DPL} \arrow[d] & \\
 & \text{DPS} \arrow[dl] \arrow[dr, "\text{o}"] & \\
\text{DS} \arrow[dr, "\text{o}"] & & \text{DRS} \arrow[dl] \\
 & {[[\text{DRS}]]_z} &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \varphi \arrow[d] & \\
 & {[\varphi]} \arrow[dl] \arrow[dr, "\text{o}"] & \\
{[[\varphi]]_{ds}} \arrow[dr, "\text{o}"] & & \text{DRS}_\varphi \arrow[dl] \\
 & {[[\text{DRS}_\varphi]]_z} &
\end{tikzcd}$$

($[\![DRS]\!]_z$ stands for the class of *DRS*-interpretations as discussed in section 3.2 and defined in the introductory chapter (also see Zeevat (1991a)).)

The diagram gives the picture that one should keep in mind. Most of the links in the diagram have not yet been defined properly. But their formal definition can easily be constructed from what has been said so far. The $\circ$-links, for example, rely on the following operation:

Definition 3.6.3 *For each referent system $\sigma = (I, R, E)$ we define the reduction of σ to a set as follows:*

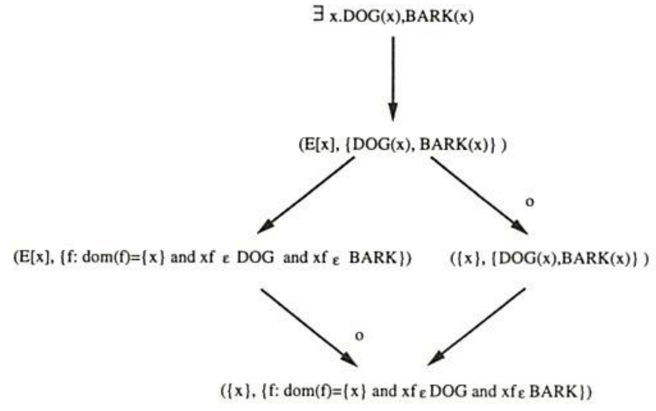
$$set((I, R, E)) \;=\; \{r \in R : \; r \in dom(E) \;\;\&\;\; r \notin range(I)\}.$$

Now if we want to make *DRSs* out of *DPSs* or *DRS*-interpretations out of *DSs*, we just have to apply this set function to the first component of the *DPS* or *DS*. Our claim is, of course, that the diagram commutes. We will make this claim precise in the following proposition, but we will omit the proofs. They are tedious but straightforward.

Proposition 3.6.4 *For each DPL-formula ϕ we claim that:*

$\lvert[\phi]\rvert \;=\; [\![\phi]\!]_{ds}$	*DPSs are intermediate between DPL and DSs;*
$\lvert[\phi]^\circ\rvert \;=\; \lvert[\phi]\rvert^\circ$	*the DRS-interpretation of a reduced DPS is the reduced DS.*

We give an example of all the links in the diagram:



$$\begin{tikzcd}
 & \exists\, \text{x.DOG(x),BARK(x)} \arrow[d] & \\
 & \text{(E[x], \{DOG(x), BARK(x)\})} \arrow[dl] \arrow[dr, "\text{o}"] & \\
\text{(E[x], \{f: dom(f)=\{x\} and xf } \varepsilon \text{ DOG and xf } \varepsilon \text{ BARK\})} \arrow[dr, "\text{o}"] & & \text{(\{x\}, \{DOG(x),BARK(x)\})} \arrow[dl] \\
 & \text{(\{x\}, \{f: dom(f)=\{x\} and xf } \varepsilon \text{ DOG and xf } \varepsilon \text{ BARK\})} &
\end{tikzcd}$$

Our conclusion from this comparison with *DRT* is that *DSs* are 'just' *DRSs* with import and export functions. We have a set of referents and a set of conditions on these referents, just as in DRT. But we also have import and export functions that make our manipulations of these referents explicit.

These manipulations are crucial for the semantics of anaphora. They handle the question which referents are to be identified and which referents are to be kept distinct. But precisely this point is usually left implicit in formulations of DRT. There it is usually assumed that we have automatically chosen the referents in a way that solves all problems. For example, it is usually assumed that the sets of new discourse markers U and U' are disjoint whenever we merge two *DRS*s (U, C) and (U', C'). This is not unreasonable if discourse markers are indeed to be compared with our referents. If they are indeed just featureless storage facilities that we have *created* during the interpretation, then—by definition—they cannot be the same.

But if referents are indeed featureless, then it is unclear how we can work with them at all: if they have no features we cannot recognise them. For example, we cannot see from two markers whether they have to be identified because of some anaphoric link unless these markers have a feature that shows this. The import and export function add such features, names, and thereby make it possible to be explicit about the interaction of the referents.

So although these details are usually ignored in formalisations of DRT, we see that this is not necessary. In our machinery we have not only the featureless storage facilities, but also labels for them, that allow us to do something with them.

Apart from allowing us to be explicit about the manipulation of referents, there are also some other pleasant consequences of our set-up. Since we have the referents we inherit all the advantages of DRT: an intuitively appealing story about how we store antecedents in memory, a situation where at each point we will only have a finite number of these antecedents, etc. Furthermore, now that we have an explicit representation of the syntactic variable *and* the semantic variable, we are in a position where we can distinguish properties of variables as syntactic objects and properties of variables as storage facilities. For example, while it is by definition impossible to create the same referent twice, we can use the same words as anaphors over and over again. Therefore it is reasonable to allow re-using element of NOM, syntactic variables, although the idea of re-using a semantic variable makes no sense. So we have obtained a more flexible machinery in which we can implement the *DRT*-ideas about anaphora in a more felicitous way.

3.6.2 DPL

The comparison with the relational *DPL* semantics is technically more straightforward. We have an interpretation of the *DPL* language as relations and another one as *DSs*. We will show that the relational interpretation can be obtained from the *DS* interpretation. We will do this in two steps: first we construct a relational interpretation in terms of referent systems from the *DS* interpretation. Then we will see that these new relations are closely related to the relational interpretation that we know already. So the picture is as follows: (*DPR* is the class of *Dynamic Predicate Relations*, that we will define shortly; $[\![DPL]\!]_{gs}$ the usual relational interpretation; $[\![DPL]\!]_{pgs}$ the extension thereof to partial functions.)

$$\begin{tikzcd}
\text{DPL} \arrow[r] & \text{DS} \arrow[r, leftrightarrow] & \text{DPR} \arrow[d] \\
 & & {[[\text{DPL}]]_{\text{pgs}}} \arrow[d] \\
 & & {[[\text{DPL}]]_{\text{gs}}}
\end{tikzcd}$$

At the *DPR* level we find pairs (σ, f) consisting of a referent system σ and an assignment f on the referents of that referent system. We create relations on these objects from *DSs* as follows:

Definition 3.6.5

1. *For each DS* $\varepsilon = (\sigma_\varepsilon,\ F_\varepsilon)$ *we define the binary relation* $[\varepsilon]$ *as follows:*[19]

   $(\sigma, g)[\varepsilon](\sigma', h)$ *iff* $dom(I_{\sigma_\varepsilon}) \subseteq range(E_{\sigma_\varepsilon})$ & $\sigma' = \sigma \bullet \sigma_\varepsilon$ & $h = g \oplus f$ *for some* $f \in F_\varepsilon$.[20]

2. *For each formula* ϕ *we define the relation* $[\phi]$ *as follows:*

   $[\phi] \ = \ [[\![\phi]\!]_{ds}]$.

The relation $[\phi]$ works with assignments that are defined on referents. The usual relational interpretation is defined on assignments that are defined on (all) variable names. We will obtain this kind of relation in two steps: first we switch from referents to names and then we switch from partial to total functions. The switch to names is made as follows:

Definition 3.6.6

1. *For each binary relation* R *on* DPR*-pairs we define the following relation on partial assignments* $f : NOM \hookrightarrow DOM$.

   $f|R|_p g$ *iff* $\exists(\sigma, f'), (\sigma', g') : \ (\sigma, f')R(\sigma', g')$ & $f = E_\sigma^{-1} f'$ & $g = E_{\sigma'}^{-1} g'$.

2. *For each* DPL *formula* ϕ *we define a relation on partial assignments as follows:*

   $|\phi|_p \ = \ |[\phi]|_p$.

Definition 3.6.6 gives us an interpretation of formulas ϕ as relations on partial functions $NOM \to DOM$. From this relational interpretation we obtain a relation on total assignments $NOM \to DOM$ by restricting $|\phi|_p$ to total functions. This gives us the usual relational interpretation. We will prove this shortly, but first we try to gain some insight in the relations $[\phi]$.

[19]Be careful: in this subsection we use the same brackets as in the previous subsection, but with a different meaning!

[20]Here $\oplus$ glues g and f together.

Lemma 3.6.7 *Let (σ, g), (τ, h) be suitable pairs. Then:*

$(\sigma,g)[\bot](\tau,h)$	*iff*	$g \neq g$;
$(\sigma,g)[P(x)](\tau,h)$	*iff*	$\sigma = \tau \ \& \ g = h \ \& \ xI_\tau h \in \mathbf{P}$;
$(\sigma,g)[\exists x](\tau,h)$	*iff*	$\tau = \sigma \bullet E[x] \ \& \ h = g \cup \{(xI_\tau, d)\}$ *for some* $d \in D$;
$(\sigma,g)[\phi.\psi](\tau,h)$	*iff*	$\tau = \sigma \bullet (\sigma_\phi \bullet \sigma_\psi) \ \& \ \exists f \in F_{\phi.\psi}: \ g \oplus f = h$;
$(\sigma,g)[(\phi \to \psi)](\tau,h)$	*iff*	$\tau = \sigma \bullet \sigma_{(\phi\to\psi)} \ \& \ h = g \ \& \ \forall(\rho,k):$ $(\sigma,g)[\phi](\rho,k) \ \exists(\mu,l):(\rho,k)[\psi](\mu,l)$.

We omit the proof of the lemma.
Now we are ready for the following proposition (proof in the appendix):

Proposition 3.6.8 *Let g and h be partial assignments $NOM \hookrightarrow DOM$. Then:*

$g\vert\bot\vert_p h$	*iff*	$g \neq g$;
$g\vert P(x)\vert_p h$	*iff*	$x \in dom(g) \ \& \ g = h \ \& \ xh \in \mathbf{P}$;
$g\vert\exists x\vert_p h$	*iff*	$x \in dom(h) \ \& \ (y \neq x \ \Rightarrow \ xg = xh)$;
$g\vert\phi.\psi\vert_p h$	*iff*	$\exists k: \ g\vert\phi\vert_p k \ \& \ k\vert\psi\vert_p h$;
$g\vert(\phi \to \psi)\vert_p h$	*iff*	$g = h \ \& \ \forall k: \ g\vert\phi\vert_p k \ \Rightarrow \ \exists l: \ k\vert\psi\vert_p l$.

The proposition says that the usual clauses for the relational interpretation of DPL define the relational interpretation with partial functions. From this it follows that the restriction of $|.|_p$ to total assignments really is the usual DPL interpretation.
The proof of the proposition will be given in some detail in the appendix, partly because there are some interesting constructions involved, partly also because we want to make up for the lack of detail in our proofs so far. From the proposition it is clear that the restriction of $|.|_p$ to total functions is indeed the usual relational DPL interpretation.
This means that we indeed get DPL by forgetting the referents themselves and retaining only their names. This confirms that DPL sticks to the Fregean notion of variable.
In DPL the basic objects in the semantics are the assignments of values to variable names. These objects are used to model the way in which we store antecedents in memory when we interpret texts with anaphors. Of course it would be better, for an intuitively acceptable explanation of this process, if finite assignments were used instead of total, infinite assignments (cf. Fernando (1991a) for discussion). But this requires but a small adaptation of the standard formulation of DPL-semantics.

Another interesting point is that DPL-variables can be re-used. This is what we can expect with a syntactic notion of variable: just as pronouns (and other anaphoric expressions) in natural language, a variable name can be used over and over again with different meanings in different contexts. But since DPL does not distinguish the variable name from the 'real' variable, the re-use of a variable name can have nasty side effects. For whenever we give a new use to a variable name x, with $\exists x$, we are forced to forget the information that was previously attached to that name. In the DSs, where we can also re-use variable names, such a re-use will only result in the creation of a local variable, a variable that no longer has a name. Thereby we no longer have access to that variable, but we will not be forced to throw this variable itself away and we save the (truth-conditional) information that was stored in it. In DPL, however, we do not have such an option. Whenever we re-use a variable we automatically lose the information that was attached to it. This causes the *non-eliminativity* problem, as discussed in the previous chapter (also see Groenendijk and Stokhof (1991b) and Vermeulen (1993c)).
It has been suggested that this problem for DPL can easily be prevented by using different variables all the time, but this is an option that really does not go very well with a syntactic notion of variable. Syntactic elements, lexical items, typically *can* be used over and over again. It seems unelegant to put a semantically motivated restriction on the syntax, especially since the whole problem can be prevented by using the machinery developed here. Again we see that having explicit representations of both the syntactic and the semantic variable makes the machinery more flexible and allows us to give a natural representation of all the phenomena involved.

We conclude that the distinction between variables and variable names allows us to formulate the crucial ideas about the dynamic semantics of anaphora. It makes it possible to show explicitly how the link between an anaphor and its antecedent is established. Furthermore, our semantics with referent systems has enabled us to keep two kinds of tasks separate: the task of manipulating the variables can be described separately from the task of computing the (truth-conditional) result of these manipulations.
This distinction also has the advantage that it allows us to recognise the fact that variables and variable names are different kinds of things with

different kinds of behaviour. This way we can do justice both to our intuitions about syntax and to our intuitions about semantics.

3.7 Discussion

In this chapter we have developed techniques for passing on information that is stored in variables. The techniques are of general interest for all situations where information is manipulated, but were designed with a special application in mind: anaphora. We have shown that the machinery of the referent systems can compare with the two major alternatives in this field: *DRT* and *DPL*.

Still some questions about the precise relation between our formal machinery and the situation in natural language remain that we would like to go into in some more detail. These questions are of a rather general nature and apply not only to our semantics for anaphora with referent systems, but to any formal treatment of anaphora. Still we feel we have to say something about these questions, since we have noticed that confusion about these general points has led to misjudgements of our intentions and our results.

Sometimes the aims of formal semantics are formulated in terms of purely practical problems, such as translating natural language into a formal language. Although the idea of translation of natural language has provoked many interesting developments in formal semantics, we do not feel that it is correct to judge all developments in semantics from this perspective. Thus the question which formalism is better for semantics, for us, is not the same as the question which formalism allows the smoothest translations. There can be improvement in the semantic representation without much improvement in terms of translation. Perhaps the system that we have developed is not much of an improvement from the translational point of view, but we have argued that it *does* improve our representation of the situation in natural language, indeed we would argue that it gives clearer insight into the mechanisms operating in natural language.

To make the point plain, we consider the issue of re-occurrence of variables. Arguably the translational perspective has led to confusion here. Some problems in dynamic semantics, such as the eliminativity problem (cf. chapter 2, Groenendijk and Stokhof (1991b)), are connected with the possibility of using a variable name more than once. Now, it has

been suggested that these problems do not require much attention since these situations can easily be avoided by a suitable translation procedure: simply choose new variables as often as you can in your translation (cf. Dekker (1993)). This is of course a sensible remark from a typically translational perspective on semantics. However, from the point of view of semantic representation in general it does not make much less sense. The strategy to avoid problems in the semantics representation by choosing 'suitable translations' is unsatisfactory for several reasons. Let's look at the eliminativity problem first: the point here is that by choosing a new variable for each (indefinite) noun phrase we obtain translations which have the required eliminativity property. So, if good translations are all we are after in semantics, the eliminativity problem is hereby solved. But as we have seen in the previous chapter, there is no real connection between the precise choice of variables and the eliminativity property that we are looking for: in chapter 2 we gave an eliminative semantics for the full *DPL*-language. This shows that in this particular situation a purely 'practical' attitude towards formal semantics leads to a misconception of the issues involved: the translation outlook suggests that eliminativity is a choice-of-variables problem, but it really is a completely independent issue.
There is another remark to be made about this fresh variable strategy for translation. If we use such an approach for the analysis of anaphora in natural language, then we are bound to misrepresent at least one other important aspect of this phenomenon: anaphora is a typical example of a situation where one and the same lexical item can have radically different denotations in different parts of a text. Think, for example, of a pronoun such as *it*. This is typically a word that we use all the time, but not always with the same denotation in mind: its denotation varies with its antecedent. In other words, one of the remarkable things about anaphors is that their denotation varies with the (linguistic) context. To us this seems to be one of the crucial properties of natural language anaphora and one of the main challenges for a formal semantics is to represent precisely this property. On the formal side the counterpart of this challenge is given by the variables: in the formal machinery it are the variables that can obtain radically different denotations depending on the context. It is clear, however, that a fresh variable translation, however practical it may be, will obscure this issue: if we choose fresh variable names all the time, then we will end up with representations in which each variable has

only one denotation. This is another example of how the practical and the principled questions in semantics can lead to different choices. It is clear that with respect to this phenomenon our formal system allows us to stay 'closer to natural language', which seems to be a good thing even if it has no clear practical use.
Of course lots of points still remain where our formal system deviates from the system we use in natural language. For example, all the connections between anaphors and antecedents are represented in our system by identity of variable names. But in the natural language an anaphoric expression is typically *not* syntactically identical with its antecedent.
Such a deviation from real life is potentially a bad thing. But in this case it is not so clear that there is actually something wrong with our system. In the formal representation we distinguish radically between on the other hand the things that are said about some discourse marker, the information content which is typically represented by some predicate, and on the other hand the machinery by which we manipulate this information, the referent names in *NOM*. This radical distinction is not made in natural language. It seems that there the same words serve both purposes. If we say, for example:

> A man and a woman came in. The man was wearing a black coat.

Then the word *man* does not only serve to give information, but can also be used when we want to *name* the referent associated with it. This is what we do when we say *the man*. So in natural language we sometimes do two things with one word. This does not mean that the distinction between these two aspects of the meaning of one word that we have made is wrong. The insight that words have this double role is an important one and we do not have to be ashamed that we make it in our formalism. However, there is a follow up to the criticism that to me seems quite correct. Although there is no harm in separating formally things that are not always explicitly separated in natural language, it is desirable that the aspects that we have separated are represented correctly. We think that it is fair to say that the contribution to the information content via predicates is sufficiently close to real life for our purposes, although it may require improvement in other situations. But our representation of the linking machinery with variables is admittedly too simple. Here we meet the resolution problem again, with all the complications that it involves (cf. chapter I). We already have admitted that the resolution problem in general may be a task for which compositional semantics is not very well suited. But if we compare the simple minded resolution strategy of our formal language and the complex mechanism that we employ in natural language, there is also clearly room for some improvement. For example, in natural language there are many different clues that can be used to link up with an antecedent. Consider for example the following alternatives for the above example:

> A man and a woman came in. He was wearing a black coat.
>
> A man and a woman came in. The former was wearing a black coat.

We see that already in this simple example there are at least three ways of establishing the anaphoric link. But in the formal system we only have one clue for each antecedent, its current name.
Fortunately there is ample room for improvement in that area within our set-up. We could, for example, have instead of one current name for each referent, a set of names. This would amount to replacing the import and export functions with functions of type: $R \rightarrow \wp(NOM)$.
This is but a simple adjustment of the definition of a referent system, representing the situation where there are several ways to refer to a variable, just as in natural language. But already this simple change allows us to do very wild things in the definition of the merger. We could, for example, identify a referent with the most suitable antecedent at this point. A first attempt to model this strategy is given by a condition such as:

> In merging (I, R, E) and (I', R', E') we identify $r' \in R'$ with $r \in R$ under the following condition:
>
> $$r \equiv r' \Leftrightarrow max\{\, s \in R : \; E(s) \cap I(r')\} \; = \; E(r) \cap I(r') \;\; \& \\ E(r) \cap I(r') \neq \emptyset.$$

(We ignore for the moment all sorts of details, such as the problem of non-unique maxima.)
Perhaps a definition of this sort could also begin to account for the unstability of anaphoric links.[21] For if a referent system is preceded by another

[21] Technically this instability will be difficult to capture.

referent system, then the choice of the preferred antecedent might thereby be influenced. In natural language we sometimes have a similar situation. Take the (extended) example:

> A man and a woman came in. He liked her.
>
> John was looking at the door. A man and a woman came in. He liked her. But he hated him.

So there is room for criticism in our overly simple representation of the linking mechanism involved in anaphora, but there is also room for improvement. The advantage of our set-up is that this kind of improvement in the context component need not have any side effects in the content component. This is so because we have neatly separated the two ways in which variables contribute to dynamic interpretations.
Now that we have distinguished in our semantics the different problems involved in interpreting anaphora—the linking problem on the one hand and the computation of information content on the other—one can easily imagine that systematic influences on the linking process could be represented in the referent system without making it necessary to re-think the definition of the other component of our meaning objects.[22] Taking the context-content distinction serious in the semantics is the big step. After that improvements of either one of the components should be just a matter of fine tuning.

Appendix

Associativity

In this appendix we pick up the question of the associativity of the merger. We will prove the following

Proposition 3.7.1 (Proposition 3.5)

$$\begin{aligned}(\sigma \bullet \tau) \bullet \rho \;&=\; \sigma \bullet (\tau \bullet \rho) \textit{ iff}\\ dom(E_\sigma^{-1}) \cap dom(I_\tau) \cap dom(I_\rho) \;&\subseteq\; dom(E_\tau^{-1}) \cup dom(I_\sigma) \textit{ and}\\ dom(I_\rho) \cap dom(E_\tau^{-1}) \cap dom(E_\sigma^{-1}) \;&\subseteq\; dom(E_\rho^{-1}) \cup dom(I_\tau).\end{aligned}$$

[22]Work by Zeevat (1991b) in this direction, in particular in unification formalisms, is in this spirit.

(We have written $dom(E_\sigma^{-1})$ instead of $range(E_\sigma)$ to make it easier to see that the two conditions are dual.)
We will see that these conditions say that there is no "clash" of names that can be prevented by a transaction under one of the bracketings.
This is exactly what *does* happen in the counterexample against associativity that we have seen:

$(E[v] \bullet E[v]) \bullet I[v]$:there is an export-clash;

$E[v] \bullet (E[v] \bullet I[v])$:clash prevented by the transaction between $E[v]$ and $I[v]$.

In such a situation the import-export behaviour will not be independent of the bracketing.

Now we can see that the second condition of the proposition captures this situation. The intersection on the left hand side of the condition says that a transaction between τ and ρ is possible and that an export-clash between τ and σ is possible. The union on the right hand side is to prevent the trouble that we have seen in the example. (In the example the rhs-union is empty.)
If something in the intersection is also in $dom(I_\tau)$, then this means that there is not only a transaction between τ and ρ, but also between σ and τ. This means that there is no clash that can be prevented. This is what happens for example in $E[v] \bullet T[v] \bullet I[v]$.
If something in the intersection is also in $dom(E_\rho^{-1})$, the clash is not really prevented, it is merely delayed. This happens for example in $E[v] \bullet E[v] \bullet T[v]$.
The first condition can be illustrated by looking at the dual of our example, i.e. $E[v] \bullet I[v] \bullet I[v]$. (Remember that to find the dual, we have to read from right to left and consider import as export and vv.)
We see that the conditions of the proposition can be understood by looking at these basic examples.

Proof:
$\Leftarrow$:
Assume that both conditions hold. Also assume that we have chosen the symbols for the referents (r, r', etc.) in such a way that no confusion can arise if we simply speak of a referent without mentioning its referent

system (i.e. we will say r and xI simpliciter instead of $r \in R$, $xI \in R'$, etc).
Notation:

$$\begin{aligned} I_1 &= I_{(\sigma\bullet\tau)\bullet\rho}, \\ I_2 &= I_{\sigma\bullet(\tau\bullet\rho)}. \end{aligned}$$

We will check that $I_1 = I_2$. Then $E_1 = E_2$ follows by duality (note that the two clauses in the proposition are dual). Strictly speaking we have to check two things: first that $dom(I_1) = dom(I_2)$ and second that for all x in the domain $xI_1 = xI_2$. We will concentrate on the second task and perform the first one implicitly.
We consider the definition of xI_1:

$$\begin{aligned} xI_1 &= xI_{\sigma\bullet\tau} && \text{if } x \in dom(I_{\sigma\bullet\tau}) \\ &= xI_\rho && \text{if } x \in dom(I_\rho)\backslash(dom(I_{\sigma\bullet\tau}) \cup dom(E^{-1}_{\sigma\bullet\tau})) \end{aligned}$$

i.e.

$$\begin{aligned} xI_1 &= xI_\sigma && \text{if } x \in dom(I_\sigma) \\ &= xI_\tau && \text{if } x \in dom(I_\tau)\backslash(dom(I_\sigma) \cup dom(E^{-1}_\sigma)) \\ &= xI_\rho && \text{if } x \in dom(I_\rho)\backslash(dom(I_\sigma) \cup (dom(I_\tau)\backslash dom(E^{-1}_\sigma)) \\ & && \cup\, dom(E^{-1}_\tau) \cup (dom(E^{-1}_\sigma)\backslash dom(I_\tau))) \end{aligned}$$

The definition of I_2 gives us:

$$\begin{aligned} xI_2 &= xI_\sigma && \text{if } x \in dom(I_\sigma) \\ &= xI_{\tau\bullet\rho} && \text{if } x \in dom(I_{\tau\bullet\rho})\backslash(dom(I_\sigma) \cup dom(E^{-1}_\sigma)) \end{aligned}$$

i.e.

$$\begin{aligned} xI_2 &= xI_\sigma && \text{if } x \in dom(I_\sigma) \\ &= xI_\tau && \text{if } x \in dom(I_\tau)\backslash(dom(I_\sigma) \cup dom(E^{-1}_\sigma)) \\ &= xI_\rho && \text{if } x \in dom(I_\rho)\backslash(dom(I_\sigma) \cup dom(I_\tau) \cup \\ & && dom(E^{-1}_\tau) \cup dom(E^{-1}_\sigma)). \end{aligned}$$

We see that the only difference between xI_1 and xI_2 can occur if the third clause of the definitions is activated. But in the proposition we see that by the first condition we have that for $x \in dom(I_\rho)$:

$$x \in dom(E^{-1}_\sigma) \cup dom(I_\tau) \;\; \Rightarrow \;\; x \in dom(E^{-1}_\tau) \cup dom(I_\sigma).$$

Hence for $x \in dom(I_\rho)$:

$x \notin dom(I_\sigma) \cup dom(I_\tau) \cup dom(E^{-1}_\tau) \cup dom(E^{-1}_\sigma)$ iff

$x \notin dom(I_\sigma) \cup (dom(I_\tau)\backslash dom(E^{-1}_\sigma)) \cup (dom(E^{-1}_\sigma) \cap dom(I_\tau)) \cup dom(E^{-1}_\tau) \cup dom(E^{-1}_\sigma)$ iff

$x \notin dom(I_\sigma) \cup (dom(I_\tau)\backslash dom(E^{-1}_\sigma)) \cup dom(E^{-1}_\tau) \cup dom(E^{-1}_\sigma)$.

If we apply the same trick again by splitting up $dom(E_\sigma^{-1})$ into: $(dom(E_\sigma^{-1})\backslash dom(I_\tau))$ and $dom(E_\sigma^{-1}) \cap dom(I_\tau)$, we find:

> $x \notin dom(I_\sigma) \cup (dom(I_\tau)\backslash dom(E_\sigma^{-1})) \cup dom(E_\tau^{-1}) \cup (dom(E_\sigma^{-1})\backslash dom(I_\tau))$ iff
>
> $x \notin dom(I_\sigma) \cup dom(I_\tau) \cup dom(E_\tau^{-1}) \cup dom(E_\sigma^{-1})$.

We conclude that: $I_1 = I_2$.

$\Rightarrow$:
Suppose that the first condition of the proposition does not hold. (The case where the second clause fails is dual.) Then we have $x \in dom(E_\sigma^{-1}) \cap dom(I_\tau) \cap dom(I_\rho)$ but $x \notin dom(E_\tau^{-1}) \cup dom(I_\sigma)$ for some $x \in NOM$. Now if we look at the import functions I_1 and I_2, we see that in both cases the first two clauses do not apply. Hence we have to consider the third clause of the definitions. We see that since $x \in dom(E_\sigma^{-1}) \cap dom(I_\tau) \cap dom(I_\rho)$, also $x \in dom(I_\tau)$ and therefore $x \in dom(I_\sigma) \cup dom(I_\tau) \cup dom(E_\tau^{-1}) \cup dom(E_\sigma^{-1})$. This means that xI_2 is undefined.
But if $x \in dom(E_\sigma^{-1}) \cap dom(I_\tau) \cap dom(I_\rho)$, then $x \notin dom(I_\sigma) \cup (dom(I_\tau)\backslash dom(E_\sigma^{-1})) \cup dom(E_\tau^{-1}) \cup (dom(E_\sigma^{-1})\backslash dom(I_\tau))$. Therefore xI_1 is defined and in fact $xI_1 = xI_\rho$.
So we find that xI_1 is defined, while xI_2 is undefined. Hence $(\sigma \bullet \tau) \bullet \rho \neq \sigma \bullet (\tau \bullet \rho)$.

This completes the proof of the proposition.□

DPL and DS

In this section of the appendix we prove the following proposition from section 5.

Proposition 3.7.2 (Proposition 5.7)
Let g and h be partial assignments $NOM \hookrightarrow DOM$. Then:

$g\vert\bot\vert_p h$	*iff*	$g \neq g$;
$g\vert P(x)\vert_p h$	*iff*	$x \in dom(g)$ & $g = h$ & $xh \in \mathbf{P}$;
$g\vert\exists x\vert_p h$	*iff*	$x \in dom(h)$ & $(y \neq x \Rightarrow xg = xh)$;
$g\vert\phi.\psi\vert_p h$	*iff*	$\exists k:\ g\vert\phi\vert_p k$ & $k\vert\psi\vert_p h$;
$g\vert(\phi \to \psi)\vert_p h$	*iff*	$g = h$ & $\forall k:\ g\vert\phi\vert_p k \Rightarrow \exists l:\ k\vert\psi\vert_p l$.

We use the following two results:

Corollary 3.7.3 *Let three referent systems ρ, σ and τ be given such that σ and τ are partially persistent and extending. Then $\rho \bullet (\sigma \bullet \tau) = (\rho \bullet \sigma) \bullet \tau$.*

Proof:
Follows immediately from the associativity result that is proved above.□

Lemma 3.7.4 (Extension lemma) *Let (ρ, k), (σ, g), (τ, h) be given:*

- *if ρ is partially persistent and extending, then* $(\sigma, g)[\phi](\tau, h) \Rightarrow (\sigma, g) \bullet (\rho, k)[\phi](\tau, h) \bullet (\rho, k)$;
- *if σ is partially persistent and extending, then* $(\sigma, g)[\phi](\tau, h) \Rightarrow (\rho, k) \bullet (\sigma, g)[\phi](\rho, k) \bullet (\tau, h)$.

Proof :
Direct. (1) requires partial persistence and extending to keep the variables linked to the same referents. In (2) the condition is used to make sure that the merger is associative (see corollary).□

Now we can prove the proposition:

Proof:
The proof is a lengthy induction in which a construction of suitable partially persistent and extending referent systems is implicitly defined.

1. The case where ϕ is $\bot$ is obvious;
2. $g|P(x_1, \ldots, x_n)|_p h$ iff
   there are (σ, g'), (τ, h') such that $(\sigma, g')[P(x_1, \ldots, x_n)](\tau, h')$ & $g = E_\sigma^{-1} g'$
   & $h = E_\tau^{-1} h'$ iff
   there are (σ, g'), (τ, h') such that $(\sigma, g') = (\tau, h')$ &
   $range(g') \supseteq \{x_1, \ldots, x_n\}$ & $(x_1 g', \ldots, x_n g') \in \mathbf{P}$ & $g = E_\sigma^{-1} g'$
   & $h = E_\tau^{-1} h'$ iff

$g = h$ & $(x_1g, \ldots, x_ng) \in \mathbf{P}$.

