Parsing as Sequential Construction of Logical Forms

Wilfried Meyer Viol

Abstract

In this paper we give a formal description of the parsing model that underlies the treatment of Long Distance Dependencies, Topic and Focus, Ellipsis and Quantification in, amongst others, the papers [1996],[1997], [1999a],[1998],[1999b]. In this model, a natural language string consists of a sequence of 'instructions packages' to construct some term in a formal representation language, the *logical form* of the string in question. Parsing, then, is the process of executing these packages in a left to right order.

Contents

Introduction

In this paper we will give a formal description of the parsing model that underlies the treatment of Long Distance Dependencies, Topic and Focus, Ellipsis and Quantification in, amongst others, the papers [1996],[1997],[1999a],[1998],[1999b]. Although the intuition behind the model is quite natural, nevertheless, it seems not to have been explored to any extent. The main idea is to view a natural language string $s = w_1 w_2 \dots w_n$ as projecting a sequence of 'instructions packages' to construct some term $\mathcal{T}$ in a formal representation language, this term being the supposed *interpretation* or *logical form* of the string in question. Parsing, then, is the process of executing these packages in a left to right order:

$$PARSE(s) = (\langle w_1, \mathcal{T}_1 \rangle, \dots \langle w_n, \mathcal{T}_n \rangle),$$

where each $\mathcal{T}_{i+1}$ is a *partial logical form*, the result of executing w_i on form $\mathcal{T}_i$, $\mathcal{T}_1$ is the result of processing w_1 on a starting form $\mathcal{T}_0$ and, provided s is grammatical, $\mathcal{T}_n$ is a *complete logical form.* The model we are going to introduce to formalize this process is shamelessly eclectic.
— The partial logical forms constitute the domain PT of a partially ordered Kripke frame $M = \langle PT, \leq \rangle$ on which the standard logical connectives and operators are given an intuitionistic interpretation.
— The partial logical forms in this frame are themselves represented as partial decorated binary trees $\mathcal{T}$ and the partial order $\leq$ reflects tree *growth.* On these trees the modal operators of the Logic of Finite Trees are interpreted. So the structure up to this point is an intuitionistic multi-modal frame $M = \langle P, \leq, \prec_i \rangle_{i \in I}$ and represents the space of partial logical forms.
— The parsing process is now a *goal-directed* movement through this space towards the set $LoFo \subset PT$ of complete Logical Forms. This goal-directedness will be represented by a *requirement function* R which adds to every node of a partial tree a (finite) set of requirements that have to be fulfilled (so, if $\mathcal{T} \in LoFo$ and n is a node in $\mathcal{T}$, then $R(n) = \emptyset$). Addition of a requirement function gives the structure $M = \langle P, \leq, \prec_i, R \rangle$.
— Finally, the words the words $w_1, \dots, w_n$ of a string natural language s and the general principles of that language will be represented as *incremental actions* which map partial logical forms to enriched ones. That is, on top of the intuitionistic modal frame with requirements, we now add a PDL like structure[1] of actions A such that $w_1, \dots, w_n \in A$. This gives our final structure $M = \langle PT, \leq, \prec_i, R, A \rangle$.

1 Terms as Decorated Trees

In order to handle partiality of logical forms in a flexible way, we represent the elements of each D_i as *decorated finite partial trees.* Logical forms are built up by one or more ways of putting together basic semantic entities.

[1]PDL stands for Propositional Dynamic Logic.

$$\underbrace{[_0[_0\ ]\ [_1[_1\ ]\ [_1\ [_0\ ][_1\ ]]]}_{Tree}$$
$$\underbrace{\{00\text{:}\mathbf{john},\ 011\text{:}\lambda x\lambda y\mathbf{read(x)(y)},\ 0111\text{:}\lambda P(\mathbf{some}P),\ 0110\text{:}(x,\mathbf{book}x)\}}_{Decorations}.$$

Figure 1: A Term as a decorated tree

Each of these modes of combination can be associated with a product type-constructor. A *term* in a language appropriate for these type-constructors can be represented as a *finite binary branching tree*, where every binary branching $\langle b_1, b_2\rangle$ reflects the presence of a subterm $O(b_1, b_2)$ for some operator binary O. As an example, let `APL` be the operation of *function application* in a typed lambda calculus. The sentence *John read a book*, represented by the formula $\mathbf{read}(\mathbf{john}, \mathbf{some}(x,\mathbf{book}(x)))$, can be seen as resulting from the unreduced lambda term

$$\texttt{APL}(\texttt{APL}(\lambda x\lambda y\mathbf{read}(y)(x), \texttt{APL}(\lambda P(\mathbf{some}P), (x, \mathbf{book}x)), \mathbf{john})$$

by β-reduction. In Figure 1. we have represented this term as a binary tree structure with an associated set of node decorations. (Here '$[_0$' means argument- and '$[_1$' function-daughter of the `APL` operator.) By representing terms in this way, we can address term *structure* and term *content* separately. In particular, we can represent partially specified trees that are, moreover, only partly decorated.
Such decorated tree structures have to be constructed in the course of a parse through an NL string. In order to deal with the partial logical forms arising during a parse we consider T-structures,

