Notre Dame Journal of Formal Logic
Volume 35, Number 2, Spring 1994

Incremental Semantics for Propositional Texts

C. F. M. VERMEULEN

Abstract In this paper we are concerned with the special requirements that a semantics of texts should meet. It is argued that a semantics of texts should be incremental and should satisfy the break in principle. We develop a semantics for propositional texts that satisfies these constraints. We will see that our requirements do not only apply to the semantics but also have consequences for the syntax. The interaction between text structure and text meaning will turn out to be of crucial importance to the semantics of texts. We develop two versions of the semantics: one representational, one in update style.

1 Introduction Traditionally in formal semantics the attention has been focussed on the interpretation of *sentences*. But since it was argued, by Kamp [4] and Heim [3] for example, that the semantics of texts requires more than a straightforward extension of the techniques developed for sentences, text semantics has become a separate topic of research. It is now quite generally recognised that special tools have to be developed for the analysis of typically text level phenomena such as anaphora.

The tools that have been developed for the semantics of texts also have been put to use in the analysis of sentences. For example in Kamp's *Discourse Representation Theory* and Heim's *File Change Semantics* it is argued that anaphors that find their antecedent within the sentence can best be treated in the same way as anaphors that find their antecedent in another sentence in the text. The so-called *donkey sentence* is a good example of a situation where this approach pays off:

If a farmer owns a donkey, he beats it.

In the approaches mentioned above, this sentence obtains the required interpretation in a natural way, while this is quite hard in traditional sentential semantics. So the semantics of texts has led to the development of new techniques which have proved useful for the study of old problems in sentential semantics.

In this paper we are concerned with the consequences of this shift of attention for the requirements on the formal methods that are used. In sentence semantics the all important methodological constraint is *compositionality*. But

Received May 5, 1993; revised October 7, 1993

it seems that the compositionality principle, as it stands, is not appropriate for the semantics of texts. Instead we propose other constraints: *incrementality, pure compositionality*, and *the break in principle.* We will develop a semantics for propositional texts that satisfies these 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. Of course, 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. (Note that it is harder to imagine a non-incremental treatment of a *spoken* text.) 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 here we are talking about the text level. We do not want to claim here that everything that happens in the semantics of natural language has to be accounted for incrementally. It is not excluded that some micro level phenomena behave differently. Intuitions about incrementality typically apply to the macro level and this is also the level for which text semantics is designed.

The incrementality constraint gives rise to an important difference with 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 how the compositionality principle works in 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 analysis of meaning and structure have to be performed at the same time. Therefore the compositionality principle is, in its usual form, not appropriate for text semantics.

Instead we will use a more modest form of compositionality than we are used to in sentence semantics. Of course the meaning of the text as a whole is composed from the meanings of the parts of the text: we do not want foreign elements to influence the interpretation of a text. But we cannot assume that information about the structure of the text will tell 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. Our modest form of the compositionality will be called *pure compositionality*: it simply states that the meaning of a text depends on nothing but the meaning of its parts.

The last 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. But then it is inevitable that also our interpretations will have this property: they are *incomplete* or *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.

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 listen in on a conversation, in a train, for example: 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, in some sense, *partial meanings.*

This principle has serious consequences in the 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 to interpret a text as we hear it.

The semantics we give in this paper incorporates the three principles: we give a compositional, 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 reasonings. 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 *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 $(\phi \to \psi)$. (Note that we only consider texts in which the assumptions are given before their conclusions.) The formal language that we will work with is defined as follows.

Definition 1.1 Let a vocabulary of atomic texts A be given. We define the texts over A, $Text_A$, as follows:

$$\begin{aligned}
&\mathit{if}, \mathit{then}, \mathit{end} \in Text_A\\
&\bot \in Text_A\\
&p \in A \Rightarrow p \in Text_A\\
&\phi \in Text_A \text{ and } \psi \in Text_A \Rightarrow \phi\psi \in Text_A.
\end{aligned}$$

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 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 present a deduction system in which proofs are considered as a special kind of text, texts of which the construction satisfies a number of additional syntactic constraints.[1] We will not develop such a deduction system in this paper, but we intend to present it in another paper.

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. We will not attempt anything like that here, nor do we pretend that it is clear how we could extend the approach in that direction. It seems that in sentences many phenomena occur that do not have a counterpart at the level of texts. (Some problems are discussed in Visser [7], who considers a very limited fragment.) But at the same time it is clear that some phenomena in sentences simply are special cases of text phenomena. Here we think of donkey anaphors, for example, but also of 'if . . . then' constructions, which seem to be nothing but an internalisation of the kind of text structure that is the topic of this paper.

2 *Texts as sequences*

In this section we present our first attempt at an incremental 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 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:

$(a \wedge b) \wedge c = a \wedge (b \wedge c)$	(associativity of $\wedge$)
$(a \vee b) \vee c = a \vee (b \vee c)$	(associativity of $\vee$)
$a \wedge b = b \wedge a$	(commutativity of $\wedge$)
$a \vee b = b \vee a$	(commutativity of $\vee$)
$a \wedge a = a$	(idempotency of $\wedge$)
$a \vee a = a$	(idempotency of $\vee$)
$a \wedge (a \vee b) = a$	(first absorption law)
$a \vee (a \wedge b) = a$	(second absorption law)

In a lattice $\mathbf{L}$ we can define an ordering by:

$$a \leq b \Leftrightarrow a \wedge b = a.$$

2. A Heyting algebra is a structure $\mathbf{I} = (I, \wedge, \vee, \perp, \rightarrow)$, such that $(I, \wedge, \vee)$ is a lattice, $\perp$ is the least element of I and $\rightarrow$ is a binary operation such that:

$$(\iota_0 \wedge \iota_1 \leq \iota_2) \Leftrightarrow (\iota_0 \leq \iota_1 \rightarrow \iota_2).$$

Note that each Heyting algebra has a top element, $(\perp \rightarrow \perp)$, which we will call $\top$. 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 [10], another is the algebra of relations that Groenendijk and Stokhof [2] use.

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. It is assumed that for each atomic text $p \in A$ an information state ι_p is given: ι_p is the information that p. (We will use postfix notation for function application and we will adapt the notation for function composition accordingly.)

