Parsing Traffic Accident Reports with Dynamic Syntax

W.Meyer-Viol

Department of Philosophy, King's College London

meyervio@dcs.kcl.ac.uk

Abstract

In this report the formal background is formulated for the application of Dynamic Syntax (6) to the parsing of free text traffic accident reports. It accompanies a first implementation of the approach using Active Database technology (8). Dynamic syntax seems to be ideally suited for this task because it concentrates on non-mediated transformation of a natural language string into a meaning representation or database structure. That is, there is no requirement to first assign syntactic structure to a string in order to determine its semantics. Dynamic syntax gets around the need for syntactic analysis by using a notion of logical form where structure and content can be addressed seperatedly and constructed piecemeal. The background grammar as well as the words of a language are viewed procedurely as transformations that map a partial logical form to an enriched one.

Contents

1 Introduction

Many interesting practical tasks in natural language processing require the *translation* of a natural language report about a given topic or knowledge domain — either in writing or in oral form — into data structures representing the knowledge domain in question, for the use by independent processes. The natural language side of such a task generally involves a quite regimented fragment of the language in which it is formulated. For instance, in many kinds of reports all sentences will be interpretated declaratively; that is, the question form will be wholly lacking and no declarative form will need to be interpreted as a question. Moreover, ideosyncratic abbreviations and constructions will generally abound in such reports.

On the other end of such a task, the data structures are generally wholly geared towards the specific task involved. That is, the *signature* of the representation language will be supplied by the task and thus the class of 'meaningful' formulas in this language. In general, the data structures generated by the natural language reports will interact with information from other sources. This interaction may involve choice in the representation signature not directly reflected in the natural language.

So, a practical natural language task consists of a pairing of an NL-fragment and a set of data structures (to be used in some independent, encompassing process) through a mapping which assigns to each item on the NL side an object on the data structure side. These two poles of the task, the natural language reports and the representational data structures or 'objects' are guaranteed to be connected: a natural language sentence is guaranteed to deal with the subject matter of the task. This makes a *goal-directed* approach viable.

In a parsing task of the kind described we are generally not interested in the syntactic structure except as a means to get at the 'meaning', and this is the only reason that syntactic analysis is involved. In the standard approach, the NL items are first translated into syntactic trees and then into semantic trees. Because the first part has no interest of its own the question is: can this not be done in a more efficient, straightforward way? **Dynamic syntax**, (see (3),(4),(5),(6),(7)), is an attempt at such a direct approach. Dynamic Syntax disposes of syntactic analysis as an intermediate stage and uses the natural language to create the representational objects directly. It does this by interpreting the words as instructions, or programs, to extend a given *partial* data structure with new features. Each word is thus 'executed' in the context of a given partial data structure, in general the product of executing other (previous) words within context. If the words of a sentence form a *grammatical* string, then executing them all (in some sequence) will result in a data structure that is in some sense *complete.*

The words in a sentence correspond to program actions on the partial data structures, but the order in which the words are offered in the reports need not correspond to the order in which the corresponding programs are executed. It may, for instance, be fruitful to start with the verb-program, or to execute the word-programs from right to left. Other tasks however, especially those involving 'on line' processing, seem most naturally to involve a left to right sequencing if the programs.

This report describes the formal details behind an application of Dynamic Syntax to one such translation task, that of parsing **traffic accident reports** based on the data described in ((1),(2)). This report has been the basis for a first implementation of the approach presented here (8), which has encoded the translation task using *event-condition-action rules* and in particular Oracle 8 triggers.

Traffic accident reports (TARs) are written by police officers at the scene of the accident and give information about the accident such as its location, time, conditions, participants, and events. These TARs have two parts: a standardized form, whose format is the same for every report, and a free-text part where the officer gives further particulars of the accident in brief and abbreviated form. One of the main aims of automating the parsing of the free-text part of the reports is to verify the inforamtion of the standardized form with respect to the free text, and to supplement it with additional information. The free text parts are typically brief, at most two or three sentences, highly abbreviated and often ungrammatical in an ideosyncratic way.
Here is a typical entry

PDN CRSSNG FRM NS BHND PKD V1, V1 RVRS HTNG PDN.

In expanded, more understandable form, this report says:

PEDESTRIAN CROSSING FROM NORTH SIDE BEHIND PARKED V1, V1 REVERSES HITTING PEDESTRIAN.

V1 denotes here 'vehicle 1'; other vehicles participating in the accident would be denoted V2, V3 etc. Typical in these reports is the use of the gerund 'CROSSING' instead of a main tensed verb 'CROSSES', a construction which is ungrammatical according to general English grammar. Now the above (two) sentences have to be **parsed**, that is a representation of their 'meaning' has to be constructed. In our approach currently, these meanings have the form of expressions in a **typed lambda calculus**. In a future formalization we intend to make the representation closer to the structure of the standardized information accompanying the free-text report. The parsing of a sentence proceeds word-by-word from left-to-right. After having worked partly through the sentence we may have more than one tentative analysis (e.g. PEDESTRIAN CROSSING — is that a pedestrian who is crossing, or is it a location where people may cross the street ?), so the various alternatives have to be maintained until they drop out (due to the fact that they can no longer accomodate the rest of the words).
Having processed the last word of a TAR string which is *grammatical* (within the confines of the TAR translation task!) the parse will have left us with one or more logical representations of the meaning of the TAR string.

In the following sections the formal apparatus underlying this parsing process will be described. The next section will give an overview of the **parsing task** as it is structured in Dynamic Syntax. This will lead to a formal definition of a **Task Model** in Section 3 as consisting of three componponents. The next sections will then discuss two of these three components. Section 4 will describe the first of these components, the representation structures. Thse will be **linked, typed lambda terms** represented as **decorated linked binary trees**. In Section 5 the second component will be described, the transformations that map the TAR string to a representational structure. Section 6 discusses the overall evaluation loop of the parser. Section 7 gives our conclusions about the effectiveness of the Dynamic Syntax approach with respect to parsing of TARs.

2 The Parsing Task in Dynamic Syntax

Within Dynamic Syntax a task like the transformation of TARs into traffic-situation representations requires an analysis of three components.

1. **Representation Language** of the task. In the case of TARs, the appropriate family of basic relation and function symbols (expressed in typed lambda terms) must be able to describe city maps, temporal and causal information, etc. On the basis of this vocabulary, a (typed) lambda calculus is defined. The situation-representations are the fully reduced terms of this language (normal forms). The set of unreduced (but reducible) lambda terms of this language are in one-one correspondence with a set, $L\lambda T$ (Linked λ-Terms), of decorated linked binary trees. This set uniquely determines a set, PT, of *partial* decorated trees (namely all those partial trees that can be homomorhically embedded in some comlete tree). So a task (like a TAR) contributes the pair $(PT, L\lambda T)$ where PT is the non-empty set of data structures, the *domain*, and $L\lambda T \subset PT$, the set of complete logical forms, meaning representations for the TAR

2. **Natural Language** of the task. Because of the nature of translation tasks, both the vocabulary and the linguistic construction of a Parsing Task may be limited and idiosyncratic with respect to the language in general. A full sepcification of the task requires a **Presentation mode** of the natural language string. This deals with the *order* in which the natural language words are associated with the transformations on giving the eventual representation.

3. **From Natural to Representation Language.** The represenation and the natural language side of the task must supply two sets of *actions* guided by these specificities. These actions are structure- and decoration-preserving partial functions on the *domain* PT. The first set, **Words**, are partial transformations associated with the *words* of the natural language: a word extends a partial decorated tree to give a new one.
The second set, **Background**, consists of transformation not connected to the words of the natural language but associated more with TAR specificities, some of which it will share with English in general and some that are particular to these kinds of reports (for instance, we don't need rules for English questions).

Representation Language of the Task

In order to handle partiality of logical forms in a flexible way, we represent them as **decorated finite trees**. Logical forms are built up by putting together basic components of the representation(al language) by means of `APL`, the operation of **function application** in a typed lambda calculus. The sentence "*Pedestrian crossed the street*", represented by the formula $\mathbf{Crosses}(\mathbf{pedes'n}, \mathbf{the}(x, \mathbf{street}(x)))$, can be seen as resulting from the **unreduced lambda term**

$$\mathtt{APL}(\mathtt{APL}(\lambda x \lambda y \mathbf{Crosses}(y)(x), \mathtt{APL}(\lambda P(\mathbf{the}P), (x, \mathbf{street}x)), \mathbf{pedes'n})$$

by β-reduction. In Figure 1 we have represented this term as a binary tree structure with an associated set of node decorations. The prime constituents of the lambda term derive from contributions of the individual words

$$\underbrace{[_0[_0\ ]\ [_1[_1\ ]\ [_1\ [_0\ ][_1\ ]]]]}_{Tree}$$

$$\underbrace{\{00{:}\mathbf{pedes'n},\ 011{:}\lambda x\lambda y\mathbf{Crosses(x)(y)},\ 0111{:}\lambda P(\mathbf{the}P),\ 0110{:}(x,\mathbf{street}x)\}}_{Decorations}.$$

Figure 1: A lambda term as a decorated binary tree

By representing terms in this way, we can address term *structure* and term *content* separatedly. In particular, we can represent partially specified trees that are only partly decorated, a necessary requirement if the left-to-right presentation of the natural language words is to monotonically construct the logical form.

In fact the natural language sentences we consider are built up of **clauses**. "*Pedestrian crossed the street*" is one such clause but the sentence "*Pedestrian who was looking the other way crossed the street*" consists of two clauses: "Pedestrian crossed the street" and the relative clause expressing *Pedestrian was looking the other way.* This is represented by a pair of **linked lambda terms** sharing the argument '**pedes'n**'.

As each lambda term is represented as a decorated binary tree, linked lambda terms will be represented by finite *partially ordered* sets such trees, where the partial order is constitued by a directed link relation between the 'head' node in one tree, the one decorated by **pedes'n**, and the root node of a second tree, where this second tree is decorated at some terminal node by the decoration of the head node.

Constructing Decorated Trees

We are going to consider the words of a clause as instructions to construct a certain part of the lambda term represented as decorated *partial* tree. Now, if the left-to-right order of a sentence could induce a construction of the linked trees in some top-down or bottom-up fashion then matters would be simple (and uninteresting): we could just 'grow' our trees by adding to the frontier. English sentences (or its reflection in traffic accident reports) however may require more complicated processing.