Definition 1 (T-Structures) A *T-structure* is a quintuple of the form $\mathcal{T} = \langle T, \prec_0, \prec_1, \prec_\downarrow, \prec_*\rangle$ where T is a non-empty domain of tree nodes and $\prec_i$,for $i \in I = \{0, 1, \downarrow, *\}$, is a (possibly empty) binary relation on T.
The set BT consists of those T-structures which are ordered as *binary trees*, where $\prec_0$ is the right-daughter and $\prec_1$ the left-daughter relation, $\prec_\downarrow$ is the immediate dominance relation ($\prec_\downarrow = \prec_0 \cup \prec_1$), and $\prec_*$ is the dominance relation, i.e., the reflexive and transitive closure of $\prec_\downarrow$.

Definition 2 (Partial Trees) A function f is a *Tr-morphism* from T-structure $\mathcal{T}$ in T-structure $\mathcal{T}'$ if it maps T in T' such that for all $n, m \in T$, for all $i \in I$: $n \prec_i m \Rightarrow f(n) \prec_i f(m)$. The set PT, of *partial trees*, consists of all T-structures $\mathcal{T}$ such that there is a Tr-morphism mapping $\mathcal{T}$ to an element of BT, i.e., a binary tree.

In a full-blown 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. The *under-specified* tree relations ($\prec_\downarrow$ and in particular $\prec_*$), play an essential role in constructing the term while traversing the string in a left to right fashion: we cannot always decide on the spot where *specifically* a certain subterm has to function in the eventual term. Given the string *A book John read*, we do not yet know what to do with *A book* at the start of the sentence.

$$\underbrace{[_0[_0\ ]\ [_*\ [_0\ ],[_1\ ]\ ]\ldots]}_{Tree} \quad \underbrace{\{00:\textbf{John}, 0*1:\lambda P(\textbf{some}P), 0*0:(x,\ \textbf{book}x)\}}_{Decorations}.$$

Figure 2: A Partial Term as a partial decorated tree

After having parsed *A book john* the partial logical form constructed is shown in Figure 2, which gives a partial tree model with an under-specified tree relation. This under-specified relation *constrains* the set of completions to those binary trees which have this relation witnessed by an immediate dominance sequence.

Partial trees are constructed in stages and node by node and we need a pointer to identify the nodes at which action is to take place, that is, our representations of partial logical forms consist of pairs $\langle \mathcal{T}, n\rangle$, which we will write as $\mathcal{T}n$, a partial tree $\mathcal{T}$ together with a *pointer* indicating some node $n \in T$.

Definition 3 (Structure of Pointed Partial Trees) Let $PPT = \{\mathcal{T}n \mid \mathcal{T} \in PT, n \in T\}$ be the set of *pointed partial trees*. We set $\mathcal{T}n \prec_i \mathcal{T}'n'$ if $\mathcal{T} = \mathcal{T}'$ and $n \prec_i n'$, and $\mathcal{T}n \leq \mathcal{T}', n'$ if there is a Tr-morphism $f : T \mapsto T'$ such that $f(n) = n'$. The frame of *Pointed Partial Tree structures* can now be defined as

$$\mathcal{PPT} = \langle PPT, \prec_i, \leq\rangle_{i\in\{0,1,\downarrow,*\}}.$$

Along $\leq$, a pair $n, m \in T$ such that $n \prec_* m$ may be mapped by Tr-morphism f to a pair such that $f(n) \prec_\downarrow f(m)$ and later, by some Tr-morphism g, to a *fully specified* relation $g(f(n)) \prec_0 g(f(m))$.

The Language DU

On Pointed Partial Tree Structures we can interpret the Language of Finite Trees, LFT (see [1]), a propositional modal language with the modalities $\langle 0\rangle\phi$ ("ϕ holds on the first daughter"), $\langle 1\rangle\phi$ ("ϕ holds on the second daughter"), $\langle\downarrow\rangle\phi$ ("ϕ holds on some daughter"), $\langle *\rangle\phi$ ("ϕ holds here or somewhere below"), $\langle L\rangle\phi$ ("ϕ holds on a linked node"), their converses $\langle i^{-1}\rangle$, and their universal variants $[\ i\ ], [\ i^{-1}\ ]$, for $i \in \{0, 1, \downarrow, *\}$. In the tree of Figure 1, for instance we have the following

— $\langle 0\rangle Fo(\textbf{John})$ and $\langle 1\rangle\langle 1\rangle\lambda x\lambda y\textbf{read}(x)(y)$ hold at the top node 0.