Definition 2.2 We define for each ϕ the update function $[\phi]$ as follows. Let an information history $\sigma = (\sigma_1, \ldots, \sigma_n)(n \geq 1)$ be given by:

$$\begin{aligned}
\sigma[\bot] &= (\sigma_1, \ldots, \sigma_{n-1}, \sigma_n \wedge \bot)\\
\sigma[p] &= (\sigma_1, \ldots, \sigma_{n-1}, \sigma_n \wedge \iota_p)\\
\sigma[\mathit{if}] &= (\sigma_1, \ldots, \sigma_{n-1}, \sigma_n, \top)\\
\sigma[\mathit{then}] &= (\sigma_1, \ldots, \sigma_{n-1}, \sigma_n, \top)\\
\sigma[\mathit{end}] &= (\sigma_1, \ldots, \sigma_{n-2} \wedge (\sigma_{n-1} \rightarrow \sigma_n))\\
\sigma[\phi\psi] &= (\sigma[\phi])[\psi].
\end{aligned}$$

Furthermore we define truth as follows:

For $\iota \in \mathbf{I}$ we define$(\iota) \models \phi$ iff $(\iota)[\phi] = (\iota)$. We say that ϕ is true in ι.
We write $\models \phi$ iff $(\top) \models \phi$. We say that ϕ is true (in $\mathbf{I}$).

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 [6]'s update semantics. He uses an information algebra that is defined as follows:

Definition 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 $\mathbf{I} = \wp(W)$. **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 ι_p $(p \in A)$ is: $\iota_p = \uparrow(\{p\}) = \{w : \{p\} \subseteq w\}$. Definition 2.2 gives us the right result for texts like *p if q then r end*: it is easy to check that now:

$$(\top)[p\ \mathit{if}\ q\ \mathit{then}\ r\ \mathit{end}] = (\iota_p \wedge (\iota_q \rightarrow \iota_r)).$$

And, since function composition is associative, the semantics clearly is incremental and associative, as required. 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: $[\mathit{if}] = [\mathit{then}]$. Thereby also $[\mathit{if}\ p\ \mathit{then}\ q\ \mathit{end}] = [\mathit{then}\ p\ \mathit{if}\ q\ \mathit{end}]$. 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 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. (In fact all partiality in the semantics of definition 2.2 originate from the partiality of *end.*) 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 in fact ever more refined incremental well formedness test will become implicitly available in our machinery as we proceed.

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 were 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.

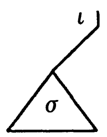
Note that after *end* we can actually construct the implication in the Heyting algebra, and we no longer need the tree structure. As a consequence not all binary trees have to occur in the semantics. We can restrict ourselves to trees of the following kind.

Definition 3.1 Let a Heyting algebra $\mathbf{I}$ be given. We define the update trees over $\mathbf{I}$, $U_{\mathbf{I}}$, as follows:

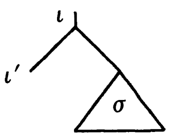
$$\begin{aligned}
&\iota \in \mathbf{I}, && \Rightarrow (\iota) \in U_{\mathbf{I}};\\
&\iota \in \mathbf{I} \text{ and } \sigma \in U_{\mathbf{I}}, && \Rightarrow (\iota,\sigma) \in U_{\mathbf{I}};\\
&\iota \in \mathbf{I},\ \iota' \in \mathbf{I},\ \sigma \in U_{\mathbf{I}} && \Rightarrow (\iota,(\iota'),\sigma) \in U_{\mathbf{I}}.
\end{aligned}$$

Maybe one does not immediately recognise these objects as binary trees. They can be read as follows: the general format is $(\iota,(\iota'),\sigma)$ where σ is itself an update tree. The first component contains the information so far, ι. We think of it as a flag at the root of the tree. The second component, (ι'), 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. Instead 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.[2] So we simply have (ι) if we are not processing an implication at the moment, and we have (ι,σ) if we are building up the antecedent of an implication. All components are filled, $(\iota,(\iota'),\sigma)$, 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.

ι° a tree of the form (ι)



a tree of the form (ι,σ)



a tree of the form $(\iota,(\iota'),\sigma)$

Each time the simplest example of such a configuration arises when σ is of the form (ι''). 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 3.2 We define for each tree τ its final segment, $seg_f(\tau)$, as follows:

$$\begin{aligned}
&seg_f((\iota)) && = (\iota);\\
&seg_f((\iota,(\iota'))) && = (\iota,(\iota'));\\
&seg_f((\iota,(\iota'),(\iota''))) && = (\iota,(\iota'),(\iota''));\\
&seg_f((\iota,\sigma)) && = seg_f(\sigma) \text{ if } \sigma \neq (\iota');\\
&seg_f((\iota,(\iota'),\sigma)) && = seg_f(\sigma) \text{ if } \sigma \neq (\iota'').
\end{aligned}$$

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 $\sigma\{\rho'\}$. This 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.[3] 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.

Definition 3.3 Let $\sigma \in U_{\mathbf{I}}$ be given. The following clauses define the update functions $[\phi]$ for $\phi \in \mathit{Text}_A$:

$$\begin{aligned}
&\sigma⟦(\iota)⟧[\bot] &&= \sigma\{(\iota \sim \bot)\};\\
&\sigma⟦(\iota',(\iota))⟧[\bot] &&= \sigma\{(\iota',(\iota \wedge \bot))\};\\
&\sigma⟦(\iota'',(\iota'),(\iota))⟧[\bot] &&= \sigma\{(\iota'',(\iota'),(\iota \wedge \bot))\};\\[1ex]
&\sigma⟦(\iota)⟧[p] &&= \sigma\{(\iota \wedge \iota_p)\};\\
&\sigma⟦(\iota',(\iota))⟧[p] &&= \sigma\{(\iota',(\iota \wedge \iota_p))\};\\
&\sigma⟦(\iota'',(\iota'),(\iota))⟧[p] &&= \sigma\{(\iota'',(\iota'),(\iota \wedge \iota_p))\};\\[1ex]
&\sigma⟦(\iota)⟧[\mathit{if}] &&= \sigma\{(\iota,(\top))\};\\
&\sigma⟦(\iota',(\iota))⟧[\mathit{if}] &&= \sigma\{(\iota',(\iota,(\top)))\};\\
&\sigma⟦(\iota'',(\iota'),(\iota))⟧[\mathit{if}] &&= \sigma\{(\iota'',(\iota'),(\iota,(\top)))\};\\[1ex]
&\sigma⟦(\iota',(\iota))⟧[\mathit{then}] &&= \sigma\{(\iota',(\iota),(\top))\};\\
&\sigma⟦(\iota'',(\iota'),(\iota))⟧[\mathit{end}] &&= \sigma\{(\iota'' \wedge (\iota' \rightarrow \iota))\};\\
&\sigma[\phi\psi] &&= (\sigma[\phi])[\psi].
\end{aligned}$$

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:

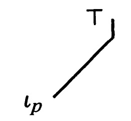
For $\iota \in \mathbf{I}$ we define $(\iota) \models \phi$ iff $(\iota)[\phi] = (\iota)$. We say that ϕ is true in ι. We write $\models \phi$ iff $(\top) \models \phi$. We say that ϕ is true (in $\mathbf{I}$).

Note that for $[\bot]$, $[p]$ and $[\mathit{if}]$ we do not need the entire final segment: only the very latest information state in the configuration, ι, is required. In $[\mathit{then}]$ and $[\mathit{end}]$ 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.