1. The left-to-right order of a sentence may supply arguments *before* their position in the tree can be created: In the sentence "*V1 which ... killed pedestrian.*" the location of '*V1*' is not yet fixed in the relative clause: "*which hit V2*" or "*which V2 hit*" are both possible. Generally this does not mean that it is completely undetermined where an argument may find a position: an argument may come with the information in what *area* of the tree it must find its place. Tree 'areas' are always defined with respect to a given node.[1] We will introduce **partial trees** where such nodes with only 'underdetermined' tree-address — which specifies an *area* instead of a *node* — live.

2. The processing may deliver functors *after* their position has been created and occupied. In the sentence "*V1 drove ...*" the verb may be modified to "*V1 drove too fast...*"; now '*drove too fast*' supplies the functor at the tree node that the '*drove*'-functor would occupy if '*drove*' would occur unmodified in the sentence. That is, nodes may have to interposed

[1]The daughters of a given node a form an area, also the nodes dominated by a given node, or all nodes that can be reached by following only left daughters, etc.

between existing ones. But this would mean that tree relation may be broken up and trees restructured in the course of the parse. This is a very undesirable situation.[2] In order to prevent the need for restructuring, a functor will be introduced in a decorated tree as occupying (only) an *area* in the tree, that is, a node in a **partial tree**.

On the space of *complete* decorated binary trees the strings of words in accident reports cannot be represented as strings of structure-preserving transformations, but when the objects involved are *partial* tree structures then the strings can be represented as tree- and decoration-*homomorphisms*. That is, the decorated-tree transformations associated with the words can be defined as respecting partial tree structure while nevertheless accomodating the various orders in which the sentence string can offer the arguments and functors of the lambda term under construction.

Being homomorphisms, these transformation can only *increase* structure, and this feature of the transformation is preserved for all mappings that we can define in terms of these transformations using operations like composition, iteration, union or intersection.

Natural Language of the Task

The Natural Language side of the task supplies a characteristic set of sentence structures. This set is characteristic in two ways: first the words that build up the sentences may form an idiosyncratic subset of the words of the language in general and secondly, the sentence structures themselves may, on the one hand, not exhibit the same variation as the language as a whole (e.g., the question form may be lacking completely) but, on the other hand, they may be ungrammatical from the perspective of the general language (e.g., sentences without subject or main verb are quit common in TAR). The Natural Language side of the task is divided into the handling of the actual words and sentences, and their association with the actions on the partial logical forms.

3 The Task Model

Given a natural language fragment **NL** (like the language of TARs) and a representation language for **NL** in terms of a set of dcorated linked binary trees **LBT**, a parsing model is fully determined by a triple

$$\mathcal{B} = ((\mathbf{W}, \mathbf{NL}, \delta, \hat{\delta}), (\mathbf{PT}, \mathbf{LBT}), (\mathbf{Background}, \mathbf{Words}, \mathbf{Start}))$$

where

1. $(\mathbf{W}, \mathbf{NL}, \delta, \hat{\delta})$ is a quadruple consisting of a set natural language words $\mathbf{W}$, a specific corpus of task sentences $\mathbf{NL} \subset \mathbf{W}^*$, a lexicon $\delta : \mathbf{W} \mapsto \mathbf{Word}$ and a presentation order $\hat{\delta} : \mathbf{NL} \mapsto \mathbf{Word}^*$.

[2] Undesirable because it means that the information that can be gathered from the partial stages of the parse is not *persistent*; for instance, the information can only be 'passed on' with the qualification "this only holds if nothing has changed in the tree in which it has been generated". Every receiver of the information will have to check this. This is not the case for persistent information; this can be given to the environment as independent of the state of its source.

A sentence or clause of TAR can be associated with a sequence of **Word** actions in a left-to-right order, in a right-to-left order, a 'head-first- ordering, etc. This presentation order is captured by a function $\hat{\delta} : NL \mapsto \mathbf{Word}^*$ defined on the basis of the lexicon $\delta : W \mapsto \mathbf{Word}$ by the stipulation: for all $w \in NL \subset W^*$:

- $|\hat{\delta}(w)| = |w|$
- $\forall i \leq |w| \exists j \leq |w| : \delta(w(i)) = (\hat{\delta}(w))(j)$

where $|w|$ gives the length of string w. That is, the mapping $\hat{\delta}$ may permute the word-order in a TAR sentence at the **Word** level. Note that if for all i: $\delta(w(i)) = (\hat{\delta}(w))(i)$, then the presentation order is left-to-right.

2. $(\mathbf{PT}, \mathbf{LBT})$ is a pair consisting of a set **PT** of partial decorated linked binary trees and a set of complete decorated linked binary trees $\mathbf{LBT} \subset \mathbf{PT}$, embodying a representation language for **NL**. The partial structures in **PT** are partially ordered by the 'growth' or 'development' relation $\leq$ where the elements of **LBT** constitute maxinal elements.

3. $(\mathbf{Background}, \mathbf{Words}, \mathbf{Start})$ where $\mathbf{Background}, \mathbf{Words} \subseteq \mathbf{PT} \times \mathbf{PT}$ are *actions*, partial functions, on the partial logical forms, and $\mathbf{Start} \in \mathbf{PT}$ is some initial partial tree for the structure and decoration preserving partial functions in **Background** and **Words** to work on.
   There will be only a *finite*, usually quite small, number of alternatives generated in **Background**. To express this, let $CL(\mathbf{Background})$ consist of the set **Background** closed under function composition, union and iteration, and for a given transformation f let $\overline{f}(n) = \{f'(f(n)) \mid f' \in CL(\mathbf{Background})\}$. That is, $\overline{f}$ closes off f under all possible applications of **Background** transformations; $\overline{f}(n)$ contains all elements of PT that can be reached from $f(n)$ by applying **Background** transformations.. The various functions in the **Background** sharing a domain element can be seen as parsing *alternatives*, various possible structural analyses of the language string up to the current word. Now we certainly want the number of possible alternative analyses to be finite at every moment. This we express by
   $$\overline{\mathbf{1}}(n) \quad \text{is finite for every } n \in PT.$$
   where $\mathbf{1}$ is the Identity action. The effect of a word $f \in \mathbf{Word}$ on a given partial tree n will be measured in terms of $\overline{f}(n)$, that is, apply the action f associated with the word to the tree n and then close off the result under the transformations from **Background**. This gives a finite number of new trees.

With this definition of a Task Model for Dynamic Syntax, the fundamental notions of **grammaticality** and **expressive adequacy** can be formulated.

Let $\mathcal{B}$ be a Task Model.

$$\mathcal{B} = ((\mathbf{W}, \mathbf{NL}, \delta, \hat{\delta}), (\mathbf{Start}, \mathbf{PT}, \mathbf{LBT}), (\mathbf{Background}, \mathbf{Words})).$$

A string $w = e_1, \ldots, e_n \in \mathbf{W}^*$ is $\mathcal{B}$-**grammatical** if

$$\overline{(\hat{\delta}(w))_1} \cdot \ldots \cdot \overline{(\hat{\delta}(w))_n}(\mathbf{Start}) \cap \mathbf{LBT} \neq \emptyset.$$

where $\overline{f_1} \cdot \overline{f_2} = \{f_1' \cdot f_2' \mid f_1' \in \overline{f_1}, f_2' \in \overline{f_2}\}$, and $\overline{f}(n) = \{f'(n) \mid f' \in \overline{f}\}$.

Task Model $\mathcal{B}$ is **expressively adequate** if for every string $w = e_1, \ldots, e_n \in \mathbf{W}^*$

$$w \in NL \iff \overline{(\hat{\delta}(w))_1} \cdot \ldots \cdot \overline{(\hat{\delta}(w))_n}(\mathbf{Start}) \cap \mathbf{LBT} \neq \emptyset$$

Thus $\mathcal{B}$ is expressively adequate if the strings of words (in $\mathbf{W}^*$) that lead to a complete decorated linked binary tree are exactly the strings in **NL**

4 Representation Structures $(\mathbf{PT}, \mathbf{LBT})$

In this Section the details of the representation of the typed lambda terms will be supplied. Terms of a typed lambda calculus are represented as **decorated linked binary trees**. We will first describe the bare tree structure (Section 4.1) and subsequently introduce the decorations (Section 4.2) and the language to talk about them (Section 4.4).

4.1 Tree Structures

Tree structures form a special class of T-structures.

Definition 1 (T-Structures) A T-*structure* is a tuple of the form $\mathcal{T} = \langle T, =, \{\prec_i \subseteq T \times T \mid i \in I\}\rangle$ where T is a non-empty set, $=$ is an equivalence relation on T, and each $\prec_i$ is a (possibly empty) binary relation on T such that T is *connected*; that is, for every $n \in T$ if there is an $n' \in T, n \neq n'$, then there is some $n' \in T, n \neq n'$ and some $i \in I : (n, n') \in \prec_i$ or $(n', n) \in \prec_i$.
If $n, n' \in T$ such that $(n, n') \in \prec_i, =$, we will write $\mathcal{T}n \prec_i \mathcal{T}n'$, $\mathcal{T}n = \mathcal{T}n'$.

The set I is the finite set

$$I = \{0, 1, \downarrow, *, L\},$$

of **relation indices**. Given the basic relations indices in I, let $I^{-1} = \{i^{-1} \mid i \in I\}$, let I^k be the set of all sequences of length k of elements from I, and let I^* be the set of all finite sequences (including the empty sequence) of I-elements. For $j \in I^k$ we define the relation $\prec_j$ as $\prec_{j(1)} \circ \ldots \circ \prec_{j(k)})^3$. For $j \in I^{-1}$, say $j = i^{-1}$, we define the relation $\prec_j$ by $\mathcal{T}n \prec_j \mathcal{T}n' \equiv_{df} \mathcal{T}n' \prec_i \mathcal{T}n$.

T-structures come with structure preserving mappings.

Definition 2 (T-morphisms) For every $J \subseteq I$, function f is a J-*morphism* from T-structure $\mathcal{T}$ in T-structure $\mathcal{T}'$ if it maps T in T' such that for all $n, m \in T$, $\mathcal{T}m = \mathcal{T}n \Rightarrow \mathcal{T}'f(m) = \mathcal{T}'f(n)$ and for all $i \in J$:$\mathcal{T}n \prec_i \mathcal{T}m \Rightarrow \mathcal{T}'f(n) \prec_i \mathcal{T}'f(m)$.
An I-morphism, that is, a mapping respecting *all* relations of a T-structure, will be called a T-morphism.
We will say

$$\mathcal{T} \leq_J \mathcal{T}'$$

[3] where $n \prec_0 \circ \prec_\downarrow n'$, for instance, means that there is some n'' such that $n \prec_0 n''$ & $n'' \prec_\downarrow n'$.

if there is some J-morphism mapping T to T'. We will use $\leq$ to denote the finest relation $\leq_I$ ($J \subset J'$ entails $\leq_{J'} \subseteq \leq_J$).