For the last 'iff' we need a construction. The following will do: take $\sigma = T[dom(g')]$, $g' = g$, $\tau = \sigma$ and $h' = g'$.

3. $g|\exists x|_p h$ iff

   there are (σ, g'), (τ, h') such that $(\sigma, g')[\exists x](\tau, h')$ and $g = E_\sigma^{-1}g'$ & $h = E_\tau^{-1}h'$ iff

   $\sigma \bullet E[x] = \tau$ & $h' = g' \oplus (x, d)$ for some $d \in D$ & $g = E_\sigma^{-1}g'$ & $h = E_\tau^{-1}h'$ iff

   $y \neq x \rightarrow yg = yh$ & $x \in dom(h)$.

   For the last 'iff' we need a construction. Take σ as above and take $\tau = \sigma \bullet E[x]$. Then $g' = g$ and $h' = h \oplus (x, d)$ will do.

4. $g|\phi.\psi|_p h$ iff

   there are (σ, g'), (τ, h') such that $(\sigma, g')[\phi.\psi](\tau, h')$ and $g = E_\sigma^{-1}g'$ & $h = E_\tau^{-1}h'$ iff

   there are (σ, g'), (τ, h') such that $\tau = \sigma \bullet \sigma_{\phi.\psi}$ & $h' = g \oplus (f \oplus f')$ for some $f \in F_\phi$ and $f' \in F_\psi$ & $g = E_\sigma^{-1}g'$ & $h = E_\tau^{-1}h'$ iff (by corollary)

   there are (σ, g') and (τ, h') such that $\tau = (\sigma \bullet \sigma_\phi) \bullet \sigma_\psi$ & $h' = (g \oplus f) \oplus f'$

   & $g = E_\sigma^{-1}g'$ & $h = E_\tau^{-1}h'$ iff (take $\rho = \sigma \bullet \sigma_\phi$ and $k' = g' \oplus f$)

   there is (ρ, k') such that $(\sigma, g')[\phi](\rho, k')$ and $(\rho, k')[\psi](\tau, h')$ &

   $g = E_\sigma^{-1}g'$ & $h = E_\tau^{-1}h'$ iff

   there is a k such that $g|\phi|_p k|\psi|_p h$.

   The last 'iff' requires a construction. The extension lemma tells us that the constructions that we find for ϕ and ψ by induction hypothesis can be combined to one for $\phi.\psi$. (Check that the construction gives partially persistent and extending referent systems!)

5. $g|(\phi \rightarrow \psi)|_p h$ iff

   there are (σ, g'), (τ, h') such that $(\sigma, g')[(\phi \rightarrow \psi)](\tau, h')$ and $g = E_\sigma^{-1}g'$ &

$h = E_{\tau}^{-1} h'$ iff

there are (σ, g'), (τ, h') such that $\sigma = \tau$ & $g' = h'$ &

$\forall(\rho, k') :\ (\sigma, g')[\phi](\rho, k')\ \exists(\mu, l') :\ (\rho, k')[\psi](\mu, l')$.

Now the induction hypothesis, the extension lemma and the construction procedure tell us that this holds iff:

for all k such that $g|\phi|_p k$ there is a l such that $k|\psi|_p l$.

□

Part II

Propositional Dynamics

Why Go Propositional?

In this part of the thesis we will concentrate on *propositional texts.* This means that we will be using propositional languages to represent texts. There are two lines of motivation for this choice:

- The phenomena that we want to discuss typically arise on the propositional level. Here we think of the structural organisation of texts. It makes good sense to look at this kind of structure in isolation and try to get a good picture of the extent to which the idea of a dynamic semantics applies here.

- In the end all genuinely dynamic representation languages will be 'propositional' in nature. In fact in addition to a lot of propositional constants a dynamic representation language ultimately will have only one connective left: concatenation. This is the ultimate consequence of the *small unit principle.*

As one can see the second line of motivation is more radical than the first: it says that the attention for propositional languages should *not* be seen as a temporary restriction, that we can give up later on. Rather it is only a first step on our way to really dynamic representation languages, which only have concatenation as a connective.
To understand this more radical motivation, it is good to remind ourselves that already *DRT* and *DPL* essentially live on the propositional level: both in *DRT* and in *DPL* the analogue of the existential quantifier behaves like just another atomic formula. The language $\mathcal{L}$ which, as we have shown, is a good language for discussing the semantic phenomena common to *DPL* and *DRT*, is really just a propositional language with connectives $\cdot$ and $\rightarrow$. The formulas $\exists x$ are propositional constants, i.e. atomic propositions with a fixed interpretation. So in a dynamic framework predicate logic arises

as a particular propositional logic. This makes clear that in a dynamic setting the restriction to the propositional level does not imply that we cannot cope with quantificational phenomena.
Still to some the propositional character of $\mathcal{L}$ may seem a bit of an accident: for example, one may feel that once other, generalised forms of quantification are considered, we will really need to look at *non-local* structures in order to represent these quantificational phenomena correctly. Then we would no longer be able to get the required quantificational force by the introduction of special propositional constants. In principle this could be the case. But notice that this would mean that we would have to give up the *small unit principle*. We have argued for the small unit principle in chapter 1 and will give further motivation for it in the next chapter. Basically the small unit principle tells us that a dynamic semantics will have to work on a small scale, making small steps at a time. We see that the treatment of $\exists x$ as a propositional constant is typically in the spirit of the small unit principle: a quantifier is not treated as a global operator, but instead the quantificational contribution is located in a (small) proposition. So we do not only observe that $\mathcal{L}$ is a propositional language: we also see that dynamic languages in general will have to be propositional, because of the small unit principle.
This shows that the choice for the propositional level can be seen as a consequence of the small unit principle. By the small unit principle one of the tasks of a dynamic semantics precisely is to obtain the effect of 'global' constructions in small steps, i.e. in 'propositional' languages. We will take this line of motivation one step further in the next chapter: in the end the only connective that we can allow ourselves to use according to the small unit principle is concatenation.
We will show in the next chapter that this restriction does not mean that other connectives cannot be expressed in the language: we will see that in a set up with only concatenation as a connective other propositional connectives — we will look at implication as an example — can be reconstructed by using appropriate constants. In fact, we will argue that our way of recovering the propositional connectives also extends to quantificational structures of the form $Qx(\phi, \psi)$: we will argue that the standard semantics of such quantificational structures can be reproduced using only *small units*.
So from the second line of motivation it follows that working with propositional languages is certainly not a restriction: even propositional structure

should be regarded as a luxury that we will have to give up, as the ultimate consequence of the small unit principle.

But even if this were not so, the first line of motivation still gives a good enough reason for looking at propositional languages, in particular because these languages allow us to focus on the macro structure of texts. In the previous chapters we have been preoccupied with the semantics of variables and this has also been the main point of focus in the literature on dynamic semantics. The reason for this preoccupation has been given in the introduction to part I: traditionally dynamic semantics is typically concerned with anaphoric phenomena and as far as compositional semantics is concerned, the semantics of anaphora *is* the semantics of variables. Still, the subject of dynamic semantics not just the semantics of anaphora: dynamic semantics is concerned with the intepretation of *texts* in general. Since semantics so far has mainly been concerned with the interpretation of *sentences*, there are several new things that a dynamic semantics will have to cope with that did not play a role traditionally in semantics. Anaphoric relations between different sentences so far have been the most popular example of such a phenomenon. But dynamic semantics will also have to deal with other text level phenomena such as the macro structure of texts. Most of the example discourses that we have considered so far consist of only one or two sentences. Usually in these examples the discourse as a whole can be interpreted as the conjunction of these sentences. In general, however, the sentences of a text can have many different relations to one another: a text is typically organised into blocks which stand in certain *discourse relations* to each other. These discourse relations are also relevant to the interpretation of the text. For example, one block or paragraph may provide an *explanation* of what was claimed in another paragraph, or perhaps it is one of the items in an *enumeration*. Such things are usually indicated by special words or phrases, such as: *therefore*, *in the third place*, etc. In the next chapter we will concentrate on the situation where one part of the text is an *assumption* of a *claim* elsewhere in the text. Again such relations are things which, at least at first sight, have a global character and therefore pose a problem for a dynamic semantics. So this is a topic that we will have to deal with at some point and it typically is something which can be studied at the propositional level. This is another way to motivate the attention for propositional texts.

We see that there is more than one good reason for looking at propositional texts. In this part we will look in particular at propositional languages that represent proof-like texts, texts consisting of assumptions and conclusions. We will first use these as an example to develop our ideas about the way in which a dynamic semantics could deal with the macro structure of texts (chapter 4). Then we will look at the possibilities of using these texts for the development of a genuinely dynamic proof theory: we try to work out the idea that proving a theorem is in fact the same as constructing a text which proves the theorem. So according to this idea the construction of a proof is a special case of the construction of a text (chapter 5). In chapter 5 the restriction to the propositional case really will amount to a drastic simplification of the situation, because there we will not only exclude 'global' constructions from the language, but also exclude all the interesting quantificational constants—such as $\exists x$— from the picture. Here the only excuse is simplicity: we have not yet been able to obtain a genuinely dynamic proof system for systems with quantificational constants, such as *DPL* or *DRT*.

Chapter 4

The Dynamics of Propositional Structure

4.1 Introduction

We have seen that dynamic semantics involves a shift of attention from the semantics of sentences to the semantics of texts. So far this shift has resulted in a new outlook on the semantics of anaphora (in Kamp (1981), Heim (1983), Groenendijk and Stokhof (1991a) etc.) and some related phenomena, such as presuppositions (Van der Sandt (1992), Beaver (1994), Van Eijck (1991b), Krahmer (1994)). But the shift to the semantics of texts also has broader consequences. In this chapter we are concerned with the consequences of this shift of attention for the requirements on the formal methods that are used. The all important methodological constraint in sentence semantics is *compositionality.* Traditionally this is the principle that fixes the treatment of sentence structure. But it seems that the compositional view on the role of structure in semantics, as it stands, is not appropriate for the semantics of texts. Therefore we propose several other constraints: *incrementality, (pure) compositionality* and *the break in principle.* All these constraints can be seen as ways of making the *small unit principle* concrete. Recall that the small unit principle was introduced as a natural constraint on a semantics of texts. Since texts can be really large objects, it is hard to imagine that someone who interprets a text works on the text as a whole. Therefore a reasonable model of text interpretation should reflect how we interpret a text bit by bit.

The three constraints that we impose here make different aspects of this idea precise. Apart from discussing the motivation of the constraints, we will also develop an semantics for propositional texts in this chapter that satisfies the three principles. Hence our semantics will illustrate the way in which these principles work for a simple, propositional language.

The incrementality principle is inspired by the observation that we can interpret texts *as we hear them.* If we want to understand a text we do not have to wait for the text to be completed before we can start our interpretation. We can simply start as soon as we hear the first word and then we build up our interpretation step by step. It is also clear that for large texts this is the only possible procedure. We cannot first read a large text, a book say, and only after that start to interpret it. So we do not only want to work in small units: we also want to process these small units incrementally. Of course, in real life we do not always choose to work strictly incrementally — sometimes it might be convenient to wait a bit, for example until the end of the sentence — but this waiting cannot be extended indefinitely. And anyway, it should never be necessary. Although it might be convenient to wait sometimes, in principle the text should *allow* us to interpret it without delay.

This is the way we want to look at our observation concerning incrementality. It simply is not true that we always *do* interpret texts incrementally. There are numerous occasions on which we chose to read a text not simply from beginning to end, but in some other order. Perhaps this is exactly what the reader has done with this text. But all the time we rely on the fact that a text *allows* for an incremental interpretation. And this will also be our constraint on the formalism: we do not demand that everything is done incrementally, but merely that everything *can* be done incrementally. Note that the incrementality constraint makes even more sense with respect to *spoken* text: it is much harder to imagine a non-incremental treatment of a spoken text.

Here we are talking about the text level. It does not necessarily follow that everything that happens in the semantics of natural language has to be accounted for incrementally. In principle it is not excluded that some micro level phenomena behave differently. Intuitions about incrementality typically are especially strong about the macro level and this is also the level for which text semantics is designed. Still, we would like to argue that also at the micro level an incremental treatment makes sense. This may seem a rather radical claim at this point, but perhaps this will have changed by the time we reach the end of the chapter. Then we will have developed our incremental treatment of the (macro) structure of texts and it will be easy to see that the techniques apply quite generally. For example, it should not be problematic to give an incremental interpretation of quantified structures of the form $Qx(\phi, \psi)$ along the lines set out in this chapter. Hopefully at that point one will start to wonder what kind of phenomenon could resist an incremental interpretation.

The incrementality constraint gives rise to an important difference with what we were used to in sentence semantics. In sentence semantics we allow ourselves to use information about the structure of the sentence in its interpretation. When we start interpreting a sentence we assume that its structure is known. Then we can let the structure tell us *how* the meanings of the sentence parts have to be glued together to form the meaning of the sentence. This is the *compositionality* principle for traditional sentence semantics. But in the current, incremental set up we cannot use this method. For we want to do justice to the observation that we can interpret a text as we hear it. Thereby we cannot let some kind of structural analysis *precede* the interpretation process. Instead it seems that the analyses of meaning and structure have to be performed at the same time. Therefore the compositionality principle, in its usual form, is not appropriate for text semantics.[1]

Instead we will use a more modest form of compositionality than we are used to in sentence semantics. We want to preserve the idea the meaning of the text as a whole is composed from the meanings of the parts of the text: we do not aim to consider the influence of 'foreign elements' on the interpretation of a text here. But we will not assume that information about the structure of the text tells us how the parts have to be put together. The structure of the text has to be discovered at the same time as the meaning of the text. Therefore we impose a modest form of the compositionality, which we will be called *pure compositionality*: it simply states that the meaning of a text depends on nothing but the meaning of its parts. Note that also the pure compositionality principle goes together nicely with the small unit principle: we can simply decompose a text into small units and the interpretation of these small units is all we need to compute the meaning of the text as a whole.

[1]Of course this does not mean that the classical compositional approach is *wrong*: it just does not work for the questions that we are interested in.

Another constraint that we impose on text semantics is the break in principle. We have argued that it is always possible to interpret a text, even if it is clear that the text is as yet incomplete and that more is to follow. This gave rise to the incrementality constraint. But then it is inevitable that also our interpretations will be incomplete: they will be *partial* in this sense. We do not mean that there will be room for doubt about the meaning of such an incomplete text. What we mean is that the interpretation of a text will allow for combination with material from other parts of the text, the parts that are to follow.

Now, if we follow this line of reasoning a little further, we see that it is not only natural to require that we be able to interpret *unfinished* texts, but also other kinds of incomplete texts. In fact we want to be able to interpret all *continuous parts*, or *segments*, of texts. It seems that not only if we have not yet heard the last part of a text, but also if we have not heard the first part of a text, we are able to understand exactly what is being said. Of course we may have missed some important clues in such a situation. So our understanding of what is being said can again in general only be partial. But this partiality is in the *result* of the interpretation only. We can interpret everything that is being said *completely*, yet the information that we get out of such a text fragment is only partial: the information becomes complete in combination with other, previous, partial interpretations.

This seems to be what happens when you hear in on a conversation: you can understand everything that is being said, even though you may have missed the beginning of the story. This leads to the formulation of the *break in principle*, that guarantees that wherever we break in in a text, we will always be able to understand what is being said. In other words the break in principle says that every segment of a text should be interpretable.

From what has been said it should be clear that the break in principle can only hold if we have in the semantics objects that are *partial meanings*. Here we use partial in the sense explained above: they are typically *unsaturated objects*, i.e. objects which like to interact with other objects. Having partial meanings is also something which corresponds nicely with the small unit principle: a text may contain some construction that is too big to handle in one go. Then we will have to interpret the parts of this construction first. These interpretations will be partial or unsaturated in this sense: later we will have to glue them together to get the

interpretation of the complete construction.
This principle has serious consequences in presence of the compositionality principle. According to the break in principle anything is a meaningful part of a text. Hence a text can be decomposed in many different ways and it seems reasonable to assume that each of these decompositions should allow us to compute the meaning of the text. It is also desirable that different decompositions lead to the same result, as long as we are not considering texts that are ambiguous. Thereby the three principles together demand that text meanings form an associative algebra: we want the meaning of the whole to be composed uniformly from the meanings of the parts and each decomposition into parts should give the same result. In particular an incremental decomposition has to be available.
So the situation is as follows:

> **Pure Compositionality**: The meaning of a text can be computed (uniformly) from the meaning of its parts.
>
> **Incrementality**: The meaning of a text can be computed by a process of interpretation that strictly follows the order of presentation.
>
> **Break in principle**: Any segment of a text can be interpreted. (In general its meaning will be partial.)

Together these requirements amount to:

> **Associativity**: Text meanings form an algebra with an associative operation (which we will call the merger) by which the meanings can be glued together.

We see that the general story for text semantics is quite different from what we are used to in sentential semantics. In sentential semantics we allow ourselves to use information about the structure of the sentence and we can postpone our interpretation process until all the structural information is available. We cannot afford to treat the structure of texts in the same way: we have to be able interpret a text as we hear it.

The semantics we give in this chapter incorporates the three principles: we give an incremental semantics of texts that satisfies the break in principle. The texts that we study are very simple: they are built up from propositional variables, the atomic texts. The only kind of text structure that we consider is the kind we find in proof-like texts. This kind of structure is usually indicated by phrases such as 'suppose that,' 'assume for the moment,' 'hence,' 'so,' etc. It also occurs at sentence level, typically in 'if... then' sentences.

In general it can be quite difficult to detect the structure of a text: often it is only indicated vaguely or implicitly. Then it can be quite hard to determine what is going on. But the problem of the *detection* of text structure does not concern us here. We will focus on the (dynamic) *interpretation* of text structure. At this point it may not be entirely clear to the reader what *interpretation of structure* is supposed to mean. But this will become clear later on when we see in practice how structure and meaning interact in our set up.

Since we are not trying to deal with the detection of (implicit) structural clues here, we might as well assume that all clues are given explicitly. In our formal language if, *then* and *end* are used for this purpose. The intended interpretation of a text of the form *if* ϕ *then* ψ *end* is the implication $(\phi \rightarrow \psi)$.[2]

The formal language that we will work with is defined as follows.

Definition 4.1.1 *Let a vocabulary of atomic texts A be given. We define the texts over A, $Text_A$, as follows:*

$$if,\ then\ , end \in Text_A$$

$$\bot \in Text_A$$

$$p \in A \ \Rightarrow\ p \in Text_A$$

$$\phi \in Text_A \ \&\ \psi \in Text_A \ \Rightarrow\ \phi\psi \in Text_A$$

As one can see, we treat *if*, *then* and *end* simply as basic texts — even though we plan to use them as structural indicators — and there are no structural restrictions on texts: they are simply built up by concatenation. Sometimes the concatenation of texts can be pronounced as 'and'.

This way we can get funny texts that have no sensible interpretation. This agrees with the view on text structure that we developed above: the

[2]Note that we only consider texts in which the assumptions are given before their conclusions.

structure of a text has to be analysed at the same time as its meaning. We cannot assume beforehand that the texts that we have to analyse are well formed. If the text is not well formed, then we will have to find this out as we proceed.

Maybe it is good to recall that an atomic text such as *if* does not only stand for the word 'if', but also for a phrase such as 'let's assume the following'. So an expression such as *if* p, which at first sight seems highly ungrammatical, can correspond to a quite sensible text such as *Let's assume that p holds.*

Proofs are a good example of texts that have this kind of structure. They typically consist of a network of assumptions and conclusions of a kind that is very similar to the structure of the texts of $Text_A$. Therefore, one of the things that we would like to do is to develop a deduction system in which proofs are considered as a special kind of texts, texts of which the construction satisfies a number of additional syntactic constraints. We will not develop such a deduction system in this chapter, but we will get back to this in the next chapter.

In the end we would also like to have a sentence level semantics that satisfies the incrementality constraint and the break in principle. We already explained above that it is not automatically clear that this can be done. But then we can just try and see which phenomena exactly resist an incremental treatment.

4.2 Texts as sequences

In this section we present our first attempt at an incremental update semantics for $Text_A$. The final version will be presented in the next section. This first attempt serves to illustrate one important feature of our approach. It can be seen as a solution to one important problem that arises in incremental semantics: *non-associativity*. It was pointed out above that an incremental semantics satisfying the break in principle will always be associative. So non-associative features of texts are problematic. In $Text_A$ an 'if... then' construction intuitively causes non-associativity. For the interpretation of a simple concatenation of basic texts $p \in A$, we do not have to worry about non-associativity: $(pq)r$ and $p(qr)$ give the same information. So any bracketing of such simple texts will do. But if the special elements *if*, *then* and *end* occur in a text, then we have to be more careful.

Consider, for example, the text *p if q then r end*. This text gives the information that *p* and also that *if q then r end*. This suggests that we have to interpret *if q then r end* first as one component of the text before we can add it to our interpretation of *p*. This corresponds to a bracketing *p* (*if q then r end*). But we have to allow for an incremental interpretation of this text. So it seems that we will only be able to handle the bracketing (((((*p if*) *q*) *then*) *r*) *end*).

The solution that we give for this problem in this section will work in general when an incremental treatment of such non-associative phenomena is needed. The solution can be summarised by one word: *memory*. In our semantics we will allow ourselves to have more than one slot where information can be stored. We will not only have a slot for our current state of information, but we will also have slots for some specific information states that we used to be in. So we remember our information *history*.

Now, when we have to interpret *p if q then r end*, we can first interpret *p*. We store the information that *p* in our memory before we interpret *q*. This information is again stored before we interpret *r*. Now we can construct from the information that we have stored the information that *if q then r*. Finally this information can be added to the information that *p*. Note that we do not need brackets to tell us how we have to store the information: the special elements *if*, *then* and *end* will tell us exactly what has to be done.

This story can be formalised as follows. In the semantics we will always assume that some Heyting algebra (**HA** for short) **I** is given to provide the basic information items.

Recall that Heyting algebras are defined as follows.

Definition 4.2.1 *1. A* lattice *is a structure* $\mathbf{L} = (L, \wedge, \vee)$*, such that the binary operations* $\wedge$ *and* $\vee$ *satisfy the following conditions:*

$$\begin{array}{rcll} (a \wedge b) \wedge c &=& a \wedge (b \wedge c) & \textit{(associativity of } \wedge\textit{)} \\ (a \vee b) \vee c &=& a \vee (b \vee c) & \textit{(associativity of } \vee\textit{)} \\ a \wedge b &=& b \wedge a & \textit{(commutativity of } \wedge\textit{)} \\ a \vee b &=& b \vee a & \textit{(commutativity of } \vee\textit{)} \end{array}$$

$$\begin{array}{ll} a \wedge a \ = \ a & \textit{(idempotency of } \wedge\textit{)} \\ a \vee a \ = \ a & \textit{(idempotency of } \vee\textit{)} \\ a \wedge (a \vee b) \ = \ a & \textit{(first absorption law)} \\ a \vee (a \wedge b) \ = \ a & \textit{(second absorption law)} \end{array}$$

In a lattice **L** *we can define an ordering by:*

$$a \leq b \ \Leftrightarrow \ a \wedge b = a.$$

2. *A* Heyting algebra *is a structure* $\mathbf{I} = (I, \wedge, \vee, \bot, \rightarrow)$, *such that* $(I, \wedge, \vee)$ *is a lattice,* $\bot$ *is the least element of* I *and* $\rightarrow$ *is a binary operation such that*

$$(\imath_0 \wedge \imath_1 \leq \imath_2) \ \Leftrightarrow \ (\imath_0 \leq \imath_1 \rightarrow \imath_2).$$

We call the elements of **I** *information states.* An *information history* is a finite, non-empty sequence of information states. We define the interpretation of texts ϕ, $[\phi]$, as a partial function on information histories. We will assume that for each atomic text $p \in A$ an information state $\imath_p$ is given: $\imath_p$ is the information that p.[3]

Definition 4.2.2 *We define for each* ϕ *the update function* $[\phi]$ *as follows.*[4] *Let an information history* $\sigma = (\sigma_1, \ldots, \sigma_n)$ $(n \geq 1)$ *be given.*

$$\begin{array}{ll} \sigma[\bot] & = (\sigma_1, \ldots, \sigma_{n-1}, \sigma_n \wedge \bot) \\ \sigma[p] & = (\sigma_1, \ldots, \sigma_{n-1}, \sigma_n \wedge \imath_p) \\ \sigma[if] & = (\sigma_1, \ldots, \sigma_{n-1}, \sigma_n, \top) \\ \sigma[then] & = (\sigma_1, \ldots, \sigma_{n-1}, \sigma_n, \top) \\ \sigma[end] & = (\sigma_1, \ldots, \sigma_{n-2} \wedge (\sigma_{n-1} \rightarrow \sigma_n)) \\ \sigma[\phi\psi] & = (\sigma[\phi])[\psi] \end{array}$$

Furthermore we define truth as follows:

For $\imath \in \mathbf{I}$ *we define* $(\imath) \models \phi$ *iff* $(\imath)[\phi] = (\imath)$. *We say that* ϕ *is true in* $\imath$. *We write* $\models \phi$ *iff* $(\top) \models \phi$. *We say that* ϕ *is true (in* **I***).*

[3] Since we are running out of brackets we will not make an attempt to keep the notation in this part consistent with the notations in part I and chapter 1. As we will not be interpreting formulas of $\mathcal{L}$ her, but only look at $Text_A$ this need not cause any confusion.

[4] We will use postfix notation for function application and we will adapt the notation for function composition accordingly.

A good example of an information algebra **I** that the reader can keep in mind in what is to follow can be found, for example, in [Veltman]'s update semantics. He uses an information algebra that is defined as follows:

Definition 4.2.3 *Let a vocabulary A of atomic expressions be given.*

Let $W=\wp(A)$. $w \in W$ is called a possible world (or possibility);

Let $\mathbf{I}=\wp(W)$. $\mathbf{I}$ is the information algebra (over A), ordered by $\subseteq$. The elements $\sigma \in \mathbf{I}$ are called information states.

Here the $w \in W$ are called *possible worlds* because each subset $w \subseteq A$ corresponds to a way the world might be: the atomic propositions, or *possible facts*, in w might be exactly the things that are true, while all other atomic propositions are false. In *information state* σ we know that one of the $w \in \sigma$ is the real world, but we do not know exactly which one. It is clear that **I** is a Heyting algebra since $\mathbf{I} = \wp(W)$ is an (atomic) Boolean algebra. So Definition 2.2 applies. The canonical choice for $\imath_p$ ($p \in A$) is: $\imath_p = \uparrow(\{p\}) = \{w : \{p\} \subseteq w\}$.
There is nothing deep behind our choice of **HA**s as information algebras. We have chosen **HA**s because we do not want to worry here about the definitions of the conjunction and implication of information states. Thus working in a **HA** allows us to concentrate on the other problems for our semantics and this is in fact all that we want from them. Therefore any other structure with well defined operations of conjunction and implication can serve equally well as **I**. One interesting example of a suitable information algebra **I** that is not a **HA** is the algebra of *DRS* meanings as defined in Zeevat (1991a).
A similar remark applies to our choice for implication as the additional connective. The choice for implication is motivated by our interest in proof-like texts, but the construction of a sequence interpretation would work for any other connective as well.[5]

Definition 4.2.2 gives us the right result for texts like $p\ if\ q\ then\ r\ end$: it is easy to check that now:

[5]It is even possible to treat quantificational constructions in this fashion if we decompose something like $Qx(\phi, \psi)$ into $start_{Qx} \cdot \phi \cdot middle_{Qx} \cdot \psi \cdot end_{Qx}$, where $start_{Qx}, middle_{Qx}$ and end_{Qx} are new constants in our language.

$$(\top)[p\ if\ q\ then\ r\ end] = (\imath_p \wedge (\imath_q \rightarrow \imath_r))$$

And the semantics is incremental and associative, as required. This is clear since function composition is associative. But the semantics is not satisfactory in every respect: the structural contribution of the special elements *if*, *then* and *end* is not represented in the best possible way. We see, for example, that in our semantics *if* and *then* get the same meaning: $[if] = [then]$. Thereby also $[if\ p\ then\ q\ end] = [then\ p\ if\ q\ end]$. This implies that for our semantics the texts *if p then q end* and *then p if q end* are equally acceptable, which intuitively, of course, they are not. So our semantics cannot distinguish a coherent from an incoherent text. This would imply that we will still have to determine in advance whether or a text is coherent or not. Which brings us back to the treatment of text structure: if we had a grammar of texts that would simply rule out *then p if q end* as ungrammatical, no problems would arise. But we have already explained that this is not the way things should be done in text semantics. Even if we have a text grammar that rules out *then p if q end* as ungrammatical, we still want to find out during the interpretation that the expression is illegal according to this grammar. We need a situation in which un-wellformedness is indicated in the semantics by some kind of failure or error behaviour.
At this point the only kind of semantic failure that occurs is partiality: some expressions generate *partial* functions. This indicates that the text is *left incomplete*, i.e. we need some preceding material to be able to make sense of the text. For example *end* will only be defined on information histories of length greater than two, indicating that it should be preceded by two expressions that generate locations in memory.[6] But unfortunately *end* is not able to distinguish *if*-locations from *then*-locations. Therefore the partiality in the semantics cannot rule out *then p if q end*.
Here we see in a concrete example how the interplay between syntax and semantics is a crucial topic in incremental semantics. We have introduced the incrementality requirement on the semantics of texts, since we feel that we can interpret texts as we hear them. But if we are only able to interpret *well formed* texts, then we also have to be able to decide about the well formedness of a text as we hear it.
In what follows we will usually concentrate on the *meaning* of texts, but

[6]In fact all partiality in the semantics of definition 2.2 originates from the partiality of *end*.

in fact ever more refined incremental well formedness tests will become implicitly available in our machinery as we proceed.

4.3 Texts as trees

In this section we attack the problem that we discovered for the semantics with information histories. We saw that we cannot see in the semantics whether a text is well formed or not. The reason for this is that the different locations in the information histories do not show why they where created: were they created by *if* in order to store an assumption or were they created by *then* in order to store a conclusion? Once we can answer this question we are done.
Therefore we want to be able to distinguish the *if* places from the *then* places in our information histories. In order to do that we simply add *structure* to the information histories: instead of using sequences to represent our memory, we will use binary trees. We will use the left branches in the trees to (temporarily) store the antecedents of implications and the right branches will be used for the conclusions. The [*end*] command will tell us that the implication is complete. Clearly this way the *if* information can be distinguished from the *then* information by its position in the structure. This will enable us to decide in the semantics whether a text is well-formed or not. We call this idea, that the information that we find in texts is structured in a tree-like configuration, the *texts as trees* perspective.
The use of tree structure in the semantics may seem far fetched at first sight. But note that the job that we are trying to do is to account for the structural interaction of text components in the semantics.[7] Therefore the introduction of structure in the semantics is quite a natural step to make. Also note that we have made this kind of move before: when we had to account for the links between the variables in different parts of a text, we added a component with the required information about these variables.
Whenever we meet an *end* we can actually construct the implication in the Heyting algebra, and we no longer need the tree structure. As a

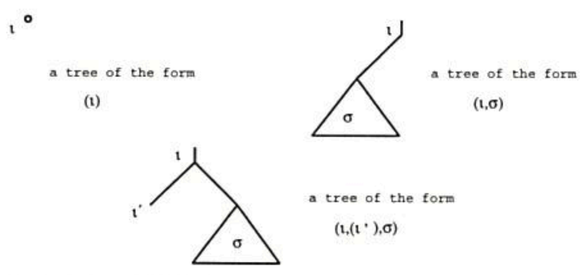
[7]This has become necessary because of the new methodological principles that we have embraced.

consequence not all binary trees have to occur in the semantics. We can restrict ourselves to trees of the following kind.