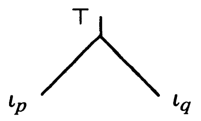
Example 3.4

$$\begin{aligned}
&(\top)[\mathit{if}\ p\ \mathit{then}\ q\ \mathit{end}] &&=\\
&(\top,(\top))[p\ \mathit{then}\ q\ \mathit{end}] &&=\\
&(\top,(\iota_p))[\mathit{then}\ q\ \mathit{end}] &&=\\
&(\top,(\iota_p),(\top))[q\ \mathit{end}] &&=\\
&(\top,(\iota_p),(\top \wedge \iota_q))[\mathit{end}] &&=\\
&(\iota_p \rightarrow \top \wedge \iota_q) &&=\\
&(\iota_p \rightarrow \iota_q).
\end{aligned}$$

We give pictures for two of the stages in the process.



$(\top)$ [if p]



$(\top)$ [if p then q]

Now we can introduce the following notions of well formedness for texts.

1. $[\phi]$ is grammatical if $[\phi]$ is a total function and $(\top)[\phi] = (\iota)$ for some $\iota \in \mathbf{I}$.
2. $[\phi]$ is right incomplete (but left complete) if $[\phi]$ is a total function and $(\top)[\phi] \neq (\iota)$.
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 case 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 paper, we will see whether it is necessary to use an update formulation of the semantics.

4 Trees as an update algebra

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 into the meaning of a sentence. In Visser [8] 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 4.1 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 steal bits of information from previous information states.

It might be helpful to think about the partiality of information in terms of evaluation: the truthvalue 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.

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 left-hand side. Visser calls this the *Once a Thief, Always a Thief*, or *OTAT* principle. The *OTAT* principle introduces an 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.

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 $Id = \Phi_{id}$. (Here $\circ$ stands for function composition.)

Now the notion of an update algebra is inspired by the following fact:

Proposition 4.2 *Let a merge algebra $(X, S, \bullet, id)$ be given. Consider the monoid of update functions $(F_S, \circ, \Phi_{id})$. Define $\Phi : (X, \bullet, id) \to (F_S, \circ, \Phi_{id})$ by $x\Phi = \Phi_x$. Then Φ is a homomorphism of monoids iff $(X, S, \bullet, id)$ is an update algebra.*

Proof: For a proof we refer to Visser [8].

The fact that Φ is a homomorphism guarantees that $\Phi_{x \bullet y} = \Phi_x \circ \Phi_y$. This implies that:

$$\Phi_{x \bullet (y \bullet z)} =$$

$$\Phi_x \circ \Phi_{y \bullet z} =$$

$$\Phi_x \circ (\Phi_y \circ \Phi_z) = (\Phi_x \circ \Phi_y) \circ \Phi_z =$$

$$(\Phi_{x \bullet y}) \circ \Phi_z$$

$$\Phi_{(x \bullet y) \bullet z}$$

(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.

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 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.

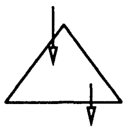
The construction process follows a fixed route through the tree: left-to-right, top-down path. If we want to analyse the construction process of some tree, we simply have to follow this path. In this way the stages of the construction process are represented in the tree by the segments of the path. (Maybe 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.) 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.)

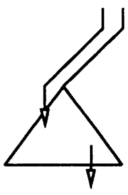
Definition 4.3 We define the (partial) trees over some *HA* $\mathbf{I}, T_{\mathbf{I}}$, inductively. In our definition we have to distinguish the subclasses $downT_{\mathbf{I}}$ for the down trees and $upT_{\mathbf{I}}$, for the up trees. (This terminology will be explained below.)

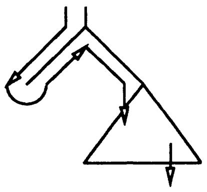
1. $\iota \in \mathbf{I} \quad \Rightarrow (\iota) \in downT_{\mathbf{I}} \cap upT_{\mathbf{I}};$
2. $\iota \in \mathbf{I},\ \sigma \in downT_{\mathbf{I}} \quad \Rightarrow (\iota,\sigma) \in downT_{\mathbf{I}};$
3. $\iota \in \mathbf{I},\ \sigma \in upT_{\mathbf{I}} \quad \Rightarrow (\sigma,\iota) \in upT_{\mathbf{I}};$
4. $\iota,\ \iota' \in \mathbf{I},\ \sigma \in downT_{\mathbf{I}} \quad \Rightarrow (\iota,(\iota'),\sigma) \in downT_{\mathbf{I}};$
5. $\iota,\ \iota' \in \mathbf{I},\ \sigma \in upT_{\mathbf{I}} \quad \Rightarrow (\sigma,(\iota),\iota') \in upT_{\mathbf{I}};$
6. $(\iota,\sigma) \in downT_{\mathbf{I}},\ (\sigma',\iota) \in upT_{\mathbf{I}} \Rightarrow (\sigma',\iota,\sigma) \in T_{\mathbf{I}};$
7. $\lambda \in upT_{\mathbf{I}},\ \rho \in downT_{\mathbf{I}} \quad \Rightarrow (\lambda,\rho) \in T_{\mathbf{I}};$
8. $0 \in T_{\mathbf{I}}$
9. $downT_{\mathbf{I}} \cup upT_{\mathbf{I}} \subseteq T_{\mathbf{I}}.$

(Note that in (6) either $\sigma = \lambda$ for some $\lambda \in upT_{\mathbf{I}}$ or $\sigma = (\iota'),\ \lambda$ for some $\iota' \in \mathbf{I}$, $\lambda \in upT_{\mathbf{I}}$. Similarly for σ'.) Each of these clauses corresponds to a kind of segment through an update tree. Note that we distinguish *down* trees and *up* trees:

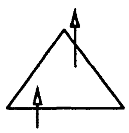
clauses (1)-(5) define these subclasses and in clauses (6)-(9) they are used to construct T_{I}. The down trees—cases (2) and (4)—are the segments that actually correspond to an update tree. These paths start at some root ι and then go down into the tree below that root. With these segments we can simply 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.

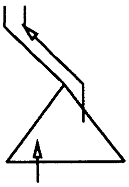

a generic down tree

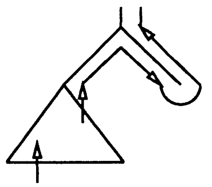

a down tree of the form (ι,σ)


a down tree of the form $(\iota,(\iota'),\sigma)$

The up trees—cases (3) and (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.


a generic up tree

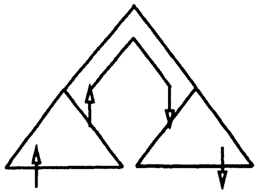

an up tree of the form (σ,ι)


an up tree of the form $(\sigma,(\iota'),\iota)$

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 complete 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 do 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.