The $\leq$-minimal elements, the *atoms*, are T-structures with a singleton domain and all relations empty. The relation is moreover *discrete*, and notice that, because of connectedness, a 'minimal' extension cannot consist of merely adding one element to the domain, it must also add a pair to one of the relations. For the class of T-structures we are interested in as underlying logical forms the relation indices in I are not independent.

Definition 3 (Linked Binary Trees) The set LBT of **linked binary trees** consists of those T-structures which are ordered as *binary trees*, where $\prec_0$ is first-daughter and $\prec_1$ the second-daughter relation, $\prec_\downarrow$ is the immediate dominance relation ($\prec_\downarrow = \prec_0 \cup \prec_1$), $\prec_*$ is the dominance relation, i.e., the reflexive and transitive closure of $\prec_\downarrow$, and $\prec_L$ is the so-called link-relation which connects a node of one tree to the root node of another. The binary relation $=$ is an equivalence relation (i.e. reflexive, transitive and symmetric) satisfying a condition $\#_1$ defined below.
The linked binary trees underly our target structures, the logical forms representing the meaning of sentences. The structures that arise on the way to these targets we call *partial trees*.

Definition 4 (Partial Trees) The set PT, of *partial trees*, consists of all T-structures $\mathcal{T}$ such that there is a T-morphism mapping $\mathcal{T}$ to an element of LBT, i.e., a binary tree.

Notice that in a full-blown linked binary tree, $n \prec_* m$ implies that there is a sequence of immediate dominance steps relating n to m, but in a *partial* tree this does not need to be the case.

So when a the term 'partial tree' is used this may mean not only that not all nodes are present but also that some tree relations are not yet fully specified.

A pair $\mathcal{T}n$ of a partial tree $\mathcal{T}$ and a node n in the domain T of $\mathcal{T}$ will be called a *pointed partial tree*. In order to describe the (computational) structure of partial trees we introduce *addressing* of nodes that will be used to move through a tree.

Addresses

In the process of growth from partial to full binary trees, tree nodes have to find locations in the tree and nodes may have to be merged. This requires *computation* on tree node addresses. The addressing of the nodes we will use is simple. The address i of any node n is just the 'strongest', the most specific, sequence $\mathcal{T}r \prec_{i(1)} \prec_{i(2)} \ldots \prec_{i(n)} \mathcal{T}n$ in the tree, where r is the (absolute) root of the tree. We will define the notion of 'strength' involved. We will set the domain D_{Adr} of **addresses** to be the set $D_{Adr} = \{0i \mid i \in I^*\} - \{i**j \in I^* \mid i,j \in I^*\}$ where $I = \{0, 1, \downarrow, *, L\}$[4]. That is, the addresses are finite strings of indices of tree relations starting with 0 and excluding strings with adjacent occurences of the symbol '$*$'. On the set D_{adr} a relation '$\triangleright$' of 'strengtening' will be defined, by means of which adresses will be assigned to the nodes in a tree.

The absolute root r in the tree, (i.e. the $n \in T$ such that $\mathcal{T}n' \prec_i \mathcal{T}n$ implies $n' = n, i = *$) is given the address 0:

[4]to also account for adjunction we include the inverses of the indices: $I = (\{0, 1, \downarrow, *, L\} \cup \{0^{-1}, 1^{-1}, \downarrow^{-1}, *^{-1}, L^{-1}\})$.

$ADR_{\mathcal{T}}(r) = 0$.

If $ADR_{\mathcal{T}}(n) = a$ and $j \in I^*$ is the $\triangleright$-strongest sequence of indices in $\{i \in I^* \mid \mathcal{T}n \prec_i \mathcal{T}n'\}$ then $ADR_{\mathcal{T}}(n') = a \cdot j$.

The addresses of tree $\mathcal{T}$ are collected in $ADR(\mathcal{T}) = \{i \in D_{Adr} \mid \exists n \in T : ADR(n) = i\}$.[5]

Definition 5 (The $\triangleright$-Relation on Addresses) We have on D_{Adr} a basic *reflexive, transitive and asymmetric* relation $\triangleright$ ('strengthening') defined by:

- For all $x, y \in D_{Adr}$:
  $x \triangleright x$,
  $(x \triangleright y \ \&\ y \triangleright x) \Rightarrow x = y$
  $(x \triangleright y \ \&\ y \triangleright z) \Rightarrow x \triangleright z$.
- $\downarrow \triangleright 0$,
- $\downarrow \triangleright 1$,
- $* \triangleright z \in \{0, 1, \downarrow, *\}^* - \{i * *j \in I^* \mid i, j \in I^*\}$.[6]

For $i \in I, j \in I^*$, we have $i \triangleright j$ if j is a more specific tree relation than i (0-daughter is stronger than daughter ($\downarrow$) and $\downarrow$-daughter is a stronger relation than a $*$-daughter).
Now for complex $i \in D_{Adr}$, we set

- $i \triangleright i' \iff i' = j_1 \dots j_k \in I^*$ such that $i \in I^k$ and for all $l \le k : i(l) \triangleright j_l$.

So the Computational structure we are considering is the tuple $(D_{Adr}, \triangleright)$ as an address ordering with the important concept of **directed set** of addresses, a set with a $\triangleright$-upper bound in D_{Adr}. That is, a set $A \subseteq D_{Adr}$ is directed if there is an element $j \in D_{Adr}$ such that $i \triangleright j$ for all $i \in A$. In a Binary Tree, the set of $j \in D_{Adr}$ such that $\mathcal{T}n \prec_j \mathcal{T}n'$ for fixed $\mathcal{T}n$ and $\mathcal{T}n'$ is such a directed set *containing its upper bound*; Here is such a set

$$\{01 * 0, 0 * 0, 0 \downarrow * \downarrow, 01*\}$$

the obvious upper bound is $01 * 0$, the most specific relation of the four.
If there is more than one tree relation between two nodes in a binary tree, then these are always ordered according to strength, that is, they cannot be mutually excusive (like being both the left and the right daughter of a given node). Here left-daughter is stronger than daughter (immediate dominance) and immediate dominance is stronger than dominance. The strongest sequence in the tree is then the one with the strongest links.

[5] If we include the inverse indices, $i^{-1}, i \in I$, then we define the relation $\prec_{i^{-1}}$ by $\mathcal{T}n \prec_{i^{-1}} \mathcal{T}n' \equiv_{df} \mathcal{T}n' \prec_i \mathcal{T}n$, and consider the address as the 'strongest' sequence of indices from the extended set of tree relations. The definition of the root r now includes that $\mathcal{T}r \prec_{i^{-1}} \mathcal{T}n$ implies $r = n$ and $i = *$.

[6] and this pattern is repeated exactly for the inverse indices. Moreover, for the complex case (here 'ϵ' is the empty sequence):

- $ii^{-1} \triangleright \epsilon$,
- $i^{-1}i \triangleright \epsilon$
- $\downarrow 0^{-1} \triangleright \epsilon$, $\downarrow 1^{-1} \triangleright \epsilon$
- $0 \downarrow^1 \triangleright \epsilon\ 1 \downarrow^1 \triangleright \epsilon$

Remark 1 (Uniqueness of the strongest connection) In our structures there will be nodes that can be reached along the same sequence of indices from the root, but the *strongest* such sequences are unique. Why is this so? First notice that the one-node tree consisting of only the root-node satisfies the uniqueness. The unique strongest address is 0 (other addresses include $*0, *$, both $\triangleright$ reducible to 0). Given a tree $\mathcal{T}$ that satisfies the principle, then new tree structure arises in two ways from $\mathcal{T}$

1. by the make(i) action applied to $\mathcal{T}n$ for some node $n \in T$. This action creates a *new* node n' with the unique connection $\mathcal{T}n \prec_i \mathcal{T}n'$. Now because $\mathcal{T}n$ has a unique address (strongest connection to the root), so does $\mathcal{T}n'$.
2. by the merg(i) action applied to $\mathcal{T}n$ for some node $n \in T$. Merging of nodes in general may create index sequences between the root and a node which do not have a unique maximum, but in our set-up, uniqueness is guaranteed by making the existence of such a maximum the *precondition* of merging two nodes. (Here we have to compute $\triangleright$ pairs).

The point is now that all trees that will appear in our computations are constructed from the root-node tree by make() and merg().

All computations of addresses will involve computing the $\triangleright$ maximum of such a set, generally with exactly two elements.

A weaker set of addresses is a directed set not containing its maximum

$$\{0 * 1, 00*\}$$

The central feature of the directed sets is that they correspond exactly to the sets of indices in D_{Adr} that can index the relations between the same two nodes in a *partial* (as opposed to a *complete*) binary tree.

Sets of addresses without an upper bound in D_{Adr} are sets that cannot index the relations between the same two nodes; the archetypical such set is

$$\{01, 00\}.$$

The $\triangleright$-maximum is the *strongest* relation that holds between the given nodes in $\mathcal{T}$. If i is the $\triangleright$-strongest element of $\{j \in D_{Adr} \mid \mathcal{T}n' \prec_j \mathcal{T}n\}$ then we define that $\mathcal{T}n \prec_i^{\mu} \mathcal{T}n'$ holds. This is the strongest relation in the sense that all other relations can be deduced. (If a 0-daughter, then immediately dominated, if immediately dominated then dominated).

Remark 2 For any T-structure $\mathcal{T}$ we define for every pair $n, n' \in T$ the set $[n, n'] \subseteq D_{adr}$

$$[n, n'] = \{i \in I^k \mid \mathcal{T}n \prec_{i(1)} \ldots \prec_{i(k)} \mathcal{T}n', k < \omega\}$$

1. $\mathcal{T}$ is a *partial tree* iff $[n, n']$ is a $\triangleright$-directed set for every n, n'. (I.e if it has an upper bound in D_{adr}).
2. $\mathcal{T}$ is a *binary tree* iff for every n, n' the set $[n, n']$ has an upper bound in $[n, n']$ itself.

Note that $[n, n']$ is a finite set for $n, n' \in T$ the domain of some T- structure. (This is so because we do not allow adjacent $*$ symbols).

Now we can formulate the constraint the binary relation $=$ must satisfy in every binary tree:

$\#_1$ for every $n, n' \in T$, the domain of $\mathcal{T}$, the set $\{i \in I^* \mid n'' \prec_i n''', n'' = n, n''' = n'\}$ has a unique $\triangleright$-maximum, the 'strongest', most specific, relation between the nodes.