Definition 4.3.1 *Let a Heyting algebra* $\mathbf{I}$ *be given. We define the update trees over* $\mathbf{I}$, $U_{\mathbf{I}}$, *as follows:*

If $\imath \in \mathbf{I}$,	*then* $(\imath) \in U_{\mathbf{I}}$;
If $\imath \in \mathbf{I}$ *and* $\sigma \in U_{\mathbf{I}}$,	*then* $(\imath, \sigma) \in U_{\mathbf{I}}$;
If $\imath \in \mathbf{I}$, $\imath' \in \mathbf{I}$, $\sigma \in U_{\mathbf{I}}$	*then* $(\imath, (\imath'), \sigma) \in U_{\mathbf{I}}$;

Maybe one does not immediately recognise these objects as binary trees. They can be read as follows: the general format is $(\imath, (\imath'), \sigma)$ where σ is itself an update tree. The first component contains the information so far, $\imath$. We think of it as a flag at the root of the tree. The second component, $(\imath')$, contains the material that we have assumed. It is stored in the left branch of the tree. The third component, the right branch, is used for the conclusion. If one of the components is not in use, we do not write it down.[8] So we simply have $(\imath)$ if we are not processing an implication at the moment, and we have $(\imath, \sigma)$ if we are building up the antecedent of an implication. All components are filled, $(\imath, (\imath'), \sigma)$, if we have arrived at the conclusion of the implication. Since we always compute the effect of an implication as soon as we can (see the definition of [*end*] below), at most one of the three components — the rightmost — is not an element of **I**. So we can keep the following pictures in mind.



[8]We could have chosen to fill the places that are not in use with a dummy tree, but we prefer not to introduce a foreign element into the picture. As it is the tree consists of elements of the Heyting algebra only. (Note that $(\top)$ cannot play the role of the dummy tree! If we use $(\top)$ as dummy we will get confused if we are processing expressions such as *if* $\top$ *then* $\top$ *end*.)

Each time the simplest example of such a configuration arises when σ is of the form $(\imath'')$.

Before we define the interpretation of our texts on these update trees, we introduce the notion of the *final segment* of a tree. This notion will be of use in the definition of the update semantics. The fact that we can distinguish the final segment in a tree from the other parts shows that the structure of the trees as we have defined them can be interpreted 'historically': from a tree we reconstruct its construction process. We can tell which parts were built first and which parts later.

Definition 4.3.2 *We define for each tree τ its final segment, $seg_f(\tau)$, as follows:*

$$\begin{array}{lll} seg_f((\imath)) & = & (\imath); \\ seg_f((\imath,(\imath'))) & = & (\imath,(\imath')); \\ seg_f((\imath,(\imath'),(\imath''))) & = & (\imath,(\imath'),(\imath'')); \\ seg_f((\imath,\sigma)) & = & seg_f(\sigma) \\ & & if\ \sigma \neq (\imath'); \\ seg_f((\imath,(\imath'),\sigma)) & = & seg_f(\sigma) \\ & & if\ \sigma \neq (\imath''). \end{array}$$

We will write $\sigma(\!|\rho|\!)$ for σ to emphasise that $seg_f(\sigma) = \rho$ and $\sigma\{\rho'/\rho\}$ for the tree that results from replacing ρ, the final segment of σ, by ρ' in σ. If it is clear from the context what ρ is, we simply write $\sigma\{\rho'\}$.[9]

We can now define the incremental semantics of our propositional texts: with each proposition ϕ we associate a partial function on update trees, $[\phi]$, as follows.

[9]The notation is analogous to the notation $\phi(x)$ in predicate logic to indicate the free variable x in ϕ and the notation $\phi(a)$ for ϕ with x substituted by a. Note that here two different notations are necessary because we do not have, in general, $seg_f(\sigma\{\rho'/\rho\}\ ) = \rho'$. Take for example $\sigma = (\imath,(\imath'),(\top,(\top))\ )$ and $\rho' = (\imath'')$. Then $seg_f(\sigma) = (\top,(\top))$ and $seg_f(\sigma\{\rho'/(\top,(\top))\ )\} = (\imath,(\imath'),(\imath''))$.

Definition 4.3.3 *Let $\sigma \in U_{\mathbf{I}}$ be given. The following clauses define the update functions $[\phi]$ for $\phi \in Text_A$.*

$$\begin{aligned}
&\sigma\llbracket(\imath)\rrbracket\ [\bot] &&= \sigma\{(\imath \wedge \bot)\}\ ;\\
&\sigma\llbracket(\imath', (\imath))\rrbracket\ [\bot] &&= \sigma\{(\imath', (\imath \wedge \bot))\}\ ;\\
&\sigma\llbracket(\imath'', (\imath'), (\imath))\rrbracket\ [\bot] &&= \sigma\{(\imath'', (\imath'), (\imath \wedge \bot))\}\ ;\\
&&&\\
&\sigma\llbracket(\imath)\rrbracket\ [p] &&= \sigma\{(\imath \wedge \imath_p)\}\ ;\\
&\sigma\llbracket(\imath', (\imath))\rrbracket\ [p] &&= \sigma\{(\imath', (\imath \wedge \imath_p))\}\ ;\\
&\sigma\llbracket(\imath'', (\imath'), (\imath))\rrbracket\ [p] &&= \sigma\{(\imath'', (\imath'), (\imath \wedge \imath_p))\}\ ;\\
&&&\\
&\sigma\llbracket(\imath)\rrbracket\ [if] &&= \sigma\{(\imath, (\top))\}\ ;\\
&\sigma\llbracket(\imath', (\imath))\rrbracket\ [if] &&= \sigma\{(\imath', (\imath, (\top)))\}\ ;\\
&\sigma\llbracket(\imath'', (\imath'), (\imath))\rrbracket\ [if] &&= \sigma\{(\imath'', (\imath'), (\imath, (\top)))\}\ ;\\
&&&\\
&\sigma\llbracket(\imath', (\imath))\rrbracket\ [then] &&= \sigma\{(\imath', (\imath), (\top))\}\ ;\\
&\sigma\llbracket(\imath'', (\imath'), (\imath))\rrbracket\ [end] &&= \sigma\{(\imath'' \wedge (\imath' \to \imath))\}\ ;\\
&\sigma[\phi\psi] &&= (\sigma[\phi])[\psi].
\end{aligned}$$

In these clauses the update functions are defined for certain configurations of the final segment of σ. If the final segment of σ does not have this configuration, the function is undefined. As before, we can define truth as follows:

> *For $\imath \in \mathbf{I}$ we define $(\imath) \models \phi$ iff $(\imath)[\phi] = (\imath)$. We say that ϕ is true in $\imath$. We write $\models \phi$ iff $(\top) \models \phi$. We say that ϕ is true (in $\mathbf{I}$).*

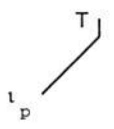
Note that for $[\bot]$, $[p]$ and $[if]$ we do not need the entire final segment: only the very latest information state in the configuration, $\imath$, is required. In $[then]$ and $[end]$ we see how the *structure* of the final segment matters in the updating process: if the final segment has the wrong shape the update functions are undefined.

In the following example we see how the updating process works.

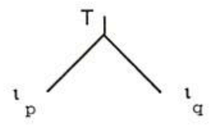
Example:

$$\begin{array}{ll}
(\top)[\mathit{if}\ p\ \mathit{then}\ q\ \mathit{end}] & = \\
(\top,(\top))[p\ \mathit{then}\ q\ \mathit{end}] & = \\
(\top,(\imath_p))[\mathit{then}\ q\ \mathit{end}] & = \\
(\top,(\imath_p),(\top))[q\ \mathit{end}] & = \\
(\top,(\imath_p),(\top\wedge\imath_q))[\mathit{end}] & = \\
(\imath_p\rightarrow\top\wedge\imath_q) & = \\
(\imath_p\rightarrow\imath_q) &
\end{array}$$

We give pictures for two of the stages in the process.



(T) [if p]



(T) [if p then q]

Now we can introduce a provisional well formedness test for the texts in $Text_A$.

1. $[\phi]$ is grammatical if $[\phi]$ is a total function and $(\top)[\phi] = (\imath)$ for some $\imath \in \mathbf{I}$.

2. $[\phi]$ is right incomplete (but left complete) if $[\phi]$ is a total function and $(\top)[\phi] \neq (\imath)$.

3. $[\phi]$ is left incomplete (and possibly right incomplete) if $[\phi]$ is a partial function but $[\phi] \neq \emptyset$.

4. $[\phi]$ is incoherent if $[\phi] = \emptyset$.

5. The texts that are not incoherent are called coherent.

Here we use *grammatical* for well formed in the strict sense. In the loose sense all coherent expressions are well formed: they can occur as a segment of a grammatical text. For example, in our terminology *if p then q* is coherent — it falls under case 2 — and *if p then q end* is grammatical. *then p if q end* is an example of an incoherent text. Note that, for now, among the left incomplete texts we cannot distinguish the right complete from the right incomplete texts: both *then q end* and *then q* fall under 3.

At this point we have an incremental semantics for our propositional texts that can distinguish *if* from *then*. This means that structural deficiencies of a text of $Text_A$ can be detected as we are interpreting it. The methodological constraints are also satisfied: for any text segment we can compute its meaning as an update function. Since composition of (partial) functions is an associative operation, associativity is satisfied. This means that we have done our job. But we have done it in a special way: using update functions as meanings. In the remaining part of the chapter we will see whether it is necessary to use an update formulation of the semantics.

4.4 Trees as an update algebra

4.4.1 Update algebras

So far we have given the semantics of texts in terms of update functions. For some purposes the *meaning as update* view is misleading. Sometimes we do not only want to see the effect of a sentence meaning: we also want to look *in*to the meaning of a sentence. In Visser (1992a) we find the notion of an *update algebra*. If an update semantics can be defined in terms of an update algebra, then there is a natural harmony between the update view on meaning and the so-called *representational* view: in this case the elements of the update algebra represent the update functions. If the update functions allow for such a representation, then clearly no conflict between the different ways of looking at meanings arises.

In this section we present trees as an update algebra. Thus a representational interpretation of texts is obtained which is in harmony with the update interpretation that we have defined in the previous section. Visser defines his update algebras as follows:

Definition 4.4.1 *A* **merge algebra** *M is a structure $(X, S, id, \bullet)$, where $id \in S \subseteq X$ and where $(X, id, \bullet)$ is a monoid (with identity element id).*

S is the set of **states** *of the algebra, $\bullet$ is called the* **merger**.

A merge algebra $M = (X, S, id, \bullet)$ is an **update algebra** *if M satisfies the following principle, called OTAT: $x \bullet y \in S \Rightarrow x \in S$.*

Intuitively, the states are the information objects that are *not* partial. They do not have to be interpreted in the light of *previous* information. They can be combined with other information objects, but this is not necessary. The other objects in X are partial: they steel bits of information from previous information states.[10] In an update algebra adding information later, on the right hand side does not help to satisfy such a demand for previous information, on the lefthand side. Visser calls this the *Once a Thief, Always a Thief*, or *OTAT* principle. The *OTAT* principle introduces the essential asymmetry in the formalism.
The elements of a merge algebra $(X, S, \bullet, id)$ generate canonical update functions on the set S of states as follows:

For each $x \in X$ we define $\Phi_x : S \to S$ as follows:

$s\Phi_x = s \bullet x$ if $s \bullet x \in S$. Otherwise $s\Phi_x$ is undefined.

It is not clear in general which functions should be allowed as update functions. Of course the set of update functions over S should contain the Φ_x. But apart from the canonical update functions that we have defined above one might want to consider other functions, for example a *might*-operator as in Veltman (1991).
It *is* clear that the class of update functions should be closed under function composition: if you update your information state with some update function and then update the result with another update function, then, surely, this whole process should also count as an update function. Hence the update functions over S form a monoid, say $(F_S, \circ, Id)$, where $\{\Phi_x : x \in X\} \subseteq F_S$ and $Id = \Phi_{id}$.[11]

[10]It might be helpful to think about the partiality of information in terms of evaluation: the truth value of the information from a state, $s \in S$ can be determined independently. But partial information ($x \in X \backslash S$) can be evaluated only if it is preceded by a suitable context.

[11]Here $\circ$ stands for function composition.

Now the notion of an update algebra is inspired by the following fact:

Proposition 4.4.2 *Let a merge algebra $(X, S, \bullet, id)$ be given. Consider the monoid of update functions $(F_S, \circ, \Phi_{id})$. Define $\Phi : (X, \bullet, id) \rightarrow (F_S, \circ, \Phi_{id})$ by $x\Phi = \Phi_x$. Then:*
Φ is a homomorphism of monoids iff $(X, S, \bullet, id)$ is an update algebra.$\Box$

The proof is elementary. It can be found in Visser (1992a).
The fact that Φ is a homomorphism guarantees that $\Phi_{x \bullet y} = \Phi_x \circ \Phi_y$. This implies that

$$\begin{aligned}
&\Phi_{x\bullet(y\bullet z)} = \\
&\Phi_x \circ \Phi_{y\bullet z} = \\
&\Phi_x \circ (\Phi_y \circ \Phi_z) = (\Phi_x \circ \Phi_y) \circ \Phi_z = \\
&(\Phi_{x\bullet y}) \circ \Phi_z \\
&\Phi_{(x\bullet y)\bullet z}
\end{aligned}$$

(by associativity of function composition). Thereby updating with the elements of an update algebra is a process that can be done incrementally and satisfies the break in principle. So in update algebras the methodological principles are always satisfied.

4.4.2 Partial trees

Now we show that update trees fit the update algebra picture. We define a monoid of trees such that we find the update functions of definition 4.3.3 among its canonical updates. Then it will be clear that the text semantics that we have developed so far can be handled in a representational semantics as well as in update style.
In order to make an update algebra of trees we have to find a suitable notion of *partial tree*. We obtain this notion by taking a different perspective on trees: instead of regarding trees as fixed objects, we now treat them as things that grow. In our set up it is the process of growth that we are mainly interested in, since this is where the update functions come in: we have seen that updates with the information that we find in texts are represented as instructions to *build* update trees.
The construction process follows a fixed route through the tree: the left-to-right, top-down path. If we want to analyse the construction process

of some tree we simply have to follow this path. In this way the stages of the construction process are represented in the tree by the segments of the path.[12]
Now we make the following step: we no longer distinguish between a tree and its construction process. So we think of trees only in terms of the left-to-right, top-to-bottom path through the tree. Then it is but a small step to consider the segments of such a path as *partial trees.* We take these segments as the elements of the update algebra. Note that among the elements of update algebra we will find segments that actually correspond to an update tree. These will be the *states* of the update algebra.
In the following definition we describe the tree segments systematically. We will use the terms *(partial) tree, (tree) path* and *(tree) segment* to refer to them.

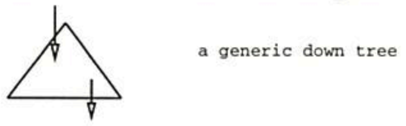
Definition 4.4.3 *We define the (partial) trees over some HA* $\mathbf{I}$, $T_{\mathbf{I}}$, *inductively. In our definition we have to distinguish the subclasses* $downT_{\mathbf{I}}$, *for the* down trees *and* $upT_{\mathbf{I}}$, *for the* up trees.[13]

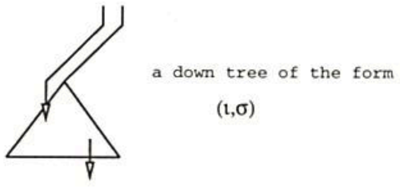
1. $\imath \in \mathbf{I}$ $\Rightarrow (\imath) \in downT_{\mathbf{I}} \cap upT_{\mathbf{I}}$;
2. $\imath \in \mathbf{I},\ \sigma \in downT_{\mathbf{I}}$ $\Rightarrow (\imath, \sigma) \in downT_{\mathbf{I}}$;
3. $\imath \in \mathbf{I},\ \sigma \in upT_{\mathbf{I}}$ $\Rightarrow (\sigma, \imath) \in upT_{\mathbf{I}}$;
4. $\imath,\ \imath' \in \mathbf{I},\ \sigma \in downT_{\mathbf{I}}$ $\Rightarrow (\imath, (\imath'), \sigma) \in downT_{\mathbf{I}}$;
5. $\imath,\ \imath' \in \mathbf{I},\ \sigma \in upT_{\mathbf{I}}$ $\Rightarrow (\sigma, (\imath), \imath') \in upT_{\mathbf{I}}$;
   $downT_{\mathbf{I}} \cup upT_{\mathbf{I}} \subseteq T_{\mathbf{I}}$
6. $(\imath, \sigma) \in downT_{\mathbf{I}},\ (\sigma', \imath) \in upT_{\mathbf{I}}$ $\Rightarrow (\sigma', \imath, \sigma) \in T_{\mathbf{I}}$;[14]
7. $\lambda \in upT_{\mathbf{I}},\ \rho \in downT_{\mathbf{I}}$ $\Rightarrow (\lambda, \rho) \in T_{\mathbf{I}}$;
8. $\mathbf{0} \in T_{\mathbf{I}}$.

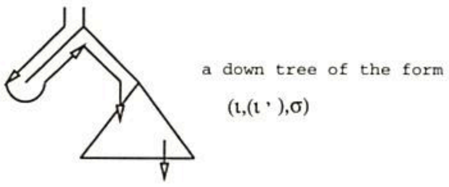
Each of these clauses corresponds to a kind of segment through an update tree. Note that we distinguish *down* trees and *up* trees. The down trees — cases (2) and (4) — are the segments that actually correspond to an update tree. These paths start at some root $\imath$ and then go down into the

[12]Maybe the reader has noticed that our habit of collapsing completed subtrees somewhat disturbs the analogy between the construction and the path. For the moment we will ignore this mismatch, but it will be taken care of later.
[13]This terminology will be explained below.
[14]Note that either $\sigma = \lambda$ for some $\lambda \in upT_{\mathbf{I}}$ or $\sigma = ((\imath'), \lambda)$ for some $\imath' \in \mathbf{I}$, $\lambda \in upT_{\mathbf{I}}$. Similarly for σ'.

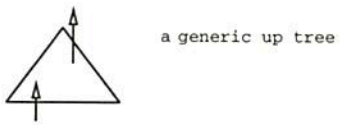
tree below that root. With these segments we can simply think of the pictures of trees that we also used in the previous section. We just have to add arrows to indicate the direction of the path.

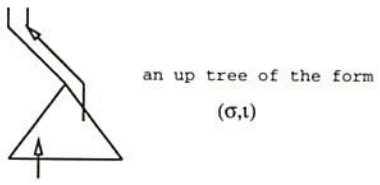


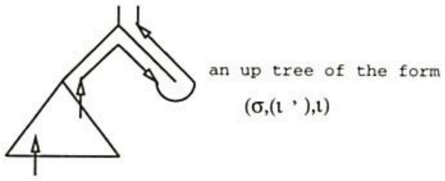




The up trees — cases (3) ans (5) — are the mirror images of the down trees. They are segments that start somewhere in a tree and then go up to its root. For up trees we use as pictures the mirror images of the pictures for down trees.



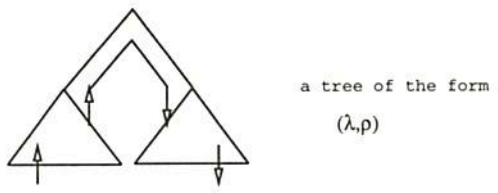




Now we have seen the path segments that start at a root and go down into the tree and the path segments that start somewhere in the tree and climb up to the root. This leaves two cases to consider: the segments that both start and finish at a root and the segments that neither start nor finish at a root.

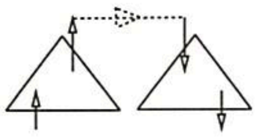
The first case gives those segments that actually describe a completed subtree. Since we are in the habit of collapsing completed subtrees, we will not find many of these paths in our trees. Only the degenerate case can occur, where a path starts at a root and does not leave it. This case is handled by (1) in the definition. Such a tree is both a down and an up tree.

The second case, of the segments that neither start nor finish at a root, can again be divided into two cases. First there are the paths that describe a jump from assumption to conclusion. These paths not meet the root of the tree in which they occur. They are the bridges between left and right branches of trees. We describe them in case (7) and we use the following pictures for them.



But there is another kind of segment that does not start or finish at a root. This case is described by (6). Here the comparison with paths in binary trees breaks down. As one can see in (6), we are in a situation where an up tree is followed by a down tree. The up tree moves up to some root, and then the down tree moves down from *this* root. In the path of a binary tree this cannot happen: each node has just one subtree below it and if we have completed the path through this subtree, the only way to continue the path is by going to the next node (on the right hand side). This is the point where we see how our habit of collapsing subtrees somewhat spoils the analogy with the paths. For in our situation, if the path through some subtree is completed, we collapse this subtree and add the result of this collapse to the node. After we have collapsed the tree, there is only a node left. Then we can simply start a new subtree from this same node. Case (6) describes this moment when one subtree is completed and the next one is built at the same node.[15] For this situation we use the following kind of picture:

[15]We will use these situations in the representation of the conjunction of two implications:
if p then q end if r then s end.
The information of both these implications should be stored at the same node.



Finally there is the tree **0**. In fact **0** is not really a tree: we will use **0** to describe the situation in which the construction process has reached the error state: something has gone wrong and we no longer know what to do. So **0** does *not* correspond to the empty tree.[16] In fact it is just the opposite: the empty tree is harmless and really does not do anything. **0**, on the other hand, is lethal in all situations.

Now we know how to think about partial trees as tree paths. Sometimes it is even easier to think of them in terms of their basic components. This brings an inductive definition of the class af partial trees within reach. We distinguish the following *basic trees*.

Definition 4.4.4 *We distinguish the following basic trees in* $T_{\mathbf{I}}$*:*

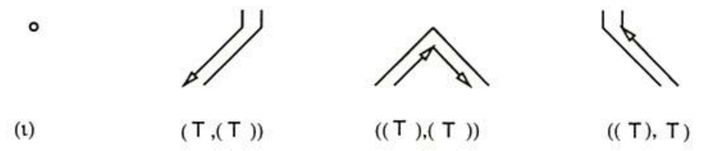
$(\imath)$ *is a basic tree for each* $\imath \in \mathbf{I}$*. (Think of an atomic text 'p'.)*

$(\top, (\top))$ *is a basic tree. (Think of the instruction 'if'.)*

$((\top), \top)$ *is a basic tree. (Think of the instruction 'end'.)*

$((\top), (\top))$ *is a basic tree. (Think of the instruction 'then'.)*

In a picture:

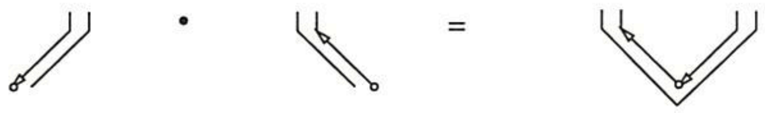


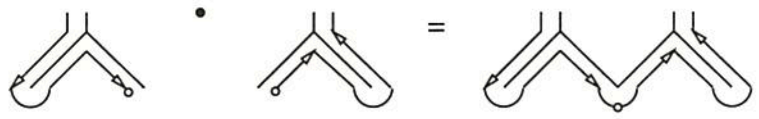
We can think of all tree segments in terms of these basic segments: big segments are obtained by glueing together these basic segments. Before we can make this precise, we have to explain *how* tree segments are glued together. This is the topic of the next section.

[16]Here $(\top)$ plays the role of the empty tree.

4.4.3 The merger of trees

In this section we describe how segments of tree paths can be merged into bigger segments. This merging operation will be the monoidal operation of the update algebra of partial trees.
The basic idea behind the merger of trees is easy: if two tree segments τ and τ' have to be merged, we first complete the path described by τ an then we simply continue along the path described by τ'. Or rather, we *try* to continue along τ'. For, if we try to merge two paths, something can go wrong.[17] Consider the following examples of such a situation. [18]



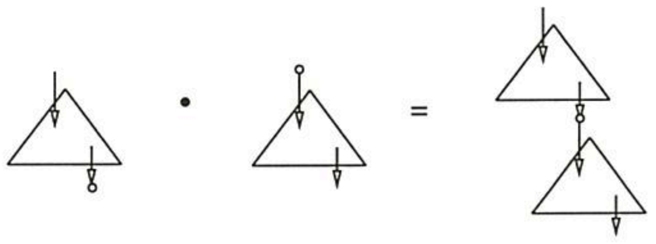


If we simply glue together the segments as indicated in the pictures, we get something which, although it makes sense geometrically, is useless in our set up. For it is clear that the result is not a segment of a (left-to-right, top-down) path through a binary tree. In these cases we reach the error state, for which we have introduced **0**. So **0** can also be read as 'undefined'.
In most cases, however, things will not go wrong. For example, if τ is a down tree, i.e. a path downwards from some root, and also τ' is a down tree, then it is clear that the result of glueing τ and τ' together, will always be a sensible path through an update tree. In fact it is clear that

[17]It may help to compare the cases where the merger goes wrong with the cases where the update functions of the previous section were undefined.

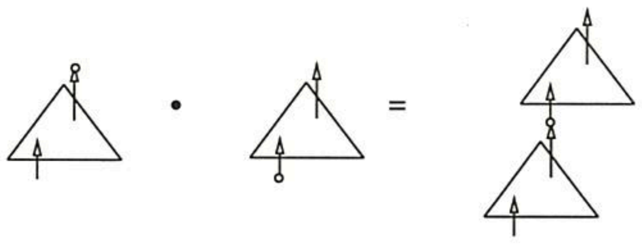
[18]We use $\bullet$ as notation for the merger and $\circ$ to indicate the point where the segments are glued together. Note that this is not really necessary since the arrows already give enough information to determine what should go where.

the result will be a down tree as well.

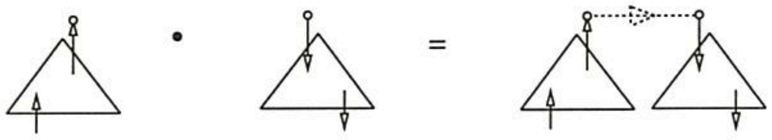


There are also cases where τ is not itself a down tree, but does look like a down tree at the point of contact. These cases — where τ is of one of the forms $(\sigma', \imath, \sigma)$ or (λ, ρ) (for a non-trivial tree ρ) — work similarly so we do not have to discuss them separately. This is why we will ignore this kind of situation in what follows. We can concentrate on what happens at the point of contact.

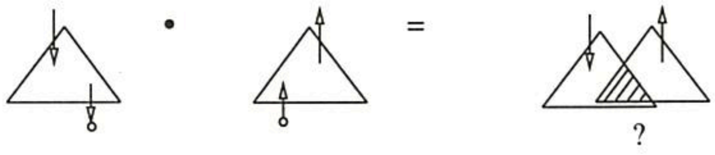
In the dual case, where two up trees meet, we cannot meet any problems either.



A third kind of situation where the merger cannot go wrong is the situation where an up tree is merged with a down tree. As was noted above, this is a case where our geometrical intuitions about paths have to be stretched a little. In these cases the first tree, which is an up tree, and the second tree, which is a down tree, should be thought of as hanging at the same root, but not at the same time. The second tree can only be built after the collapse of the first tree. In pictures this looks as follows.



But the merger of trees can give rise to problems when a down tree and an up tree meet. In such a situation the second path, up the tree, has to fit in the tree associated with the first path.

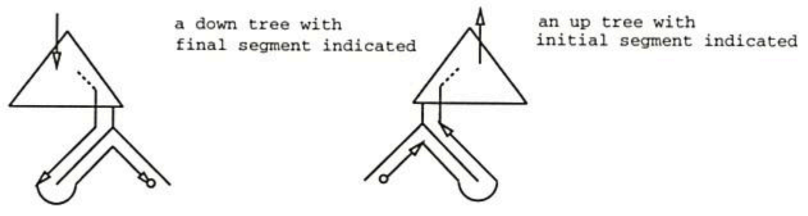


We have already seen situations where this goes wrong. In both examples it was easy to see in advance that something would go wrong, but in general this can be quite difficult. Fortunately we do not have to see it in advance. We can simply check it step by step, as we are performing the merger. In each step of the merging process our actions will be determined by what we find locally, at the point of contact. There we just have to check whether the *final segment* of the first tree and *initial segment* of the second tree match. We have already defined the notion of the final segment for update trees, i.e for down trees. Here we extend this notion to partial trees. We also define the dual notion of the initial segment of a tree, that gives for each path the configuration that we find at the beginning of the path. As long as we were dealing with down trees only, we always found a root at the beginning of our paths. But since we also have up trees, there are more ways in which a path can start.

Definition 4.4.5 *We define the functions seg_f and seg_i on trees in $T_{\mathbf{I}}$ as follows.*

$$
\begin{array}{ll}
seg_f((\imath)) = (\imath), & seg_i((\imath)) = (\imath); \\
seg_f((\sigma, \imath)) = (\imath); & seg_i((\imath, \sigma)) = (\imath), \\
seg_f((\imath, (\imath'))) = ((\imath, (\imath'))); & seg_i(((\imath'), \imath)) = ((\imath'), \imath); \\
seg_f((\imath, \sigma)) = seg_f(\sigma), & seg_i((\sigma, \imath)) = seg_i(\sigma), \\
\quad \text{if } \sigma \neq (\imath'); & \quad \text{if } \sigma \neq (\imath'); \\
seg_f((\imath, (\imath'), (\imath''))) = (\imath, (\imath'), (\imath'')); & seg_i(((\imath''), (\imath'), \imath)) = ((\imath''), (\imath'), \imath); \\
seg_f((\imath, (\imath'), \sigma)) = seg_f(\sigma), & seg_i((\sigma, (\imath'), \imath)) = seg_i(\sigma), \\
\quad \text{if } \sigma \neq (\imath''); & \quad \text{if } \sigma \neq (\imath''); \\
seg_f((\sigma', \imath, \sigma)) = seg_f((\imath, \sigma)); & seg_i((\sigma', \imath, \sigma)) = seg_i((\sigma', \imath)); \\
seg_f((\lambda, (\imath))) = (\lambda, (\imath)); & seg_i((\imath), \rho)) = ((\imath), \rho); \\
seg_f((\lambda, \rho)) = seg_f(\rho), & seg_i((\lambda, \rho)) = seg_i(\lambda), \\
\quad \text{if } \rho \neq (\imath); & \quad \text{if } \lambda \neq (\imath); \\
seg_f(\mathbf{0}) = \mathbf{0}; & seg_i(\mathbf{0}) = \mathbf{0}.
\end{array}
$$

We will keep the same notation for final segments, writing $\tau(\!|\rho|\!)$ to indicate that $seg_f(\tau) = \rho$ and $\tau\{\rho'\}$ if we have substituted ρ' for the final segment of τ. For initial segments we introduce similar notation: $(\!|\lambda|\!)\tau$ and $\{\lambda'\}\tau$. If we want to indicate the final or initial segment in a picture, this looks as follows.

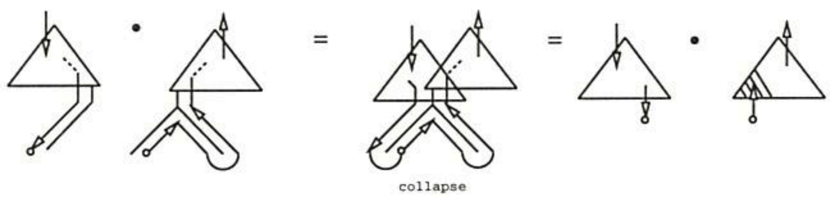


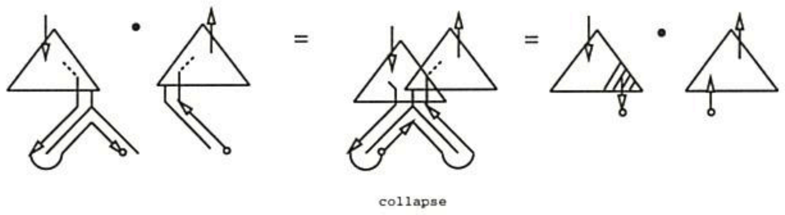
We need the final and the initial segment to keep track of shape of the path at its end and beginning respectively. Note that up trees have a trivial final segment, of the form $(\imath)$, and that down trees have a trivial initial segment. Another interesting case are the trees of the form (λ, ρ), but we will not discuss this case until we need it.

We have included a clause for $\mathbf{0}$ in the definition. Of course the final or initial segment of the undefined tree is not a particularly meaningful notion, but this way seg_f and seg_i become total functions. This will make some technical details slightly more elegant later on.

Now we can get back to our description of the merger of a down tree with an up tree. We see that when a down tree and an up tree meet, there are two possibilities. Either the final segment of the first tree and the initial segment of the second tree clash as in the examples above. Then we reach the error state: there is a local mismatch between the two paths.