— $\langle\downarrow^{-1}\rangle\langle *\rangle Fo(x, \textbf{book}x))$ holds at node 00 decorated by $Fo(\textbf{John})$.

where the atomic formulas are designed to describe *Declarative Units* decorating binary (linked) tree structures. Declarative units are pairs consisting of a sequence of labels followed by a content formula. We have seen examples of content formulas in the denotations **john**, $(x, \textbf{Book}x)$ and $\lambda x\lambda y\textbf{read}(x)(y)$. The types e, t and $e \to t$ from these examples are instances of labels. The descriptions of declarative units determine the *atomic vocabulary*. So, our language has *monadic* predicates $La_1, \ldots La_n, Fo$, standing for n label dimensions and a formula dimension and individual constants from $D_{La_1}, \ldots D_{La_n}, D_{Fo}$ respectively, denoting values on these dimensions. The *atomic propositions* of the language then have the form $La_i(t)$ or $Fo(t)$ where t is either an element of the appropriate domain D_{La_i}, D_{Fo}, or it is a *meta variable.* A declarative unit can then be completely represented by a finite set of atomic propositions.

$$\{La_1(l_1), \ldots La_n(l_n), Fo(\Psi)\},$$

and a *partial* declarative unit, an object naturally arising in the course of a parse, is merely a subset of a description of a declarative unit.

The language, DU, we have settled on to describe partial Declarative units and their developments towards logical forms, includes the tree modalities from LFT, the standard Boolean constants and connectives and existential and universal quantifiers ranging over the set of label and formula *values.*

Definition 4 (The Representation Language DU) A proposition A of the language DU has one of the following shapes:

$$A \ ::= \top \mid \bot \mid La_1(l_1) \mid \ldots \mid La_n(l_n) \mid Fo(\phi) \mid Eq(t_1, t_2) \mid A \wedge A \mid A \vee A \mid \\ \mid A \rightarrow A \mid \exists \mathbf{x} A \mid \forall \mathbf{x} A \mid \langle \# \rangle A \mid [\#] A$$

for $VAR = \{\mathbf{x}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{y}, \ldots\}$ a denumerable set of individual variables, MV a denumerable set of *meta variables*, predicate values $l_i \in D_{La_i} \cup VAR \cup MV$, for each i: $1 \leq i \leq n$, ϕ a logical form in $D_{Fo} \cup Var \cup MV$, $t_1, t_2 \in D_{La_i} \cup D_{Fo} \cup VAR \cup MV$ and $\#$ a modality i or i^{-1} for $i \in \{0, 1, \downarrow, *\}$. The quantifier variables are rendered in boldface to distinguish them from the variable bound by quantifiers in the domain D_{Fo}, i.e., variables occurring in the logical forms under construction.

As is standard, this language is interpreted over Pointed Partial Trees by means of *Valuation* functions V.

Definition 5 (Pointed Partial Tree Models) A *Pointed Partial Tree Model* M is a pair $M = \langle \mathcal{PPT}, V \rangle$ consisting of a Pointed Partial Tree Structure $\mathcal{PPT}$ and a valuation V assigning finite sets of atomic formulas to elements $\mathcal{T}n \in PPT$, and satisfying the following principle

$$\mathcal{T}n \leq \mathcal{T}'n' \Rightarrow V(\mathcal{T}n) \subseteq V(\mathcal{T}'n').$$

This principle guarantees that once an atomic proposition has been established at some node in a partial tree, this proposition will remain to hold there throughout all future developments of that tree. These pointed partial tree will now be used to represent (unreduced) partial lambda terms as in Figures 1 and 2.

Definition 6 (Truth Definition for DU) Given a Pointed Partial Tree Model $M = \langle \mathcal{PPT}, V, \rangle$, a set $\mathcal{D} = \bigcup_{i \leq n} D_{La_i} \cup D_{Fo}$ of label and formula values, a set MV of meta variables, $t_1, t_2 \in \mathcal{D} \cup MV$ we say that pointed tree $\mathcal{T}n \in PPT$ of M *satisfies* formula ϕ, with the notation

$$\mathcal{T}n \models_M \phi,$$

if

$$\begin{array}{lll}
\phi \text{ is atomic and } \phi \in V(\mathcal{T}n) & & \\
\phi \neq \bot & & \\
\phi = \top & & \\
\phi = Eq(t_1, t_2) & \text{and} & t_1 = t_2 \\
\phi = \psi \wedge \chi & \text{and} & \mathcal{T}n \models_M \psi \;\&\; \mathcal{T}n \models_M \chi \\
\phi = \psi \vee \chi & \text{and} & \mathcal{T} \models_M \psi \text{ or } \mathcal{T}n \models_M \chi \\
\phi = \psi \to \chi & \text{and} & \text{for all } \mathcal{T}'n' : \mathcal{T}n \leq \mathcal{T}'n', \\
 & & \text{if } \mathcal{T}'n' \models_M \psi \text{ then } \mathcal{T}'n' \models_M \chi \\
\phi = \exists \mathbf{x}\psi & \text{and} & \text{there is a } t \in \mathcal{D} : \mathcal{T}n \models_M \psi[t/\mathbf{x}] \\
\phi = \forall \mathbf{x}\psi & \text{and} & \text{for all } \mathcal{T}'n' : \mathcal{T}n \leq \mathcal{T}'n' \text{ and all } t \in \mathcal{D} \\
 & & \mathcal{T}'n' \models_M \psi[t/\mathbf{x}]
\end{array}$$