Merging Nodes

For every partial tree $\mathcal{T}$ with nodes n, n' we may introduce a function $f_{n,n'}$ mapping $\mathcal{T}$ the tree $f_{n,n'}(\mathcal{T})$ which is exactly like $\mathcal{T}$, except that $f_{n,n'}(\mathcal{T})n = f_{n,n'}(\mathcal{T})n'$, that is, it has added to $\mathcal{T}$ that the nodes n, n' are equivalent.

> So $f_{n,n'}$ is a *partial* function, because not every two nodes in a tree can be identified; This is because conditions $\#_1$ and $\#_2$ (see Section 4.2 below) will in general not be satisfied.

Supposing that $\mathcal{T}n = \mathcal{T}n$ is given for all $\mathcal{T}, n \in T$; then for every $\mathcal{T}, n \in T$ the =-equivalence class $[n]_= = \{n' \in T \mid \mathcal{T}n = \mathcal{T}n'\}$ is well-defined and non-empty (as it contains at least n itself). Now for $\mathcal{T}$ a T-structure, the structure $[\mathcal{T}]$ consist of $[T] = \{[n]_= \mid n \in T\}$ and $[l] \prec_i [n]$ iff $\exists l' \in [l], n' \in [n] : l' \prec_i n'$. By condition $\#_1$ above there is a unique strongest relation between nodes in $\overline{\mathcal{T}}$.

Then the operation $MER_{n,n'}$ which given a partial tree delivers a partial tree in which the nodes n, n' have been identified, can be defined by

$$MER_{n,n'}(\mathcal{T}) = [f_{n,n'}(\mathcal{T})]$$

Definition 6 (Pointed Partial Trees) A *pointed partial tree* $\mathcal{T}n$ is a partial Tree $\mathcal{T}$ together with an element n of T, the domain of $\mathcal{T}$.The set of all pointed partial trees will be denoted by PPT. A pointed partial tree $\mathcal{T}n$ will also be identified by $\mathcal{T}p$ where $p \in ADR(\mathcal{T})$ is *the* address of node n in $\mathcal{T}$.

Actions will be defined on elements of PPT and formulas of our metalanguage evaluated at elements of PPT, that is, at tree *nodes*. For instance, the merg action $\texttt{merg}(i)$ that will be defined has a domain pointed partial trees and

$$\texttt{merg}(i)(\mathcal{T}n) = MER_{n,n'}(\mathcal{T}),$$

where $ADR_{\mathcal{T}}(n') = i$.

4.2 Node decorations

The Node decorations are assigned to the nodes by one of the functions in

$$DEC = \{TY, FO, LAB\}$$

For $F \in DEC, F : PPT \mapsto D_F$. Moreover, all functions except ADR are partial. Here the values assigned by TY, FO, LAB are in the respective domains D_{Ty}, D_{Fo} and D_{Lab} the domains of *types*, *formulas* and (additional) labels respectively. All these domains are structured by a $\triangleright$ partial order. These will be defined below. These functions respect the equivalence relation = in the sense that

$\#_2$ for $F \in \{FO, TY, LAB\}$: the set $\{F(\mathcal{T}n') \mid n' \in [n]_=\}$ has a unique $\triangleright$ maximum a (in the $(D_F, \triangleright)$ structure); this maximum is the value of F on $[\mathcal{T}][n]$, i.e. $F([\mathcal{T}][n]) = a$.

So, for $=$ to be well-defined on a structure the sets

— $\{i \in I^* \mid [n]_= \prec_i [n']_=\}$,

and

— $\{a \in D_F \mid F_{\mathcal{T}}(n') = a, n' \in [n]_=\}$

for $F \in DEC$ must all have a unique $\triangleright$-maximum.

Types

The domain D_{Ty} is built by closing off the set of basic types e, t and cn under the operations:

- if $A, B \in D_{Ty}$ then $A \rightarrow B \in D_{Ty}$.
- $A, A \rightarrow B \in D_{Ty}$ then $ModPon(A, A \rightarrow B) \in D_{Ty}$.

The basic types e, t, cn stand for 'entity', 'truth', and 'set' denotations' respectively. With these basic types we classify *all* linguistic expressions using the combinator '$\rightarrow$'.

On D_{Ty} the relation $\triangleright$ is only given for

$$ModPon(A, A \rightarrow B) \ \triangleright \ B$$

(it 'executes' Modus Ponens).

Formulas

The domain D_{Fo} consist of our lambda terms representing 'meanings' of sentence parts. And if $\lambda x\phi, \psi \in D_{Fo}$ then $APL(\lambda x\phi, \psi) \in D_{Fo}$.

On D_{Fo} the relation $\triangleright$ is given for

$$APL(\lambda x\phi, \psi) \ \triangleright \ \phi[\psi/x]$$

(it substitutes the argument ψ for all free occurrences of x in the functor body ϕ).

In the description of the linguistic data FO-values uniquely determine TY-values. In particular, 'having a FO value' and 'having a TY-value' coincide.

4.3 Node requirements

A further function, REQ , assigns finite sets of so-called DU-formulas to tree nodes,

$$REQ(\mathcal{T}n) = A \iff A \text{ is a finite set of } DU\text{-formulas,}$$

In section 4.4 the DU language will be defined. The elements of $REQ(\mathcal{T}n)$ represent requirements to be fulfilled at that node. For equivalence classes $[n]_=$ we set:

$$REQ([\mathcal{T}][n]_=) \bigcup_{\mathcal{T}'n' \in [\mathcal{T}][[n]_=} REQ(\mathcal{T}'n').$$

Requirements are formulas of the language DU. This language is designed to express configurations within a decorated binary tree. A configuration is roughly a a connected piece of (partial) tree structure plus a selection of decorations for the nodes in the structure.

A **parsing alternative** is a pair

$$(\mathcal{T}n, DEC, REQ, s)$$

consisting of a pointed partial tree $\mathcal{T}n \in PPT$ (a partial tree $\mathcal{T}$ with pointer located at node n) and a string $s \in STRING$ (the remainder of the sentence to be parsed), together with a set of decoration functions $DEC = \{FO, TY, LAB, ADR\}$ and a requirement function REQ.

In an alternative $(\mathcal{T}n, DEC, REQ, s)$ all information is present to decide on actions to undertake

4.4 A language to talk about decorated trees

We will introduce a meta language DU to describe configurations of partial decorated tree structures. This language is evaluated on tree nodes. The atomic vocabulary of this language consists of the possible node decorations. The full set of operators on this language contains the usual **Boolean connectives**, it contains **modal operators** (evaluated along the tree relations) and **quantifiers** which quantify over the domain of feature values that can decorate a node. Finally, the language has the operator '?' which turns a ?-free DU assertion ϕ into a **requirement** $?\phi$. This language will be used for the following purposes:

1. In terms of this language *requirements* or *goals* will be expressed. These will decorate tree nodes and will direct the parsing activity.

2. Pre- and post-conditions of actions can be formulated in DU. In fact in our framework, *all* actions have pre-conditions in DU. As the DU language has both quantifiers and modal operators, quite strong pre- and post-conditions can be formulated and it becomes interesting to consider pre- and post-conditions formulated in restricted versions of DU (quantifier free and/or modality free, for instance).

3. On a meta level, we use the language to axiomatize our data structures and transformations.

Definition 7 (The language DU) The DU-terms are variables $x, y, \ldots \in VAR$, or elements from $D = D_{Fo} \cup D_{Ty} \cup \ldots$. DU has monadic predicate symbols Fo, $Ty, Adr, Lab,$ and sentential operator ?. So the FORM(DU) is defined by

$$A ::= Fo(a) \mid Ty(b) \mid lab(c) \mid Adr(d) \mid \top \mid \bot \mid ?\emptyset \mid \neg A, \mid A \wedge A \mid ?A \mid \exists x A \mid \langle i \rangle A \mid \langle i^{-1} \rangle A.$$

for $a \in VAR \cup D_{Fo}$ analogously for b, c and d. Furthermore $x \in VAR$, $i \in \{0, 1, \downarrow, *, L\}$ and $\top, \bot, ?\emptyset$ special constants (the meaning of which is given by the definition below)

The function REQ assgins to each node n a list of DU-formulas, the *requirements* at n. In order to define the important notion of when *requirement R is fullfilled* we have to define what it means for a node in a tree to *force* a DU-formula.

Interpretation of the language

The language is interpreted with respect to the models as defined. In an alternative $(\mathcal{T}n, DEC, REQ, s)$ all information is present to decide on actions to undertake. What information can be accessed is given by the following *forcing* definition.

Definition 8 (Forcing Definition for DU) Given a pair of a Pointed Partial Tree $\mathcal{T}n$ and a string s, a set of functions $DEC = \{FO, TY, LAB, ADR\}$, and a requirement function REQ assigning values to (tree)nodes as described we say that pointed tree $\mathcal{T}n$ *forces* DU-formula ϕ, with the notation

$$(\mathcal{T}n, DEC, REQ, s) \Vdash \phi,$$

if

$\phi \neq \bot$
$\phi = \top$
$\phi = Fo(a)$ (is atomic) and $FO(\mathcal{T}n) = a$
$\phi = Ty(a)$ (is atomic) and $TY(\mathcal{T}n) = a$
$\phi = Lab(a)$ (is atomic) and $LAB(\mathcal{T}n) = a$

for $i \in I, a \in I^*$:
$\mathcal{T}n \Vdash Adr(0)$ iff there is no $n' \neq n$, no i such that $\mathcal{T}n' \prec_i \mathcal{T}n$
$\mathcal{T}n \Vdash Adr(a \cdot i)$ iff there is $\mathcal{T}n'$ such that $\mathcal{T}n' \prec_i^\mu \mathcal{T}n$ and $\mathcal{T}n' \Vdash Adr(a)$.