Or else the final and initial segments have one of the following shapes.





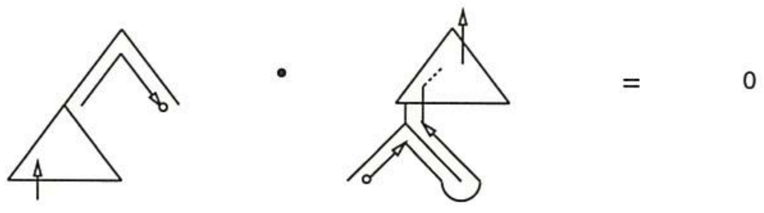
In the two cases indicated here we see that locally the paths match. The final segment and the initial segment together form a complete subtree. Now we can simply compute the information of this subtree, collapse the subtree itself and add the information at its root.[19] Then the two trees will have a new final and initial segment, and we can check again whether

[19]We have to chose in which of the two trees to store the information of the subtree. We will prove later on that the choice does not really matter. If the reader cannot wait for this, she can also store the information in both trees.

these match. Continuing in this fashion, we either reach the error state at some point or we reach a situation which is no longer of this type. In that case either the down tree of the first path is absorbed by the second path, or the up tree of the second path is absorbed by the first path. Then we are in a situation where at the point of contact it is no longer the case that a down tree and an up tree meet, i.e. we are no longer in a problematic case.

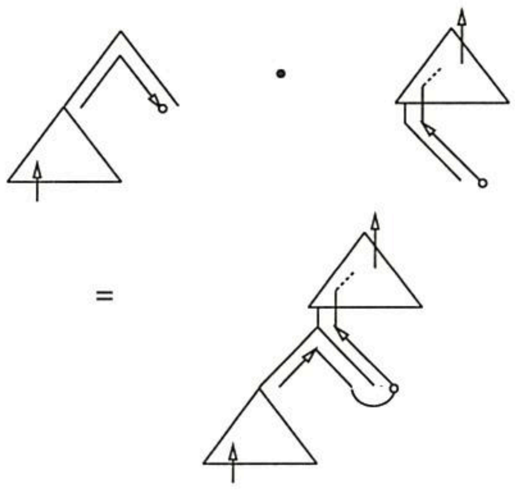
Before we make formal sense of this pictorial explanation, we have to consider one more case. This is the case where one (or two) of the trees is of the shape (λ, ρ). We already explained above that, if the first tree is of this form and in case ρ is not a trivial tree, (λ, ρ) behaves just like a down tree at the point of contact. But if $\rho = (\imath)$ for some $\imath \in \mathbf{I}$, then the bridge shape of (λ, ρ) is relevant. This explains the definition of the final segment: if $\rho = (\imath)$, then $seg_f((\lambda, \rho)) = (\lambda, \rho)$. Else we just get $seg_f((\lambda, \rho)) = seg_f(\rho)$.

In such a bridge shaped situation a clash can occur, for example in:



If we simply glue together these paths, then we get something that cannot occur in a binary tree. What we get reminds us of the second example of a mismatch discussed above.

There can also be a match between the bridge and the initial segment. This again reminds us of something that we have seen before. But now, even if the trees match, we never get a complete subtree. So we do not get a collapse. In a picture:



We see that we do not get a complete subtree, so there is no collapse. Of course the symmetrical situation, where the second tree is $((\imath), \rho)$, is handled analogously.

Now we are ready for the formal definition of the merger of partial trees. It will simply be summary of the explanation above.

Definition 4.4.6 *We define the merger of two trees* τ *and* τ', $\tau \bullet \tau'$. *We distinguish the following cases, considering all combinations of final and initial segments:*

$$\tau \bullet \mathbf{0} \overset{1}{=} \mathbf{0} = \mathbf{0} \bullet \tau'$$

$$\tau[\![(\imath_0)]\!] \bullet [\![(\imath_1)]\!]\tau'$$

$We\ distinguish\ four\ subcases:$

$$\begin{aligned}
(\imath_0) \bullet (\imath_1) &\overset{a}{=} (\imath_0 \wedge \imath_1) \\
(\sigma, \imath_0) \bullet (\imath_1) &\overset{b}{=} (\sigma, \imath_0 \wedge \imath_1) \\
(\imath_0) \bullet (\imath_1, \sigma') &\overset{c}{=} (\imath_0 \wedge \imath_1, \sigma') \\
(\sigma, \imath_0) \bullet (\imath_1, \sigma') &\overset{d}{=} (\sigma, \imath_0 \wedge \imath_1, \sigma')
\end{aligned}$$

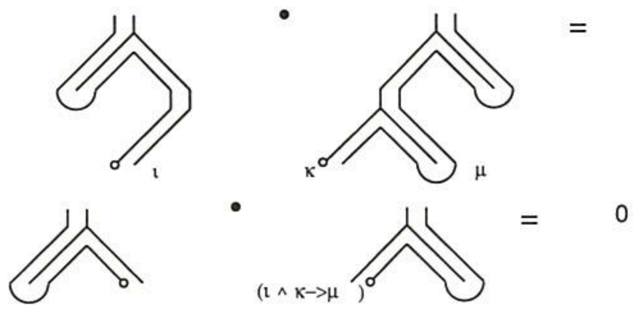
$$\begin{aligned}
\tau[\![(\imath_0, (\imath_1))]\!] \bullet [\![(\imath_2)]\!]\tau' &\overset{3}{=} \tau\{(\imath_0,\ (\imath_1) \bullet [\![(\imath_2)]\!]\tau')\} \\
\tau[\![(\imath_0)]\!] \bullet [\![((\imath_1), \imath_2)]\!]\tau' &\overset{4}{=} \{(\tau[\![(\imath_0)]\!] \bullet (\imath_1)\ , \imath_2)\}\tau'
\end{aligned}$$

$$\begin{aligned}
\tau\llbracket(\imath_0,(\imath_1),(\imath_2))\rrbracket \bullet \llbracket(\imath_3)\rrbracket\tau' &\overset{5}{=} \tau\{(\imath_0,(\imath_1),\ (\imath_2)\bullet\llbracket(\imath_3)\rrbracket\tau')\}\\
\tau\llbracket(\imath_0)\rrbracket \bullet \llbracket((\imath_1),(\imath_2),\imath_3)\rrbracket\tau' &\overset{6}{=} \{(\tau\llbracket(\imath_0)\rrbracket \bullet(\imath_1)\ ,(\imath_2),\imath_3)\}\tau'\\
\tau\llbracket(\imath_0,(\imath_1),(\imath_2))\rrbracket \bullet \llbracket((\imath_3),\imath_4)\rrbracket\tau' &\overset{7}{=} (\tau\{(\imath_0)\} \bullet(\imath_1\to\imath_2\wedge\imath_3)) \bullet \{(\imath_4)\}\tau'\\
\tau\llbracket(\imath_0,(\imath_1))\rrbracket \bullet \llbracket((\imath_2),(\imath_3),\imath_4)\rrbracket\tau' &\overset{8}{=} \tau\{(\imath_0)\} \bullet ((\imath_1\wedge\imath_2\to\imath_3)\bullet\{(\imath_4)\}\tau')\\
(\lambda,(\imath_0))\bullet \llbracket(\imath_1)\rrbracket\tau' &\overset{9}{=} (\lambda,\ (\imath_0)\bullet\llbracket(\imath_1)\rrbracket\tau')\\
\tau\llbracket(\imath_0)\rrbracket \bullet((\imath_1),\rho) &\overset{10}{=} (\tau\llbracket(\imath_0)\rrbracket \bullet(\imath_1)\ ,\rho)\\
(\lambda,(\imath_0))\bullet \llbracket((\imath_1),\imath_2)\rrbracket\tau' &\overset{11}{=} \{(\lambda,(\imath_0)\bullet(\imath_1)\ ,\imath_2)\}\tau'\\
\tau\llbracket(\imath_0,(\imath_1))\rrbracket \bullet((\imath_2),\rho) &\overset{12}{=} \tau\{(\imath_0,(\imath_1)\bullet(\imath_2)\ ,\rho)\}\\
\tau\llbracket(\imath_0,(\imath_1))\rrbracket \bullet \llbracket((\imath_2),\imath_3)\rrbracket\tau' &\overset{13}{=} \mathbf{0}\\
\tau\llbracket(\imath_0,(\imath_1),(\imath_2))\rrbracket \bullet \llbracket((\imath_3),(\imath_4),\imath_5)\rrbracket\tau' &\overset{14}{=} \mathbf{0}\\
\tau\llbracket(\imath_0,(\imath_1),(\imath_2))\rrbracket \bullet((\imath_3),\rho) &\overset{15}{=} \mathbf{0}\\
(\lambda,(\imath_0))\bullet \llbracket((\imath_1),(\imath_2),\imath_3)\rrbracket\tau' &\overset{16}{=} \mathbf{0}\\
(\lambda,(\imath_0))\bullet((\imath_1),\rho) &\overset{17}{=} \mathbf{0}
\end{aligned}$$

The definition contains a lot of cases: one for each combination of final and initial segment. For each case there is also a symmetrical one. In our presentation each case is followed by its mirror image.

Each case in this definition has already been covered in the pictorial explanation above. The cases (3), (5) and (9) are cases where the second tree is a down tree. These are easy cases, where we can just glue the paths together and nothing can go wrong. The cases (4), (6) and (10) are dual: here the first tree is an up tree. Case (2) is the situation where an up tree is followed by a down tree. This is also a situation in which nothing can go wrong. The real work has to be done in the remaining cases, (11)-(17), where either a down tree meets an up tree or else one of the trees has a bridge shape. Here we can make one step of the computation as indicated and then we continue with the new situation.

We give one last example of how this works in pictures. In the example we see how a down tree and an up tree are merged. In the first step the final and initial segment match. So a subtree is completed and the result, $(\iota \wedge \kappa \to \mu)$, is added to the second tree. In the next step we see that the final segment and the initial segment do not match. We reach the error state, $\mathbf{0}$, and the computation stops.



Now it is not difficult to prove our claim that the class of the trees over **I** can be generated from the basic trees with the merge operation •.

Lemma 4.4.7 (Generation Lemma) *If $\tau \in T_{\mathbf{I}}$, then τ can be constructed from basic trees with a finite number of applications of* •.

Proof: In the proof we follow the inductive definition of $T_{\mathbf{I}}$. We will assume that the conditions of its clauses are satisfied. (Recall that for all $\tau \in downT_{\mathbf{I}}$, $seg_i(\tau)$ is of the form $(\imath)$ and that for $\tau \in upT_{\mathbf{I}}$, $seg_f(\tau)$ is of the form $(\imath)$.)

1. $(\imath) \in T_{\mathbf{I}}$ is a basic tree.

2. If σ is constructed from basic trees, then $(\imath) \bullet ((\top, (\top)) \bullet \sigma)$ gives a construction of $(\imath, \sigma)$ from basic trees.

3. If σ is constructed from basic trees, then $(\sigma \bullet ((\top), \top)) \bullet (\imath)$ gives a construction of $(\sigma, \imath)$ from basic trees.

4. If σ is constructed from basic trees, then $(\imath) \bullet ((\top, (\top)) \bullet ((\imath') \bullet (((\top), (\top)) \bullet \sigma)))$ gives the required construction of $(\imath, (\imath'), \sigma)$.

5. If σ is constructed from basic trees, then $(((\sigma \bullet ((\top), (\top))) \bullet (\imath')) \bullet ((\top), \top)) \bullet (\imath)$ gives the required construction of $(\sigma, (\imath'), \imath)$.

6. If σ' and σ are constructed from basic trees, then $(\sigma' \bullet (((\top), \top) \bullet ((\imath) \bullet ((\top, (\top)) \bullet \sigma))))$ gives the required construction of $(\sigma', \imath, \sigma)$.

7. If λ and ρ are constructed from basic trees, then $\lambda \bullet (((\top), (\top)) \bullet \rho)$ gives the required construction of (λ, ρ).

8. $\mathbf{0} = ((\top), (\top)) \bullet ((\top), (\top))$.

□

4.4.4 Associativity

Now we go on to prove that the merger is an associative operation on partial trees, thus ensuring that what we have defined is a monoid. We find that, because of the generation lemma, the following result suffices to prove associativity.

Proposition 4.4.8 (Basic Associativity) *Let two trees τ and τ' and a basic tree β be given. Then $(\tau \bullet \beta) \bullet \tau' = \tau \bullet (\beta \bullet \tau')$.*

Proof: Appendix. □

We can extend this associativity result as follows:

Proposition 4.4.9 (Full Associativity) *Let three trees $\tau_0,\ \tau_1,\ \tau_2 \in T_{\mathbf{I}}$ be given. Then:*

$$(\tau_0 \bullet \tau_1) \bullet \tau_2 \;=\; \tau_0 \bullet (\tau_1 \bullet \tau_2).$$

Proof: By the generation lemma we can write the τ_i as products of basic trees. Let n_1 be the number of basic trees we need for τ_1. The proof will be by induction on n_1.
If $n_1 = 1$, then τ_1 is a basic tree and we are done by the previous proposition. So let $n_1 = n + 2$ and assume that the statement holds whenever $n_1 < n + 2$. Then τ_1 is a product of basic trees and can be written (by the induction hypothesis) $\tau_1 = \tau \bullet \beta$ for some tree τ and a basic tree β. Now:

$$\begin{aligned}
(\tau_0 \bullet \tau_1) \bullet \tau_2 &= \\
(\tau_0 \bullet (\tau \bullet \beta)) \bullet \tau_2 &= \quad \text{(by induction hypothesis on } \tau_1\text{)} \\
((\tau_0 \bullet \tau) \bullet \beta) \bullet \tau_2 &= \quad \text{(by induction hypothesis on } \tau_0 \bullet (\tau \bullet \beta)\text{)} \\
(\tau_0 \bullet \tau) \bullet (\beta \bullet \tau_2) &= \quad \text{(by induction hypothesis (for } n_1 = 1\text{))}
\end{aligned}$$

$$\begin{aligned}
\tau_0 \bullet (\tau \bullet (\beta \bullet \tau_2)) &= \text{(by induction hypothesis (τ is smaller} \\
&\qquad \text{than τ_1!))} \\
\tau_0 \bullet ((\tau \bullet \beta) \bullet \tau_2) &= \text{(by induction hypothesis on $\tau \bullet (\beta \bullet \tau_2)$)} \\
\tau_0 \bullet (\tau_1 \bullet \tau_2).&
\end{aligned}$$

This proves the proposition.
□

Now it is clear that the partial trees as we have defined them in this section form a monoid. This means that the partial trees *may* provide a suitable setting for text semantics: in section 4.1 associativity was introduced as the methodological constraint on text semantics.
The next step is to check that the partial trees actually form an update algebra, as the title of this section promised, with as states the update trees of the previous section. After that we have to see whether the update functions of section 4.3 really can be represented in this update algebra.

Proposition 4.4.10 *$(T_{\mathbf{I}},\ downT_{\mathbf{I}},\ \bullet,\ (\top))$ is an update algebra.*

Proof: We know that $(T_{\mathbf{I}},\ \bullet)$ is a monoid. It is clear that $(\top)$ is its unit. It is not difficult to check that *OTAT* holds:

$$\text{if } \tau \bullet \tau' \in downT_{\mathbf{I}}, \text{ then already } \tau \in T_{\mathbf{I}}.$$

□

4.5 Trees and texts

In section 4.3 we have seen that texts can be interpreted as update functions on down trees and in section 4.4 we have seen how trees form an update algebra. In this section we make the relation between the semantics of section 4.3 and the trees of section 4.4 precise. First we define the tree representation of a text.

Definition 4.5.1 *For a text $\phi \in Text_A$ we define its tree representation $[\![\phi]\!] \in T_{\mathbf{I}}$.*

$$[\![\bot]\!] = (\bot);$$

$$[\![p]\!] = (\imath_p);$$

$$[\![if]\!] = (\top, (\top));$$

$$[\![then]\!] = ((\top), (\top));$$

$$[\![end]\!] = ((\top), \top);$$

$$[\![\phi\psi]\!] = [\![\phi]\!] \bullet [\![\psi]\!].$$

Now we can check that this tree representation indeed generates the update functions from section 4.3.

Proposition 4.5.2 *Let a text $\phi \in Text_A$ be given. Then $[\phi] = \Phi_{[\![\phi]\!]}$.*

Proof: The proof for the basic cases $\bot$, p, if, *then* and *end* consists of a careful comparison of the clauses of definition 4.3.3 with the corresponding clauses in definition 4.4.6.[20] For compound texts, $\phi\psi$, the result is a direct consequence of the fact that $T_{\mathbf{I}}$ is an update algebra:

$$[\phi\psi] = [\phi] \circ [\psi] = \Phi_{[\![\phi]\!]} \circ \Phi_{[\![\psi]\!]} = \Phi_{[\![\phi]\!] \bullet [\![\psi]\!]} = \Phi_{[\![\phi\psi]\!]}.$$

□

So we have an equivalent representational semantics for the update semantics of section 4.3.
Thereby we also inherit the notion of truth from section 4.3. The following corollary can even been seen as an explanation of the notion of truth we defined there: it turns out that texts that are true (in **I**) have $(\top)$ as representation.

Corollary 4.5.3 *Let a text $\phi \in Text_A$ be given. Then ϕ is true iff $[\![\phi]\!] = (\top)$.*

[20] Of course we have to read **0** as undefined (or vice versa).

Proof: Recall that ϕ is true iff $(\top)[\phi] = (\top)$. Hence ϕ is true iff $(\top) \bullet [\![\phi]\!] = (\top)$ iff $[\![\phi]\!] = (\top)$. $\Box$

With the tree representation of texts we have obtained a more refined test of well formedness: the grammatical texts have a trivial tree representation — $(\imath)$ for some $\imath \in \mathbf{I}$ — the coherent texts are precisely the texts that are not represented by $\mathbf{0}$ and the left (respectively right) complete texts are the texts that have a down (up) tree as a representation. The advantage over the test with the update functions is that we can now easily distinguish among the trees that are both left and right incomplete from the texts that are just left incomplete.

4.6 Discussion

The main conclusion of this chapter is that an incremental semantics (of texts) is feasible, even if typically non-associative phenomena occur. We have used implication as an example of a non-associative operation, but if we look at the techniques that we have used, we see that these techniques do not depend on the nature of implication. They are applicable to any n-ary operation which has a well defined semantics. Suppose that we have an n-ary construction $op(\phi_1, \ldots, \phi_n)$ that corresponds in the semantics to some n-ary operation OP. Then we simply add sufficiently many extra constants to the language, $start_{op}$, $mark_{i_{op}}$ and end_{op} say. Now we can represent $op(\phi_1, \ldots, \phi_n)$ as $start_{op}\phi_1\ mark_{1_{op}}\ \ldots\ mark_{n-1_{op}}\ \phi_n\ end_{op}$. In the case that we have worked out here, the binary operation is implication and we have added constants if as a start symbol, *then* as a marker between the argument places and *end* as a closing symbol. This binary operation gave rise to binary trees in the semantics, in the general case we will have n-ary trees. In the interpretation of this expression we store the n arguments of OP in the n branches of such a tree. When we meet the constand end_{op}, we compute the operation OP on the n branches. So we have developed a general strategy to deal with these phenomena by exchanging non-associativity for structured memory.
It was also shown that the update view on semantics and our tree semantics are compatible. We have been able to fit our update functions in the general frame of Visser's update algebras. In an update algebra the static meanings generate update functions canonically, but it is not

excluded that also other update functions exist. This seems to represent a very reasonable view on the relation between static and dynamic semantics: it is hard to imagine static meanings that do not give rise naturally to update functions, but, at least at first sight, it is not clear that all ways to update information states should be representable statically, as the meaning of some text:[21] our text language simply may not be rich enough.

We have used *binary* trees to represent the slots in memory that we need for proof-like texts. For the kind of texts we consider this is not an unreasonable choice. But our ways of reasoning do not always fit the binary format.[22] For example, we tend to use *intermediate* conclusions, as in

> Suppose Mary shows up. Then she will bring her dog along with her. And therefore Bob and his cat will be forced to leave.

If we want to represent such situations in our approach, binary trees will not be sufficient. We would need structures of flexible length to handle an arbitrary number of intermediate conclusions. This would make the objects in our semantics more complex. Another problem would be the semantics of *end*: there we would have to compute the content of such a complex structure. But it is not obvious how this should be done. The relation between the three statements in the example clearly is not very simple and there is room for discussion about what exactly this relation is. For example, is the fact that *'if Mary shows up, then she will bring her dog'* part of the evidence on which we base our conclusion that *'Bob and his cat will leave'*? Or does the conclusion only depend on the information that *'Mary shows up'* and that *'she will bring her dog along with her'* and **not** on the connection between these events?

Perhaps we should also make a remark about other kinds of texts. The texts in this chapter are all of the same kind, the kind that comes with *if ... then* structure. But in the general case different types of texts are mixed. We find small *arguments* in long *stories*, in which not only a course

[21] Although the idea is already implicit in Visser (1992a), it was Patrick Blackburn who pointed out to me that one can think about the relation between update semantics and static meanings in terms of representable functions.

[22] In Zeinstra (1990) an attempt is made to work with a more flexible language. She also makes an attempt at an incremental semantics.

of events is described, but also more or less extensive *comments* on these events are included. Each of these kinds of texts has its own peculiarities which have to be taken into account in the semantics. In fact it seems that in a text we find a nesting of these types of texts, each of which has features that are crucial for the interpretation of the text and the sentences of which it is made up. In our approach this will give rise to different kinds of trees in the semantic universe. In such a mixed semantic universe we will have to include information that tells us what kind of tree we are working in currently. In this chapter this is not yet necessary, since there only is one kind of tree. But once this kind of information is added, the extension of the approach should be straightforward.

So we see that the techniques that we have developed are quite powerful. They can be adapted easily to treat other operations instead of implication and with a little extra help they should also suffice for a dynamic semantics of 'mixed texts'. Still there seem to be some generalisations that we are missing. If we compare the semantics that we have developed here with a *DRT*-style solution for the semantics of anaphora, then we see that in both cases a similar move is made. In *DRT* the meanings of texts are context-content pairs, (X, F). Here the component X was added to the semantics in order to account for anaphoric phenomena in texts. So the truth-conditional information F had to be embedded into a context before the anaphoric interaction of the text meanings could be described satisfactorily. Here we see a similar move: instead of just working with the truth-conditional information $\imath \in \mathbf{I}$, we have to embed this information in bits of tree structure. These bits of tree structure serve exactly the same purpose as the context sets of *DRT*: they allow us to represent the interaction of the truth-conditional information in different parts of the text, depending on their place in the structure of the text. So we see that there is a general problem in dynamic semantics: representing the interaction of information-in-context. We think that this point is so crucial to dynamic semantics that it deserves to be the main point on the agenda for years to come. The main contribution of a dynamic semantics for natural language can then be a systematic account of the way in which on different levels of meaning contextual information interacts with information content.[23] It is well known that this is important for the

[23]Note that here we use context and content as relative notions: we have seen that the context-content pairs of *DRT* together can serve as the content component in our

semantics of anaphora and presuppositions. We have shown that it is also of great use in the interpretation of text structure. And probably there are even more phenomena in the semantics of natural language which could benefit from such an approach. On the technical side this means that it would be a good idea to develop general techniques for the combination of contexts and contents. In recent years Visser ((Visser 1992a), (Visser 1992b), (Visser 1992c)) has made a start with the development of such a theory, but it is clear that a lot of work in this area still remains to be done.

Appendix

We use this appendix to present the proof of the basic associativity result (proposition 5.7) that is essential for the associativity of $\bullet$. The notation is as in section 5.

Proposition 4.6.1 (Basic Associativity) *Let two trees τ and τ' and a basic tree β be given. Then $(\tau \bullet \beta) \bullet \tau' = \tau \bullet (\beta \bullet \tau')$.*

Proof:
We can assume that $\tau = \tau(\!|\rho|\!)$ and $\tau' = (\!|\lambda|\!)\tau'$.
Now we distinguish two situations:

either:[24] $seg_f(\tau \bullet \beta) = seg_f(\rho \bullet \beta)$ and $seg_i(\beta \bullet \tau') = seg_i(\beta \bullet \lambda)$.

or

not: $seg_f(\tau \bullet \beta) = seg_f(\rho \bullet \beta)$ and $seg_i(\beta \bullet \tau') = seg_i(\beta \bullet \lambda)$.

First we discuss the situation where not $seg_f(\tau \bullet \beta) = seg_f(\rho \bullet \beta)$ and $seg_i(\beta \bullet \tau') = seg_i(\beta \bullet \lambda)$.
Assume first that $seg_f(\tau \bullet \beta) \neq seg_f(\rho \bullet \beta)$. This can only happen if $\tau \bullet \beta$ gives rise to a collapse. Then either

$$\rho = (\imath_0, (\imath_1), (\imath_2)) \text{ and } \beta = ((\top), \top)$$

or

$$\rho = (\imath_0, (\imath_1)) \text{ and } \beta = ((\top), (\top), \top).$$

tree semantics.

[24]This first case includes the case where one of ρ, λ is equal to $\mathbf{0}$.

We will discuss the first case. The second one is handled analogously. So $\tau \bullet \beta = \tau\{(\imath_0)\} \bullet (\imath_1 \to \imath_2)$.
Note that it is not possible that also $\beta \bullet \tau'$ gives rise to a collapse. So we know that $seg_i(\beta \bullet \tau') = seg_i(\beta \bullet \lambda)$. This means that $seg_i(\beta \bullet \tau') = ((\top), \top \wedge \imath_\lambda) = \beta \bullet (\imath_\lambda)$, where $\imath_\lambda$ is the leftmost node of λ.[25]
This gives us:

$$(\tau \bullet \beta) \bullet \tau' =$$

$$(\tau\{(\imath)\} \bullet (\imath' \to \imath'')) \bullet \tau'$$

and

$$\tau \bullet (\beta \bullet \tau') =$$

$$\tau \bullet \ [\![((\top), \imath_\lambda)]\!] \beta \bullet \tau' =$$

$$(\tau\{(\imath)\} \bullet (\imath' \to \imath'')) \bullet \ \{(\imath_\lambda)\}(\beta \bullet \tau').$$

Since $seg_i(\beta \bullet \tau') = \beta \bullet (\imath_\lambda)$, we see that $\{(\imath_\lambda)\}(\beta \bullet \tau') = \tau'$. So the result follows.
The case where $seg_i(\beta \bullet \tau') \neq seg_i(\beta \bullet \lambda)$ follows by symmetry.

In the second case $seg_f(\tau \bullet \beta) = seg_f(\rho \bullet \beta)$ and $seg_i(\beta \bullet \tau') = seg_i(\beta \bullet \lambda)$. Now it suffices to check that for all choices of ρ, β, λ we have

$$(\rho \bullet \beta) \bullet \lambda = \rho \bullet (\beta \bullet \lambda).$$

It is clear that this suffices, since $\bullet$ is specified entirely in terms of the final and initial segments. We have to check the following 64 combinations[26] of $\rho_i \bullet \beta_k \bullet \lambda_j$.

	$\rho = seg_f(\tau)$	β	$\lambda = seg_i(\tau')$
(1)	$(\imath_0)$	$(\imath)$	$(\imath'_0)$
(2)	$(\imath_0, (\imath_1))$	$(\top, (\top))$	$((\imath'_1), \imath'_0)$
(3)	$(\imath_0, (\imath_1), (\imath_2))$	$((\top), \top)$	$((\imath'_2), (\imath'_1), \imath'_0)$
(4)	$(\lambda, (\imath_0))$	$((\top), (\top))$	$((\imath'_0), \rho)$

[25]The terminology *leftmost node* should be clear: it is the point where the path segment λ starts. We could define this notion properly, but feel that this would only confuse matters.

[26]We skip the really trivial cases where $\rho_i, \lambda_j = \mathbf{0}$.

We distinguish cases according to the value of k. For each case we handle the easy combinations, i.e. the combinations where either the error state, $\mathbf{0}$, is reached or else three trees with the same 'direction' have to be merged. For also if ρ_i, β_k and λ_j are all down trees (or symmetrically all up trees), then associativity is obvious. For each case we will specify which are the remaining combinations.

$k = 1$: Note that now β does not change the form of the final or the initial segment. Therefore $\rho_i \bullet \beta_1 \bullet \lambda_j = \mathbf{0}$ iff $\rho_i \bullet \lambda_j = \mathbf{0}$. This is the case if: $i = j = 2$, $i = j = 3$, $i = j = 4$ or $\{i, j\} = \{3, 4\}$. Also if all three trees are down trees or all three trees are up trees, no problem can arise. This is the case if one of i, j is equal to 1.

There are four remaining cases: $\rho_i \bullet \beta_1 \bullet \lambda_j$ for $i = 2$ and $j \in \{3, 4\}$ or, symmetrically, $j = 2$ and $i \in \{3, 4\}$.

$k = 2$: Note that β_2 is a down tree. Hence the final segment of $\rho_i \bullet \beta_2$ will have the same shape as β_2. This implies that $\rho_i \bullet \beta_2 \bullet \lambda_j = \mathbf{0}$ iff $\beta_2 \bullet \lambda_j = \mathbf{0}$. This is the case precisely when $j = 2$. The other easy combinations are those where all trees are down trees. This is the case whenever $j = 1$.

The eight remaining combinations are those where $j \in \{3, 4\}$.

$k = 3$: By symmetry with the previous case we may conclude that the cases $\rho_i \bullet \beta_3 \bullet \lambda_j$ with $i \in \{3, 4\}$ remain.

$k = 4$: Note that β_4 has both a non-trivial initial and final segment. This means that it behaves both as a down tree (when merging with λ_j) and as an up tree (when merging with ρ_i). As a consequence there are no easy cases with just three up trees or just three down trees: we only have $\mathbf{0}$ cases as easy cases.

We see that $\rho_i \bullet \beta_4 \bullet \lambda_j = \mathbf{0}$ can be the case only if already $\rho_i \bullet \beta_4 = \mathbf{0}$ or $\beta_4 \bullet \lambda_j = \mathbf{0}$. This is the case whenever $i \in \{3, 4\}$ or $j \in \{3, 4\}$.

The four remaining cases are those in which $\{i, j\} \subseteq \{1, 2\}$.

We find that there are $4 + 8 + 8 + 4 = 24$ cases left to consider. By symmetry it suffices to check twelve of these (if these twelve are chosen carefully). For example, checking the following twelve cases suffices to finish the proof.

$\rho_2 \bullet \beta_1 \bullet \lambda_3$	$(\imath_0, (\imath_1))$	$(\imath)$	$((\imath'_2), (\imath'_1), \imath'_0)$
$\rho_2 \bullet \beta_1 \bullet \lambda_4$	$(\imath_0, (\imath_1))$	$(\imath)$	$((\imath'_0), \rho)$
$\rho_1 \bullet \beta_2 \bullet \lambda_3$	$(\imath_0)$	$(\top, (\top))$	$((\imath'_2), (\imath'_1), \imath'_0)$
$\rho_2 \bullet \beta_2 \bullet \lambda_3$	$(\imath_0, (\imath_1))$	$(\top, (\top))$	$((\imath'_2), (\imath'_1), \imath'_0)$
$\rho_3 \bullet \beta_2 \bullet \lambda_3$	$(\imath_0, (\imath_1), (\imath_2))$	$(\top, (\top))$	$((\imath'_2), (\imath'_1), \imath'_0)$
$\rho_4 \bullet \beta_2 \bullet \lambda_3$	$(\lambda, (\imath_0))$	$(\top, (\top))$	$((\imath'_2), (\imath'_1), \imath'_0)$
$\rho_1 \bullet \beta_2 \bullet \lambda_4$	$(\imath_0)$	$(\top, (\top))$	$((\imath'_0), \rho)$
$\rho_2 \bullet \beta_2 \bullet \lambda_4$	$(\imath_0, (\imath_1))$	$(\top, (\top))$	$((\imath'_0), \rho)$
$\rho_3 \bullet \beta_2 \bullet \lambda_4$	$(\imath_0, (\imath_1), (\imath_3))$	$(\top, (\top))$	$((\imath'_0), \rho)$
$\rho_4 \bullet \beta_2 \bullet \lambda_4$	$(\lambda, (\imath_0))$	$(\top, (\top))$	$((\imath'_0), \rho)$
$\rho_1 \bullet \beta_4 \bullet \lambda_1$	$(\imath_0)$	$((\top), (\top))$	$(\imath'_0)$
$\rho_1 \bullet \beta_4 \bullet \lambda_2$	$(\imath_0)$	$((\top), (\top))$	$((\imath'_1), \imath'_0)$

We leave it to the industrious reader to check these cases. (In fact the cases where either $\rho_i \bullet \beta_k$ or $\beta_k \bullet \lambda_j$ collapses have already been discussed above.)
This completes the proof of the proposition. □

We have to admit that the proof is a bit clumsy. But at least it is pretty straightforward as well: the main work is a lot of trivial case checking. By general observations we have been able to reduce the number of cases that actually have to be checked to twelve.
An alternative proof has been proposed by Albert Visser. It is possible to embed the partial trees in a term rewriting system, *Termtree* say, such that the term rewriting procedure actually computes the merger. The terms of *Termtree* would be sequences of basic terms among which we find our basic trees. A typical rewriting rule for *Termtree* would look something like:

$$(\imath_0, (\imath_1)) \bullet ((\imath_2), (\imath_3)) \rightsquigarrow (\imath_0, (\imath_1 \wedge \imath_2), (\imath_3)).$$

Now the proof of the associativity would follow from two observations about *Termtree*:

- The normal forms of *Termtree* are exactly the partial trees of $T_{\mathbf{I}}$;
- *Termtree* has strong normalisation.