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

$$\begin{array}{lll}
\phi = \langle i \rangle \psi & \text{and} & \exists \mathcal{T}'n' \in PPT : \mathcal{T}n \prec_i \mathcal{T}'n' \text{ and } \mathcal{T}'n' \models_M \psi \\
\phi = \langle i^{-1} \rangle \psi & \text{and} & \exists \mathcal{T}'n' \in PPT : \mathcal{T}'n' \prec_i \mathcal{T}n \text{ and } \mathcal{T}'n' \models_M \psi \\
\phi = [i]\psi & \text{and} & \text{for all } \mathcal{T}'n' : \mathcal{T}n \leq \mathcal{T}'n' \text{ and all } \mathcal{T}''n'' \in PPT \\
 & & \text{if } \mathcal{T}'n' \prec_i \mathcal{T}''n'' \text{ then } \mathcal{T}''n'' \models_M \psi \\
\phi = [i^{-1}]\psi & \text{and} & \text{for all } \mathcal{T}'n' : \mathcal{T}n \leq \mathcal{T}'n' \text{ and all } \mathcal{T}''n'' \in PPT \\
 & & \text{if } \mathcal{T}''n'' \prec_i \mathcal{T}'n' \text{ then } \mathcal{T}''n'' \models_M \psi
\end{array}$$

As usual, we can introduce negation by the definition

$$\neg\phi \equiv_{df} \phi \to \bot.$$

The operators and modalities with universal force (‘ $\to$′, ‘$\forall$′, ‘$[\#]$′) quantify not only over (nodes of the) current (partial) decorated trees, but over possible developments of the current structure. For instance, the top node of the current decorated partial tree need not be the *root* node of the eventual tree. Given that we have a falsum $\bot$ (satisfied by no node) and verum $\top$ (satisfied by all nodes) in our language we can ‘close off’ the top node by $\mathcal{T}n \models_M [\downarrow^{-1}] \bot$. This closing off is an operation that can take place on a tree the moment all words of the NL string have been processed: the node that happens to be the top one at that moment is turned into a root node. At the other end of the tree we can declare bottom nodes to be terminal nodes by annotating them with $[\downarrow]\bot$. This is a task of the lexical entries associated with the words: a word closes off a branch downwards.

By definition we have *persistence* of atomic DU-formulas. By the form of the Truth definition, this can be lifted to the whole of DU. So, if ϕ is a DU-formula, $\mathcal{T}n \models_M \phi$ and $\mathcal{T}n \leq \mathcal{T}'n'$, then $\mathcal{T}'n' \models_M \phi$.

It may be illuminating to view some typical interactions between the tree modalities and connectives, like implication, with universal force. In a model M we can have $\mathcal{T}n \models_M \langle * \rangle \phi$ without there being a sequence $\mathcal{T}n \prec_\downarrow \ldots \prec_\downarrow \mathcal{T}n'$ such that $\mathcal{T}n' \models_M \phi$. In a *partial* tree the relation $\prec_*$ between two nodes entails only that the path between them can always be completed to a fully specified one. So, on Pointed Partial Tree Models the basic logic of finite trees holds ‘under double negation’: if ϕ is an LFT theorem, i.e., $\models_{LFT} \phi$, then $\mathcal{T}n \models_{\mathcal{M}} \neg\neg\phi$ for every model M.

The meta variables can be distinguished from the proper values by the fact that only for a proper value $l_i \in D_{La_i}$ we have the satisfaction of $\exists \mathbf{x} La_i(\mathbf{x})$. $\exists \mathbf{x}\psi$

holds at a node if $\psi[t/\mathbf{x}]$ holds there for some $t \in \mathcal{D}$. That is, a meta variable $\mathbf{U}$ won't do as this is not an element of $\mathcal{D}$. We want the existential quantifier to be able express that some label or feature predicate has a proper value.
Apart from the LFT principles we will have to introduce axioms regulating the behaviour of the Fo and Ty predicates on the trees. For instance, a node may be annotated by at most one type. This requires a principle of the form

$$\forall \mathbf{x} \forall \mathbf{y} (Ty(\mathbf{x}) \wedge Ty(\mathbf{y}) \rightarrow \neg Eq(\mathbf{x}, \mathbf{y})).$$