$\phi = Root$ and there is no $n' \neq n$, no i such that $\mathcal{T}n' \prec_i \mathcal{T}n$
$\phi = Term$ and there is no $n' \neq n$, no i such that $\mathcal{T}n \prec_i \mathcal{T}n'$

$\phi =?\emptyset$ and $REC(\mathcal{T}n) = \emptyset$
$\phi =?\psi$ (is a requirement) and $\psi \in REQ(\mathcal{T}n)$

Reference to the string:

$\phi = Lang(w)$ and $w = s(1)$

$\phi = \neg\psi$ and *not* $(\mathcal{T}n, s) \Vdash \psi$
$\phi = \psi \wedge \chi$ and $(\mathcal{T}n, s) \Vdash \psi$ & $(\mathcal{T}n, DEC, REQ, s) \Vdash \chi$
$\phi = \exists \mathbf{x}\psi$ and there is a $t \in \mathcal{D} : (\mathcal{T}n, DEC, REQs) \Vdash \psi[t/\mathbf{x}]$

if $i \in I = \{0, 1, \downarrow, *, L\}$ and

$\phi = \langle i \rangle \psi$ and $\exists n' \in T : \mathcal{T}n \prec_i \mathcal{T}n'$ and $(\mathcal{T}n', DEC, REQ, s) \Vdash \psi$
$\phi = \langle i^{-1} \rangle \psi$ and $\exists n' \in T : \mathcal{T}n' \prec_i \mathcal{T}n$ and $(\mathcal{T}n', DEX, REQ, s) \Vdash \psi$

Ordering of parsing alternatives

The parsing alternatives will be the structures on which the words and the general rules will work. These actions are (decorated tree)-structure preserving; they only increase the information collected from the string and **fulfill requirements**. This is represented in the following sense. Every action α satisfies

$$\alpha(\mathcal{T}, n, DEC, REQ, s)) \leq (\mathcal{T}, n, DEC, REQ, s)$$

by the following definition.

$$(\mathcal{T}, n, DEC, REQ, s) \leq (\mathcal{T}', n', DEC', REQ', s')$$

holds if

1. there is a mapping f from T in T' such that $f(n) = n'$ and for all $i \in I: \; m \prec_i m' \Rightarrow f(m) \prec_i f(m')$.
2. For all $F_i \in DEC: \; F_i(\mathcal{T}n) \triangleright F_i'(\mathcal{T}'n')$.
3. $REQ(\mathcal{T}n) \subseteq REQ'(\mathcal{T}'n') \cup \{\phi \in DU \mid (\mathcal{T}'n', DEC', REQ', s') \Vdash \phi\}$.
4. $\exists a \in W^* : s = a \cdot s'$.

An *appropriate* task structure $(\mathbf{PT}, \mathbf{LBT})$ satisfies the following:

1. for all $a \in PT : \mathbf{Start} \leq a$,
2. for all $a \in \mathbf{PT} \exists b \in \mathbf{LBT} : a \leq b$
3. for all $a \in \mathbf{LBT}, F \in DEC : (\forall b : a \leq b \exists c : b \leq c : F_c(c) = \phi) \Rightarrow F_a(a) = \phi$.
4. for all $a \in \mathbf{LBT}, F \in DEC : F_a(a) \neq \phi \Rightarrow (\exists b : a \leq b \forall c : (b \leq c \Rightarrow F_c(c) \neq \phi)$.

We can introduce a modality to address the most general partial order.

For ϕ a DU-formula

$$\begin{aligned}(\mathcal{T}, n, DEC, REQ, s) \Vdash \Diamond\phi \quad \Longleftrightarrow \quad & \exists(\mathcal{T}', n', DEC', REQ', s') : \\ & (\mathcal{T}, n, DEC, REQ, s) \leq (\mathcal{T}', n', DEC', REQ', s') \\ & \text{and } (\mathcal{T}', n', DEC', REQ', s') \Vdash \phi\end{aligned}$$

And we can now introduce strong variants of universal operators, for instance $(\phi \mapsto \psi) =_{df} \Box(\phi \rightarrow \psi)$.
Persistence of atomic formulas can be guaranteed by the following definition

$$(\mathcal{T}, n, DEC, REQ, s) \Vdash Fo(a) \quad \Longleftrightarrow \quad a \triangleright FO(\mathcal{T}n).$$

and analogously for the Ty and Lab predicates. I. e., with this definition

$$Fo(a) \rightarrow \Box Fo(a).$$

With this notion of persistence, we cannot *test* for values of the F-function, for all $\triangleright$-weaker values are also accepted.

The set of data structures BT then is identified by

$$\mathcal{T}n \in BT \iff \forall \phi : \mathcal{T}n \models \Box\Diamond\phi \rightarrow \phi.$$

That is, the set $\mathbf{LBT} \subset \mathbf{PT}$ consist of a set of maxima, where $\neg\neg\phi$ reduces to ϕ and $\phi \vee \neg\phi$ holds for every ϕ.

5 From Language to Representation (Start, Background, Words)

Actions map a pointed partial tree to a new one. A global action may be factored into 1) actions that map one tree to another one, and 2) actions that map one pointer to another. The elements of **Background, Words** are generally complex actions composed of a small number of atomic actions put together by the regular operations plus a form of a conditional test. The atomic actions and the modes of composition are the subject of Section 5.1. In fact, all elements α of **background** $\cup$ **words** will be **partial functions**, complex actions on **PT**, such that there is a DU-formula ϕ:

$$(\mathcal{T}n, DEC, REQ, s) \in dom(\alpha) \iff (\mathcal{T}n, DEC, REQ, s) \Vdash \phi.$$

Sections 5.2 and 5.3 will discuss these.

5.1 Actions on Trees

An action α is of the form

$$\alpha \subseteq PPT \times PPT,$$

that is, actions are *relations* over PPT. An action α may map a Pointed Partial Tree to a number of other such trees. The actions we will introduce are fitted to the Pointed Partial Tree Models in that they are *incremental*, they never delete structure once it has been created. The actions are based on a set of *basic* or *atomic* actions. The basic actions consist of: creation of new nodes relative to old ones; decoration of nodes, moving to nodes, and merging of nodes. These action may now be combined to complex ones.

Definition 9 (Actions) For PPT a Pointed Partial Tree Model with Requirements, $i \in \{0, 0^{-1}, 1, 1^{-1}, \downarrow, \downarrow^{-1}, *, *^{-1}, L, L^{-1}\}$, and ϕ either an atomic DU formula or of the form $?\psi$ for arbitrary $\psi \in DU$, the set ACT of actions contains the following elements[7] :

Basic Actions

1. $1 = \{\langle \mathcal{T}n, \mathcal{T}n \rangle \mid \mathcal{T}n \in PPT\}$,
   $AB = \emptyset$.
   These represent the halting action and the Abort action respectively.

2. $\texttt{make}(i)$. For $i \in I$ the action $\texttt{make}(i) : PPT \mapsto PPT$ creates new nodes. This action creates a new node n' and a tree $\mathcal{T}' = \mathcal{T} \cup \{n'\}$ such that $\mathcal{T}'n \prec_i \mathcal{T}'n'$.

3. $\texttt{put}(\phi)$. For ϕ an atomic DU-formula (i.e. of the form $Fo(a), Ty(b), Lab(c)$) or a DU-formula of the form $?\psi$ the action $\texttt{put}(\phi) : PPT \mapsto PPT$ is defined. This action leaves the tree structure unchanged but adds values to the Node function FO, TY, LAB, REQ so that, for instance, $\texttt{put}(Fo(a))(\mathcal{T}n) = \mathcal{T}'n$ where $FO(\mathcal{T}'n) = a$ and $\texttt{put}(?\psi)(\mathcal{T}n) = \mathcal{T}'n$ where $\psi \in REQ(\mathcal{T}'n)$.

4. $\texttt{go}(i)$. for $i \in I$ $\texttt{go}((i)) : PPT \mapsto PPT$ such that $\texttt{go}((i))(\mathcal{T}n) = \mathcal{T}'n' \iff \mathcal{T} = \mathcal{T}'$ & $ADR_{\mathcal{T}}(n') = ADR_{\mathcal{T}}(n) \cdot i$,

[7]We define the actions only on the relevant coordinates of the parsing alternatives, the ones that change. If unmentioned than the values of DEC and REQ mappings are unchanged.

5. $\texttt{merg(i)}$. For $i \in D_{Adr}$, the action $\texttt{merg(i)} : PPT \mapsto PPT$ is defined as follows. The action $\texttt{merg(i)}$ is defined on $\mathcal{T}n$ if $\mathcal{T}n' \Vdash Adr(i)$ and $\mathcal{T}n \Vdash Adr(j)$ and the sets $\{i,j\}$ and $\{F_{\mathcal{T}}(n), F_{\mathcal{T}}(n')\}$ for $F \in DEC$ have unique $\triangleright$-maxima.
   If $\texttt{merg(i)}$ is defined on $\mathcal{T}n$ then $\texttt{merg(i)}(\mathcal{T}n)$ is the tree $\mathcal{T}'[n]$ which is identical to $\mathcal{T}n$ except that $\mathcal{T}'n = \mathcal{T}'n'$ and $\mathcal{T}'m \prec_i \mathcal{T}'[n]$ means that i is the maximal element in $\{i \in I^* \mid n' \in [n] : \mathcal{T}'m \prec_i \mathcal{T}'n'\}$, analgously for $\mathcal{T}'[n] \prec_i \mathcal{T}'m$. Also $F_{\mathcal{T}}([n]) = a$ means that a is the maximal element in $\{F(\mathcal{T}m) \mid m \in [n]\}$

Complex Actions We can put actions together by executing one after the other (sequential composition ';') and doing that any finite number of times (finite iteration '$*$'), or by indeterministically choosing between them (indeterministic choice '$+$').

6. if α, α' are actions, then so are $\alpha;\alpha', \alpha+\alpha', \alpha^*$.

Finally, we can put actions together in a conditional IF THEN ELSE statement. This action supplies the second reason for introducing the language *DU* describing configurations on a pointed partial tree: an action can be *guarded* by or be *conditionalized* w.r.t a tree-configuration. A conditional action can do more, it can *bind* variables. Let $[a/x]$, for $a \in D$ a *variable-free* term and $x \in VAR$, be a *substitution* of a for x. For A a *DU*-formula let $A[a/x]$ be the result of replacing every *free* occurrence of the variable x in A by the term a. The conditinal actions use these substitutions

7. If Σ is a set of formulas all variables of which occur in $\overline{x}$ and α, α' are actions then $\langle \Sigma(\overline{x}), \alpha, \alpha' \rangle : PPT \mapsto PPT$ is a *conditional* action with the definition:
   $\langle \Sigma(\overline{\mathbf{x}}), \alpha, \alpha' \rangle =$
   $$\{\langle \mathcal{T}n, \mathcal{T}'n' \rangle \in \alpha[\overline{t}/\overline{x}] \mid \overline{t} \in (\mathcal{D} \cup VAR)^*, \mathcal{T} \Vdash \Sigma[\overline{t}/\overline{x}]\} \cup \{\langle \mathcal{T}n, \mathcal{T}'n' \rangle \in \alpha' \mid \neg \exists \overline{t} \in (\mathcal{D} \cup VAR)^*, \mathcal{T}n \Vdash \Sigma[\overline{t}/\overline{x}]\}.$$

That is, if Σ is forced by $\mathcal{T}n$, for some substitution of t for variable x, then action α is executed, where in the body of this action the variable x is also replaced by t. If Σ is not forced by $\mathcal{T}n$ for any substitution for x, then action α' is executed.