Then we would know that different ways of rewriting the terms (or: computing the merger) would give the same result.
We have chosen not to present this proof in detail, although it is more elegant than the direct proof. One reason is that we would have to introduce a lot of notions for no other reason than to make the proof readable. Another reason is that the resulting proof is not really shorter: the term rewriting system has a lot of rules (it has to do the same thing as the definition of the merger which has 17 cases), which makes the normalisation proof tedious.

Chapter 5

Proofs as Texts

5.1 Introduction

In the previous chapters we have seen how the dynamic approach provides a new perspective on semantics. We have seen how several assumptions that are traditionally made in sentence semantics have to be adjusted in a satisfactory account of the semantics of texts. The crucial idea behind many of the adaptations that we discussed was the *small unit principle*: since texts are big, any reasonable model of the way we interpret texts should work with small units. Any formal model of text interpretation has to show that it is possible to work through a large text in small steps. As a result of this small unit principle we came to the conclusion that the crucial ingredient of a dynamic semantics is the way it accounts for the interaction between different parts of the text. Since dynamic semantics can only work with small units, the global relations between different parts of a text, will have to be coded up locally in a dynamic set up. We have seen that this applies to the semantics of variables but also to other levels of interpretation, such as the interpretation of text structure.
In this chapter we make a shift from semantics to proof theory: we try to develop a dynamic proof theory. This means first of all that we are looking for the proof theory for the formalisms that we use in dynamic semantics. For example, a dynamic proof theory in the end should tell us how we can build proofs for *DPL*. But we do not only want the proof theory to be dynamic in the sense that it works for dynamic semantics: we also want to find back some of the crucial ideas about dynamics in the way the proof theory works. In fact it is this second aspect of dynamic proof theory on

which we will concentrate: we will develop a proofs system that treats its *proofs as texts* and models proof building as text construction. Then the dynamic perspective on the way we deal with texts will automatically apply to our proofs system. For example, the small unit principle will tell us that we have to build up the proof texts bit by bit, just as the other texts.

Note that it is not hard to give a deduction system for *DPL* if we are prepared to drop our second aim in dynamic proof theory. If all we want is a way of deciding whether:

$$\phi \models_{DPL} \psi$$

then things are easy. We know that $\phi \models_{DPL} \psi$ iff $\models_{DPL} (\phi \rightarrow \psi)$.[1] Since we have a truth-preserving translation from *DPL*-formulas to formulas in standard predicate logic, we can simply translate this problem to predicate logic and answer the question there.[2] So it is clear that it is the second aim that makes dynamic proof theory interesting: it only makes sense to give a deduction system for *DPL* if this deduction system itself also has a dynamic flavour. But before we can start to do this we have to make up our mind as to what a deduction system with a dynamic flavour is. This is what we intend to do in this chapter: we will propose the *proofs as texts* perspective as the suitable form of dynamics in proof theory. Then the second step would be to actually develop a deduction system for *DPL* that has a dynamic flavour. Unfortunately we will not be able to make this second step in this thesis: this will have to wait until some other occasion.

So in this chapter we will restrict ourselves to developing the idea of a dynamic proof theory. We will first do this informally in section 5.2. Then we will make the informal ideas precise in an example: we will give a dynamic proofs system for a fragment of intuitionistic propositional logic.

[1]Here we ignore our earlier remarks in chapter 3 about partiality for a moment: we consider the notion of validity as defined in Groenendijk and Stokhof (1991a) (cf. section 1.5).

[2]Also see Van Eijck and De Vries (1992b), who give use a Hoare-calculus for *DPL* to do this in an 'on-line' manner.

5.2 Dynamic proof theory: proofs as texts

The origins of the proofs as texts perspective lie in our attempts to come to grips with the proof theory of the formalisms that are currently used in dynamic semantics. Dynamic semantics originated as an attempt to find an elegant and natural representation of phenomena of text coherence, such as anaphora. These dynamic investigations have led to a different look on the formal objects that are used in the semantics to represent the meanings of natural language expressions. The formulas used to be treated as if they were sentences, but in dynamic semantics it is more appropriate to treat formulas as if they were (small) texts. This view of *formulas as texts* has had several interesting consequences for the architecture of formal semantics. We will not go into the details of these developments, but we will try to give a feel for the issues that arise when we try to give a proof theory for such formalisms.
First we note that the kind of anaphoric links that we saw in our example can also occur between the assumptions and the conclusions of some argument in a text. For example in:

> Assume a man owns a house. Then he also owns a garden.

the man in the conclusion is definitely supposed to be the same man as the man in the assumption. It seems natural to require that a proofs system for dynamic semantics should take such possibilities into account. This means for example that one of the theorems of dynamic logic should be:

> If a man owns a house, then he owns a house.

Therefore, when we try to decide in our formalism whether $\phi \vdash \psi$, we should allow for anaphoric links between ϕ and ψ to occur *as if* ϕ *occurred before* ψ in some text. These anaphoric phenomena give rise to unexpected complications in the logic. For example, a logic that can handle anaphoric links should be able to explain what happens in the following example:[3]

> If a man owns a house, he owns a garden.
>
> If he owns a garden, he waters it.
>
> Therefore: if a man owns a house, he waters it.

[3]Due to Johan van Benthem.

It is quite clear, intuitively, what goes wrong here: the anaphoric link between *a house* and *it* that we find in the conclusion is not justified by the assumptions. The other link, between *a man* and *he*, is perfectly all right. It turns out to be rather difficult to find a formulation of the logic that explains these things in a natural and elegant way.[4]
These complications have led us to believe that a different design of the deduction system could be of great use in improving our understanding of the logic of these formalisms. For example, as was already pointed out, in $\phi \vdash \psi$ one should treat ϕ as if it occurred *before* ψ in a text. Therefore a presentation of the logic in which this order is reflected is to be preferred in a deduction system for dynamic logic. This shows that anaphoric links do not only suggest changes in the design of the *semantic* machinery, but also in the set up of the *deduction systems* for dynamic logic: at the very least these deduction systems should respect the order of the formulas in some way or other.
Here we will go one step further: we will not only consider the expression $\phi \vdash \psi$ *as if* ϕ occurred before ψ in some text, we will in fact treat the whole expression as (the representation of) a text. For example, we read the expression $\phi \vdash \psi$ as:

Let's assume that ϕ holds. Then we may conclude that ψ.

If we do this, the anaphoric links between ϕ and ψ will be represented correctly and later also other phenomena of text coherence that we may want to represent will automatically fall into place.
Note that it is not at all unreasonable to think of a proof as a text. For usually proofs are presented to us as texts. They are a special kind of text, of course: they are the kind of text that has been written in such a way that every step that is made in the text preserve validity. It is only natural to try and use these proof-texts in a deduction system for a formalism in which the formulas themselves are also (representations of) texts. Designing the proof objects as if they were these proof texts has the additional advantage that we obtain a representation of such proof texts. It is a well known fact (cf. Fine (1985)) that the way we like to present our proofs in real life—e.g. in math books— is quite different from the way proofs are built in the deduction systems we use for, say,

[4]Cf. Kamp and Reyle (1991), Vermeulen (1989), Saurer (1993), De Vrijer (1990).

predicate logic. But the official criterion for the correctness of a mathematical result still is the question whether it can be recast into a formal deduction in some (sound and complete) deduction system. Typically the way mathematicians like to present their proofs is very far from the way proofs work in formal deduction systems, but here the proofs as texts idea could help. If we succeed in designing a proofs as texts deduction system, this may help to bridge the annoying gap between the proof theory and proof practice.
These are the two main reasons for choosing a proofs as texts presentation of a dynamic logic: many proofs in real life are indeed presented to us as a special kind of text and by treating proofs as texts we will automatically be in a situation where we can discuss all phenomena of text coherence that we may want to cover in our logic. Note that by chosing to regard proofs as texts, we immediately make the methodological principles that we have discussed for texts in general applicable to proofs systems. So it is clear from the start that the proofs as texts idea is going to mean: we will have to present a picture of proof building that is in agreement with the small unit principle.
In this chapter we concentrate on developing this idea as such and we will apply it to a very simple example. We will not try to give deduction rules that can cope with anaphoric links: our language will be too poor to represent such links. Here we will present a prototype of the kind of deduction system that we will try to use later to get at the logic of anaphora. So we intend to develop a satisfactory account of the logic of texts in several steps: we first give a rudimentary proofs system and then we intend to refine this later to include (other) phenomena in the machinery that are especially important in texts. Anaphora is the most salient example of such a phenomenon. An aspect of texts that we will be concerned with from the start is the structural coherence of a text. We will see that the structural organisation of a text is the main tool in the formulation of a logic in proofs as texts style.

The choice for the proofs as texts perspective has interesting consequences. We are now in a situation where we have a formalism in which we represent texts, while at the same time the proofs for this formalism are also texts. So the proofs will form a special subset of the set of all formulas and the usual convenient distinction between the level of the formulas and the meta-level of the proofs over these formulas is no longer available. We

represent the proofs at the same level as the formulas.
This situation could easily lead to a proliferation of connectives in the representation language: besides the usual connectives that occur in a sentence we now also have to represent connectives between sentences, i.e. ways in which sentences can be combined into texts. But fortunately the two ways of combining sentences into texts that we are interested in here have a natural counterpart on sentence level with more or less the same meaning. Therefore we can use just one connective on both levels. An example of this situation is conjunction: this is represented within one sentence by expressions such as *and* and *but*. But also different sentences can be presented in such a way that it is clear that they should be interpreted in a conjunctive manner. In fact this seems to be the default option if two sentences occur after each other. So one symbol for conjunction is enough.
Another example of such a situation is provided by implication and valid inference. Within one sentence implications occur in the guise of if-then constructions, but there are several constructions that do the same thing between different sentences. Think of constructions such as

Assume that ... . Then it is clear that ... .

Also here we can use just one symbol for both situations: both if-then sentences and the entailment relation between different sentences will be represented with the same symbol. The technical justification for this move is the fact that we have the deduction theorem for our logic. If this were not the case, we might be forced to work in a language with two implications, one for ordinary implication and one for the entailment relation. Conjunction and implication are the only two connectives that we will use in this chapter.
So we see that we get a representation on the level of formulas of all sorts of operations that we used to think of as operations on the level of proofs. For example, we now have a connective to represent the notion of valid inference in our formulas, namely the connective that we also use for implication. But we also find that notions that we used to think of a sentence level notions now apply to proofs. For example, now that a proof is 'just another formula' it makes sense to talk about the meaning of a proof in the same sense as about the meaning of any other text. This is an interesting observation that we will not follow up here, but the reader

may wish to consult chapter 4, Vermeulen (1994b) for more details on the issue of the meaning of proof texts.
We already pointed out that in a proofs as texts deduction system the criteria that we discussed in chapter 4 apply to our proof objects. This gives the other constraints on the way the proofs system should work. In particular, texts are usually written incrementally or, to be more precise, usually we expect texts to be written in such a way that they can be read incrementally. In chapter 4 we have argued that this means that texts should allow for an incremental interpretation: the intuition that texts have this incremental quality seems too strong to ignore. Here we will also try to incorporate it in the design of our proofs system: the texts that are the proofs of our deduction system will be built up incrementally. Thus we get a picture in which proofs are texts that are built up step by step, taking care that validity is preserved along the way: theorem proving is modeled as text construction.
The rest of this chapter will be devoted to the development of an example of a proofs as texts deduction system. We will apply the ideas discussed above to the $\{\wedge, \rightarrow, \perp\}$-fragment of intuitionistic propositional logic. First we will define the language of this fragment in a particularly suitable way and we will discuss the structural notions that will play an important role throughout the chapter. Then we will go on and give a characterisation of derivability in terms of these structural notions. We will prove that our calculus does indeed give us precisely the theorems of intuitionistic logic. We will use the calculus to present an incremental proofs system for the logic. We have included an appendix with some familiar proof theoretic results about the fragment of intuitionistic propositional logic that we discuss.

5.3 The language of proof texts

In this section we define the language that we will use for the representation of proofs. As was announced above, we will restrict ourselves to a propositional language and we will identify sentence level and text level operations. We have chosen to keep the sentence level notation. Hence our formulas will contain $\rightarrow$ as a sign both for implication and for entailment. Similarly $\wedge$ will be used for conjunction within a sentence as well as for concatenation of sentences. We will ignore all other connectives.

The objects of our proofs system will be (special) formulas of our language. We want to formulate the rules of our proofs system as conditions on the construction of these formulas. The conditions will have to guarantee that the formulas we build are exactly the valid formulas of the language. These construction conditions will depend heavily on the structure of the formulas. The information about the structure of a formula tells us how the connectives in the formula are nested. In particular it will tell us how the implications in the formula are nested and thereby we will be able to tell which part of the text contains the assumptions that we can use at some point in the construction. We will use a representation of formulas as trees to characterise the structural information that we will need. Thus we will not only have a left to right order in the proof texts, but also a hierarchical order. These two ways to impose order on a text together will allow us to formulate proof rules as rules for building valid texts.
So the language that we will use is well known: it is simply the $\{\wedge, \rightarrow, \perp\}$-fragment of propositional logic. But instead of the usual inductive definition of this language, we define the formulas to be a special kind of *ordered labelled trees.* Of course there is an implicit tree structure in the formulas as we usually define them: their *construction* or *parsing* tree. Here we choose to make this structure explicit and identify the formulas with their parsing trees.
We make this move for simplicity only: since there is a one-one correspondence between formulas in the usual sense and the (parsing) trees that we will use as formulas, none of the results depend on this move. But as it is the structure of the parsing trees that we use for our characterisation of validity (in section 5.4.3), we can save some work by making the identification of formulas with their parsing trees from the start.

Definition 5.3.1

- *A tree is a pair $\langle N, \leq\rangle$, where N is a set, the set of nodes of the tree, and $\leq$ is an ordering on N such that:*
  - ▷ *there is a maximum element $r \in N$, called the root of the tree, i.e.*
    $\exists r \in N \forall n \in N : n \leq r$.
  - ▷ *for each $n \in N$ the set of nodes above n, $\uparrow(n) = \{m \in N : n \leq m\}$, is linearly ordered by $\leq$. So:*
    $\forall n \in N : \uparrow(n) \times \uparrow(n) \cap \leq$ *is a linear order on* $\uparrow(n)$.

- *Given a set LAB of labels we can define the notion of a tree with labels in LAB, or, for short, a labelled tree, as follows.*

  A labelled tree is a triple $\langle N, \leq, l\rangle$, such that $\langle N, \leq\rangle$ is a tree and l is a mapping $l : N \to LAB$.

- *An ordered labelled tree is a quadruple $\phi = \langle N, \leq, l, \{\prec_n : n \in N\}\ \rangle$, where $\langle N, \leq, l\rangle$ is a labelled tree and $\{\prec_n : n \in N\}$ is a family of relations such that each $\prec_n$ is a linear order on the daughters of n.*

Now we focus on a special set of ordered labelled trees: the formulas of our propositional language. In the definition we use $D(n)$ for $\{m \in N : m \leq n \ \&\ \neg\exists k : m \leq k \leq n\}$, the set of daughters of n.

Definition 5.3.2 *Let $ALPH = \{\wedge,\ \to,\ \bot,\ p_0,\ p_1,\ \ldots\}$ be the set of labels. Now we define $\mathcal{L}_p$ to be a subset of the ordered labelled trees with labels in $ALPH$:*
an ordered labelled tree $\langle N, \leq, l, \{\prec_n : n \in N\}\ \rangle \in \mathcal{L}_p$ iff the following conditions are satisfied:

- *If $D(n) = \emptyset$, then $l(n) \notin \{\wedge,\ \to\}$.*
- *If $D(n) \neq \emptyset$, then $D(n)$ contains exactly two elements and $l(n) \in \{\wedge, \to\}$.*

In the definition of $\mathcal{L}_p$ we have ensured that each tree in $\mathcal{L}_p$ is binary branching and that the labelling is such that the leaves of the tree are labelled with atomic propositions and all the internal nodes are labelled with a connective. This way we make sure that all the trees in $\mathcal{L}_p$ look like the parsing trees of the formulas in the usual sense.[5] So although we have chosen to define the language as a special set of labelled trees, it is clear that a formula in the usual sense can be obtained from a labelled tree by reading off the labels of the nodes.
Also the usual notions defined on formulas can easily be reformulated in tree terminology. When ϕ is given, the nodes of ϕ give us all the information about the subformulas of ϕ and their occurrences in ϕ. A formula ψ is a subformula of ϕ iff there is a node in ϕ such that ψ is (isomorphic to)

[5]Note that this way of defining construction trees works for every context free language. For more details on the correspondence of context free grammars and labelled trees we refer to Kracht (1993b).

the tree below this node. Each node that has this property determines a different occurrence of ψ in ϕ. The root of the tree corresponds to the only occurrence of ϕ as a subformula of ϕ.
In the rest of this chapter we will use the lower case Greek letters ϕ, ψ, χ, ... as variables ranging over nodes. Thereby, strictly speaking, they correspond to *occurrences* of subformulas. But we will allow ourselves to use the same variables for subformulas if we feel that no confusion can arise. Note that the tree ordering $\leq$ automatically gives us an ordering on the subformula occurrences of ϕ, the suboccurrence relation. We will also use $\leq$ for the subformula relation that can easily be obtained from the suboccurrence relation. In our notation we will use $=$ for the equality of subformulas and $\equiv$ for the equality of subformula occurrences, i.e. of nodes. $\stackrel{\text{def}}{=}$ is used in defining equations. Another important notation convention is that we will write down the formulas in the 'usual' way. Although officially formulas are trees we will use the traditional linear notation. So $(p \to q)$ stands for the tree with only three nodes: a root node and two daughter nodes, where the left daughter is labelled p, the right daughter is labelled q and the root has label $\to$. We will not spell out in detail how the traditional notation can be obtained for an arbitrary tree in $\mathcal{L}_p$: we trust that the reader can see how this is done.

Recall that we will not think of the formulas of this language as sentences, but more generally, as texts. A formula such as $(p \to q)$, for example, can stand for a text fragment such as:

> Let's assume p. Then it is safe to conclude q.

Note that in the texts that we use in real life to represent proofs all sorts of things are allowed that cannot be represented in our simple formal language. For example, in a proof text it can happen that the assumptions for some claim are given after the claim itself, as in:

> Then n^2 will be an even number if n itself is even.

We cannot represent such a situation in $\mathcal{L}_p$. Another thing that is quite common in proof texts, but that will not be considered here, is the use of intermediate conclusions. In a proof text we usually find constructions such as:

> Assume that n is even. Then n^2 is even as well. So $n^2 + 1$ is odd. Therefore $\frac{1}{2}(n^2 + 1)$ is not an integer.

Here the second and the third sentence are intermediate conclusions that lead up to the final conclusion that $\frac{1}{2}(n^2 + 1)$ is not an integer. Such intermediate conclusions do not exist in $\mathcal{L}_p$. In the implications of $\mathcal{L}_p$ we only have room for the assumptions and the final conclusion. If we want to represent the intermediate conclusions we have to use a trick: for example, we could form a conjunction of both final and intermediate conclusions.

We choose to work within these limitations because we want to keep our representation language $\mathcal{L}_p$ as simple as possible. Thus we will be able to concentrate on the main point of the *proofs as texts* perspective: everything is a text.

As was explained above, in this approach we will represent theorem proving as text construction. And while we are constructing a text we will have no other information at our disposal than what we find in the formula itself. In fact all that is left for us to use in formulating proof rules is the structure of the formulas. In the next sections we will see that this is really all we need: the interaction of the linear and the hierarchical structure of texts together gives all the information we need in our deduction system.

The deduction system that we will develop for intuitionistic propositional logic gives us an example of a deduction system in the *proofs as texts* style. The system is developed in three steps: first we will discuss some important notions concerning the structure of formulas. Then we will show how these structural notions allow us to formulate criteria for the validity of the formulas of $\mathcal{L}_p$ (section 5.4.3). We will prove soundness and completeness theorems for these criteria in section 5.5. Next we will actually define the deduction system in proofs as texts style (section 5.6): we will formulate rules for building formulas in such a way that validity is preserved. The rules will rely heavily on the criteria of section 5.4.3. Finally we will consider an alternative presentation of the deduction system that is reminiscent of the linear notation for natural deduction proofs.

5.4 The structure of formulas

In this section we define some notions concerning the structure of formulas. We will use these notions later to give a structural characterisation of the valid formulas. The main idea of this characterisation can be illustrated with a few examples.

- For a formula of the form $(\phi \to \psi)$ to be valid, the information that we find in ϕ has to justify the information in ψ. For example in $(p \to p)$, the first occurrence of p justifies the second occurrence of p, but in $(q \to p)$ there is no information — only q — to justify the occurrence of p.

- In $(\phi \to (\psi \to \chi))$, $(\psi \to \chi)$ as a whole has support ϕ. We saw above that in $(\psi \to \chi)$ ψ is support for χ. Hence both ϕ and ψ may serve as evidence to justify the occurrence of χ. For example, both in $(p \to (q \to p))$ and in $(q \to (p \to p))$, the second occurrence of p is justified by the first occurrence of p.

- In $((\phi \to \psi) \to \chi)$, $(\phi \to \psi)$ is the support for χ. This implies that we can use ψ to support χ, provided we have sufficient support for ϕ. For example in $((p \to (q \to r)) \to (q \to (p \to r)))$ the final occurrence of r has as support p, q and $(p \to (q \to r))$. From $(p \to (q \to r))$ we may conclude $(q \to r)$, since we have support for p. But then we may conclude r, since we have support for q. Hence there is sufficient support for r. We may conclude that the second occurrence of r in this formula is justified.

This kind of reasoning suggests that in a formula we can systematically locate the subformulas that may be used to justify the occurrence of other subformulas. In $(\phi \to \psi)$, ϕ is there to justify ψ. But this observation generalises to other formulas: we will develop a general notion of 'justification' along these lines that will allow us to recognise all valid formulas. In section 5.6 we will use this notion to give instructions for *building* valid formulas. Since we regard these formulas as texts, such a system of instructions is in fact a proofs system: it tells us how to build texts with preservation of validity.

5.4.1 Command relations on trees

Recall that the set of subformula occurrences of ϕ is ordered by the relation $\leq$. Each node in (the derivation tree of) ϕ represents an occurrence of a subformula of ϕ. Thus we can describe the structure of formulas in terms of structural notions on trees. Such notions are well known from the literature. The two kinds of relations on trees that will be crucial for us are familiar from the literature on natural language syntax: *command* relations and *precedence* relations. For a general discussion we refer to Kracht (1992), Kracht (1993b), here we will restrict ourselves to the relevant examples of such relations.
We will use these relations to characterise the *support relation* on subformulas: we will say that one occurrence of a subformula supports another occurrence of a(nother) subformula, if we can use the first occurrence to justify the second occurrence. In the definition of the support relation we use precedence relations and command relations. The precedence relations are used for the following reason: a subformula occurrence ψ of ϕ can only support a subformula occurrence χ if the two are separated by an $\rightarrow$ sign. ψ has to be in the antecedent—i.e. left subtree—of this implication and χ has to be on the conclusion side, the right subtree. This will be what the $\rightarrow$-precedence relation gives us. Note that here the linear structure of texts is important: a subformula can support another subformula only if it occurs to the left of that formula.
Let us call the set of nodes in ϕ that have as label $\rightarrow$, $Sub_{\rightarrow}(\phi)$. Then we can define the relation $Prec_{\phi}^{\rightarrow}$ as follows:

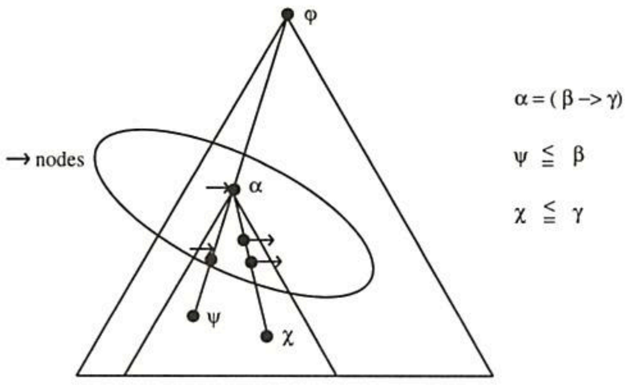
Definition 5.4.1
Let a formula ϕ be given and let ψ, $\chi \leq \phi$. Then $Prec_{\phi}^{\rightarrow}$ is the smallest solution of the following equation:[6]

$$\psi Prec_{\phi}^{\rightarrow} \chi \overset{\text{def}}{=}$$
$$\exists(\beta \rightarrow \gamma) \in Sub_{\rightarrow}(\phi) : \ \psi \leq \beta \ \& \ \chi \leq \gamma.$$

Here the notation $\exists(\beta \rightarrow \gamma) \in Sub_{\rightarrow}(\phi)$ is used as shorthand for $\exists\alpha \in Sub_{\rightarrow}(\phi) : \exists\beta, \gamma : \ D(\alpha) = \{\beta, \gamma\} \ \wedge \ \beta \prec_{\alpha} \gamma$. The following picture

[6]In the rest of the chapter we will also only be interested in the smallest solution of such equations, even if we don't mention this explicitly.

illustrates the situation.



The other structural relation that is important for the characterisation of the support relation is the command relation. We know that for ψ to support χ they have to be on alternate sides of an $\rightarrow$ and this is what the precedence relation gives. But this is not enough: for example in $((p \rightarrow q) \rightarrow p)$, p and p are on alternate sides of an implication sign, but clearly p does not provide support for p in this situation. p can only support the material that occurs after the *first* implication sign that follows p. This means in the tree terminology that when ψ supports χ, the first $\rightarrow$-node above ψ will be above χ as well. This is known as a *command* relation. The command relation keeps an eye on the nesting of implications in a formula: it codes the hierarchical structure of a text. We give the following definition.

Definition 5.4.2 *Fix $\phi \in \mathcal{L}_p$ and consider an arbitrary $\psi \leq \phi$.*

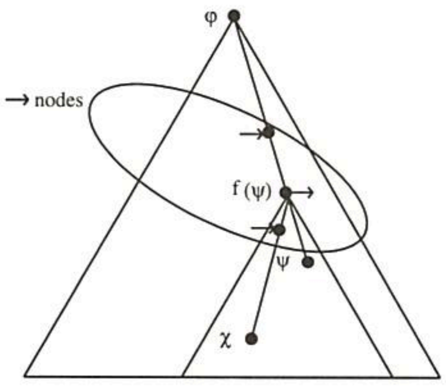
- *$f_{\rightarrow}(\psi)$ is the smallest subformula of ϕ that is larger than ψ and of which $\rightarrow$ is the main connective;*

  $$\begin{array}{l} f_{\rightarrow}(\psi) \equiv \chi \ \stackrel{\text{def}}{=} \\ \exists \chi_1, \chi_2\, (\chi \equiv (\chi_1 \rightarrow \chi_2)\ \&\ \ \psi \leq \chi \leq \phi\ \& \\ \quad \forall \chi', \chi_1', \chi_2'\, ((\chi' \equiv (\chi_1' \rightarrow \chi_2')\ \&\ \psi \leq \chi' \leq \phi)\ \Rightarrow \chi \leq \chi') \\ \vee \\ (\chi \equiv \phi\ \&\ \neg \exists \chi', \chi_1', \chi_2'\, (\chi' \equiv (\chi_1' \rightarrow \chi_2')\ \&\ \psi \leq \chi' \leq \phi)) \end{array}$$

∘ $\psi\ Com^{\rightarrow}_{\phi}\ \chi$, *we say* ψ *→-commands* χ *in* ϕ*;*

$$\psi\ Com^{\rightarrow}_{\phi}\ \chi \ \stackrel{\text{def}}{=} \chi \leq f_{\rightarrow}(\psi)$$

The following picture illustrates the situation: $f(\psi)$ is the first →-node above ψ that is also above χ. Note that it is not excluded that there are →-nodes in between χ and $f(\psi)$. We only demand that there are no →-nodes between ψ and $f(\psi)$.



We have argued that for ψ to support χ, ψ has to →-precede and →-command χ: the supporting subformula has to occur before the supported subformula and it should be nested into the implicational structure of the formula in the right way. Therefore we define the →-precede-and-command relation.

Definition 5.4.3 *Let* ϕ, ψ, $\chi \in \mathcal{L}_p$ *be given.* $\psi\ PC^{\rightarrow}_{\phi}\ \chi$

$$\psi\ PC^{\rightarrow}_{\phi}\ \chi \ \stackrel{\text{def}}{=} \ \psi\ Com^{\rightarrow}_{\phi}\ \chi\ \&\ \psi\ Prec^{\rightarrow}_{\phi}\ \chi.$$

So, for example, in $\phi = ((p \rightarrow q) \rightarrow p)$ we see a situation where $p\ Prec^{\rightarrow}_{\phi} p$, but not $p\ Com^{\rightarrow}_{\phi} p$. Therefore p does not precede and command p in this formula. On the other hand, in $(p \rightarrow (q \rightarrow p))$ we see that both $p\ Prec^{\rightarrow}_{\phi} p$ and $p\ Com^{\rightarrow}_{\phi} p$, so in this formula the first occurrence of p does precede and command the second occurrence of p. Note that the first occurrence is not preceded and commanded by the second occurrence in either case.

It is important to notice that in the definition of the command relation we do not find an existence condition: we take ϕ itself as a default value for $f_{\rightarrow}(\psi)$. But the precedence relation, as we defined it, demands that there is at least one subformula in $Sub_{\rightarrow}$ of which both ψ and χ are subformulas. Hence in $\psi \wedge \chi$, ψ will $\rightarrow$-command χ but it does not $\rightarrow$-precede χ. Hence ψ will not precede and command χ in this case. This way we can make sure that ψ does not count as support for χ in $\psi \wedge \chi$.
We have defined precedence and command relations with respect to the set of $\rightarrow$-nodes in ϕ. But the notions of precedence and command make sense for any set of nodes S in ϕ. Hence the definitions generalise to definitions of S-precedence, written $Prec^S_\phi$, and S-command, written Com^S_ϕ, if we replace $Sub_{\rightarrow}(\phi)$ by S in the definitions. In what follows we will treat $S = Sub_{\rightarrow}(\phi)$ as default. We will allow ourselves to omit the superscripts unless $S \neq Sub_{\rightarrow}(\phi)$.[7]

5.4.2 Support relations on subformulas

In this subsection we apply the structural notions $\rightarrow$-precedence and $\rightarrow$-command to define the support relation $\psi \sqsubset_\phi \chi$ on subformula occurrences of ϕ. We define $\sqsubset_\phi$ in such a way that $\psi \sqsubset_\phi \chi$ implies that the subformula χ really *needs* support. For example in $(p \rightarrow q)$ it is q, not p, that needs support. p is just an assumption an it is always allowed to make extra assumptions. Therefore we do not want to find $\psi \sqsubset p$ for any ψ. But for q the situation is different: if we intend to write down q, we have to be careful to preserve validity. q really needs support. We will call such subformula occurrences that need support *claims*. For each ϕ the set of claims of ϕ can be defined as follows.