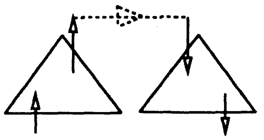


a tree of the form (λ, ρ)

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. Such a moment occurs, for example, when we are interpreting 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. For such a situation we use the following kind of picture:



a tree of the form (σ', ι, σ)

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. (In fact $(\top)$ plays the role of the empty tree.) **0** is just the opposite of the empty tree: 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 easier to think of them in terms of their basic components. We distinguish the following *basic trees*.

Definition 4.4 We distinguish the following basic trees in $T_{\mathbf{I}}$:

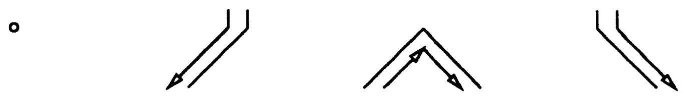
(ι) is a basic tree for each $\iota \in \mathbf{I}$. (Think of an atomic text '*p*'.)

$(\top,(\top))$ 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*'.)

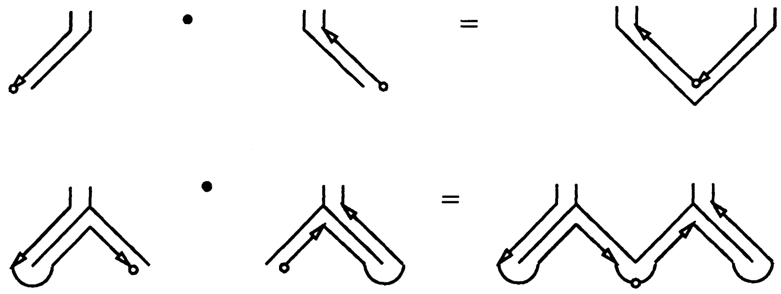
In a picture:



(ι) $(\top,(\top))$ $((\top),(\top))$ $((\top),\top)$

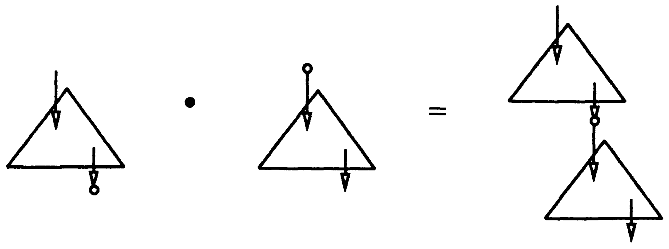
We can think of all tree segments in terms of these basic segments: big segments are obtained by glueing together these basic segments. Before we can make this precise, we have to explain *how* tree segments are glued together. This is the topic of the next section.

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 and 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. (It may help to compare these cases where the merger goes wrong with the cases where the update functions of the previous section were undefined.) Consider the following examples of such a situation. In the pictures we use • as notation for the merger and ∘ 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.



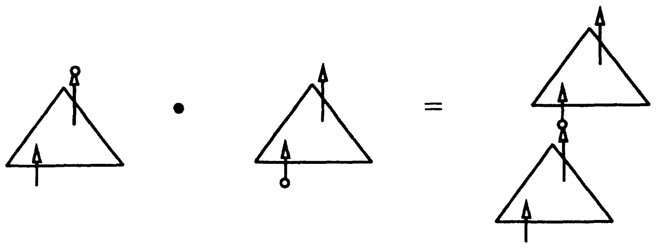
If we simply glue together the segments as indicated in the pictures, we get something which, although it makes sense geometrically, is useless in our set up. For it is clear that the result is not a segment of a (left-to-right, top-down) path through a binary tree. In these cases we reach the error 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 the result will be a downtree as well.

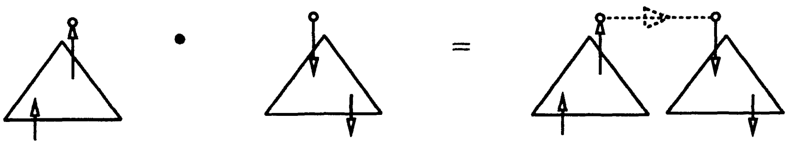


There are also cases where τ is not itself a down tree, but does look like a down tree at the point of contact. These cases—where τ is of one of the forms (σ', ι, σ) or (λ, ρ) (for a non-trivial tree ρ)—work similarly so we do not have to discuss them separately. This is why we will ignore this kind of situation in what follows. We can concentrate on what happens at the point of contact.

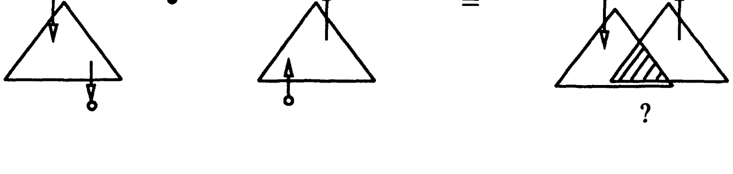
In the dual case, where two up trees meet, we cannot meet any problems either.



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 uptree, 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.



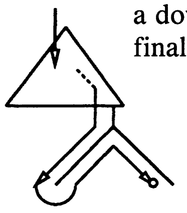
We have already seen situations where this goes wrong. In both examples it was easy to see in advance that something would go wrong, but in general this can be quite difficult. Fortunately we do not have to see it in advance. We can simply check it step by step, as we are performing the merger. In each 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. If we were dealing with down trees only, we would always find a root at the beginning of our paths. But since we also have up tree, there are more ways in which a path can start.

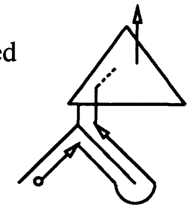
Definition 4.5 We define the function seg_f and seg_i on trees in T_{I} as follows:

$$\begin{array}{ll}
seg_f((\iota)) = (\iota), & seg_i((\iota)) = (\iota); \\
seg_f((\sigma,\iota)) = (\iota); & seg_i((\iota,\sigma)) = (\iota), \\
seg_f((\iota,(\iota'))) = ((\iota,(\iota'))); & seg_i(((\iota'),\iota)) = ((\iota'),\iota); \\
seg_f((\iota,\sigma)) = seg_f(\sigma), & seg_i((\sigma,\iota)) = seg_i(\sigma), \\
\quad \text{if } \sigma \neq (\iota'); & \quad \text{if } \sigma \neq (\iota'); \\
seg_f((\iota,(\iota'),(\iota''))) = (\iota,(\iota'),(\iota'')); & seg_i(((\iota''),(\iota'),\iota)) = ((\iota''),(\iota'),\iota); \\
seg_f((\iota,(\iota'),\sigma)) = seg_f(\sigma), & seg_i((\sigma,(\iota'),\iota)) = seg_i(\sigma), \\
\quad \text{if } \sigma \neq (\iota''); & \quad \text{if } \sigma \neq (\iota''); \\
seg_f((\sigma',\iota,\sigma)) = seg_f((\iota,\sigma)); & seg_i((\sigma',\iota,\sigma)) = seg_i((\sigma',\iota)); \\
seg_f((\lambda,(\iota))) = (\lambda,(\iota)); & seg_i(((\iota),\rho)) = ((\iota),\rho); \\
seg_f((\lambda,\rho)) = seg_f(\rho), & seg_i((\lambda,\rho)) = seg_i(\lambda), \\
\quad \text{if } \rho \neq (\iota); & \quad \text{if } \lambda \neq (\iota); \\
seg_f(0) = 0; & seg_i(0) = 0.
\end{array}$$

We will keep the same notation for final segments, writing $\tau\llbracket\rho\rrbracket$ 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: $\llbracket\lambda\rrbracket\tau$ and $\{\lambda'\}\tau$.