For instance, the action $\langle \{\langle\downarrow_1\rangle Fo(x), \langle\downarrow_0\rangle Fo(y)\}, \texttt{put}(Fo(APL(x,y))), 1\rangle$ has the semantics: if

$$\mathcal{T}n \Vdash \langle\downarrow_1\rangle\ Fo(a)\ \wedge\ \langle\downarrow_0\rangle Fo(b)$$

then execute $\texttt{put}(Fo(APL(a,b)))$ else execute 1, i.e. do nothing.

5.2 Words

Elements of **Words**, lexical actions, map one tree to a next one adding information in the process. These transitions are projected by the Natural Language *words*. They are defined as conditional actions of the IF THEN ELSE variety. A lexical transition tests IF some finite set of formulas (the *condition*) holds at the node where the pointer is located — this may include modal statements about decorations located at nodes related to the pointed one and it may also involve requirements. If the condition holds there, THEN a (sequence of) action(s) is undertaken resulting in a new Tree Description, ELSE (i.e. if the condition does not hold) a second action is undertaken, usually an 'abort' action.

The actions projected by a single lexical item may carry out a whole range of different updates. Some items project only an annotation as a decoration (proper names), or only a requirement (case particles), or an annotation/requirement (tense specifications) or, even, only movement of the pointer (expletives).

Definition 10 (Lexical Actions) A *Lexical Action* w is an action of the form

$$w = \langle \Sigma(x), \alpha, \alpha' \rangle$$

where Σ is a finite set of DU-formulas, x is a sequence containing all variables occurring in Σ, and α, α' complex actions as given by Definition 9.

For example, we have *V1*, and *hit* project the following actions

V1	IF	$\{?Ty(e)\}$
	THEN	$\texttt{put}(Fo(\mathbf{V1}), Ty(e), [\downarrow]\bot)$
	ELSE	AB
hit	IF	$\{?Ty(e \rightarrow t)\}$
	THEN	$\texttt{make}(\langle 1 \rangle); \texttt{go}(()\langle \downarrow_1 \rangle)$
		$\texttt{put}(Fo(\mathbf{hit})), Ty(e \rightarrow (e \rightarrow t)), [\downarrow]\bot;$
		$\texttt{go}(\langle 1^{-1} \rangle); \texttt{put}(?\langle \downarrow_0 \rangle (Ty(e)))$
	ELSE	AB

In case of the proper name *V1* the actions are released by the presence of the requirement for an object of type e. These actions then consist in annotating the node with the content formula **V1** the type label e, and closing it off as a terminal node. In the case of the transitive verb '*hit*', the presence at the pointed node of a requirement for $Ty(e \rightarrow t)$ releases the actions: creation of a function daughter node, annotation of that node with a type and a formula and the imposition on the mother node, the predicate node, of a requirement for an argument daughter of type e. Actions annotating the node with facts are the analogue of lexical substitution processes in other frameworks (though it should be remembered that it is not lexical items which are inserted in the tree in this framework, but their logical correlate). Actions decorating a node with requirements are analogous to sub-categorisation statements. However, in this framework all nodes are introduced with requirements, so unlike standard sub-categorisation statements which are imposed on sisters to terminal nodes as a requirement on lexical insertion in a tree, in this framework, the use of such requirements is much more widespread.

Words are assigned to categories which combine a set of features common to all words in that category.

Noun phrases

Noun phrases are simple like *V1, V2*, *John* and complex, like *a car* (a determiner and a common noun), or *a red car* (a determiner, adjective and common noun) or, in the accident reports often *car* with implicit determiner'*a*'.

Noun phrases: V1, V2, the pedestrian,...

V1	IF	$\{?Ty(e)\}$
	THEN	$\mathtt{put}(Fo(\mathbf{V1}), Ty(e), [\downarrow]\bot)$
	ELSE	AB

For noun phrases, the general rules give

Expansion NP+	IF	$\{?Ty(e), Term\}$
	THEN	1+DetNou
	ELSE	1
DetNou+	IF	$\{?Ty(e), Term\}$
	THEN	$\mathtt{make}(\langle\downarrow_0\rangle); go\langle\downarrow_0\rangle; \mathtt{put}(?Ty(cn));$ $\mathtt{make}(\langle\downarrow_1\rangle); \mathtt{go}(\langle\downarrow_1\rangle); \mathtt{put}(?Ty(cn \to e)); \mathtt{go}(\langle\uparrow_1\rangle)$
	ELSE	1
Followed by		$\mathtt{go}(\langle\downarrow_1\rangle); \mathtt{put}(Fo(\epsilon), Ty(cn \to e)); \mathtt{go}(\langle\uparrow_1\rangle); \mathtt{go}(\langle\downarrow_0\rangle)$ $+$ $\mathtt{go}(\langle\downarrow_1\rangle)$

So, if the pointer is at a node with $?Ty(e), Term$ then either the pointer stays there (**1**) (for cases like *V1*) or it expands the node by the program DetNou which makes a node for a determiner and a node for a common noun. Now, either it fills in the determiner (for loose standing *car*) and moves the pointer to the common noun node, or it awaits a determiner.

Determiners the/a/some

DET	IF	$\{?Ty(cn \to e)\}$
	THEN	$\mathtt{put}(Fo(\mathbf{DET}), Ty(cn \to e), [\downarrow]\bot)$
	ELSE	AB

Common Nouns car, speed, child, parent(s)...

CN	IF	$\{?Ty(cn)\}$
	THEN	$\mathtt{put}(Fo(\mathbf{CN}), Ty(cn), [\downarrow]\bot)$
	ELSE	AB

Adjectives (great as in 'with great speed'), sufficient, opposite

ADJ	IF	$\{?Ty(cn)\}$
	THEN	$\mathtt{make}(\langle\downarrow_1\rangle); \mathtt{go}(\langle\downarrow_1\rangle); \mathtt{put}(Fo(\mathbf{Adj}), Ty(cn \to cn), [\downarrow]\bot)$ $\mathtt{go}(\langle\uparrow_1\rangle); \mathtt{make}(\langle\downarrow_0\rangle); \mathtt{go}(\langle\downarrow_0\rangle); \mathtt{put}(?Ty(cn))$
	ELSE	AB

Transitive Verbs

A transitive verb like *distracts, collides* belongs to the category TV. The basic form, the *stem*, of the verb *distracting* is *distract*

TV		
TV	**TV**ing	**TV**ed
distract distracts distracted	distracting	distracted
collide collides collided	colliding	

Which has the following programs associated with it

$$\begin{array}{lll}
\textbf{TV} & \text{IF} & \{?Ty(e \to t)\} \\
 & \text{THEN} & \texttt{make}(\langle\downarrow_1\rangle); \texttt{go}(\langle\downarrow_1\rangle) \\
 & & \texttt{put}(Fo(\textbf{TV})), Ty(e \to (e \to t)), [\downarrow]\bot; \\
 & & \texttt{go}(\langle 1^{-1}\rangle); \texttt{make}(\langle\downarrow_0\rangle); go\langle\downarrow_0\rangle; \texttt{put}(?Ty(e))) \\
 & \text{ELSE} & \text{AB}
\end{array}$$

where TV has to be filled in.

bus distracting driver.

$$\begin{array}{lll}
\textbf{TV}\text{ing} & \text{IF} & \{Ty(e), Fo(x)\} \\
 & \text{THEN} & \texttt{make}(L); \texttt{go}(L); \texttt{put}(?Ty(t)); \texttt{make}(\langle\downarrow_0\rangle); \texttt{go}(\langle\downarrow_0\rangle); \\
 & & \texttt{put}(Fo(x)); \texttt{go}(\uparrow_0); \texttt{make}(\langle\downarrow_1\rangle); \texttt{go}(\langle\downarrow_1\rangle); \texttt{put}(?Ty(e \to t)) \\
 & \text{ELSE} & \text{AB}
\end{array}$$

Followed by the program for **TV**. (Notice the **TV**ing program leaves the pointer at the right node for **TV** to start).

Driver distracted by bus

$$\begin{array}{lll}
\textbf{TV}\text{ed} & \text{IF} & \{Ty(e), Fo(x)\} \\
 & \text{THEN} & \texttt{make}(L); \texttt{go}(L); \texttt{put}(?Ty(t)); \texttt{make}(\langle\downarrow_1\rangle); \texttt{go}(\langle\downarrow_1\rangle); \texttt{put}(?Ty(e \to t)) \\
 & & \texttt{make}(\langle\downarrow_0\rangle); \texttt{go}(\langle\downarrow_0\rangle); \texttt{put}(Fo(x), Ty(e)); \texttt{go}(\langle\uparrow_0\rangle); \texttt{make}(\langle\downarrow_1\rangle); \texttt{go}(\langle\downarrow_1\rangle); \\
 & & \texttt{put}(Fo(\textbf{TV}), Ty(e \to (e \to t))); \texttt{go}(\langle\uparrow_1\rangle); \texttt{go}(\langle\uparrow_1\rangle); \\
 & & \texttt{make}(\langle\downarrow_0\rangle)\texttt{put}(?Ty(e), Lang(\texttt{by})) \\
 & \text{ELSE} & \text{AB}
\end{array}$$

Some transitive verbs have an idiomatic preposition going with the object e.g. *to collide* **with**. This requires some extra action for this specific **TV**. Now the program is the **TV** program followed by $\texttt{put}(Lang(\texttt{with}))$ (see Definition 8). Notice that **TV** leaves the pointer at the object node ($?Ty(e)$) of the predicate. So the extra action puts a requirement for the word *with* at that node

$$\begin{array}{ll}
\text{IF} & Ty(e \to t) \\
\text{THEN} & \text{ADJUNCT} \\
\text{ELSE} & \begin{array}{ll} \text{IF} & ?Ty(e), ?Lang(w) \\ \text{THEN} & \text{NP} \\ \text{ELSE} & \text{ABORT} \end{array}
\end{array}$$

A preposition *with, on, to,* ..., initiates either a VP-adjunct or a V-argumemt (with idiomatic PP).

Intransitive verbs

ITV		
ITV	**ITV**ing	**ITV**ed
park parks parked	parking	parked
turn turns turned	turning	turned

Notice that the **ITV**ed form can be classified in two ways: *V1 parked in front NS* (past tensed verb) and *behind parked car* (verb used as adjective).

Which has the following program associated with it:

ITV	IF	$\{?Ty(e \to t)\}$
	THEN	$\mathtt{put}(Fo(\mathbf{ITV})), Ty(e \to t), [\downarrow]\bot;$
	ELSE	AB

where ITV has to be filled in.