Definition 5.4.4 *Let ϕ, $\psi \in \mathcal{L}_p$ be given. We define $Cl_\phi(\psi)$, to express that ψ is claimed in the proof text ϕ;*

$$Cl_\phi(\phi) \ \&$$

$$Cl_\phi((\psi_1 \wedge \psi_2)) \ \Rightarrow \ Cl_\phi(\psi_1) \ \& \ Cl_\phi(\psi_2) \ \&$$

$$Cl_\phi((\psi_1 \rightarrow \psi_2)) \ \Rightarrow \ Cl_\phi(\psi_2)$$

[7]Cf. Kracht (1992), Kracht (1993b) for more on command relations in general.

Examples are:

- ∘ $p \wedge q$: $Cl_{p\wedge q} = \{\, p,\ q,\ p \wedge q\}$
- ∘ $(p \rightarrow q)$: $Cl_{p\rightarrow q} = \{\, q,\ p \rightarrow q\}$

Now we define the support relation as follows.

Definition 5.4.5 *Let $\psi, \chi \leq \phi$ be given. Then*

$$\psi \sqsubset_\phi \chi \quad \overset{\text{def}}{=} \quad Cl_\phi(\chi) \ \ \& \ \ \psi\ PC_\phi \chi$$

So we say that a subformula occurrence supports a claim, precisely if it precedes and commands the claim. We can illustrate this with the following example: in the previous subsection we saw that in $(p \rightarrow (q \rightarrow p))$ the first occurrence of p precedes and commands the second occurrence of p. Since the second occurrence is in a claim position we may conclude that p supports p in this case. We can also see that the first occurrence of p precedes and commands q. But since q is not a claim of the formula p does not *support* q.
It is on this notion of support that our characterisation of valid formulas will be based. The basic idea is that a formula is valid iff all its claims can be justified. The notion of justification will be defined recursively: either a claim is justified directly because the formula contains an occurrence of the claim that supports it — as in $(p \rightarrow p)$ — or the claim is supported indirectly, as will be explained shortly.
The connection of this notion of support with our interest in texts that represent proofs is as follows: in reading a proof text we might at some point want to check a claim by the author. At such a point we have to know where we can find the standing assumptions: does the author still assume that ϕ holds or has this assumption already been cancelled? To check this we have to look at the place in the text where this assumption was made and see whether it was cancelled between that point and the point where we are now. This is exactly what our support relation does for us: given two places in the text, it tells us whether the assumption that we find in one place is still in force at the other place. Whenever we are reading a proof text we implicitly use the support relation to keep track of the standing assumptions.

5.4.3 Justifying claims

In this subsection we define a predicate $Just(\psi, \phi)$ that will express that the claim ψ of the text ϕ is justified. Our way of checking this was indicated at the beginning of this section: for a claim ψ we check whether there is sufficient support. This means that we search among the formulas that support the claim — as defined above — for material that actually justifies the occurrence of the claim. For example, in $(p \rightarrow p)$, the supporting material p clearly justifies the occurrence of the claim p. We will have several proof rules that determine which conclusions can be drawn from some set of supporting material.
In the examples that we have given we see that the justification of a claim consists of a trip through the formula searching for support. Thus a successful attempt at justification can be represented by the subformulas that we meet on such a trip. We could include a representation of these trips. That way we can have some kind of annotation of proofs in our system. Then we would have to define a three place predicate $Just(\xi, \psi, \phi)$, that expresses that ξ represents a justifying trip for the claim ψ that occurs in ϕ. Here we will not follow this direction. Right now we are content to ignore to possibility to add annotation, in order to keep things simple. We already made a similar decision when we chose our representation language. Thereby our language of proofs itself has become too poor to actually *contain* any kind of annotation. Adding the annotation now would be a rather artificial attempt to make up for this limitation. Therefore we will just stick to our strategy of keeping things simple: we define the *two*-place predicate $Just(\psi, \phi)$ to express that ψ is a justified claim of ϕ. If ϕ is fixed by the context we will write $Just_\phi(\psi)$ or even simply $Just(\psi)$, instead. We will adopt a similar convention for Cl_ϕ. Recall that the variables ϕ, ψ, χ, ξ ... range over nodes, i.e. subformula occurrences. If we use $=$ this means that we are only interested in the subformulas below the nodes.

Definition 5.4.6 *Let $\phi \in \mathcal{L}_p$.*

$Just(\psi, \phi) \quad \overset{\text{def}}{=}$

(direct proof)

$$\exists \xi : \ \xi \sqsubset_\phi \psi \ \ \& \ \ (\xi = \psi \ \ \vee \ \ \xi = \bot)$$

(conjunction of proofs)

$$\begin{array}{ll} \vee & \exists\psi_1\psi_2 : \ \psi \equiv (\psi_1 \wedge \psi_2) \ \ \& \\ & Just(\psi_1, \phi[\psi := \psi_1]) \ \ \& \\ & Just(\psi_2, \phi[\psi := \psi_2]) \end{array}$$

(proof of an implication)

$$\begin{array}{ll} \vee & \exists\psi_1\psi_2 : \ \psi \equiv (\psi_1 \rightarrow \psi_2) \ \ \& \\ & Just(\psi_2, \phi) \end{array}$$

(generalised modus ponens)

$$\begin{array}{ll} \vee & \exists\xi : (\ Cl_\xi(\psi) \ \ \& \\ & \quad Just(\bigwedge\{\chi : \chi \sqsubset_\xi \psi\}, \ \phi[\psi := \ \bigwedge\{\chi : \chi \sqsubset_\xi \psi\}]) \ \ \& \\ & \quad Just(\xi, \ \phi[\psi := \ \xi]) \ \ ) \\ & \vee \ (\ Cl_\xi(\bot) \ \ \& \\ & \quad Just(\bigwedge\{\chi : \chi \sqsubset_\xi \bot\}, \ \phi[\psi := \ \bigwedge\{\chi : \chi \sqsubset_\xi \bot\}]) \ \ \& \\ & \quad Just(\xi, \ \phi[\psi := \ \xi]) \ ) \end{array}$$

Here $\phi[\psi := \chi]$ stands for the result of replacing the occurrence ψ in ϕ by χ. We write $\bigwedge A$ for the conjunction of all the formulas in the set A. We can either rely on the fact that specific bracketings and orderings of conjunctions are irrelevant or else use some fixed bracketing and ordering strategy.

This is the structural characterisation of validity that we were after. We will prove this in the next section, by proving soundness and completeness. But first we discuss the definition in some more detail. The definition looks rather threatening, but the examples later on will surely help to get a feel for the system.
The definition of $Just$ distinguishes four cases. The first case is the case of direct justification. Here the claim that has to be justified is one of the subformulas that support it or else the claim is supported by $\bot$. Prototypical examples are

$Just(p, (p \rightarrow p))$ and

$Just(p, (\bot \rightarrow p))$ and

$Just(p, ((p \wedge q) \rightarrow p))$.

The second case says that we can justify a claim of the form $(\psi_1 \wedge \psi_2)$ by justifying both ψ_1 and ψ_2. An example of this is

$$Just((p \wedge q), (p \to (q \to (p \wedge q)))).$$

The third case treats claims of the form $(\phi \to \psi)$. In these cases it suffices to justify the claim of this claim, i.e. ψ. Note that case two and three are similar in that they both say that to justify a complex claim it suffices to justify the sub-claims of this complex claim. In case of a conjunction this means that we have to justify both conjuncts, in case of an implication we have to justify the consequent of the implication. Thereby one justification strategy that one could follow, is to look for justification of the atomic claims of a formula only. According to cases two and three this will suffice to justify all claims.[8] But this is not always the most efficient strategy. For example in

$$((p \to q) \to (p \to q))$$

it suffices to justify the second occurrence of q, the only atomic claim of this formula, and in fact this can be done, using generalised modus ponens. But it is easier to justify $(p \to q)$ as a whole, using the direct proof rule.
Case four can be seen as a generalised version of modus ponens. It can be applied in case the claim that has to be justified occurs as a claim of some other formula that would itself be a justified claim at this point. So if we have no direct evidence for a claim, ψ say, but we do have evidence for ξ, which has ψ as a claim, then we can try to apply generalised modus ponens (gmp) with this ξ. In this case we call ξ the *major premise* of the application of generalised modus ponens. For the rule to be applicable it must be the case that all the formulas that support ψ in ξ, $\{\chi : \chi \sqsubset_\xi \psi\}$, are now justifiable.
Note that we also have the case where not ψ, but $\perp$ occurs as a claim of ξ. So in general we are looking for a ξ that has a claim that gives a *direct proof* of our current claim.
A prototypical case of an application of gmp is

$$((p \wedge (p \to q)) \to q).$$

Here the second occurrence of q is a claim that has to be justified. There is no direct support for q, but one of its supporting formulas contains q

[8]More on this topic can be found in the appendix where we prove an *Inversion Lemma* that makes this statement precise.

as a claim, namely $(p \to q)$. Now the direct proof rule tells us that this supporting formula would be a justified claim at this point. According to the generalised modus ponens rule it suffices to justify all the subformulas of this alternative claim that support the claim. In our example this means that it suffices to justify p, which is the only support for q in $(p \to q)$. Since p is in fact one of the formulas that supports our claim q we are done.

The example shows that the rule has modus ponens as a special case. But the rule allows for more than that. Basically it generalises modus ponens in three ways. First note that in a nested implication the rule allows us to eliminate several assumptions in one sweep. To justify

$$(((p \wedge q) \wedge (p \to (q \to r))) \to r)$$

we only need one application of generalised modus ponens. In a natural deduction formulation of the logic we would have to apply modus ponens twice: once to eliminate p and once to eliminate q from $(p \to (q \to r))$. But here both assumptions are eliminated at once.

The second kind of generalisation that is built in is that there can be some 'implicit conjunction elimination' going on in generalised modus ponens: our version of modus ponens will also work if the required conclusion is only one of several claims of the major premise. For example in

$$((p \wedge (p \to (r \wedge q))) \to q)$$

q can be justified with one application of generalised modus ponens.

These first two generalisations of modus ponens are an advantage of our system: they allow us to ignore parts of the structural organisation of a text that are not relevant now. We only have to know which formulas support the current claim and not the precise way in which these formulas occur in the organisation of the text.

With the third generalisation that is built into the generalised modus ponens rule things are different. This third generalisation is the recursive nature of the rule. If we apply generalised modus ponens with some major premise ξ, we have to justify both ξ itself and all the formulas in ξ that support our claim ψ. Each of these justifications can in principle be a justification by generalised modus ponens. We are then in a situation where we have to apply generalised modus ponens with one major premise in order to be able to apply modus ponens on another major premise. The simplest example of such a situation is

$$(((p \to q) \wedge (q \to r)) \to (p \to r))$$

Here the justification of the claim r might proceed by modus ponens on $(q \to r)$. This major premise itself is easily justified (by direct proof), but we also have to justify q. In order to justify q we can then again apply modus ponens, with major premise $(p \to q)$ this time. In this second application the major premise itself is easily justified (by direct proof) and this time also the justification of the supporting material p is easy (also direct proof). So we see that in the application of generalised modus ponens on $(q \to r)$ another application of generalised modus ponens is invoked.

This third generalisation is not so attractive. It is because of this kind of implicit recursion that it is quite hard at times to recognise a text in our system as a proof. It would be nice if we could make some of this recursion explicit in the text, but as it stands we do not have this option: we have chosen to work with a poor representation language in which such information cannot be represented. So we will have to accept this kind of implicit recursion for now. But, clearly, we can improve on this at some point in the future by enriching the representation language.

Another way of viewing the situation is as follows: at this point we have a proofs system in which we can check whether validity is preserved throughout the text. This certainly is an important issue that any proofs system will have to take care of in some way. But if we think of a proof as a text, then we would like to be able to make a distinction between a proof and an arbitrary valid text. Certainly a proof is more than a text in which the validity of each step is guaranteed: it is a text in which the validity of each step is *obvious*. In a proof all the ideas have been elaborated on to the point where only very simple deduction steps remain. The validity of each proof step by itself should be obvious. Since we are using a simple representation language we have a system with proof texts of which the semantics is obvious, but which are not so obvious *qua* proofs because of the recursions that may be involved in applications of the gmp rule.

Probably the best choice in the end will be to include some of the annotation in the texts. For example, in an application of generalised modus ponens the major premise has the flavour of an intermediate conclusion. We already said that we may want to include intermediate conclusions in the proof texts at some point, so this is a good candidate for annotation that we may want to include in the texts. But for now we stick to the strategy of keeping all the annotation out of the text.

Now we are at a point where we have defined a system by which we can check whether a certain claim in a text is justified. In the following subsection we give some more examples of the way in which this system works. Our goal in the end is to *construct* texts which are themselves justified, i.e. we want to construct texts which have the following property:

Definition 5.4.7

$$Jf(\phi) \overset{\text{def}}{=} Just(\phi, \phi)$$

expressing that ϕ is a justified claim of ϕ, i.e. ϕ is a justified text.

This is what we discuss in section 5.6.

5.4.4 Examples

In this section we show how justifications of formulas can be obtained. We give two extensive examples of formulas and their justification. The reader who wants to know what happens to formulas that do not have a justification is referred to the appendix, where a decision procedure is discussed and it is shown that *Peirce's Law* cannot be justified in our system.

In these examples we use the inductive character of the definition: $Just$ is the smallest set that is closed under the inductive clauses. So we can show that some pair (ψ, ϕ) is justified by producing a sequence $Just(\psi_1, \phi_1) \ldots Just(\psi_n, \phi_n)$ where $\psi_n = \psi$, $\phi_n = \phi$ and each component of the sequence is either an instance of direct proof or follows from previous components in the sequence by an application of one of the inductive clauses. We will construct these sequences in the reverse order: starting from our goal $Just(\psi, \phi)$ we look for subgoals $Just(\psi_i, \phi_i)$ until we get instances of direct proof.

The first example is the law of contraposition: $\gamma = ((\phi \to \psi) \to ((\psi \to \bot) \to (\phi \to \bot)))$.[9] Our aim is to show that $Jf(\gamma)$. To do this we show

[9]Actually this is not a formula, but a formula scheme. As a consequence we will not find a justification, but a scheme for justification. Note also that in intuitionistic logic this formula is true but the implication cannot be reversed: we do not have $(((\psi \to \bot) \to (\phi \to \bot)) \to (\phi \to \psi))$.

that:

$$Just(\gamma, \gamma).$$

We start by justifying the smallest claim of γ, i.e. $\perp$. This can be done as follows.

We try to use generalised modus ponens. We need a ξ such that:

- $Cl_\xi(\perp)$
- $Just(\xi, \gamma[\perp := \xi])$
- $Just(\bigwedge\{\chi : \chi \sqsubset_\xi \perp\}, \gamma[\perp := \bigwedge\{\chi : \chi \sqsubset_\xi \perp\}])$

We take $\xi = (\psi \to \perp)$, so we now need:

- $Just((\psi \to \perp),\ ((\phi \to \psi) \to ((\psi \to \perp) \to (\phi \to (\psi \to \perp)))))$
- $Just(\psi,\ ((\phi \to \psi) \to ((\psi \to \perp) \to (\phi \to \psi))))$

We see that the first statement follows by direct proof. For the second statement we perform another application of generalised modus ponens. This time we will use $(\phi \to \psi)$ as major premise:

- $\xi_1 = (\phi \to \psi)$
- $Just(\xi_1, ((\phi \to \psi) \to ((\psi \to \perp) \to (\phi \to \xi_1))))$
- $Just(\phi, ((\phi \to \psi) \to ((\psi \to \perp) \to (\phi \to \phi))))$

Now we see that in both cases we can use *direct proof.*

Now we have justified the smallest claim of γ, $\perp$. It follows (by several applications of *proof of an implication*) that γ itself is justified.

The second example we present will be used as an axiom in the formulation of intuitionistic logic that we use in the next section. We consider the axiom $\alpha = ((\phi \to (\psi \to \chi)) \to (\phi \to \psi) \to (\phi \to \chi))$ and try to show $Jf(\alpha)$. So we have to show that:

$$Just(\alpha, \alpha)$$

Again we start by justifying the smallest claim first:

$$Just(\chi, ((\phi \to (\psi \to \chi)) \to (\phi \to \psi) \to (\phi \to \chi))).$$

This can be done as follows:

We apply generalised modus ponens:

- $\xi = (\phi \to (\psi \to \chi))$,
- $Just(\xi, ((\phi \to (\psi \to \chi)) \to (\phi \to \psi) \to (\phi \to \xi)))$,
- $Just(\phi \wedge \psi, ((\phi \to (\psi \to \chi)) \to (\phi \to \psi) \to (\phi \to (\phi \wedge \psi))))$.

These two justifications are obtained as follows:

- the first statement holds by direct proof.
- the second statement follows by conjunction of proofs, as follows:
  - ▷ $Just(\phi, ((\phi \to (\psi \to \chi)) \to (\phi \to \psi) \to (\phi \to \phi)))$ and
  - ▷ $Just(\psi, ((\phi \to (\psi \to \chi)) \to (\phi \to \psi) \to (\phi \to \psi)))$

again the first statement follows by direct proof.

for the second statement we need another application of generalised modus ponens, with $\xi_1 = (\phi \to \psi)$ as major premise. Now both

- $Just(\xi_1, ((\phi \to (\psi \to \chi)) \to (\phi \to \psi) \to (\phi \to \xi_1)))$ and
- $Just(\xi_1, ((\phi \to (\psi \to \chi)) \to (\phi \to \psi) \to (\phi \to \phi)))$

follow by direct proof.

From this justification of the claim χ in α we get a justification for α itself by several applications of the rule for the proof of an implication. This way we get:

$$Just(\alpha, \alpha)$$

as required.[10]

[10]Perhaps the reader will find it useful to check which routes through the tree/text are made in these examples of justifications.

5.4.5 Other logics

So far we have concentrated on intuitionistic logic, and this is also what we will do in the remaining part of the chapter. But the choice for intuitionistic logic over some other logic was completely arbitrary: the proofs as texts approach makes sense for any logic. So now that we have given the definitions for intuitionistic logic it may be helpful to think about using a similar system for other logics, just to make clear that our choice really was made for convenience only.

For example, if we want to obtain a system for classical propositional logic, we will have to include a rule for double negation one way or the other. The easiest way is probably to first add a direct proof rule that says that if $((\psi \to \bot) \to \bot) \sqsubset_\phi \psi$, then ψ is justified. Then we can extend the rule for generalised modus ponens: at present we search for ξ such that either $Cl_\xi(\psi)$ or $Cl_\xi(\bot)$ whenever we want to prove a claim ψ with generalised modus ponens. If we extend the rule to allow for ξ with $Cl_\xi(((\psi \to \bot) \to \bot))$, then we will obtain a system that is equivalent to classical logic.[11]

Thus we can obtain a stronger logic by strengthening the rules. Similarly a weaker logic could be obtained by weakening certain rules. For example, we could try to make the system resource sensitive by checking exactly which part of the supporting material of a claim is actually used in its justification. Then appropriate restrictions on the use and re-use of this material will have to be formulated. We will not go into details here. Suffice it to say that in the case of substructural logic the need for annotation of proofs becomes even more urgent, but once suitable annotation is added nothing seems to prevent us from formulating so-called substructural logics in proofs as texts style.

5.5 Soundness and completeness

In this section we prove that our system for justification of claims in formulas gives us exactly the (intuitionistic) tautologies. In particular we will show that:

[11]This can be shown with a straightforward extension of the soundness and completeness proofs in the following section.

$\vdash_{IL} \psi \;\Rightarrow\; Jf(\psi)$

and

$Jf(\phi) \;\Rightarrow\; \vdash_{IL} \phi.$

In other words, ψ is a theorem of intuitionistic logic if and only if there is a justification for ψ.
To prove this we first fix some formulation of IL. We will use a formulation of the calculus in Hilbert style. The details of the formulation are not essential, but it will help us fix our thoughts.

Definition 5.5.1 *We define the calculus IL as follows:*

- *IL is based on the following axiom schemata:*
  - ▷ $((\phi \wedge \psi) \rightarrow \phi)$
  - ▷ $((\phi \wedge \psi) \rightarrow \psi)$
  - ▷ $(\phi \rightarrow (\psi \rightarrow (\phi \wedge \psi)))$
  - ▷ $(\phi \rightarrow (\psi \rightarrow \phi))$
  - ▷ $((\phi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\phi \rightarrow \psi) \rightarrow (\phi \rightarrow \chi)))$
  - ▷ $(\bot \rightarrow \phi)$
- *The only rule of IL is modus ponens:*

  $$\vdash_{IL} (\phi \rightarrow \psi) \;\&\; \vdash_{IL} \phi \;\Rightarrow\; \vdash_{IL} \psi$$

- *ϕ is a theorem of IL, $\vdash_{IL} \phi$, iff ϕ is an instance of one of the axiom schemata or ϕ is derived from such instances by a finite number of applications of modus ponens.*

Since the calculus IL is sound and complete for the intended interpretation, we can confuse it with any other such calculus or with the (semantic) entailment relation of intuitionistic logic whenever this is convenient.

5.5.1 Soundness

Now we can prove the following proposition.

Proposition 5.5.2 (Soundness) *Let $\phi \in \mathcal{L}_p$ be given. Then:*

$$Jf(\phi) \;\; \Rightarrow \models_{IL} \phi$$

Since the Hilbert system presented above is complete for the intended interpretation, the proposition does in fact imply that what we can justify in our system can also be proved in the Hilbert system. In other words from the proposition we get:

Corollary 5.5.3

$$Jf(\phi) \;\; \Rightarrow \vdash_{IL} \phi$$

The proposition is a direct consequence of the following lemma.

Lemma 5.5.4 *Let ϕ, ψ, $\xi \in \mathcal{L}_p$ be given. Then*

$$Just(\psi, \phi) \;\; \Rightarrow \;\; \{\chi : \; \chi \sqsubset_\phi \psi\} \models_{IL} \psi$$

Proof:
Note that $Just(\psi, \phi)$ is defined inductively. Thereby there is for each ψ, ϕ such that $Just(\psi, \phi)$ a sequence

$$Just(\psi_1, \phi_1) \ldots Just(\psi_n, \phi_n)$$

such that $\psi_n = \psi$, $\phi_n = \phi$ and each component of the sequence is either an instance of direct proof or can be obtained by one of the other proof rules from components that precede it in the sequence.
The proof is by induction on the length of these sequences. In the proof we will omit the subscript in $\models_{IL}$.
In some steps we will make use of the following lemma:

Lemma 5.5.5
$\chi \sqsubset_\phi \psi$ *iff* $\chi \sqsubset_{\phi[\psi:=\alpha]} \alpha$

This lemma says that $\sqsubset$ only depends on the structure of the tree, not on the material that is stored there. The proof of the lemma is omitted.

Now the inductive proof proceeds as follows:

- Basic cases:
  - ▷ $\bot \sqsubset_\phi \psi$.
    Since $\bot \models \psi$, clearly also $\{\chi : \chi \sqsubset_\phi \psi\} \models \psi$.
  - ▷ $\psi \sqsubset_\phi \psi$.
    Since $\psi \models \psi$, clearly also $\{\chi : \chi \sqsubset_\phi \psi\} \models \psi$.
- Conjunction of proofs:

  Now $\psi = (\psi_1 \wedge \psi_2)$ and $Just(\psi_i, \phi[\psi := \psi_i])$.

  By the lemma we know that:

  $$\{\chi : \chi \sqsubset_\phi \psi\} = \{\chi : \chi \sqsubset_{\phi[\psi:=\psi_1]} \psi_1\} = \{\chi : \chi \sqsubset_{\phi[\psi:=\psi_2]} \psi_2\}.$$

  Now the induction hypothesis gives us (for $i = 1, 2$):

  $$\{\chi : \chi \sqsubset_{\phi[\psi:=\psi_i]} \psi_i\} \models \psi_i.$$

  Hence, by the lemma:

  $$\{\chi : \chi \sqsubset_\phi \psi\} \models \psi_i.$$

  From which we may conclude

  $$\{\chi : \chi \sqsubset_\phi \psi\} \models (\psi_1 \wedge \psi_2)$$

  as required.
- Proof of implication:

  We know that $Just(\psi_2, \phi)$.

  The key observation is that

  $$\{\chi : \chi \sqsubset_\phi \psi_2\} \text{ and } \{\psi_1\} \cup \{\chi : \chi \sqsubset_\phi \psi\}$$

  are logically equivalent. (This can be checked easily.)

  The induction hypothesis guarantees:

  $$\{\chi : \chi \sqsubset_\phi \psi_2\} \models \psi_2$$

  By the key observation this reads as

$$\{\chi :\ \chi \sqsubset_\phi \psi\}\ \cup\ \{\psi_1\}\ \models \psi_2$$

and thereby

$$\{\chi :\ \chi \sqsubset_\phi \psi\}\ \models (\psi_1 \to \psi_2)$$

as required.

- Generalised modus ponens:

  Now a major premise ξ is given and either $Cl_\xi(\psi)$ or $Cl_\xi(\bot)$. We consider the case $Cl_\xi(\psi)$: the other case is similar.

  $Just(\bigwedge\{\chi;\ \chi \sqsubset_\xi \psi\},\ \phi[\psi := \bigwedge\{\chi;\ \chi \sqsubset_\xi \psi\}])$ and $Just(\xi, \phi[\psi := \xi])$.

  If we apply the lemma to the induction hypothesis we get first:

  $$\{\chi :\ \chi \sqsubset_\phi \psi\} \models\ \bigwedge\{\chi :\ \chi \sqsubset_\xi \psi\}$$

  and second:

  $$\{\chi :\ \chi \sqsubset_\phi \psi\} \models \xi.$$

  Since $Cl_\xi(\psi)$, we know that ξ is of the form

  $$(\ldots \wedge (\alpha_1 \to (\ldots (\alpha_n \to (\ldots \wedge \psi \wedge \ldots)) \ldots)) \wedge \ldots),$$

  where each α_i is a χ such that $\chi \sqsubset_\xi \psi$.

  Our first conclusion from the induction hypothesis guarantees that

  $$\{\chi :\ \chi \sqsubset_\phi \psi\} \models \alpha_i.$$

  for $1 \le i \le n$.

  Now we can apply modus ponens n times to the second conclusion from the induction hypothesis to obtain

  $$\{\chi :\ \chi \sqsubset_\phi \psi\} \models (\ldots \wedge \psi \wedge \ldots)$$

  From this we obtain

  $$\{\chi :\ \chi \sqsubset_\phi \psi\} \models \psi$$

  by several eliminating the superfluous conjuncts, as required.

□

5.5.2 Completeness

Now we go on to prove the following proposition:

Proposition 5.5.6 (Completeness) *Let $\phi \in \mathcal{L}_p$ be given. Then:*

$$\models_{IL} \phi \ \Rightarrow \ Jf(\phi).$$

The proposition can be proved as follows:

Proof:
In fact we will show that $\vdash_{IL} \phi \ \Rightarrow \ Jf(\phi)$. Then the completeness will follow from the completeness of IL.
The proof is an induction on the length of the derivation of $\vdash_{IL} \phi$. In the proof we will omit the subscript of $\vdash_{IL}$.

- First we justify the central claims of the axiom schemata.
  - ▷ $Just(\phi,\ (\phi \wedge \psi) \rightarrow \phi)$;
  - ▷ $Just(\psi,\ (\phi \wedge \psi) \rightarrow \psi)$;
  - ▷ $Just(\phi \wedge \psi,\ \phi \rightarrow (\psi \rightarrow (\phi \wedge \psi)))$;
  - ▷ $Just(\phi,\ \phi \rightarrow (\psi \rightarrow \phi))$;
  - ▷ $Just(\chi,\ ((\phi \rightarrow (\psi \rightarrow \chi)) \rightarrow (\phi \rightarrow \psi) \rightarrow (\phi \rightarrow \chi)))$;
    This has been done in section 3.4.
  - ▷ $Just(\phi,\ (\bot \rightarrow \phi))$;

  Now the justifications for the axiom schemata themselves can be obtained by several applications of the 'proof of an implication' rule. For example, since $Just(\phi,\ (\bot \rightarrow \phi))$, we have $Just((\bot \rightarrow \phi),\ (\bot \rightarrow \phi))$, as required.

- Next we consider an application of modus ponens in IL.

  Suppose $\vdash\ (\phi \rightarrow \psi)$ and $\vdash\ \phi$.

  Then (by induction hypothesis):

  $Just((\phi \rightarrow \psi),\ (\phi \rightarrow \psi))$ and $Just(\phi,\ \phi)$.

  Now we can use generalised modus ponens to conclude that

  $Just(\psi,\ \psi)$, as required. □

We can conclude that the predicate $Jf(.)$ that we have defined in the previous section picks out exactly the theorems of intuitionistic logic.[12] This shows that our enterprise of regarding proofs as (special) formulas is not without a chance: we have shown that, if we regard our formulas as texts, we will actually be able to check for each text whether it is the kind of text that represents a proof. We are able to do this by using the structure of the text. The structure of the text tells us which assumptions are in force at each point in the text. Once we know this, things are easy: we just check whether the assumptions warrant the conclusion, either directly or indirectly via some applications of modus ponens.
The next step will be to use the criteria that we have developed to *check* whether some given text is valid as criteria for *building* texts that are valid. This is the topic of the next section.

5.6 Proof construction as text construction

In this section we will use the structural characterisation of validity to give rules for the (incremental) construction of valid formulas. Since the formulas are considered as representations of proof-like texts and the proofs are the valid proof-like texts, this will in fact give us a deduction system for our logic: we will haves rule for building up proof texts.
At the end of this section we will give a presentation of the deduction system that is reminiscent of linear presentations of natural deduction.

5.6.1 Proofs

As we are constructing valid formulas of $\mathcal{L}_p$, we will sometimes have to work with constructs that are not strictly speaking in $\mathcal{L}_p$. As we are building up the proof step by step we will go through stages in which what we have cannot be represented as a formula of $\mathcal{L}_p$. Consider for example a situation where we have just made a few assumptions and have not yet drawn any conclusions from these assumptions. Then we are in a

[12]In the appendix we prove some interesting properties of our system. The properties we discuss are familiar from (intuitionistic) proof theory. In the appendix we will see how they look in the new set up.

situation which cannot be described properly in $\mathcal{L}_p$: we are building up an implication, $(\phi \to \psi)$, and have just completed the antecedent ϕ of the implication, but we have not yet drawn any conclusion. So our construct looks something like $(\phi \to \ldots$. In chapter 4 we have developed special machinery for describing and interpreting precisely these incomplete texts. The machinery could very well be used in the present situation, but this would perhaps lead to confusion. Therefore we use the following trick to represent incomplete texts instead: we introduce a new propositional constant $\star$ to represent incompleteness. The resulting language will be called ${\mathcal{L}_p}^\star$. ${\mathcal{L}_p}^\star$ inherits all structural notions, such as the definitions of precede and command relations, from $\mathcal{L}_p$. In ${\mathcal{L}_p}^\star$ we can distinguish the complete texts from the incomplete text by testing whether $\star \leq \phi$.
Note that this way we do not only get unfinished — i.e. *right-incomplete* — formulas in ${\mathcal{L}_p}^\star$, but also formulas with holes in the middle or even at the start. We do not only get

$$(\phi \to (\psi \to \chi) \wedge \star),$$

but also

$$(\phi \to (\star \to \chi) \wedge \xi),$$
$$(\star \to (\psi \to \chi) \wedge \xi), \text{ etc.}$$

For the proofs system we will not need formulas with holes: we will build up the proof texts step by step from left to right, so only unfinished formulas can occur. We will call such formulas prooflike.
The following definition summarises this discussion:

Definition 5.6.1

- *${\mathcal{L}_p}^\star$ is defined as $\mathcal{L}_p$, but we replace $ALPH$ by $ALPH^\star = ALPH \cup \{\star\}$.*
- *We call $\phi \in {\mathcal{L}_p}^\star$* **incomplete** *iff $\star \leq \phi$. Else we call ϕ* **complete**.
- *We call $\phi \in {\mathcal{L}_p}^\star$* **prooflike** *iff for all $\psi \leq \phi$:*

$$\star \; Prec^{\to}_{\phi} \; \psi \;\; \Rightarrow \;\; l(\psi) \notin ALPH \text{ and}$$
$$\star \; Prec^{\wedge}_{\phi} \; \psi \;\; \Rightarrow \;\; l(\psi) \notin ALPH.$$

We see that prooflike formulas are such that wherever a $\star$ occurs, everything to the right of this is also a $\star$. This way no holes can occur in a prooflike formula.