If we want to indicate the final or initial segment in a picture, this looks as follows:


a down tree with final segment indicated


an up tree with initial segment indicated

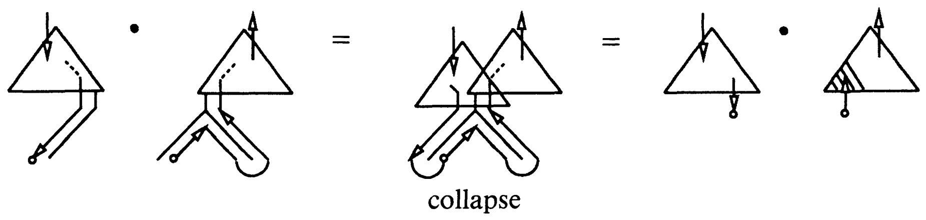
We need the final and the initial segment to keep track of shape of the path at its end and beginning, respectively. Note that up trees have a trivial final segment, of the form (ι), and that down trees have a trivial initial segment. Another interesting case are the trees of the form (λ,ρ), but we will not discuss this case until we need it.

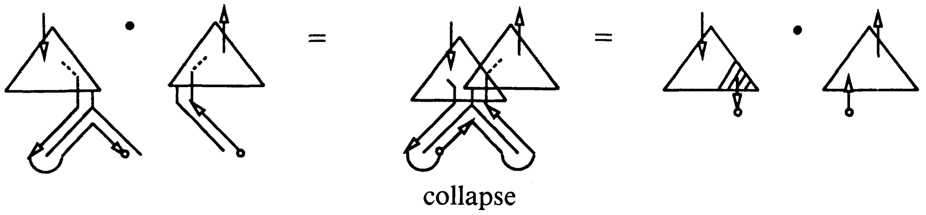
We have included a clause for **0** in the definition. Of course the final or initial segment of the undefined tree is not a particularly useful notion, but this way

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.

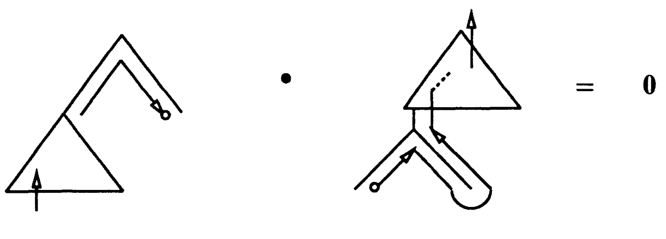




In the two cases indicated here we see that locally the paths match. The final segment and the initial segment together form a complete subtree. Now we can simply compute the information of this subtree, collapse the subtree itself and add the information at its root.[4] Then the two trees will have a new final and initial segment, and we can check again whether 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 (i.e., 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 *not* the case that a down tree and an up tree meet.

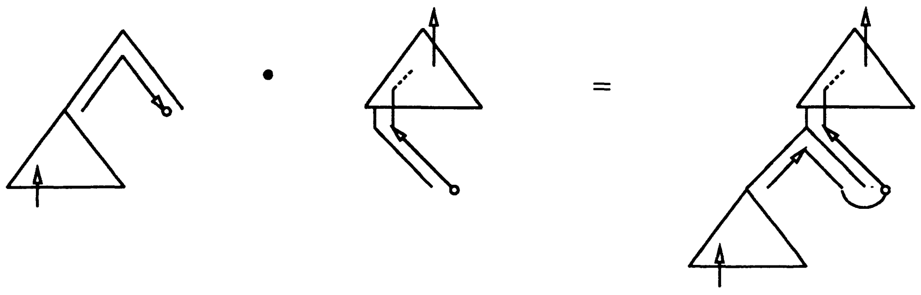
Before we make formal sense of this pictorial explanation, we have to consider one more case. This is the case where one (or two) of the trees is of the shape (λ, ρ). We already explained above that, if the first tree is of this form and in case ρ is not a trivial tree, (λ, ρ) behaves just like a down tree at the point of contact. But if $\rho = (\iota)$ for some $\iota \in \mathbf{I}$, then the bridge shape of (λ, ρ) is relevant. This explains the definition of the final segment: if $\rho = (\iota)$, 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 we also have the symmetrical situation, where the second tree is $((\iota),\rho)$, which 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. The reader may find it useful to look at the appropriate picture for each of the clauses.

Definition 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:

$$\tau \bullet 0 \overset{1}{=} 0 = 0 \bullet \tau'$$
$$\tau\llparenthesis(\iota_0)\rrparenthesis \bullet \llparenthesis(\iota_1)\rrparenthesis\tau'.$$

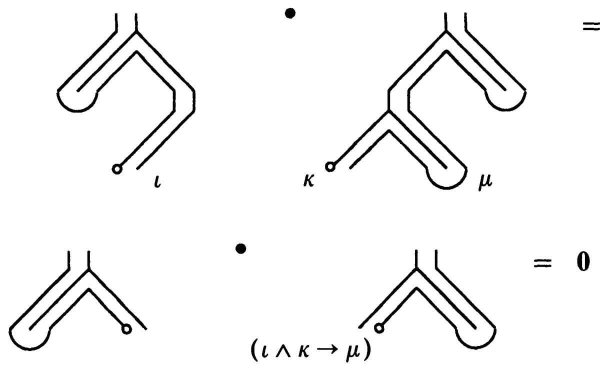
We distinguish four subcases:

$$(\iota_0) \bullet (\iota_1) \overset{a}{=} (\iota_0 \wedge \iota_1)$$
$$(\sigma,\iota_0) \bullet (\iota_1) \overset{b}{=} (\sigma,\iota_0 \wedge \iota_1)$$
$$(\iota_0) \bullet (\iota_1,\sigma') \overset{c}{=} (\iota_0 \wedge \iota_1,\sigma')$$
$$(\sigma,\iota_0) \bullet (\iota_1,\sigma') \overset{d}{=} (\sigma,\iota_0 \wedge \iota_1,\sigma')$$