TVing	IF	$\{Ty(e), Fo(x)\}$
	THEN	$\mathtt{make}(L); \mathtt{go}(L); \mathtt{put}(?Ty(t)); \mathtt{make}(\langle\downarrow_0\rangle); \mathtt{go}(\langle\downarrow_0\rangle);$ $\mathtt{put}(Fo(x)); \mathtt{go}(\uparrow_0); \mathtt{make}(\langle\downarrow_1\rangle); \mathtt{go}(\langle\downarrow_1\rangle); \mathtt{put}(?Ty(e \to t))$
	ELSE	AB

Followed by the program for **ITV**.

ITVed	IF	$\{Ty(e), Fo(x)\}$
	THEN	$\mathtt{make}(L); \mathtt{go}(L); \mathtt{put}(?Ty(t)); \mathtt{make}(\langle\downarrow_0\rangle); \mathtt{go}(\langle\downarrow_0\rangle);$ $\mathtt{put}(Fo(x)); \mathtt{go}(\uparrow_0); \mathtt{make}(\langle\downarrow_1\rangle); \mathtt{go}(\langle\downarrow_1\rangle); \mathtt{put}(?Ty(e \to t))$
	ELSE	AB

Followed by the program for **ITV**. So this form has the same program as **ITV**ing.
This is the way words can be categorized and defined. It is, however, possible (and common) that particular transitive verbs need specific information added. We must allow for that possibility.

Prepositions

The word class w of class $Prep$. This contains words like *with, in, between, on, over, ...* They can occur in phrases (*with great speed*) modifying the verb (*drive with great speed*) or they can occur in phrases (*with a broken window*) modifying an noun phrase (*(a) car with a broken window*)

prepositions with, on, alongside, against. . .

IF	$Ty(e \to t)$		
THEN	ADJUNCT		
ELSE	IF	$?Ty(e), ?Lang(w)$	
	THEN	NP	
	ELSE	ABORT	

A preposition *with, on, to, ...*, initiates either a VP-adjunct or a V-argumemt (with idiomatic PP).

Relative Pronouns

Linked structures are initiated by Relative pronouns *which, that,...* or by gerund constructions (**TV**ing)

Relative Pronouns which, that....

IF	$\{Ty(e), Fo(x)\}$
THEN	$\mathtt{make}(L); \mathtt{go}(L); \mathtt{put}(?Ty(t)); \mathtt{make}(*); \mathtt{go}(*);$ $\mathtt{put}(Fo(x)); \mathtt{go}(*^{-1}); \mathtt{SubObj}$
ELSE	AB

5.3 Background

The **Background** consist of a number of actions that set the stage for processing of the **words**.The actions in **background** are executed at the start of the parse and after every processed word. Then it creates a **finite** number of possible continuations. We will describe 7 such actions.

1. **Set-up from start**. This expands **start** to create a set of initial structures.

$\mathtt{SubObj}+$	IF	$\{?Ty(t), Term\}$
	THEN	$\mathtt{make}(\langle\downarrow_0\rangle); \mathtt{go}(\langle\downarrow_0\rangle); \mathtt{put}(?Ty(e)); \mathtt{go}(\langle\uparrow_0\rangle);$ $\mathtt{make}(\langle\downarrow_1\rangle); \mathtt{go}(\langle\downarrow_1\rangle); \mathtt{put}(?Ty(e \to t)); \mathtt{go}(\langle\uparrow_1\rangle)$
	ELSE	1
$ExpansionNP+$	IF	$\{?Ty(e), Term\}$
	THEN	1+DetNou
	ELSE	1
$DetNou+$	IF	$\{?Ty(e), Term\}$
	THEN	$\mathtt{make}(\langle\downarrow_0\rangle); go\langle\downarrow_0\rangle; \mathtt{put}(?Ty(cn));$) $\mathtt{make}(\langle\downarrow_1\rangle;; \mathtt{go}(\langle\downarrow_1\rangle); \mathtt{put}(?Ty(cn \to e)); \mathtt{go}(\langle\uparrow_1\rangle)$
	ELSE	1
$Followedby$	$\mathtt{go}(\langle\downarrow_1\rangle); \mathtt{put}(Fo(\epsilon), Ty(cn \to e)); \mathtt{go}(\langle\uparrow_1\rangle); \mathtt{go}(\langle\downarrow_0\rangle)$ $+$ $\mathtt{go}(\langle\downarrow_1\rangle)$	

2. **Function application**. This evaluates the information gathered at two daughters.

$$\langle\{\langle\downarrow_1\rangle Fo(x), \langle\downarrow_0\rangle Fo(y)\}, \mathtt{put}(Fo(APL(x,y))), 1\rangle$$

3. **Expanding the trees by formal rules**.This creates new tree structure by developing requirements of a modal kind.

$$\langle\{?\langle i\rangle\phi, [i]\bot\}, \mathtt{make}(i); \mathtt{go}(i); \mathtt{put}(?\phi); \mathtt{go}(i^{-1})\rangle$$

$\langle\{Term, ?Ty(e)\}, \mathtt{make}(*); \mathtt{go}(*)\mathtt{put}(?Ty(e \rightarrow t))\rangle$.

This creates a copy of a $?Ty(e)$-node connected to it by a $\prec_*$ relation.

4. **Construction of Linked Structures**. This prepares for relative clauses. Linked structures are initiated by relative pronouns *which, that,...* or by gerund constructions

IF	$\{Ty(e), Fo(x)\}$
THEN	$\mathtt{make}(L); \mathtt{go}(L); \mathtt{put}(?Ty(t)); \mathtt{make}(*); \mathtt{go}(*);$ $\mathtt{put}(Fo(x)); \mathtt{go}(*^{-1}); \mathtt{SubObj}$
ELSE	AB

   In this action the address of the 'dangling node' is assigned to the variable *Gap*

5. **Construction of Adjunct Structures**. There are three kinds of actions involved to create Adjunct structures. First the preparation for an adjunct: splitting up the VP node

   PREPARATION $\langle\{?Ty(e \rightarrow t)[*^{-1}]\bot\}, \mathtt{make}(*); \mathtt{go}(*); \mathtt{put}(?Ty(e \rightarrow t))\rangle$.

   This creates a copy of a $?Ty(e \rightarrow t)$-node connected to it by a $\prec_*$. Notice that this action cannot be repeated due to the condition $[*^{-1}]\bot$ which does not hold on the newly constructed node.

   Secondly, the wrapping up, the final operation is the incorporation of the Adjunct structure in the tree.

   WRAPPING UP $\langle\{Adr(\mathtt{x}*), Ty(e \rightarrow t)\}, \mathtt{go}(*^{-1}); \mathtt{merg}(\mathtt{x}*), 1\rangle$

   This moves the pointer up and reconnects the split nodes.

   In between the preparation and the wrapping up the structure is introduced, generally by lexical entries of the form **PREP**:

IF	$\{Adr(\mathtt{x}*), Ty(e \rightarrow t)\}$
THEN	$\mathtt{make}(\langle\uparrow_0\rangle); \mathtt{go}(\langle\uparrow_0\rangle); \mathtt{make}(*^{-1}); \mathtt{go}(*^{-1}); \mathtt{merg}(\mathtt{x});$ $\mathtt{go}(*); \mathtt{make}(\langle\downarrow_1\rangle); \mathtt{go}(\langle\downarrow_1\rangle); \mathtt{make}(\langle\downarrow_1\rangle);$ $\mathtt{go}(\langle\downarrow_1\rangle); \mathtt{put}(Ty(e \rightarrow ((e \rightarrow t) \rightarrow (e \rightarrow t))),$ $Fo(PREP)); \mathtt{go}(1^{-1}); \mathtt{make}(0); \mathtt{go}(0); \mathtt{put}(?Ty(e))$
ELSE	AB

   This is a complicated tree transformation where the same node has a different address in the source and target tree.
   There are some points to notice in this program. The node A with address $\mathtt{x}*$ at the start, the position of the pointer, has no longer this address after the action $\mathtt{merg}(\mathtt{x})$. After that action there are 2 indices, $\mathtt{x}*$ and $\mathtt{x}*0$ connecting the root node to A, and the latter one is the stronger index, so it is *the* address of A in the new situation. The action $\mathtt{go}(*)$ following the merge, now goes to B, the newly created node, (through the path $\mathtt{x}*0^{-1}$) and not to A for this can now be reached from B by $\mathtt{go}(*); \mathtt{go}(0)$.
   After the preposition '*with*' in '*with great speed*' has been processed the pointer is left at the embedded $?Ty(e)$ node expecting a NP '*great speed*', thus giving a functor (with decoration $Ty(e \rightarrow t)$) at address $\mathtt{x}*$, the basis for recursive application of adjuncts.

6. **Moving the pointer**. The movement after a subject:

IF	$\{Ty(e), \langle 0^{-1}\rangle Root\ \}$
THEN	$1\ +\ \mathsf{go}(0^{-1}); \mathsf{go}(1)$
ELSE	AB

Pointer Movement in general (i.e. not lexically induced):

MovP = EVALUATE;

IF	$\neg Root$			
THEN	IF	$\exists x Ty(x)$		
	THEN	if $\langle j^{-1}\rangle\top$ then $\mathsf{go}(j^{-1})$, $j \in \{0, 1, L\}$;MovP		
	ELSE	if $\exists x\langle j\rangle ?Ty(x)$ then $\mathsf{go}(j)$, $j \in \{0, 1\}$		
ELSE	IF	$Ty(t)$		
	THEN	IF	$?\emptyset$	
		THEN	FINISH	
		ELSE	AB	
	ELSE	if $\exists x\langle j\rangle ?Ty(x)$ then $\mathsf{go}(j)$, $j \in \{0, 1\}$		

6 Evaluation

In this section the **evaluation loop** of the parser is described. Words of a NL string are processed one by one on a left-to-right basis. The consumption of a word is associated with a non-empy sequence of new trees for every given pointed tree that is $ALIVE$ and accepts the current word. The fact that this is a sequence and not a single tree derives from the facts that 1) a word may have more than one entry in the lexicon, in which case all entries are applied as transformations giving rise to alternative parse paths. 2) The result of processing a word is 'closed off' under application of all the computational transformation in the background. Stages are mapped to other, following stages by *actions*. These actions are of two kinds: lexical actions (**words**), generated by the words of the sentence, and computational transformations (**background**) which don't involve the processing of a word.

A lexical transition is induced by $s(1)$, the first word of the string s. This word is tested at every alternative at the pointer position. This gives a new stage with the new developments of each alternative and string s' which is s minus the first element.
After a word has been processed, the stage is *evaluated* before the next word is considered. The word has left fresh tree structure with new facts and new requirements.