In our proofs system we will be building up these prooflike formulas. Here the act of building up will be represented by substituting material for occurrences of $\star$. By substituting only for the leftmost occurrence of $\star$ we can make sure that the substitution of prooflike formulas in prooflike formulas results in a prooflike formula. Note that intuitively the leftmost $\star$ indicates the point where we are in the construction process.
There are two kinds of construction steps to which two kinds of substitution will correspond: first there will be the substitution of a complete formula for a $\star$, which actually makes the construct 'more complete'. But we will also need to substitute of more $\star$s for a $\star$ sometimes. These substitutions will represent decisions about the structure of the proof. For example, each proof will start from scratch: we start with just one $\star$. Then we may decide that we want to make some assumptions. This will lead to substituting $(\star \to \star)$ for $\star$ to get $(\star \to \star)$. Now the first $\star$ indicates the point where we actually are and we see that this is indeed a point where assumptions can be made. If we now decide that in fact the assumption we want to make is a conjunction, we will replace the first $\star$ by $(\star \wedge \star)$, after which we can start filling in the first conjunct of the assumption. Filling in the first conjunct of our assumption, p say, will then be represented by $\star := p$, a substitution of the first kind which results in the more complete formula: $((p \wedge \star) \to \star)$.
Now defining the proofs system will simply amount to restricting the substitutions of the first kind in case the leftmost $\star$ is in a claim position in the prooflike formula. Of course we do not want any restrictions on the substitutions if we are not in a claim position, because we want to be able to make any kind of assumption. It is only the conclusions, which will be in claim positions, that we have to be careful with. There we will want to check whether we can justify the claim, before we actually make it. And for this purpose we can simply use the techniques that were developed in the previous section.
The following definition makes this description of the proofs system precise. We will define 'being-a-proof' as a property of formulas in $\mathcal{L}_p{}^\star$, but the recursive definition of this property can be read as a set of instructions for building proofs.

In what follows we will use the notation $\langle\phi := \psi\rangle$ for a substitution of ψ for the leftmost occurrence of ϕ. If ϕ does not occur, the operation $\langle\phi := \psi\rangle$ has no effect. We will say *claim*(ϕ) if the leftmost $\star$ in ϕ is a claim of ϕ.

Definition 5.6.2 *We define Proof, the set of proofs, as the smallest set such that:*

1	*(start)*	$\star \in \mathit{Proof}$
2	*(construct $\rightarrow$)*	$\phi \in \mathit{Proof}$
		$\Rightarrow\ \phi\langle\star := (\star \rightarrow \star)\rangle \in \mathit{Proof}$
3	*(construct $\wedge$)*	$\phi \in \mathit{Proof}$
		$\Rightarrow\ \phi\langle\star := (\star \wedge \star)\rangle \in \mathit{Proof}$
4	*(assume)*	$\phi \in \mathit{Proof}\ \&\ \neg \mathit{claim}(\phi)$
		$\Rightarrow\ \phi\langle\star := \psi\rangle \in \mathit{Proof}$
		for any formula $\psi \in \mathcal{L}_p$
5	*(direct proof)*	$\phi \in \mathit{Proof}\ \&\ \mathit{claim}(\phi)$
		$\Rightarrow\ \phi\langle\star := \psi\rangle \in \mathit{Proof}$
		if $\psi \sqsubset_\phi \star$ *or* $\bot \sqsubset_\phi \star$
6	*(modus ponens)*	$\phi \in \mathit{Proof}\ \&\ \mathit{claim}(\phi)$
		$\Rightarrow\ \phi\langle\star := \psi\rangle \in \mathit{Proof}$
		if there is a ξ *with* $\phi\langle\star := \xi\rangle \in \mathit{Proof}$
		and either:
		$\mathit{Cl}_\xi(\psi)$ *and* $\phi\langle\star := \bigwedge\{\chi :\ \chi \sqsubset_\xi \psi\ \}\rangle \in \mathit{Proof}$
		or:
		$\mathit{Cl}_\xi(\bot)$ *and* $\phi\langle\star := \bigwedge\{\chi :\ \chi \sqsubset_\xi \bot\ \}\rangle \in \mathit{Proof}$

We see that we have 6 rules for building proofs. The first rule gets us started. Each construction starts with an application of that rule. It represents the stage where we have an empty page before us and decide that we want to build a proof.[13] The second and third rule are construction rules. They allow us to construct implications and conjunctions in any part of the proof. Rule 4 tells us how to make assumptions in a proof: we can fill in any assumption ψ at a place that is not a claim of the formula.

[13]Usually in such a situation we also have a theorem in mind that we want to prove, but we cannot write this theorem down in our system. We always write the conclusions *after* the assumptions, so in our representation we will not see the theorem that we want to prove until the proof is completed.

The rules 5 and 6 are the real proof rules: they tell us which conclusions we may draw. By convention an occurrence of a claim $\star$ is always justified. So, given the discussion in the previous section, we simply could have replaced 5 and 6 by:

$$\begin{array}{llll} \textit{(conclude)} & \phi \in \textit{Proof} \ \& \ \textit{claim}(\phi) & \Rightarrow & \phi\langle \star := \psi \rangle \in \textit{Proof} \\ & \textit{and } \textit{Just}(\psi\ , \phi\langle \star := \psi \rangle). & & \end{array}$$

It is easy to check that this would give an equivalent proofs system. We have chosen for our formulation, because it makes the proof steps more explicit.

Note that in steps 4, 5 and 6 we allow ourselves to substitute complex formulas for some $\star$. This is in contradiction with the small unit principle: according to the small unit principle it should only be allowed to draw atomic conclusions and make atomic assumptions. But fortunately the system that we get when we restrict 4, 5 and 6 to atomic ψ gives us the same proof strength: if we only allow the substitution of atomic material in 4, 5 and 6, then the other rules will enable us to build up complex assumptions and conclusions step by step. Here the Inversion Lemma (proved in the appendix) is essential. It guarantees that all formulas that are justified can be justified via their *atomic* claims.

Of course our claim is that this system is in fact 'the same' as the system in the previous section. If we try to make this precise then we see that there are two ways of making a comparison between the system of the previous section and this deduction system. One is by simply restricting the attention to the complete formulas in $\mathcal{L}_p{}^\star$. Then we can compare:

$$\textit{Jf}(\phi) \text{ and } \textit{Proof}(\phi)$$

for $\phi \in \mathcal{L}_p \cap \mathcal{L}_p{}^\star$.
But we can also extend the comparison to incomplete formulas. Then the comparison works by defining a completion operation on formulas of $\mathcal{L}_p{}^\star$. For example, let for each $\phi \in \mathcal{L}_p{}^\star$ the completion of ϕ, ϕ^c be defined by:

$$\phi^c = \phi\langle \star := \top \rangle^n.$$

Here $\top$ is some fixed tautology and n is the number of $\star$s in ϕ. So $\phi^c \in \mathcal{L}_p$. Then we can go on and compare:

$$Jf(\phi^c) \text{ and } Proof(\phi^c)$$

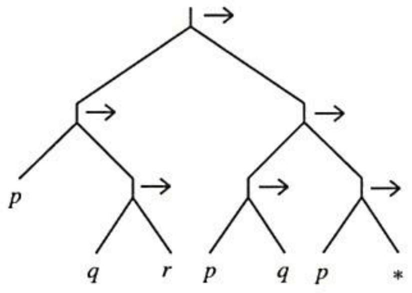
for $\phi \in \mathcal{L}_p{}^\star$. Both approaches are essentially the same and in both cases the similarity between the definition of Jf and the definition of *Proof* makes it immediately clear that these predicates coincide. Therefore the soundness and completeness of this system is simply inherited from the soundness and completeness of $Jf(\phi)$.

5.6.2 Natural deduction

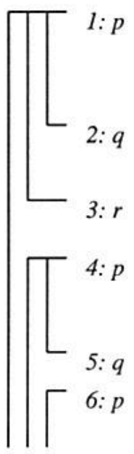
In this subsection we show how the building up of prooflike formulas can be represented in a linear style natural deduction system (cf. GAMUT (1991)). In such a system a proof looks like a column of formulas with vertical lines to its left—they indicate the scope of the assumptions—and annotation is added to the right of the column to indicate which rules have been applied.

Such a presentation for our proofs system can be obtained by performing several geometric transformations to the prooflike trees. The first step is to turn a tree in such a way that left to right order turns into top to bottom order. Now the atoms at the leafs of the tree appear in a (vertical) column and the atoms that used to occur to the left of some atom now can be found above the atom in the column. The second step is to ignore the conjunction structure: the conjuncts are simply listed on consecutive rows of the column. The third step is to replace the triangular shapes of the branches of the tree by squared braces, just as in linear natural deduction systems. Then finally we delete the part of the tree where only $\star$s occur.

We give an example to illustrate the idea. First consider the proof $((p \to (q \to r)) \to ((p \to q) \to (p \to \star)))$. This is part of the proof of one of the axiom schemata from our Hilbert system for IL.

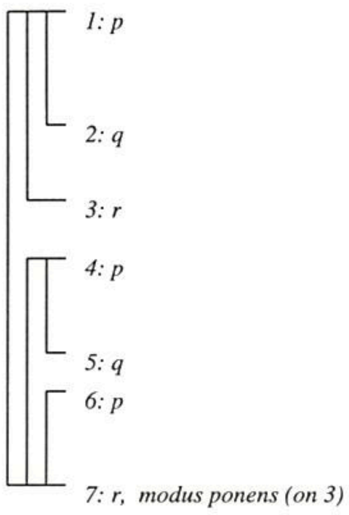


When we turn this around in the appropriate way we obtain:



Let's agree that *assume* is the default rule. Then we only have to write comments to the right of *claims* to indicate which rule is used to derive it. Here there are no claims yet: all atoms have been obtained by the *assume*-rule, that allows us to add material in positions that are not claims. (In the picture we are not explicit about the *construct* rules that have been involved in the proof.)

We could now complete the proof of the axiom by applying modus ponens. Annotation is added to indicate which formulas are involved in the application of the modus ponens rule.



Of course the proof rules on the trees can be converted into proof rules on these natural deduction-like objects. For example, the basic structural notion of one formula preceding another can be replaced by the notion of one formula being above another. Thus we can obtain a more familiar format for the proofs-as-texts system. Of course, from the *proofs as texts* point of view, the horizontal, tree-like representation is preferable for those who tend to write their texts from left to right.

5.7 Conclusion

We can conclude that it is possible to give content to the slogan *proofs as texts*. Indeed in this chapter we already have given an indication of what the slogan can lead to.

Our starting point has been the assumption that the formulas of our language represent texts. Since proofs can also be represented as texts, we concluded that in our situation both proofs and sentences should be represented in the same language. Once the distinction between an object language (for representing sentences) and a meta formalism (for representing proofs) is removed, we find that the only thing left to rely on is the structure of the formulas. In fact we are able to use the structure of the

formula to formulate rules for building texts in such a way that validity is preserved, i.e. for building proofs. This is one of the points of the proofs as texts perspective: we model *theorem proving as text construction.*
So the main consequence of the *proofs as texts* perspective that we have been able to illustrate is how the information about the structure of a text comes to rescue once the distinction between an object level and a meta level is lost. We have also made an attempt to implement the fact that texts are built up incrementally. The way in which this has been done here — by introducing a new propositional constant $\star$ — is not very elegant. We have proposed another strategy to deal with incomplete formulas in chapter 4, but we have refrained from implementing this strategy here, since this would take up too much space.
Other properties that are typical of texts have not yet been implemented. In particular we have postponed the discussion of the use of anaphora in proofs by restricting ourselves to a propositional language. So the obvious next step in our programme is the extension of the approach to predicate logic. Here we could either try to give a system for (static) predicate logic, or we could aim to give a proof theory for *Dynamic Predicate Logic*, a variant of predicate logic developed especially to deal with the semantics of anaphora. In both cases we will have extra labels for the quantifiers. The idea that we would like to pursue is that nodes that carry these labels systematically regulate the use of the material that we find at the nodes in their environment. For example, if a node is labelled with $\exists x$, then it will not allow us to use all information with a variable x freely. The nodes labelled with $\exists x$ tell us which nodes that have information about x can be used at some point in the text. This will depend on the relative positions of the $\exists x$ node and the node with information information about x. The difference between ordinary predicate logic and dynamic predicate logic should then turn out to be a difference in the way in which the quantifier nodes regulate the use of information at other nodes. We realise that this gives only a very rough indication of how the extension of the system to predicate logic should proceed, but it will have to do for now. We hope to discuss it in more detail elsewhere.
Other points that have not been dealt with here are the treatment of other connectives, such as disjunction, the proper treatment of intermediate conclusions, situations where conclusions precede assumptions, etc. For some of these issues it will be easy to find a treatment in our system, others will require more work. For example, if we include new connectives in our

language, then this can seriously complicate the structure of the formulas. As a consequence the definition of the support relation will become more complicated.
But as it stands there is no reason to suspect that any of the topics mentioned will lead to problems that cannot be solved within the *proofs as texts* perspective.

Appendix: a little proof theory

In this appendix we give some properties of the system that was defined in section 5.4.3. The properties will discuss have natural analogues in intuitionistic proof theory and will lead up to a decidability result:[14] we will be able to give a decision procedure to find out whether $Jf(\phi)$ for any $\phi \in \mathcal{L}_p$.
At this point the development of the proof theory of our system is in a stage where we reproduce familiar results about intuitionistic logic in our setting. But we hope that in time our new approach to the proof theory of IL will lead to new theorems about IL or will lead to considerable improvements of the proofs of some familiar theorems.

Inversion Lemma

In this section we show that two of our proof rules, conjunction of proofs and proof of an implication, can be inverted. As it stands these rules allow us to build up justifications of complex claims from justifications of smaller claims. Here we show the converse: if we have a justification for some claim, then we can construct from it a justification for all its subclaims.

Lemma 5.7.1 (Inversion Lemma) *Let $\psi_i \leq (\psi_1 \circ \psi_2) \leq \phi$ be given ($i = 1, 2$, $\circ = \wedge, \rightarrow$). Then:*

- *$Just((\psi_1 \wedge \psi_2), \phi)$ iff $Jble(\psi_i, \phi)$ $(i = 1, 2)$*
- *$Just((\psi_1 \rightarrow \psi_2), \phi)$ iff $Jble(\psi_2, \phi)$*

[14] Cf. for example, Troelstra and Van Dalen (1988) for discussion of such results.

Proof:
$\Leftarrow$: This is given by the rules conjunction of proofs and proof of an implication.
$\Rightarrow$: Assume that $Just((\psi_1 \wedge \psi_2), \phi)$ has been established. We distinguish the following cases according to the last rule that has been applied in this justification:

- (direct proof) Either $(\psi_1 \wedge \psi_2) \sqsubset_\phi (\psi_1 \wedge \psi_2)$ or $\perp \sqsubset_\phi (\psi_1 \wedge \psi_2)$. In the first case we see that $\psi_i \sqsubset_\phi \psi_i$, so we have a direct proof of each ψ_i. In the second case we see that $\perp \sqsubset_\phi \psi_i$, so again we have a direct proof of ψ_i.

- (conjunction of proofs) Then we knew before the last step in the justification that: $Just(\psi_i, \phi)$. So we are done.

- (proof of an implication) Does not apply.

- (generalised modus ponens) Let ξ be the major premise. There are two cases: $Cl_\xi(\psi_1 \wedge \psi_2)$ or $Cl_\xi(\perp)$. We see that ξ also allows for the conclusion of each of the ψ_i (since either $Cl_\xi(\psi_i)$ or $Cl_\xi(\perp)$), so the same application of generalised modus ponens also justifies the ψ_i.

Next assume $Just((\psi_1 \rightarrow \psi_2), \phi)$. We distinguish the following cases:

- (direct proof) Either $(\psi_1 \rightarrow \psi_2) \sqsubset_\psi (\psi_1 \rightarrow \psi_2)$ or $\perp \sqsubset_\psi (\psi_1 \rightarrow \psi_2)$. In the first case we apply generalised modus ponens with major premise $(\psi_1 \rightarrow \psi_2)$. We know (via the hypothesis) that this major premise is justified and we see that ψ_1 is justified by direct proof. So this gives a justification of ψ_2.

  In the second case we see that $\perp \sqsubset_\psi \psi_2$, so in this case we have a direct proof of ψ_i.

- (conjunction of proofs) Does not apply.

- (proof of an implication) Now we know that $Just(\psi_2, \phi)$. So we are done.

- (generalised modus ponens) As above. □

The Inversion Lemma is interesting by its own right, because it shows that there is a kind of symmetry in our system. But the main reason why we are interested in the Inversion Lemma is that it gives rise to the following corollary.

Corollary 5.7.2 *A formula can be justified iff all its atomic claims can be justified.*

This is the first step on our way to a decision procedure: we now know that to decide whether a claim can be justified it suffices to check whether all its atomic subclaims can be justified.

Subformula Property

In this subsection we prove the *Subformula Property* for our system. The subformula property guarantees that a justification of some claim in a formula can be built up from justifications of other parts of the same formula. So we do not have to leave a formula/text to justify it.
For most proof rules this is clear: the rule for direct proof does not require other justifications at all and the rules for conjunction of proofs and proof of an implication typically rely on justifications of subclaims of the current claim. The crucial rule is generalised modus ponens: here we use justifications for the major premise and for some subformulas of the major premise, but there is no restriction on which formulas we can use as major premise. So for the proof of the subformula property will have to show that applications of generalised modus ponens where the major premise is a subformula suffice to justify all justifiable claims.
In fact we will prove something slightly stronger: we will show that the ξ_3 that we use in generalised modus ponens can always be chosen from the formulas that support the current claim. Justifications in which only such applications of generalised modus ponens occur will be called *non-alien*. Other justifications are called *alien*.
In the proof we will frequently use a slightly stronger version of the Inversion Lemma: we will use that a non-alien justification of a claim exists iff non-alien justifications of its subclaims exist. This follows immediately from the proof of the Inversion Lemma as we have given it. So we state without further proof:

Lemma 5.7.3 (Strong Inversion Lemma)

- *There exists a non-alien justification of* $Just((\psi_1 \wedge \psi_2), \phi)$ *iff there exist non-alien justifications of* $Just(\psi_i, \phi)$ $(i = 1, 2)$
- *There exists a non-alien justification of* $Just((\psi_1 \rightarrow \psi_2), \phi)$ *iff there exist non-alien justifications of* $Just(\psi_i, \phi)$ $(i = 1, 2)$

Proposition 5.7.4 (Subformula Property) *If* $\mathit{Jble}(\psi, \phi)$, *then there exists a non-alien justification of* ψ *in* ϕ.

Proof:
We prove the statement by showing that the deepest alien application of gmp (generalised modus ponens) can be eliminated. Since there can only be finitely many alien applications in a justification, this shows that all alien justifications can be eliminated.
So we are in a situation where $Just(\psi, \phi)$ has been established by alien generalised modus ponens, but no other, deeper alien application of modus ponens is required. By the Strong Inversion Lemma we may assume $\psi = p$. Then we have a major premise ξ such that:

- $Cl_\xi(p)$ (or $Cl_\xi(\bot)$),
- $Just(\xi, \phi[p := \xi])$,
- $Just(\bigwedge\{\chi : \ \chi \sqsubset_\xi p\}, \phi[p := \bigwedge\{\chi; \ \chi \sqsubset_\xi p\}])$
  (or $Just(\xi_1, \bigwedge\{\chi : \ \chi \sqsubset_\xi \bot\}, \phi[p := \bigwedge\{\chi : \ \chi \sqsubset_\xi \bot\}]))$.

Here the last two statements can be verified without alien generalised modus ponens.
In what follows we will ignore the case $Cl_\xi(\bot)$, but it is easy to extend the proof to include this case.
Since $Cl_\xi(p)$, also $Cl_{\phi[p:=\xi]}(p)$. The strong inversion lemma (applied to $Just(\xi, \phi[p := \xi])$) gives a justification of this claim: $Just(p, \phi[p := \xi])$.
Now we can consider this justification of p: how does it work?

1. It was obtained by direct proof. So $p \sqsubset_{\phi[p:=\xi]} p$. We distinguish two cases: the supporting occurrence of p occurs in ϕ or the supporting

occurrence occurs in ξ. In the first case we find an alternative justification for p in ϕ (namely by direct proof) that does not uses alien rule applications.

In the second case we also find such an alternative. For then $p \in \{\chi : \ \chi \sqsubset_\xi p\}$ and by assumption there is a justification of p in ϕ that does not use alien rule applications.

2. Conjunction of proofs or proof of an implication do not apply, since p is atomic.

3. We used non-alien generalised modus ponens.

We see that we can eliminate the alien gmp unless it is followed by a non-alien application of gmp. So it suffices to show that such non-alien gmps can be eliminated. We will do this by replacing the two gmps by one big gmp. We see to it that this big gmp will again be the deepest alien gmp in the resulting justification. This will prove that case 3 can be avoided. So only case 1 remains and we can eliminate the alien gmp. Since we are past the deepest alien gmp, the next application of generalised modus ponens must be non-alien. So the major premise ζ is such that:

- $Cl_\zeta(p)$
- $\zeta \sqsubset_{\phi[p:=\xi]} p$,
- $Just(\chi_1 \wedge \ldots \wedge \chi_n, (\phi[p := \xi])[p := \chi_1 \wedge \ldots \wedge \chi_n])$
  where $\chi_1 \wedge \ldots \wedge \chi_n = \bigwedge\{\chi : \ \chi \sqsubset_\zeta p\}$

Now we construct a big gmp that does on its own what ξ and ζ do together.

By the strong inversion lemma we get a non-alien justification of each of the $\xi[p := \chi_i]$ in $\phi[p := \xi[p := \chi_i]]$. We also have the non-alien justification of $\bigwedge\{\chi : \ \chi \sqsubset_\xi p\}$ in $\phi[p := \bigwedge\{\chi : \ \chi \sqsubset_\xi p\}]$ in the big gmp. From these we can construct a non-alien justification of $\xi[p := \chi_1] \wedge ... \wedge \xi[p := \chi_n] \wedge \bigwedge\{\chi : \ \chi \sqsubset_\xi p\}$ in:

$$\phi[p := (\xi[p := \chi_1] \wedge ... \wedge \xi[p := \chi_n] \wedge \bigwedge\{\chi : \ \chi \sqsubset_\xi p\})].$$

By hypothesis we also get a non-alien justification of $\xi[p := \zeta]$ in $\phi[p := \xi[p := \zeta]]$. Now if we replace each of the χ_i in this claim by $\xi[p := \chi_i]$, we will be able to make a non-alien justification of this claim too. This construction is similar to the construction of a justification of $(\alpha \to ((\alpha \to \beta) \to \gamma))$ from a justification of $(\alpha \to (\beta \to \gamma))$.
So we get a non-alien justification of the claim

$$\xi[p := (\zeta[\chi_1 := \xi[p := \chi_1]] \dots [\chi_n := \xi[p := \chi_n]])]$$

in $\phi[p := \xi[p := (\zeta[\chi_1 := \xi[p := \chi_1]] \dots [\chi_n := \xi[p := \chi_n]])]]$.
This is the major premise of the big (alien) gmp that we are looking for. Note that we have made sure that this alien gmp is again a deepest alien gmp in the resulting justification.
So we can eliminate each deepest alien gmp. We may conclude that for all claims ψ a justification without applications of alien generalised modus ponens is available.
$\square$

The proof of the *Subformula Property* is the second big step on our way to finding a decision procedure. From the Inversion Lemma we know that we only have to consider atomic claims. Now we know that in the justification of these claims no alien applications of modus ponens have to be considered.

Decidability

At this point we can see that the following three kinds of steps will suffice to find a justification of a claim, if there is one.

1. If the claim we aim to justify is complex, we may break it down into atomic claims.

2. Then we check whether there is a direct justification for these atomic claims.

3. If not, then we try to apply generalised modus ponens with some supporting formula that has the same atomic claim, or that has $\bot$ as a claim.

The three steps give a search procedure for justifications, but as it stands it is not guaranteed that this procedure will terminate. If we are not careful we might end up trying to apply generalised modus ponens infinitely many times.
Consider, for example, $\phi = (((p \to q) \to q) \to (p \to q))$. If we start looking for a justification for this formula, we will, after some steps in the procedure, have the following goal:

$$Just(q, \phi).$$

At this point the only option we have is to use generalised modus ponens to justify q, i.e. we perform step 3. Then our new goal is:

$$Just((p \to q), (((p \to q) \to q) \to (p \to (p \to q)))\ ).$$

Then, again after some steps in the procedure, we have to justify:

$$Just(q, (((p \to q) \to q) \to (p \to (p \to q)))\ ).$$

And again there is no other option then generalised modus ponens. If we do not check for loops now, we can go on like this for ever, always substituting $(p \to q)$ for q. The following lemma will help us to prevent such infinite searches.

Lemma 5.7.5 (Finiteness Lemma) *If in a branch of the search procedure where we try to justify some claim ψ with supporting material $\{\chi_1, \ldots, \chi_n\}$ we arrive at a position where we have to justify ψ again with supporting material $\{\chi_1, \ldots, \chi_n\}$, then we may close this branch of the search tree.*

Proof:
At each point in the search procedure the search after that point is completely determined by the claim that we are trying to justify at that point and the supporting material that is available at that point. So the search tree after the first point where we try to justify ψ from $\{\chi_1, \ldots, \chi_n\}$ is isomorphic to the search tree after the second point where we try to justify ψ from $\{\chi_1, \ldots, \chi_n\}$. Therefore if we are looking for a finite branch in the search tree we need not look beyond the second point where we try to justify ψ from $\{\chi_1, \ldots, \chi_n\}$: any finite branch that exists there can

also be found in the isomorphic tree after the first point where we try to justify ψ from $\{\chi_1, \ldots, \chi_n\}$. □

This lemma will give us the possibility of cutting of all infinite branches of the search tree. To be precise:

Corollary 5.7.6 *If we are on an infinite branch in the search procedure, then this branch can be dismissed after finitely many steps.*

Proof:
Suppose that we find an infinite branch in the tree. Since we only have finitely many (atomic) subformulas in the original formula and each claim that we meet in the search is such a subformula, we can be sure that the infinite branch contains infinitely many attempts to prove the same claim, ψ say. Since all the supporting material that we find during the search will consist of subformulas of the original formula, the set of supporting material can only be one of finitely many sets. Therefore it is clear that at two points in the infinite branch we will try to prove ψ from the same supporting material, $\{\chi_1, \ldots, \chi_n\}$ say. At this point we can dismiss this branch according to the previous lemma. □

So the condition of the lemma allows us to eliminate infinite search paths. From this we may conclude:

Proposition 5.7.7 *$Jf(\phi)$ is decidable.* □

Note that this does indeed eliminate the counterexample that we considered above.
As a last illustration we show how the decision procedure works for Peirce's Law.

Example:
We apply the procedure to $(((p \to q) \to p) \to p)$, an instance of Peirce's Law. By step one of the procedure we try to find a justification for the atomic claim p. Since this claim has no direct proof (step 2), we have to apply generalised modus ponens to some supporting formula with claim p or $\perp$. There is one such formula, $((p \to q) \to p)$. We have to justify all the formulas that support p in this formula, i.e $(p \to q)$. So we now have as a goal to justify: $(((p \to q) \to p) \to (p \to q))$.

Now we apply step one again and try to find a justification for the atomic claim q in this formula. Since there is no direct proof for q (step 2), we try to perform step 3 again. But now we see that the procedure ends, since there is no supporting formula that has q or $\perp$ as a claim. We may conclude that Peirce's Law is not a theorem of intuitionistic logic.

Conclusion

Here our explorations of the dynamic environment end. So let's take some time to look back at what we have discovered and think about further steps that we could make in the future.

We have set out our view on dynamic semantics as a genuinely new approach to semantics. The attention for discourse phenomena naturally leads to a new outlook on interpretation: we can no longer afford to think of the expressions that we aim to interpret as small, complete units that we can toss and turn at liberty before we actually start to interpret them. In general we are dealing with large, incomplete natural language expressions for which a different style of interpretation is preferred. In particular this style of interpretation should show how we go through the process of interpretation step by step, always only working on one small bit of text at a time. Therefore we propose the *small unit principle* for dynamic interpretation.

The treatment of variables in dynamic semantics is a perfect illustration of this principle. Instead of treating $\exists x$ as an operator that has scope over a large piece of text, dynamic theories have managed to localise the contribution of the existential quantifier. Thus we can work with small units and have binding effects over long distances at the same time.

Such a localised treatment of existential quantification requires a careful representation of the different roles that variables play. First of all variables in dynamic semantics are information carriers: we use them to store the truth-conditional information that we find in texts. But secondly they also regulate the communication between the different parts of a texts: it is through the variables that information gets passed on from one part of text to another. In dynamic semantics we have to model both the accumulation of information through interpretation and the communication of information between different parts of a text. Therefore we propose *referent systems*, in which we can represent both roles at the same time. The *small unit principle* also has consequences for the treatment of text structure. It is a well-known fact in semantics that the structure of an expression carries information that is essential for its interpretation. Because of our commitment to the small unit principle we face the challenge of representing the influence of the structural organisation of texts in a localised way. We have illustrated this challenge using *if*... *then* constructions as an example. Our solution amounts to a sort of exchange mechanism: we can do without structural information in the syntax by introducing structure in the semantics.

This way of introducing more structure into the semantics and the use of non-truth-conditional information for the dynamic treatment of $\exists x$ follow a similar pattern: in both cases we face the challenge of coping with two kinds of information in one system. In a *discourse structure* these two kinds of information are the two roles that a variable has to play: first as storage facility and secondly as information channel from one part of text to another. In the *partial trees* we deal at the same time with the information content of formulas and with information about their structure. In both cases one kind of information controls the way in which the other kind of information is processed: the structural information in chapter 4 tells us how to combine the information content of subformulas and the way the variables in part I get linked determines how the information that is stored in them is added up.

This is a crucial observation about the dynamic approach: in dynamic semantics we have to develop good models to describe the interaction between information of different kinds. Both the partial trees of chapter 4 and the discourse structures of chapter 3 are examples of such models. Although these models indeed do what they were made for, more general questions about the combination of kinds of information remain. In Visser (1992a) and Visser (1992c) we find first investigations of this more general question, but a lot of work in this direction remains to be done. Dynamic semantics will have to develop a coherent theory about such heterogeneous information structures.

Finally we have tried to make a start with the development of a dynamic proof theory. Here the first job was to get a clear picture of what a dynamic proof theory should look like. We have proposed the *proofs as texts* perspective as an answer to this question. In chapter 5 we have illustrated this perspective in a toy example. The development of the dynamic proof theory of, for instance, *DPL* has not yet been completed. This will be taken up in future research.

Bibliography

R. Ahn and H-P Kolb, 1990. 'Discourse Representation Meets Constructive Mathematics'. In *Papers from the Second Symposium on Logic and Language*, L. Kálmán and L. Pólos (eds), pp. 105–124, Budapest. Akadémiai Kiadoó.

H. Andréka, I. Németi and I. Sain, 1993. 'Algebras of Relations and Algebraic Logic: an introduction'. unpublished manuscript.

N. Asher, 1986. 'A Treatment of Belief Sentences in Discourse Representation Theory'. *Journal of Philosophical Logic*, **15**, pp. 127–189.

N. Asher, 1993. *Reference to Abstract Objects in Discourse*, *Studies in Linguistics and Philosophy*, vol. 50. Kluwer, Dordrecht.

N. Asher and A. Lascarides, 1991. 'Discourse Relations and Defeasible Knowledge'. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pp. 55–63.