$$\tau\llparenthesis(\iota_0,(\iota_1))\rrparenthesis \bullet \llparenthesis(\iota_2)\rrparenthesis\tau' \overset{3}{=} \tau\{(\iota_0,(\iota_1) \bullet \llparenthesis(\iota_2)\rrparenthesis\tau')\}$$
$$\tau\llparenthesis(\iota_0)\rrparenthesis \bullet \llparenthesis((\iota_1),\iota_2)\rrparenthesis\tau' \overset{4}{=} \{(\tau\llparenthesis(\iota_0)\rrparenthesis \bullet (\iota_1),\iota_2)\}\tau'$$
$$\tau\llparenthesis(\iota_0,(\iota_1),(\iota_2))\rrparenthesis \bullet \llparenthesis(\iota_3)\rrparenthesis\tau' \overset{5}{=} \tau\{(\iota_0,(\iota_1),(\iota_2) \bullet \llparenthesis(\iota_3)\rrparenthesis\tau')\}$$
$$\tau\llparenthesis(\iota_0)\rrparenthesis \bullet \llparenthesis((\iota_1),(\iota_2),\iota_3)\rrparenthesis\tau' \overset{6}{=} \{(\tau\llparenthesis(\iota_0)\rrparenthesis \bullet (\iota_1),(\iota_2),\iota_3)\}\tau'$$
$$\tau\llparenthesis(\iota_0,(\iota_1),(\iota_2))\rrparenthesis \bullet \llparenthesis((\iota_3),\iota_4)\rrparenthesis\tau' \overset{7}{=} (\tau\{(\iota_0)\} \bullet (\iota_1 \rightarrow \iota_2 \wedge \iota_3)) \bullet \{(\iota_4)\}\tau'$$
$$\tau\llparenthesis(\iota_0,(\iota_1))\rrparenthesis \bullet \llparenthesis((\iota_2),(\iota_3),\iota_4)\rrparenthesis\tau' \overset{8}{=} \tau\{(\iota_0)\} \bullet ((\iota_1 \wedge \iota_2 \rightarrow \iota_3) \bullet \{(\iota_4)\}\tau')$$
$$(\lambda,(\iota_0)) \bullet \llparenthesis(\iota_1)\rrparenthesis\tau' \overset{9}{=} (\lambda,(\iota_0) \bullet \llparenthesis(\iota_1)\rrparenthesis\tau')$$
$$\tau\llparenthesis(\iota_0)\rrparenthesis \bullet ((\iota_1),\rho) \overset{10}{=} (\tau\llparenthesis(\iota_0)\rrparenthesis \bullet (\iota_1),\rho)$$
$$(\lambda,(\iota_0)) \bullet \llparenthesis((\iota_1),\iota_2)\rrparenthesis\tau' \overset{11}{=} \{(\lambda,(\iota_0) \bullet (\iota_1),\iota_2)\}\tau'$$
$$\tau\llparenthesis(\iota_0,(\iota_1))\rrparenthesis \bullet ((\iota_2),\rho) \overset{12}{=} \tau\{(\iota_0,(\iota_1) \bullet (\iota_2),\rho)\}$$

$$\tau\llbracket(\iota_0,(\iota_1))\rrbracket \bullet \llbracket((\iota_2),\iota_3)\rrbracket\tau' \overset{13}{=} 0$$
$$\tau\llbracket(\iota_0,(\iota_1),(\iota_2))\rrbracket \bullet \llbracket((\iota_3),(\iota_4),\iota_5)\rrbracket\tau' \overset{14}{=} 0$$
$$\tau\llbracket(\iota_0,(\iota_1),(\iota_2))\rrbracket \bullet ((\iota_3),\rho) \overset{15}{=} 0$$
$$(\lambda,(\iota_0)) \bullet \llbracket((\iota_1),(\iota_2),\iota_3)\rrbracket\tau' \overset{16}{=} 0$$
$$(\lambda,(\iota_0)) \bullet ((\iota_1),\rho) \overset{17}{=} 0.$$

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 uptree are merged. In the first step the final and initial segment match. So a subtree is completed and the result, $(\iota \wedge \kappa \rightarrow \mu)$, 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, **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 4.7 (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 (ι) and that for $\tau \in upT_{\mathbf{I}}$, $seg_f(\tau)$ is of the form (ι).)

1. $(\iota) \in T_{\mathbf{I}}$ is a basic tree;
2. If σ is constructed from basic trees, then $(\iota) \bullet ((\top,(\top)) \bullet \sigma)$ gives a construction of (ι,σ) from basic trees.

3. If σ is constructed from basic trees, then $(\sigma \bullet ((\top),\top)) \bullet (\iota)$ gives a construction of (σ,ι) from basic trees.
4. If σ is constructed from basic trees, then $(\iota) \bullet ((\top,(\top)) \bullet ((\iota') \bullet (((\top),(\top)) \bullet \sigma)))$ gives the required construction of $(\iota,(\iota'),\sigma)$.
5. If σ is constructed from basic trees, then $(((\sigma \bullet ((\top),(\top))) \bullet (\iota')) \bullet ((\top),\top)) \bullet (\iota)$ gives the required construction of $(\sigma,(\iota'),\iota)$.
6. If σ' and σ are constructed from basic trees, then $(\sigma' \bullet (((\top),\top) \bullet ((\iota) \bullet ((\top,(\top)) \bullet \sigma))))$ gives the required construction of (σ',ι,σ).
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 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 semi-group. We find that, because of the generation lemma, the following result suffices to prove associativity.

Proposition 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: See appendix.

We can extend this associativity result as follows:

Proposition 4.9 (Full Associativity) *Let three trees $\tau_0, \tau_1, \tau_2 \in T_1$ 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:

$$\begin{aligned}
&(\tau_0 \bullet \tau_1) \bullet \tau_2 = \\
&(\tau_0 \bullet (\tau \bullet \beta)) \bullet \tau_2 && = \text{(by induction hypothesis on } \tau_1) \\
&((\tau_0 \bullet \tau) \bullet \beta) \bullet \tau_2 && = \text{(by induction hypothesis on } \tau_0 \bullet (\tau \bullet \beta)) \\
&(\tau_0 \bullet \tau) \bullet (\beta \bullet \tau_2) && = \text{(by induction hypothesis (for } n_1 = 1)) \\
&\tau_0 \bullet (\tau \bullet (\beta \bullet \tau_2)) && = \text{(by induction hypothesis (} \tau \text{ is smaller than } \tau_1!)) \\
&\tau_0 \bullet ((\tau \bullet \beta) \bullet \tau_2) && = \text{(by induction hypothesis on } \tau \bullet (\beta \bullet \tau_2)) \\
&\tau_0 \bullet (\tau_1 \bullet \tau_2).
\end{aligned}$$

This proves the proposition.