The evaluation proceeds, recursive on the nodes, starting from the node where the pointer is located. If the evaluation of a node results in the existence of a type value and it is the $\prec_0$-daughter or $\prec_1$-daughter of some node n', then the pointer moves to n' and repeats the evaluation process there.
If this process stops because there is no type feature, then there is still processing to do below the current node and this can only be supplied by fresh words. The computational part consists only in moving the pointer to the right nodes which still have outstanding requirements.
If the process stops because there is a type feature but no $0, 1$-mother is present, then a general rule will have to tell us how to resolve the underdetermination at that point.

The body of the evaluation procedure consist of a sequence of conditional actions, where the conditions test for configurations orientated by the pointer node. The conditions are supposed to be mutually exclusive, so when one condition 'fires' we need not evaluate any further.

Evaluation creates alternatives through the *indeterminate choice* operator, +, on actions. That is,

$$\langle \alpha_1 + \alpha_2 \rangle (\overline{\mathcal{T}}_i \mathcal{T}_i, p_i), s) = (\alpha_1(\overline{\mathcal{T}}_i \mathcal{T}_i, p_i), \alpha_2(\overline{\mathcal{T}}_i \mathcal{T}_i, p_i), s)$$

A **Parsing State** is a finite sequence of the form

$$\langle < \mathcal{T} >, s, (\mathcal{T}_1, p_1, p'_1, DEC_1, REQ_1, l_1), \ldots (\mathcal{T}_k, p_k, p'_k, DEC_k, REQ_k, l_k) \rangle$$

Where

- $< \mathcal{T} >$ is a sequence of decorated binary trees representing the 'meanings' of the sentences of the TAR that have been processed already. This is necesarry, because TARs can consist of more than sentence.
- s is (the remainder of) the string to be parsed,
- the components
$$(\mathcal{T}_i, p_i, p'_i, DEC_i, REQ_i, l_i)$$
represent the pointed partial trees resulting from **alternative parses** up to the current word.
  - $\mathcal{T}_i$ the meaning of the part of the sentence that has been parsed upto now.
  - The pointer position ($p_i \in D_{adr}$).
    If $ADR_{\mathcal{T}}(n) = p_i$ then $(\mathcal{T}_i n, DEC_i, REQ_i, s)$ is a parsing alternative.
  - $p'_i = Gap_i \in D_{adr}$
  - $l_i \in \{DEAD, ALIVE\}$ gives the status of this parsing alternative. If the status is $DEAD$ then this alternative won't reoccur in the next stage.

Noitice that a control variable $p'_i = Gap_i$ is added that does not decorate any tree node. The **parsing evaluation** is defined by determining in the space of **parse states**:

1. a start state $\texttt{START}(w)$ of the parse of w,
2. the transition steps $S(w) \vdash S'(w')$ leading from one state to the next,
3. the **GOAL** states where a sentence has been parsed completely leaving at least one decorated linked binary tree without outstanding requirements.

START(w) is given by:

$$\texttt{START}(w) = (< \mathcal{T} >, w, \{n\}, \emptyset, (\{n\}n, 0, Gap, \emptyset, \{?Ty(t)\}, ALIVE)\rangle.$$

GOAL

Let

$$S(w) = \langle < \mathcal{T} >, w, K_1, \ldots K_k) \rangle \in GOAL$$

If

— $w = \epsilon$, the empty string
— for at least one $K_i = (\mathcal{T}, p, Gap, DEC, REQ, l)$ we have

- $p = 0$,
- $Gap \in \{0, 1, L\}^*$
- $REQ(\mathcal{T}0) = \emptyset$
- $l = ALIVE$.

The Transition Relation

Here we define the one-step transition relation between parsing states. We say that

$$S(w) \ \vdash \ S'(w')$$

holds, if

- $S(w) = \langle < \mathcal{T} >, w, (\mathcal{T}_1, p_1, p'_1, DEC_1, REQ_1, l_1), \ldots (\mathcal{T}_k, p_k, p'_k, DEC_k, REQ_k, l_k) \rangle$
- $S'(w') = \langle < \mathcal{T} >, w', K_1, \ldots K_k) \rangle$

where $w' = w(g) \ldots w(n)$ and for each $K_j, 1 \leq j \leq k$:

1. $K_j = \epsilon$ and $g = 1$ if $l_j = DEAD$. Otherwise,
2. $K_j = (\mathcal{T}_j, p_j, DEC_j, REQ_j, DEAD)$ and $g = 1$ if $l_j = ALIVE$ and
   (a) $w(1)(\mathcal{T}_j n) = ABORT$ where $ADR(\mathcal{T}_j n) = p_j$ and
   (b) $MERG_{Gap_j, p_j}(\mathcal{T}_j) =\uparrow$.

   Otherwise,
3. $K_j = (\mathcal{T}', p', p', DEC'_j, REQ'_j, l_j)$ and $g = 1$ if $w(1)(\mathcal{T}_j n) = ABORT$ and $\mathcal{T}' n' = MERG_{Gap_j, p_j}(\mathcal{T}_j) n'$ where $ADR(\mathcal{T}'_j n) = p_j$ and $ADR(\mathcal{T}' n') = p'$. Otherwise,
4. $g = 2$ and

   $$\overline{(\hat{\delta}(w)(1))}((\mathcal{T}_j, p_j, Gap_j, DEC_j, REQ_j, l_j)) =$$
   $$\langle (\mathcal{T}_{j_1}, p_{j_1}, Gap_j, DEC_j, REQ_j, l_j), \ldots (\mathcal{T}_{j_m}, p_{j_m}, Gap_j, l_j) \rangle$$

   and $ADR(\mathcal{T}_j) = p_j$.

As usual we let

$$S(w) \ \vdash^* \ S'(w)$$

mean that $S'(w)$ can be reached from $S(w)$ by a finite number of transition steps.

So the Transition relation has four parts. If K_i is $DEAD$ in state S_i then it will no longer appear in state S_{i+1}. If K_i is $ALIVE$ but neither the current word nor the location-underdetermined argument can be accomodated, the alternative is declared $DEAD$ and, consequently, will not appear in S_{i+2}. If on the other hand the word cannot be processed but the extra argument can, in S_{i+2} the address of the argument will specify some definite node in the specified area. Finally, if the current word can be accomodated then the transformation that it is associated with it (through the Lexicon) is applied to the current partial tree. Notice that this is the only alternative that consumes a word ($g = 2$).

The activity of the actions in **Background** is hidden in the closure $\overline{\hat{\delta}(w)}$ of an action $\hat{\delta}(w)$. So the definition of a transition step does not involve the *order* in which **Background** actions are executed.

7 Conclusions

Dynamic Syntax seems to be very well suited to dealing with Traffic Accident Reports. The general framework of Dynamic Syntax can be restricted to the limited number of syntactic forms occurring in the reports. For instance, a large body of Dynamic Symtax deals with preposed and topicalized objects and with questions. These forms do not occur in the TARs and consequently the instruments of Dynamic Syntax dealing with these phenomena are not needed. On the other hand TARs use constructions that are ungrammatical from a wider perspective, for instance subjectless sentences, noun phrases without determiners, and the use of gerunds instead of tensed verbs. It has been straightforward to adapt Dynamic Syntax to deal with these idiosyncratic constructions.

In the present set-up we are able to deal with

- simple clauses: VI TURNS RIGHT FROM BLB RD.
- gerunds: V1 TURNS RIGHT HITTING CYCLIST
- relative clauses: TRAILER WHICH BECOMES DETACHED
- that trace: PEDESTRIAN DISTRACTED BY
- vp adjuncts: ATTEMPTS TO DO U TURN ACCROSS TOP OF ROADWORKS

At present we cannot deal in all generality with

- representation of temporal and causal relations between events
- many prepositions: INTO, FROM, ONTO,..
- coordinating constructions: AND, INSTEAD OF

In order to be able to hadle the above, more linguistic analysis is needed, especially for the coordinating constructions. Also the representation language needs to be extended, to cover causal and temporal relations. And thirdly, a large number of words need to be incorporated into the Lexicon. For future work, we also need to conduct a morphological analysis of the TAR words in order to reduce and organise the number of entries in the Lexicon.

The project report (8) discusses a protype implementation of a TAR parser which covers the following subset of word categories: adjective, intransitive verb (simple and continuous tenses), transitive verb (continuous and past tense), determiner, pronoun, preposition and common noun. This system parses succesfully simple TAR sentences and can deal with predictable text omissions. The restrictions of the present implementation are mainly due to lack of word implementation.

8 Acknowledgements

This work has been funded by EPSRC grant GR/L 26872, and many discussions have been held with the PI, Alex Poulovassilis. The author is grateful to Professor Benjamin Heydecker at UCL Centre for Transport Studies for discussions relating to the parsing of TARs, and for providing for this investigation sample text that is representative of the language of real traffic accident reports.

References

(1) J. Wu and B.G. Heydecker. Natural language understanding in road accident data analysis. In B.H.V.Topping (ed.) *Developments in Artificial Intelligence for Civil and Sturctural Engineering.* Civil-Comp Press, Edinburgh, 1995.

(2) J. Wu and B.G. Heydecker. Natural language understanding in road accident data analysis. *Advances in Engineering Software*, 29(7-9), 599-610.

(3) R. Kempson, W. Meyer-Viol. Topic and Focus: The Dynamics of Tree Growth. In Bouma. G, et al (eds.) *Constraints and Resources in Natural Language Syntax and Semantics.* CLSI, 2000.

(4) R. Kempson, W. Meyer-Viol, and D. Gabbay. VP ellipsis: towards a dynamic structural account. In Lappin, S. & Benmamoun, I. (eds), *Fragments: Studies in Ellipsis and Gapping.* Oxford University Press, Oxford.

(5) R. Kempson, W. Meyer-Viol, D. Gabbay. Syntactic Computation as Labelled Deduction: WH a case study. In Borsley, R. & Roberts, I. (eds.) *Syntactic Categories.* Academic Press, 2000.

(6) R. Kempson, W. Meyer-Viol, D. Gabbay. *Dynamic Syntax.* Blackwell, 2000.

(7) W. Meyer-Viol, Sequential Construction of Logical Forms. In Lecomte A. (ed), *Logical Aspects of Computational Linguistics*, Springer Lecture Notes in Computer Science, 1998.

(8) Willis, D., A Natural Language Parsing System using the Oracle Active DBMS. MSc Computing Science Project Report, Department of Computer Science, Birckbeck College, University of London, 2000.