Furthermore, the Fo and Ty values at the daughters of some node have to be related to the values on those predicates at the node itself

$$\forall \mathbf{x} \forall \mathbf{y} (\langle 0 \rangle Ty(\mathbf{x}) \wedge \langle 1 \rangle Ty(\mathbf{x} \rightarrow \mathbf{y}) \rightarrow Ty(\mathbf{y})).$$

2 Goal-Directedness

Within the domain PPT of a partial tree model $M = \langle \mathcal{PPT}, V \rangle$, we can identify the set $LoFo$ consisting of the decorated partial trees that correspond to (unreduced) terms of the typed lambda calculus. Grammatical strings, the words of which project actions mapping one pointed partial tree to a next one, must create elements of this subset of PPT. In all elements of $LoFo$, the root node, for instance, will be annotated by a lambda term of type t. Thus, in any partial stage, that root node will have a *requirement* that it be annotated by a lambda-term of type t, a sub-node of the root that it be annotated by a term of type e, and so on.[2] All nodes are introduced with requirements. Requirements form an essential feature of the tree - they determine a set of 'successful' extensions of a given (partial) tree, namely those in which all requirements are satisfied. Consequently, our basic data structures are tree structures, the nodes of which consist of (partial) declarative units paired with finite sets of requirements.
To model these requirements we add a *requirement function* R to the model assigning a finite number of (arbitrary) DU-formulas to elements $\mathcal{T}n$. These formulas represent a finite number of requirements on that node (as opposed to the *facts* at that node assigned by V). Figure 3 represents a decorated partial tree resulting from having parsed *John read*. This tree includes a node with a requirement for an object of type e (this node-plus-requirement is introduced, "sub-categorized for", by the verb **read**).

Definition 7 (Models with Requirements) A Pointed Partial Tree Model with *requirements* is a tuple $\mathcal{M} = \langle M, R \rangle$, where $M = \langle \mathcal{PPT}, V \rangle$ is a Pointed Partial Tree Model and R is a function assigning finite sets of DU formulas to elements of PPT and satisfying the following constraint:

$$\mathcal{T}n \leq \mathcal{T}'n' \Rightarrow R(\mathcal{T}n) \subseteq (Th(\mathcal{T}'n') \cup R(\mathcal{T}'n')),$$

[2] This use of requirements on the development of a tree node has some resemblance to the familiar concept of 'sub-categorisation', as a node decorated by the labelled formula $Ty(e \rightarrow (e \rightarrow t)))$, $Fo(\mathbf{read})$ within a tree may have a mother node which is decorated with a requirement $\langle 1 \rangle Ty(e)$ (that is, a requirement for an internal argument for '*read*').

$$\underbrace{[_0\ \ [_0\ \ ],\, [_1[_0],\, [_1]]]}_{Tree}\quad \underbrace{\{00:\mathbf{John}, 011:\lambda x\lambda y\mathbf{read}(\mathbf{x})(\mathbf{y}), 010:?\mathbf{Ty}(\mathbf{e})\}}_{Decorations}$$

Figure 3: Partial decorated tree with requirements

where $Th(\mathcal{T}n) = \{\phi \in DU \mid \mathcal{T}n \models_M \phi\}$. The successful developments, that is, the developments in which all requirements are satisfied, of a pointed partial tree $\mathcal{T}n$ we collect in the set $LoFo(\mathcal{T}n)$ of (supposed) Logical Forms into which $\mathcal{T}n$ can develop.

$$LoFo(\mathcal{T}n) = \{\mathcal{T}'n' \in PPT \mid \mathcal{T}n \leq \mathcal{T}'n' : R(\mathcal{T}'n') = \emptyset\}.$$

Unlike the valuation functions, a requirement function is not restricted to atomic propositions; we are free to require any (finite number of) DU-formula(s) at some tree node. Along the growth relation $\leq$ requirements may disappear, but only by becoming facts. For instance, we have

$$[_a\ \ ?Ty(e)\ ]\ \leq\ [_a\ \ Fo(\phi), Ty(e), ?Ty(e)\ ]\ \leq\ [_a\ \ Fo(\phi), Ty(e)\ ].$$

But also, (for $\mathbf{U}$ a meta variable),

$$[_a\ Fo(\mathbf{U}), ?\exists\mathbf{x}Fo(\mathbf{x}), ?Ty(e)\ ]\ \leq\ [_a\ Fo(\mathbf{U}), Fo(John), Ty(e)\ ].$$

Having introduced the concept of a requirement over Pointed Partial Tree Models we will exploit it by introducing some constants to the language DU which address the status of the requirements.