Now it is clear that the partial trees as we have defined them in this section form a semi-group. This means that the partial trees *may* provide a suitable setting for text semantics: in Section 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 really can be represented in this update algebra.

Proposition 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: if $\tau \bullet \tau' \in downT_{\mathbf{I}}$, then already $\tau \in T_{\mathbf{I}}$.

5 ***Trees and texts*** In Section 4 we have seen that texts can be interpreted as update functions on down trees and in Section 5 we have seen how trees form an update algebra. In this section we make the relation between the semantics of Section 4 and the trees of Section 5 precise. First we define the tree representation of a text.

Definition 5.1 For a text $\phi \in Text_A$ we define its tree representation $[\![\phi]\!] \in T_{\mathbf{I}}$.

$$[\![\bot]\!] = (\bot);$$

$$[\![p]\!] = (\iota_p);$$

$$[\![if]\!] = (\top, (\top));$$

$$[\![then]\!] = ((\top), (\top));$$

$$[\![end]\!] = ((\top), \top);$$

$$[\![\phi\psi]\!] = [\![\phi]\!] \bullet [\![\psi]\!].$$

Now we can check that this tree representation indeed generates the update functions from Section 4.

Proposition 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 3.3 with the corresponding clauses in definition 4.6, where we read **0** as undefined (or vice versa). 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 3. Thereby we also inherit the notion of truth from Section 3. The following corollary can even be seen as an explanation of the notion of truth we gave there: it turns out that texts that are true (in **I**) have $(\top)$ as representation.

Corollary 5.3 *Let a text $\phi \in Text_A$ be given. Then ϕ is true iff $[\![\phi]\!] = (\top)$.*

Proof: Recall that ϕ is true iff $(\top)[\phi] = (\top)$. Hence ϕ is true iff $(\top) \bullet [\![\phi]\!] = (\top)$ iff $[\![\phi]\!] = (\top)$.

With the tree representation of texts we have obtained a more refined test of well formedness: the grammatical texts have a trivial tree representation — (ι) for some $\iota \in \mathbf{I}$ — the coherent texts are precisely the texts that are not represented by **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.

6 Concluding remarks

The main conclusion of this paper is that an incremental semantics (of texts) is feasible, even if typically non-associative phenomena occur. The general strategy to deal with these phenomena is to exchange non-associativity for structured memory.

We have introduced a non-trivial kind of structure on the memory slots in order to be able to distinguish semantically the contribution of the different kinds of items that we store in the slots, in our case assumptions and conclusions. As a result we have obtained *trees as texts*. In the general case other, more complex, structures will probably be called for, but the strategy of using structured memory will still work.

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,[5] 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[6]: our text language simply might not be rich enough.

We have used *binary* trees to represent slots in memory. 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.[7] For example, we tend to use *intermediary* 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 intermediary 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 causal connection between these events?

There is another point to be made about the kind of structure that we use. It is clear that we use binary trees (or rather we use segments of depth first pathways through binary trees), but not all binary trees occur in our semantics. This is a consequence of the strategy to compute the content of an implication as soon as this is possible. It is because of this strategy that our structures remain relatively simple. But maybe this is an unjustified simplification: maybe there is reason to distinguish the step of actually computing the content of an implication from the step of simply interpreting it. The step where we distroy the *if*... *then* structure and compute its information content in the Heyting algebra could be postponed for a while. Then we can remember the structure of the text explicitly until we choose to perform that computation.[8] It is not difficult to adapt our semantics to this effect.

We have already made a remark about the connection of our work with *Discourse Representation Theory*, *DRT* for short: instead of propositional texts we could also use basic *Discourse Representation Structures* (*DRSs*). Thus we would obtain a formulation of *DRT* which satisfies the constraints mentioned in the introduction. Also in Kamp's formulation tree structures (*DRSs*) are introduced for the interpretation of texts. But the way in which they are handled is not very elegant. Complete *if*... *then* structures—subtrees—are treated as *conditions*, the kind of thing that can only be added to a *DRS* in one sweep. This is a clear disadvantage if we also want to consider *if*... *then* structures that are stretched out over several sentences. It also is a rather counterintuitive procedure for the interpretation of (deeply) nested *if*... *then* constructions. So our tree algebras can be used to improve *DRT* in this respect.

There is also a relation between this work and the work of Visser[9]: he also tries to obtain an incremental semantics by using structured semantics. Instead of trees he uses *stacking cells* as a way to represent structure in the semantics. Stacking cells are in a sense quite similar to our trees. A stacking cell consists of a number of *pop*-levels, a stem, and a number of *push*-levels. So the general format is (*pop*, *stem*, *push*). This corresponds to the general format (σ', ι, σ) of the trees in T_{I}: the *up tree* σ' corresponds to the *pop* in a stacking cell, the *root* ι corresponds to the *stem* and the *down tree* σ corresponds to the *push*-levels. Our structures are less elegant because we allow for a split in each level, splitting it in an *if* and a *then* part. This results in a nesting of levels which makes the interaction between trees more complicated than the interaction between Visser's stacking cells. Still it can be shown that the partial tree structure can be simulated with a special kind of stacking cells.

Finally a remark about other kinds of texts. The texts in this paper 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 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. This nesting is not unlike the nesting of different kinds of modalities, each of which introduces us into a different kind of situation in which different kinds of laws rule. Each of these kinds of texts has features that are crucial for the interpretation of the text and the sentences of which it is made up.

Appendix We use this appendix to present the proof of the basic associativity result (proposition 4.8) that is essential for the associativity of •. The notation is as in Section 4.

Proposition A.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: $seg_f(\tau \bullet \beta) = seg_f(\rho \bullet \beta)$ and $seg_i(\beta \bullet \tau') = seg_i(\beta \bullet \lambda)$.
(This includes the case where one of λ, ρ equal $\mathbf{0}$.) 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

$$\rho = (\iota_0, (\iota_1), (\iota_2)) \text{ and } \beta = ((\top), \top)$$

or

$$\rho = (\iota_0, (\iota_1)) \text{ and } \beta = ((\top), (\top), \top).$$

We will discuss the first case. The second one is handled analogously. So $\tau \bullet \beta = \tau\{(\iota_0)\} \bullet (\iota_1 \rightarrow \iota_2)$. 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 $seg_i(\beta \bullet \tau') = ((\top), \top \wedge \iota_\lambda) = \beta \bullet (\iota_\lambda)$, where ι_λ is the leftmost node of λ. (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.) This gives us:

$$(\tau \bullet \beta) \bullet \tau' = (\tau\{(\iota)\} \bullet (\iota' \rightarrow \iota'')) \bullet \tau'$$

and

$$\tau \bullet (\beta \bullet \tau') = \tau \bullet ⦃((\top), \iota_\lambda)⦄\beta \bullet \tau' = (\tau\{(\iota)\} \bullet (\iota' \rightarrow \iota'')) \bullet \{(\iota_\lambda)\}(\beta \bullet \tau').$$

Since $seg_i(\beta \bullet \tau') = \beta \bullet (\iota_\lambda)$, we see that $\{(\iota_\lambda)\}(\beta \bullet \tau') = \tau'$. 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 of $\rho_i \bullet \beta_k \bullet \lambda_j$. (We skip the really trivial cases where $\rho_i, \lambda_j = \mathbf{0}$.)