N. Asher, A. Lascarides and J. Oberlander, 1992, June. 'Inferring Discourse Relations in Context'. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pp. 1–8.

J. Baeten and W. Weijland, 1990. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press.

J. Barwise, 1987. 'Noun Phrases, Genralized Quantifiers and Anaphora'. In *Generalized Quantifiers: Linguistic and Logical Approaches*, P. Gärdenfors (ed.), pp. 1–29. Reidel, Dordrecht.

J. Barwise, J. Gawron, G. Plotkin, and S. Tutiya (eds), 1992. *Situation Theory and Its Applications.* CSLI Lecture Notes 26. CSLI, Stanford.

J. Barwise and L. Moss, Summer 1991. 'Lectures on Situation Theory and its Foundations'. Notes for the European Summerschool in Logic, Language and Information, Saarbrücken.

D.I. Beaver, 1992. 'The Kinematics of Presupposition'. In *Proceedings of the 8th Amsterdam Colloquium*, P. Dekker and M. Stokhof (eds).

D.I. Beaver, 1994. *What Comes First in Dynamic Semantics.* PhD thesis, University of Edinburgh.

D. Ben-Shalom, 1994, December. 'Natural Language, Generalized Quantifiers and Modal Logic'. In *Proceedings of the 9th Amsterdam Colloquium*, P. Dekker and M. Stokhof (eds).

J. van Benthem, 1989. 'Modal Logic as a Theory of Information'. Technical Report LP-89-05, ILLC, University of Amsterdam.

J. van Benthem, 1991. 'Logic and the Flow of Information'. Technical Report LP-91-10, ILLC, University of Amsterdam.

J. van Benthem and J. van Eijck, 1982. 'The Dynamics of Interpretation'. *Journal of Semantics*, **11**, pp. 3–20.

J. van Benthem, 1989a. 'Modal Logic as a Theory of Information'. Technical Report LP-89-05, ITLI, University of Amsterdam.

J. van Benthem, 1989b. 'Semantic Parallels in Natural Language and Computation'. In *Logic Colloquium, Granada 87*, H-D Ebbinghaus *et al* (ed.), pp. 331–375. North-Holland, Amsterdam.

J. van Benthem, 1991. 'General Dynamics'. *Theoretical Linguistics*, **17**, pp. 159–201.

J. van Benthem, 1993. *Language in Action.* North-Holland, Amsterdam.

J. van Benthem, 1994. 'A Note on Dynamic Arrow Logic'. In *Logic and Information Flow*, J. van Eijkc and A. Visser (eds), pp. 15–29. MIT press, Cambridge, Massachusetts.

J. van Benthem and J. Bergstra, 1993. 'Logic of Transition Systems'. Programming Research Group Report P9308, University of Amsterdam.

J. van Benthem, J. van Eijck and V. Stebletsova, 1993. 'Modal Logic, Transition Systems and Processes'. Computer Science/Department of Software Technology Report CS-R9321, CWI (Research Institute for Mathematics and Computer Science), P.O.Box 4079, 1009 AB, Amsterdam. To appear in the Journal of Logic and Computation.

M. van den Berg, H. Prüst and R. Scha, to appear. 'A Formal Discourse Grammar and Verb Phrase Anaphora'. *Linguistics and Philosophy.* (also Technical Report, Department of Computational Linguistics, University of Amsterdam).

P. Blackburn, C. Gardent and M. de Rijke, 1994. 'Back and Forth through Time and Events'. In *Proceedings of the 9th Amsterdam Colloquium*, P. Dekker and M. Stokhof (eds).

P. Blackburn and Y. Venema, 1993, June. 'Dynamic Squares'. Logic Group Preprint Series 92, Department of Philosophy, Utrecht University, Heidelberglaan 8, 3584 CS Utrecht.

M. Böttner, 1992. 'Variable Free Semantics for Anaphora'. *Journal of Philosophical Logic*, **21**, pp. 129–169.

G. Chierchia, 1991. 'Anaphora and Dynamic Binding'. *Linguistics and Philosophy*, **15**, pp. 111–183.

R. Cooper, K. Mukai and J Perry (eds), 1990. *Situation Theory and its Applications*, *CSLI Lecture Notes*, vol. 1. CSLI, Stanford.

K. van Deemter, 1991, March. *On the Composition of Meaning.* PhD thesis, University of Amsterdam.

P. Dekker, 1993, May. *Transsentential Meditations, Ups and Downs in Dynamic Semantics.* ILLC-dissertation series 1993-1, University of Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam.

J. van der Does, 1994, December. 'The Dynamics of Sophisticated Laziness'. In *Proceedings of the 9th Amsterdam Colloquium*, P. Dekker and M. Stokhof (eds).

J. van Eijck, 1993. 'The Dynamics of Description'. *Journal of Semantics*, **10**, pp. 239–267.

J. van Eijck (ed.), 1991a. *Logics in AI - Jelia '90, Lecture Notes in Artificial Intelligence*, vol. 478. Springer, Berlin.

J. van Eijck, 1991b. 'Presupposition Failure — A Comedy of Errors'. unpublished manuscript.

J. van Eijck, 1991c. 'The Dynamics of Description'. Technical Report Report CS-R9143, CWI, Amsterdam.

J. van Eijck and G. Cepparello, 1993, December. 'Dynamic Modal Predicate Logic'. Technical Report OTS-WP-CL-93-005, OTS (Research Institute for Language and Speech), Utrecht University, Trans 10, 3512 JK Utrecht. To appear in M. Kanazawa and C.J. Piñon (eds.), Dynamics, Polarity, and Quantification, CSLI, Stanford.

J. van Eijck and H. Kamp, to appear. 'The Representation of Discourse in Context'. In *Handbook of Logic and Linguistics*, J. van Benthem and A. ter Meulen (eds). North-Holland.

J. van Eijck and A. Visser (eds), 1994. *Logic and Information Flow.* MIT press, Cambridge, Massachusetts.

J. van Eijck and F.J. de Vries, 1992a. 'A Sound and Complete Calculus for Update Logic'. In *Proceedings of the Eighth Amsterdam Colloquium*, pp. 133–152. ILLC, Amsterdam.

J. van Eijck and F.J. de Vries, 1992b. 'Dynamic Interpretation and Hoare Deduction'. *Journal of Logic, Language, and Information*, **1**, pp. 1–44.

G. Evans, 1980. 'Pronouns'. *Linguistic Inquiry*, **11**, pp. 337–362.

T. Fernando, 1991a. 'Transition Systems and Dynamic Semantics'. Technical Report CS-R9217, CWI, Amsterdam.

T. Fernando, 1991b. 'Transition Systems Over First Order Models'. Manuscript, CWI, Amsterdam.

T. Fernando, 1993a, March. 'Bisimulations and predicate logic'. Technical Report CS-R9333, CWI, Amsterdam. To appear in the Journal of Symbolic Logic.

T. Fernando, 1993b, April. 'The Donkey Strikes Back'. In *Proceedings of the 6th Conference of the European Chapter of the Association of Computational Linguistics*, S. Krauwer, M. Moortgat and L. des Tombe (eds), pp. 130–138.

T. Fernando, 1994, December. 'Generalized Quantifiers as Second Order Programs - "Dynamically" Speaking, Naturally'. In *Proceedings of the 9th Amsterdam Colloquium*, P. Dekker and M. Stokhof (eds).

K. Fine, 1985. *Reasoning with Arbitrary Objects*, *Aristotelian Society Series*, vol. 3. Blackwell, Oxford.

G. Frege, 1979. 'Logische Mängel in der Mathematik'. In *Posthumous Writings*, H. Hermes *et al* (ed.). University of Chicago Press, Chicago.

D. Gabbay, 1990, December. 'Labelled Deductive Systems: a Position Paper - part I'. Technical Report CIS 90-22, Centr. Info. & Sprach., Universität München.

GAMUT, 1991. *Logic, Language and Meaning*, vol. I & II. University of Chicago Press. (Dutch version: Logica, Taal en Betekenis, 1982, published by Het Spectrum, De Meern, the Netherlands).

C. Gardent, 1991. 'Dynamic Semantics and VP Ellipsis'. In *Logics in AI - Jelia '90*, J. van Eijck (ed.), *Lecture Notes in Artificial Intelligence*, vol. 478, pp. 251–266. Springer, Berlin.

C. Gardent, 1993. 'A Unification-Based Approach to Multiple VP Ellipsis Resolution'. In *Proceedings of the 6th Conference of the European Chapter of the Association of Computational Linguistics*, S. Krauwer, M. Moortgat and L des Tombe (eds).

J. Gawron, J. Nerbonne and S. Peters, 1992. 'The Absorption Principle and E-type Anaphora'. In *Situation Theory and Its Applications, vol II*, J. Barwise et al (ed.), CSLI Lecture Notes 26, Stanford.

J. Gawron and S. Peters, 1990a. *Anaphora and Quantification in Situation Semantics*, *CSLI Lecture Notes*, vol. 19. CSLI, Stanford.

J. Gawron and S. Peters, 1990b. 'Some Puzzles about Pronouns'. In *Situation Theory and Its Applications, vol I*, R. Cooper et al (ed.), CSLI Lecture Notes 22.

P. Geach, 1962. *Reference and Generality.* Cornell University Press, Ithaca, New York.

R. Goldblatt, 1987. *Logics of Time and Computation, CSLI Lecture Notes*, vol. 7. CSLI, Stanford.

J. Groenendijk and M. Stokhof, 1990. 'Dynamic Montague Grammar'. In *Papers from the Second Symposium on Logic and Language*, L. Kálmán and L. Pólos (eds), pp. 3–48, Budapest. Akadémiai Kiadó.

J. Groenendijk and M. Stokhof, 1991a. 'Dynamic Predicate Logic'. *Linguistics and Philosophy*, **14**, pp. 39–100.

J. Groenendijk and M. Stokhof, 1991b. 'Two Theories of Dynamic Semantics'. In *Logics in AI — European Workshop JELIA '90*, J. van Eijck (ed.), Springer Lecture Notes in Artificial Intelligence, pp. 55–64, Berlin. Springer.

W. Groeneveld, 1991, July. 'Dynamic Semantics and Circular Propositions'. ITLI-prepublication series LP-91-03, Department of Philosophy, University of Amsterdam, Nieuwe Doelenstraat 15, 1012 CP Amsterdam.

D. Harel, 1984. 'Dynamic Logic'. In *Handbook of Philosophical Logic*, D. Gabbay and F. Günthner (eds), vol. II, pp. 497–604. Reidel, Dordrecht.

I. Heim, 1983. 'File Change Semantics and the Familiarity Theory of Definiteness'. In *Meaning, Use and Interpretation of Language*, R. Bäuerle, C. Schwarze and A. von Stechow (eds), pp. 164–189. De Gruyter, Berlin.

I. Heim, 1990. 'E-type Pronouns and Donkey Anaphora'. *Linguistics and Philosophy*, **13**, pp. 137–178.

I. Heim, 1992. 'Presupposition Projection and the Semantics of Attitude Verbs'. *Journal of Semantics*, **9**, pp. 183–221.

H. Hendriks, 1993, December. *Studied Flexibility.* ILLC-dissertation series 1993-5, University of Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam.

H. Hoekstra, 1992. 'Subsectional Anaphora in DRT'. In *OTS Yearbook 1992*, M. Everaert, B. Schouten and W. Zonneveld (eds), pp. 53–62. LEd & OTS, Trans 10, 3512 JK Utrecht.

N. Kadmon, 1990. 'Uniqueness'. *Linguistics and Philosophy*, **13**, pp. 273–324.

H. Kamp, 1981. 'A Theory of Truth and Semantic Representation'. In *Formal Methods in the Study of Language*, J. Groenendijk *et al.* (eds), Amsterdam. Mathematisch Centrum.

H. Kamp and U. Reyle, 1991. 'A Calculus for First Order Discourse Representation Theory'. Technical report, sonderforschungsbereich 340, Wissenschaftliches Zentrum der IBM Deutschland GmbH, Schlossstrasse 70, 7000 Stuttgart 1.

H. Kamp and U. Reyle, 1993. *From Discourse to Logic*, vol. I, II. Kluwer, Dordrecht.

H. Kamp and C. Rohrer, 1983. 'Tense in Texts'. In *Meaning, Use and Interpretation of Language*, R. Bäuerle, C. Schwarze and A. von Stechow (eds), pp. 250–269. De Gruyter.

M. Kanazawa, 1993. 'Dynamic Generalized Quantifiers and Monotonicity'. ILLC-prepublication series LP-93-02, University of Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam.

M. Kanazawa, 1994, December. 'Completeness and Decidability of the Mixed Style of Inference with Composition'. In *Proceedings of the 9th Amsterdam Colloquium*, P. Dekker and M. Stokhof (eds).

D. Kaplan, 1979. 'On the logic of demonstratives'. In *Contemporary Perspectives in the Philosophy of Language*, French, Uehling and Wettstein (eds), pp. 401–412. University of Minnesota Press.

L. Karttunen, 1976. 'Discourse Referents'. In *Syntax and Semantics*, J. McCawley (ed.), vol. 7, pp. 363–385. Academic Press, New York.

M. Kracht, 1989a. 'On the Logic of Category Definitions'. *Computational Linguistics*, **152**, pp. 111–113.

M.A. Kracht, 1989b, February. 'When can you say 'it'?'. Bericht der Gruppe Logik, Wissenstheorie und Information 3/89, II. Department of Mathematics, Freie Universität Berlin, Arnimallee 2-6, 1000 Berlin 33, Deutschland.

M.A. Kracht, 1990, December. 'Traditional Linguistics Can Solve Logical Puzzles'. Bericht der Gruppe Logik, Wissenstheorie und Information 12/90, II. Department of Mathematics, Arnimallee 2-6, 1000 Berlin 33, Deutschland.

M.A. Kracht, 1992, February. 'The Theorie of Syntactic Domains'. Logic Group Preprint Series 75, Department of Philosophy, Utrecht University, Heidelberglaan 8, 3584 CS Utrecht, the Netherlands.

M.A. Kracht, 1993a, April. 'Mathematical Aspects of Command Relations'. In *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics*, S Krauwer, M. Moortgat and L. des Tombe (eds), pp. 240–249.

M.A. Kracht, 1993b. 'Nearness and Syntactic Influence Spheres'. Technical report, II. Department of Mathematics, Arnimallee 2-6, 1000 Berlin 33, Deutschland.

M.A. Kracht, 1994a. 'Logic and Control: How They Determine the Behabiour of Presuppositions'. In *Logic and Information Flow*, J. van Eijck and A. Visser (eds), pp. 89–111, Cambridge, Massachusetts. MIT Press.

M.A. Kracht, 1994b. 'Syntactic Coding'. In *Proceedings of the 9th Amsterdam Colloquium*, M. Stokhof, P. Dekker and H. Hendriks (eds).

E. Krahmer, 1993, May. 'Donkeys Galore, Approaches to Discourse Semantics'. ITK Research Memo 17, ITK (Institute for Language Technology and Artidicial Intelligence), PO Box 90152, 5000 LE Tilburg.

E. Krahmer, 1994, March. 'Partial Dynamic Predicate Logic'. ITK Research Report 48, ITK (Institute for Language Technology and Artificial Intelligence), PO Box 90152, 5000 LE Tilburg.

N. Kurtonina, 1994 (to appear). 'The Lambek Calculus, Relational Semantics and the Method of of Labeling'. *Studia Logica*.

I. Lewin, 1994. 'Indexical Dynamics'. In *Applied Logic: How, What and Why?*, M. Masuch and L. Pólós (eds), Dordrecht. Kluwer. Proceedings of the Applied Logic Conference Logic@Work, 1992.

R. Maddux, 1983. 'A sequent calculus for relation algebra'. *Annals of Pure and Applied Logic*, **25**, pp. 73–101.

A. ter Meulen and J. Seligman, 1994. 'Dynamic Aspect Trees'. In *Applied Logic: How, What and Why?*, M. Masuch and L. Pólós (eds), Dordrecht. Kluwer. Proceedings of the Applied Logic Conference, Logic@Work, 1992.

W. Meyer Viol, 1994. 'Instantial Logic'. manuscript, to appear as PhD. thesis.

R. Montague, 1970. 'Universal Grammar'. *Theoria*, **36**.

L. Moss and J. Seligman, 1994. 'Classification Domains and Information Links: A Brief Survey'. In *Logic and Information Flow*, J. van Eijck and A. Visser (eds), pp. 112–124, Cambridge, Massachusetts. MIT Press.

R. Muskens, 1991. 'Anaphora and the Logic of Change'. In *Logics in AI - JELIA '90*, J. van Eijck (ed.), *Lecture Notes in Artificial Intelligence*, vol. 478, pp. 414–430, Berlin. Springer.

R. Muskens, 1994. 'A Compositional Discourse Representation Theory'. In *Proceedings 9th Amsterdam Colloquium*, P. Dekker and M. Stokhof (eds), pp. 467–486. ILLC, Amsterdam.

R. Muskens, to appear. 'Tense and the Logic of Change'. In *Interface Aspects of Syntax, Semantics and the Lexicon*, U. Egli et al (ed.). Benjamins, Philadelphia.

P. Pagin and D. Westerståhl, 1993. 'Predicate Logic with Flexibly Binding Operators'. *Journal of Logic, Language and Information*, **2**, pp. 89–128.

L. Polanyi and R. Scha, 1988. 'An Augmented Context Free Grammar'. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pp. 573–577.

V. Pratt, 1991. 'Action Logic and Pure Induction'. In *Logics in AI — European Workshop JELIA '90*, J. van Eijck (ed.), pp. 97–120. JELIA, Springer, Berlin.

W.C. Purdy, 1992a. 'A Variable-Free Logic for Anaphora'. Manuscript, Syracuse University.

W.C. Purdy, 1992b. 'Surface reasoning'. *Notre Dame Journal of Formal Logic*, **33**, pp. 13–36.

A. Ranta, 1991. 'Intuitionistic Categorial Grammar'. *Linguistics and Philosophy*, **14**, pp. 203–239.

T. Reinhart, 1983. *Anaphora and Semantic Interpretation.* Croom Helm Linguistics Series. Croom Helm Ltd., London.

U. Reyle, 1993. 'Vagueness and Ambiguity in DRT'. *Journal of Semantics.*

M. de Rijke, 1992. 'A system of dynamic modal logic'. ILLC-prepublication lp-92-08, University of Amsterdam.

M. de Rijke, 1993, December. *Extending Modal Logic.* ILLC-dissertation series, University of Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam.

C. Roberts, 1989. 'Modal Subordination and Pronominal Anaphora in Discourse'. *Linguistics and Philosophy*, **12**, pp. 683–721.

V. Sánchez Valencia, 1990. *Studies on Natural Logic and Categorial Grammar.* PhD thesis, University of Amsterdam, Amsterdam.

R. van der Sandt, 1989. 'Presupposition and Discourse Structure'. In *Semantics and Contextual Expressions*, R. Bartsch, J. van Benthem and P. van Emde Boas (eds). Foris, Dordrecht.

R. van der Sandt, 1992. 'Presupposition Projection as Anaphora Resolution'. *Journal of Semantics*, **9**, pp. 333–377.

W. Saurer, 1993. 'A Natural Deduction System for Discourse Representation Theory'. *Journal of Philosophical Logic*, **22**.

D. Schmidt, 1988. *Denotational Semantics: a methodology for language development.* Brown, Dubuque, Iowa.

P. Seuren, 1985. *Discourse Semantics.* Blackwell, Oxford.

C. Sidner, 1983. 'Focusing in the Comprehension of Definite Anaphora'. In *Computational Models of Discourse*, M. Brady and R. Berwick (eds), pp. 267–330. MIT Press, Cambridge, Massachusetts.

R. Stalnaker, 1972. 'Pragmatics'. In *Semantics of Natural Language*, D. Davidson and G. Harman (eds), pp. 380–397. Reidel.

R. Stalnaker, 1978. 'Assertion'. In *Syntax and Semantics 9: Pragmatics*, P. Cole (ed.), vol. 9. Academic Press.

G. Sundholm, 1984. 'Proof Theory and Meaning'. In *Handbook of Philosophical Logic*, D. Gabbay and F. Günthner (eds), vol. III, pp. 471–506. Reidel, Dordrecht.

A. Tarski, 1941. 'On the Calculus of Relations'. *Journal of Symbolic Logic*, **6**, pp. 73–89.

R. Thomason (ed.), 1974. *Formal Philosophy, selected papers of Richard Montague.* Yale University Press, New Haven.

A. Troelstra, 1992. *Lectures on Linear Logic.* CSLI Lecture Notes 29. CSLI, Stanford.

A. Troelstra and D. van Dalen, 1988. *Constructivism in Mathematics, Studies in Logic and the Foundations of Mathematics*, vol. I and II. North-Holland, Amsterdam.

F. Veltman, 1991. 'Defaults in Update Semantics'. In *Conditionals, Defaults and Belief Revision*, H. Kamp (ed.). Dyana Deliverable R2.5A, Edinburgh.

F. Veltman, J. Groenendijk and M. Stokhof, 1993. 'Coreference and Modality'. Philosophy Department, University of Amsterdam.

Y. Venema, 1992. *Many-Dimensional Modal Logic.* PhD thesis, University of Amsterdam.

Y. Venema, 1993, February. 'A Crash Course in Arrow Logic'. Technical report, Department of Philosophy, Utrecht University, Heidelberglaan 8, 3584 CS Utrecht.

H.J. Verkuyl, 1993. *A theory of Aspectuality; The Interaction between Temporal and Atemporal Structure.* Cambridge Studies in Linguistics 64. Cambridge University Press.

H.J. Verkuyl and C.F.M. Vermeulen, 1993, December. 'Shifting Perspectives in Discourse'. unpublished manuscript.

C.F.M. Vermeulen, 1989, September. *A Dynamic Analysis of Reasoning.* Master's thesis, Department of Philosophy, Utrecht University.

C.F.M. Vermeulen, 1991, December. 'Merging without Mystery, Variables in Dynamic Semantics'. OTS-working paper OTS-WP-CL-91-003, O.T.S. (Research Institute for Language and Speech), Utrecht University, Trans 10, 3512 JK Utrecht, the Netherlands. Also appeared as: idem, Logic Group Preprint Series, 70, Department of Philosophy, Utrecht University.

C.F.M. Vermeulen, 1993a, January. 'Incremental Semantics for Propositional Texts'. OTS-working paper OTS-WP-CL-93-001, OTS (Research Institute for Language and Speech), Utrecht University, Trans 10, 3512 JK Utrecht, the Netherlands. Also as Logic Group Preprint Series 85, Department of Philosophy, Utrecht University; will appear in Notre Dame Journal of Formal Logic.

C.F.M. Vermeulen, 1993b, November. 'Proofs as Texts, Dynamic Proof Theory for Intuitionistic Propositional Logic'. OTS-working paper OTS-WP-CL-93-007, OTS (Research Institute for Language and Speech), Utrecht University, Trans 10, 3512 JK Utrecht, the Netherlands. Also appeared as Logic Group Preprint Series 102, Department of Philosophy, Utrecht University.

C.F.M. Vermeulen, 1993c. 'Sequence Semantics for Dynamic Predicate Logic'. *Journal of Logic, Language and Information*, **2**, pp. 217–254. after: idem, Logic Group Preprint Series, 60, Department of Philosophy, Utrecht University, January 1991.

C.F.M. Vermeulen, 1994a, March. 'Three Theories of Dynamic Semantics'. handout for AiO-course of OZSL.

C.F.M. Vermeulen, 1994b. 'Update Semantics for Propositional Texts'. In *Applied Logic: How, What and Why?*, M. Masuch and L. Pólós (eds), Dordrecht. Kluwer. Proceedings of the Applied Logic Conference, Logic@Work, 1992.

A. Visser, 1989. 'De ezel van Geach'. Lecture notes, Philosophy Department, Utrecht University, last update: Spring Semester 1994.

A. Visser, 1992a. 'Actions under Presuppositions'. In *Logic and Information Flow*, J. van Eijck and A. Visser (eds). MIT Press, Cambridge, Mass.

A. Visser, 1992b, November. 'Lazy and Quarrelsome Brackets'. Logic Group Preprint Series 82, Department of Philosophy, Utrecht University, Heidelberglaan 8, 3584 CS Utrecht, the Netherlands.

A. Visser, 1992c. 'Meanings in Time'. unpublished manuscript.

R. de Vrijer, 1990. 'Natural Deduction for DPL'. unpublished manuscript, Philosophy Department, University of Amsterdam.

H. Zeevat, 1987a. 'A Treatment of Belief Sentences in DRT'. In *Studies in Discourse Representation Theory and the Theory of Generalized Quantifiers*, D. de Jongh, J. Groenendijk and M. Stokhof (eds), pp. 189–215. Foris, Dordrecht.

H. Zeevat, 1987b. 'Combining Categorial Grammar and Unification'. In *Natural Language Parsing and Linguistic Theory*, U. Reyle and C. Rohrer (eds), pp. 202–229. Kluwer, Dordrecht.

H. Zeevat, 1991a. 'A Compositional Approach to DRT'. *Linguistics and Philosophy*, **12**, pp. 95–131.

H. Zeevat, 1991b, May. *Aspects of Discourse Semantics and Unification Grammar*. PhD thesis, University of Amsterdam.

H. Zeevat, 1992a. 'Presupposition and Accommodation in Update Semantics'. *Journal of Semantics*, **94**, pp. 379–412. Special Issue: Presupposition, Part 2.

H. Zeevat, 1992b. 'Presuppositions in Update Semantics'. *Journal of Semantics*, **9**.

H. Zeevat, E. Klein and J. Calder, 1987. *Unification Categorial Grammar*, pp. 195–233, in *Categorial Grammar, Unification Grammar and Parsing.* Edinburgh Working Papers in Cognitive Science. E. Klein and N. Haddock and G. Morrill.

L. Zeinstra, 1990. *Reasoning as Discourse.* Master's thesis, Philosophy Department Utrecht University.

Samenvatting

Verkenningen van Dynamische Contexten

Onderwerp van het proefschrift is de dynamische semantiek. In het proefschrift wordt de dynamische semantiek beschouwd als een wezenlijk nieuwe benadering van de interpretatie van natuurlijke taal: dynamische semantiek biedt een ander perspectief op het interpretatieproces en introduceert daarbij nieuwe technieken en methoden. Het proefschrift is gedeeltelijk een poging om tot de kern van het nieuwe van deze benadering, formeel en informeel, door te dringen en gedeeltelijk een poging om binnen de nieuwe benadering een bijdrage te leveren aan de ontwikkeling van de benodigde formele methoden en technieken.

In hoofdstuk 1 schetsen we de dynamische benadering van semantiek. Eerst wordt aan de hand van enkele voorbeelden een beeld gegeven van de verschillende formalismen—*DRT* en *DPL*—die de dynamische semantiek haar gezicht hebben gegeven. Daarna wordt in informele termen besproken waarin het cruciale verschil ligt tussen deze dynamische benaderingen en de gebruikelijke benadering in de traditie van Montague: de dynamische semantiek beperkt zich niet tot de (compositionele) analyse van de waarheidscondities van de expressies, maar wil ook het *stap voor stap* karakter van de interpretatie van natuurlijke taal correct modelleren.

We kunnen deze extra nevenvoorwaarde begrijpen als we zien dat de dynamische semantiek zich richt op de interpretatie van *teksten*, terwijl de Montague traditie vooral gericht was op de interpretatie van *zinnen*. Het is verleidelijk om aan zinnen te denken als relatief kleine, afgeronde eenheden, waar we rustig onze gedachten over kunnen laten gaan alvorens we ze interpreteren. Maar in het geval van teksten is dat beeld niet langer houdbaar: teksten zijn bij uitstek grote, onaffe expressies, waarvan we ons de interpretatie alleen kunnen voorstellen als een stapsgewijs pro-

ces, waarin steeds alleen een klein stukje van de tekst tegelijk bekeken wordt. Daarom stellen we het *small unit principle* (*principe van de kleine eenheden*) voor als karakteristiek voor dynamische semantiek: in een dynamische semantiek wordt interpretatie gemodelleerd als een process waarbij steeds slechts op één klein stukje van de expressie tegelijk geopereerd wordt.
We constateren dat de dynamische interpretatie van $\exists x$, zoals we die in hoofdstuk 1 gezien hebben, hiervan een uitstekend voorbeeld is. In plaats van een traditionele analyse van de existentiële kwantor als operator waarvan het globale bereik vooraf gespecificeerd moet worden, komt een semantiek waarin de bijdrage van $\exists x$ locaal beschreven kan worden.
In deel I wordt verder stil gestaan bij de gevolgen van deze locale representatie van existentiële kwantificatie. In een dynamische semantiek wordt het noodzakelijk de verschillende functies van variabelen zorgvuldig te onderscheiden. In de eerste plaats zijn variabelen registers waarin we informatie kunnen opslaan. Maar variabelen spelen ook een cruciale rol bij de communicatie tussen verschillende delen van een tekst. Het transport van informatie van het ene deel van de tekst naar het andere wordt door de variabelen gereguleerd. In de dynamische semantiek moet zowel het aspect van informatieopslag als het aspect van verplaatsing van informatie worden weergegeven. We stellen *referent systems* voor, als formaat waarin beide rollen van variabelen afdoende beschreven kunnen worden.
In deel II laten we zien dat het *small unit principle* ook consequenties heeft voor de analyse van de *structuur* van teksten. Informatie over de structuur van een expressie speelt een essentiële rol bij de interpretatie ervan. Maar het *small unit principle* zegt dat we slechts aan een klein deel van die structuur tegelijk kunnen werken. Het is daarom onvermijdelijk dat we een deel van de structurele informatie opslaan in onze semantische representaties. We illustreren hoe dit werkt aan de hand van *als... dan* constructies in teksten.
In feite is hier sprake van een soort uitwisselingsprogramma tussen syntaxis en semantiek: we hoeven steeds minder structurele informatie te specificeren in de syntaxis als we bereid zijn structurele informatie een plaats te geven in de semantiek.
Het is belangrijk op te merken dat de manier waarop in de dynamische semantiek de $\exists x$ kwantor locaal wordt gemaakt en de manier waarop we de *als... dan* constructies analyseren, volgens een gemeenschappelijk patroon verlopen: in beide gevallen moet er met twee soorten informatie

tegelijk worden gemanipuleerd, waarbij de ene soort informatie de manipulatie van de andere soort informatie stuurt. In deel I gaat het daarbij om de twee rollen die variabelen spelen: de communicatieve rol van de variabele bepaald de accumulatie van informatie. In deel II gaat het om structurele informatie enerzijds en waarheidsconditionele informatie anderzijds: de structuur van de expressie bepaalt hoe de waarheidscondities van de subexpressies gecombineerd moeten worden tot de waarheidsconditie van de expressie in zijn geheel.

Dit is een cruciale observatie over de dynamische benadering van semantiek: in de dynamische semantiek is behoefte aan goede formele modellen om de systematische interactie van verschillende soorten informatie te beschrijven. Het proefschrift geeft twee voorbeelden van zulke modellen, maar het valt te verwachten dat een meer algemene benadering van dit probleem vruchten zal afwerpen. Dit lijkt de logische volgende stap in de ontwikkeling van de dynamische semantiek te zijn: het ontwikkelen van een abstract, formeel kader waarbinnen de interactie tussen verschillende soorten van informatie flexibel gemodelleerd kan worden.

Tenslotte is in hoofdstuk 5 een begin gemaakt met de studie van dynamische bewijssystemen. In dit hoofdstuk wordt besproken hoe de dynamische benadering van semantiek ook zijn gevolgen zou kunnen hebben voor de bewijstheorie. Met name wordt voorgesteld om *bewijzen als teksten* te beschouwen. Hierbij worden bewijsregels dan instructies om bewijsteksten stap voor stap op te bouwen. Het idee van *bewijzen als teksten* wordt geïllustreerd aan de hand van een eenvoudig voorbeeld: er wordt een dynamisch bewijssysteem beschreven voor een fragment van de intuïtionistische propositie logica. Hierbij wordt vooral duidelijk dat het traditionele onderscheid tussen de syntax van de *formules* (op het ene niveau) en de syntax van de *bewijzen* van deze formules (op een hoger niveau) in een dynamische benadering van bewijzen niet langer voor de hand ligt.