Definition 8 (Truth Definition for Requirements) Given a Pointed Partial Tree Model with *requirements*, $\mathcal{M} = \langle M, R\rangle$, we say that pointed tree $\mathcal{T}n$ of $\mathcal{M}$ *satisfies* formula ϕ, with the notation

$$\mathcal{T}n \models_{\mathcal{M}} \phi,$$

if

$\phi \in DU$	and	$\mathcal{T}n \models_M \phi$,
$\phi = ?\psi$	and	$\psi \in R(\mathcal{T}n)$
$\phi = ?\emptyset$	and	$R(\mathcal{T}n) = \emptyset$
$\phi = \mathbf{F}\psi$	and	$\exists \mathcal{T}'n' \in LoFo(\mathcal{T}n) : \mathcal{T}'n' \models_{\mathcal{M}} \psi$
$\phi = \mathbf{G}\psi$	and	$\forall \mathcal{T}'n' \in LoFo(\mathcal{T}n) : \mathcal{T}'n' \models_{\mathcal{M}} \psi$

A requirement $?\phi$ holds if the formula ϕ occurs on the the requirement list of that node, and $?\emptyset$ is a constant which holds at a node if it has an empty requirement list. Proposition $\mathbf{F}\psi$ holds at node $\mathcal{T}n$ if there is at least one development of $\mathcal{T}n$ to (a node in) a logical form where ψ holds, and $\mathbf{G}\psi$ holds at node $\mathcal{T}n$ if ψ holds at all developments of $\mathcal{T}n$ to logical forms.
A tree node $\mathcal{T}n$ has requirements that can be satisfied iff $\mathbf{F}\top$ holds at $\mathcal{T}n$. This allows us to define a conditional: $\phi \rightarrow_{gr} \psi \equiv_d (\mathbf{F}\top \wedge \phi) \rightarrow \psi$. This conditional expresses invariances shared only by *successful* developments.
If the procedure that leads from Axiom to an element of $LoFo$ is *sound*, then the final tree represents, is *isomorphic* to, an unreduced term in the language of our logical forms.

The uses of requirements

Consider all terms of the typed lambda calculus we use to formulate our logical forms, the representations of the formal meanings of the natural Language strings under consideration. The elements of this set are our target structures the NL strings have to construct. We represent these terms as trees by their applicative structure (that is, a subterm $(\lambda x\phi, \psi)$ is represented as a binary branching point with $\lambda x\phi$ annotating the function - and ψ the argument daughter). At the nodes of these representations we hang empty sets. These empty sets represent empty sets of requirements. This we now turn into the set $LoFo$ of some Pointed Partial Tree Model $\mathcal{M}$, by adding all *partial* versions that can be extended to elements of the $LoFo$. Now, in order to *guarantee* that the set $LoFo$, defined in terms of fulfilled requirements, and the set of representations of terms in our typed lambda calculus coincide, every abstraction of such a term to a partial object has to be compensated by the introduction of a requirement. Starting, for instance, from the term $\mathbf{read}(\mathbf{John}, (\mathbf{a}, x, \mathbf{book}x))$ we can create a partial term by abstraction over **John**. This abstraction is not a term of our typed lambda calculus, so it should not belong to $LoFo$. By definition, this means that there must be some unfulfilled requirement associated with it. But this does not have to be a requirement for exactly '**John**'. It may (and, in fact, will) be merely a requirement for type e. So here is where the invariances of the process come in.

Given a specific feature of the logical forms we are interested in, the first question is now always: can we devise a system of requirement introduction such that the fulfillment of all requirements annotating a given tree corresponds exactly to a completing this tree to a term of our typed Lambda Calculus with the desired feature?. We will give an example of a requirement strategy to the effect that the set $LoFo$ coincides with the set of a binary tree structures. That is, no under-specified tree relations of the form $\prec_*$ or $\prec_\downarrow$ are left in $LoFo$ that are not the reflexive transitive closure of the immediate dominance relation or the union of argument and function daughter relation, respectively. The idea is to introduce a *Tree Node* label, a monadic predicate 'Tn' with values in $D_{Tn} = \{a \cdot x, x \mid a \in A, x \in \{0, 1, L\}^*\}$. That is, a value of this predicate is a finite sequence of elements of $\{0, 1, L\}$ possibly preceded by a constant from a set A. When trees are constructed in the parsing process, in general it is not known whether a description that starts off as a top node will remain so (and thus be the *root* node of the eventual tree. This is why we introduce a new node invariably with 'address" $a \in A$, satisfying a formula $Tn(a)$, a is a constant not yet occurring in the construction. In interaction with the tree modalities various constellations are expressible. So, given the formula $Tn(a)$, expressing the location of a node description in the tree under construction, we can fix

$$Tn(a0) \leftrightarrow \langle 0^{-1}\rangle Tn(a),$$

$$Tn(a1) \leftrightarrow \langle 1^{-1}\rangle Tn(a),[3]$$

and we can fix the root node of a tree as follows,

[3]We will also need $Tn(aL) \leftrightarrow \langle L^{-1}\rangle Tn(a)$ when we consider Linked Trees.