	$\rho = seg_f(\tau)$	β	$\lambda = seg_i(\tau')$
(1)	(ι_0)	(ι)	(ι'_0)
(2)	$(\iota_0, (\iota_1))$	$(\top, (\top))$	$((\iota'_1, \iota'_0)$
(3)	$(\iota_0, (\iota_1), (\iota_2))$	$((\top), \top)$	$((\iota'_2), (\iota'_1), \iota'_0)$
(4)	$(\lambda, (\iota_0))$	$((\top), (\top))$	$((\iota'_0), \rho)$

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 case $\rho_i \bullet \beta_3 \bullet \lambda_j$ 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 uptree (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.

$\rho_2 \bullet \beta_1 \bullet \lambda_3$	$(\iota_0,(\iota_1))$	(ι)	$((\iota'_2),(\iota'_1),\iota'_0)$
$\rho_2 \bullet \beta_1 \bullet \lambda_4$	$(\iota_0,(\iota_1))$	(ι)	$((\iota'_0),\rho)$
$\rho_1 \bullet \beta_2 \bullet \lambda_3$	(ι_0)	$(\top,(\top))$	$((\iota'_2),(\iota'_1),\iota'_0)$
$\rho_2 \bullet \beta_2 \bullet \lambda_3$	$(\iota_0,(\iota_1))$	$(\top,(\top))$	$((\iota'_2,(\iota'_1),\iota'_0)$
$\rho_3 \bullet \beta_2 \bullet \lambda_3$	$(\iota_0,(\iota_1),(\iota_2))$	$(\top,(\top))$	$((\iota'_2),(\iota'_1),\iota'_0)$
$\rho_4 \bullet \beta_2 \bullet \lambda_3$	$(\lambda,(\iota_0))$	$(\top,(\top))$	$((\iota'_2,(\iota'_1),\iota'_0)$
$\rho_1 \bullet \beta_2 \bullet \lambda_4$	(ι_0)	$(\top,(\top))$	$((\iota'_0),\rho)$
$\rho_2 \bullet \beta_2 \bullet \lambda_4$	$(\iota_0,(\iota_1))$	$(\top,(\top))$	$((\iota'_0),\rho)$
$\rho_3 \bullet \beta_2 \bullet \lambda_4$	$(\iota_0,(\iota_1),(\iota_3))$	$(\top,(\top))$	$((\iota'_0),\rho)$
$\rho_4 \bullet \beta_2 \bullet \lambda_4$	$(\lambda,(\iota_0))$	$(\top,(\top))$	$((\iota'_0),\rho)$
$\rho_1 \bullet \beta_4 \bullet \lambda_1$	(ι_0)	$((\top),(\top))$	(ι'_0)
$\rho_1 \bullet \beta_4 \bullet \lambda_2$	(ι_0)	$((\top),(\top))$	$((\iota'_1),\iota'_0)$

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:

$$(\iota_0,(\iota_1)) \bullet ((\iota_2),(\iota_3)) \rightsquigarrow (\iota_0,(\iota_1 \wedge \iota_2),(\iota_3)).$$

Now the proof of the associativity would follow from two observations about the rewriting system *Termtree*:

- The normal forms of *Termtree* are exactly the partial trees of T_{I};
- *Termtree* enjoys strong normalization.

Then we would know that all 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 normalization proof tedious.

NOTES

1. We like to call this view on proofs the *proofs as texts* perspective. Together with the *texts as trees* perspective that will be developed in this paper, we get the slogan: *proofs as texts as trees.*

2. 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.*

3. Note that here two different notations are necessary because we do not have, in general, $seg_f(\sigma\{\rho'/\rho\}) = \rho'$. Take for example $\sigma = (\iota,(\iota'),(\top,(\top)))$ and $\rho' = (\iota'')$. Then $seg_f(\sigma) = (\top,(\top))$ and $seg_f(\sigma\{\rho'/(\top,(\top))\}) = (\iota,(\iota'),(\iota''))$.

4. We have to choose 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.

5. The fact that most static notions of meaning give rise to update functions is also noticed by Van Benthem [1].

6. Although the idea is already implicit in Visser [8], 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.

7. In Zeinstra [11] an attempt is made to work with a more flexible language. She also makes an attempt at an incremental semantics.

8. I would like to thank Marcus Kracht for discussion on this point. Kracht [5] defends a similar distinction for other connectives as well.

9. In fact the author is not only influenced by Visser [8],[9],[7], in text, but also by Albert Visser, in person. He has made many useful suggestions and has asked many useful questions.

REFERENCES

[1] Van Benthem, J., "General Dynamics," *Theoretical Linguistics*, vol. 17 (1991), pp. 159-201.

[2] Groenendijk, J., and M. Stokhof, "Dynamic Predicate Logic," *Linguistics and Philosophy*, vol. 14 (1991), pp. 39-100.

[3] Heim, I., "File change semantics and the familiarity theory of definites," pp. 164-189 in *Meaning, use and interpretation of language*, edited by R. Bauërle *et al.*, Walter de Gruyter, Berlin, 1983.

[4] Kamp, H., "A theory of truth and semantic representation," pp. 277-322 in *Formal Methods in the Study of Language*, edited by J. Groenendijk *et al.*, Mathematisch Centrum, Amsterdam, 1981.

[5] Kracht, M., "When can you say 'it'," pp. 1-30 in Technical report II, Mathematisches Institut, Freie Universität Berlin, 1989.

[6] Veltman, F., "Defaults in update semantics," pp. 1-57 in Technical report LP-91-02, Institute for Logic, Language, and Computation, University of Amsterdam, 1991.

[7] Visser, A., "Meanings in time," manuscript.

[8] Visser, A., "Actions under presupposition," Logic group Preprint Series 76, Philosophy Department, Utrecht University, 1992.

[9] Visser, A., *Lazy and quarrelsome brackets*, Logic group Preprint Series 82, Utrecht, 1992.

[10] Zeevat, H., "A compositional version of discourse representation theory," *Linguistics and Philosophy*, vol. 12 (1989), pp. 95-131.

[11] Zeinstra, L., *Reading as Discourse*, Master's thesis, Philosophy Department, Utrecht University, 1990.

Research Institute for Language and Speech (O.T.S.)
Arts Department, Utrecht University, Trans 10
3512 JK Utrecht, The Netherlands
e-mail: Kees.Vermeulen@let.ruu.nl