$Tn(0) \leftrightarrow [\downarrow^{-1}]\bot.$

(Significantly, we do *not* 'internalize' the under-specified modalities $\langle * \rangle$ and $\langle \downarrow \rangle$ as values of the Tn predicate.) The Tree Node formula $Tn(0)$ holds at a node if it is a top node and remains so throughout all developments. (Note the use of the 'falsum' - "At every node above the current one $\bot$ holds." As $\bot$ is satisfied by no node at all (Definition 6) this means that there are no nodes above the current one.)
Now when we introduce a tree node with an under-specified relation to the source node, as we do in Figure 5, we add the requirement for $\exists \mathbf{x} Tn(\mathbf{x})$ to the node with under-specified address:

$$\frac{[_a\ ?Ty(t)], a}{[_a\ ?Ty(t), [_*\ ?Ty(e), ?\exists \mathbf{x} Tn(\mathbf{x})]]}$$

The point is that this requirement can only be satisfied when the node it decorates is merged with one that has a fully specified relation to the top node of the tree: only in that case will there be a $t \in D_{Tn}$ such that $Tn(t)$ holds at the node. So if, starting from the Axiom, we introduce tree nodes with under-specified tree *only* if they are accompanied by requirements for values on the Tn predicate, then in all elements of $LoFo$ that we can reach all underspecification with respect to tree relations will have been resolved.

As a second example, an analogous mechanism allows us to introduce meta variables or *placeholder values* projected by *pronominals* with the requirement that they have to be substituted by a proper value in order for the term to be complete.

$$\frac{[_a\ ?Ty(e)\ ]}{[_a\ Fo(\mathbf{U}), Ty(e), ?\exists \mathbf{x} Fo(\mathbf{x})\ ]}$$

An object of type e has been supplied, but a new, weaker, requirement has taken its place. Only a development that supplies a concrete value for the meta variable $\mathbf{U}$ in $Fo(\mathbf{U})$ can end up in $LoFo$.

3 Actions

In a final enrichment we now extend the Pointed Partial Tree models with requirements by a set A of *actions*. An action α is an element of $A \subseteq PPT^{PPT}$, that is, actions are *functions* 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* in the sense that $\mathcal{T}'n' = \alpha(\mathcal{T}n)$ entails that either $\mathcal{T}n \leq \mathcal{T}'n'$ or $\mathcal{T} = \mathcal{T}'$. So an incremental action is either some *construction* or a *pointer movement*. The actions are put together from *basic* or *atomic* actions. The basic actions consist of: creation of new nodes relative to old ones; decoration of nodes, moving to nodes, and substitution at nodes. These action may now be combined to complex ones by, essentially, the PDL operations.[4]

[4] PDL is Propositional Dynamic Logic.

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

Basic Actions

1. $1 = \{\langle \mathcal{T}n, \mathcal{T}n\rangle \mid \mathcal{T}n \in \mathcal{M}\}$,
   $AB = \emptyset$.
   These represent the halting action and the Abort action respectively.
2. $\texttt{make}(\#)$. This action creates a node $\mathcal{T}'n'$ such $\mathcal{T}' = \mathcal{T} \cup \{n'\}$ and that $\mathcal{T}'n \prec_{\#} \mathcal{T}'n'$.
3. $\texttt{go}(\#)$. Here $\texttt{go}(\mathcal{T}n) = (\mathcal{T}'n')$ implies $\mathcal{T}n \prec_{\#} \mathcal{T}'n'$.
4. $\texttt{put}(\phi)$. Here, $\texttt{put}(\phi)(\mathcal{T}n) = \mathcal{T}'n'$ implies that $\mathcal{T} = \mathcal{T}'$ except that $V(\mathcal{T}'n') = V(\mathcal{T}n) \cup \{\phi\}$ if ϕ is atomic and $R(\mathcal{T}'n') = R(\mathcal{T}n) \cup \{\psi\}$ if $\phi = ?\psi$.

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 '$+$').

5. 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.

7. If Σ is a set of formulas all free variables of which occur in $\overline{\mathbf{x}}$ and α, α' are actions then $\langle \Sigma(\overline{\mathbf{x}}), \alpha, \alpha'\rangle$ is a *conditional* action with the definition:

$$\langle \Sigma(\overline{\mathbf{x}}), \alpha, \alpha'\rangle =$$
$$\{\langle \mathcal{T}n, \mathcal{T}'n'\rangle \in \alpha[\overline{t}/\overline{\mathbf{x}}] \mid \overline{t} \in (\mathcal{D} \cup MV)^*, \mathcal{T} \models_{\mathcal{M}} \Sigma[\overline{t}/\overline{\mathbf{x}}]\} \cup$$
$$\{\langle \mathcal{T}n, \mathcal{T}'n'\rangle \in \alpha' \mid \neg\exists \overline{t} \in (\mathcal{D} \cup MV)^*, \mathcal{T}n \models_{\mathcal{M}} \Sigma[\overline{t}/\overline{\mathbf{x}}\}.$$

That is, if Σ holds at $\mathcal{T}n$, for some substitution of t for variable $\mathbf{x}$, then action α is executed where in the body of this action the variable $\mathbf{x}$ is also replaced by t. If Σ does nor hold at $\mathcal{T}n$ for any substitution for $\mathbf{x}$, then action α' is executed.[5] For instance we can define actions which transport features from one node (e.g. a 'head') to another.

- $\langle \{Fo(\mathbf{x})\}, \texttt{go}(\langle 0\rangle); \texttt{put}(Fo(\mathbf{x})), AB\rangle$ maps the value of the Fo feature at the current node, if there is such a value, to the Fo feature at the left daughter, otherwise it aborts.

We may also define an action that enables pointer movement to the closest clausal node:

- $\texttt{gofirst}(?Ty(t)) = \langle \{?Ty(t)\}, 1, \texttt{go}(\langle \downarrow^{-1}\rangle)\rangle^*; \langle \{?Ty(t)\}, 1, AB\rangle$.
  The action $\texttt{gofirst}(X)$ moves the pointer upwards to the first higher node with annotation X, i.c. the requirement $Ty(t)$, and then stops.

The actions projected by the words of a language are conditional statements like

- *John* = IF $\{?Ty(e)\}$
  THEN $\texttt{put}(Fo(\mathbf{john}), Ty(e), [\downarrow]\bot)$
  ELSE AB

[5] The PDL tests do not suffice over this propositional logic because we want the ability to formulate IF THEN ELSE statements and only in the context of excluded middle can these be defined in terms of the PDL test.

The word test whether the pointed node requires an object of type e. If so, then it places the Formula *John* there and closes off the node downwards.
We have two 'pure' classes of actions. Complex *constructions* (that is, compositions of actions without the `go` action) induce complex *tree growth.* Complex *movements* on the other hand (the family of $\texttt{go}(\langle\#i\rangle)$ instructions closed under the given operations) represent *pointer strategies.* In general, for instance, in case of actions projected by words, the actions are a mixture of these classes.

4 The Parsing Process

The object of the parsing process is to construct a binary tree structure the top node of which is decorated by a formula of type t while using all information in an NL string. The minimal element in the model $\mathcal{M}$, the starting point of every parse[6] is the *Axiom*

Axiom: $$[_a \ ?Ty(t)], a.$$

The Axiom is the Pointed Partial Tree Model consisting of a single node (thus the location of the pointer), the putative top node, with empty valuation function and requirement for an object of type t.

In the course of a parse starting from the Axiom the description of a tree is created. A parse ends essentially after the last word of an NL string has been processed. A *successful* parse of a *grammatical* NL string must result in a logical form, that is, a Pointed Binary Tree in the set *LoFo* of $\mathcal{M}$. That is, it must end with a Tree where all nodes are associated with an empty list of requirements.

Goal $$[_a, \ldots, Ty(t)\ ], [_{a0} \ldots\ ], [_{a1} \ldots\ ] \ldots], a \in LoFo.$$

If the procedure that leads from Axiom to an element of *LoFo* is *sound*, then the final tree represents, is *isomorphic* to, an unreduced term in the language of our logical forms. Now, this (representation of an) unreduced lambda term may be normalized in ways depending on a variety of labels collected on the tree during the parse.

Figure 4 shows the structures arising in the course of parsing "*John read a book*". Notice that each new structure is a development under the $\leq$ relation.

The course from *Axiom* to *Goal* must be licensed by the natural language under consideration within the context of some language external, pragmatic, mechanisms. That is, with a natural language $\mathcal{L}$ there are associated three sets of actions.
— A set C of **Computational rules**. These are actions which bring out information contained in the current Tree and make hypotheses about structure.
— A set L **Lexical actions**. These actions map one Tree to a next one adding information in the process. The elements of L are projected by the Natural Language *words.* Lexical transition are defined as conditional actions of the IF

[6]of an NL string in isolation, i.e., without a context.

$$\begin{array}{lll}
 & [_a\ ?Ty(t)], a & \mathit{Axiom} \\
\Rightarrow_C & [_a\ [_0\ ?Ty(e)], [_1\ ?Ty(e \rightarrow t)]], a0 & \downarrow \\
\Rightarrow_L & [_a\ [_0\ , Fo(\mathbf{John}), Ty(e)], [_1\ ]], a1 & \texttt{John} \\
\Rightarrow_L & [_a\ [_0\ ], [_1\ [_1\ Fo(\lambda x \lambda y \mathbf{read}(x)(y)), Ty(e \rightarrow (e \rightarrow t))], [_0\ ?Ty(e)]]], a10 & \texttt{read} \\
\Rightarrow_{LC} & [_a[_0\ ], [_1[_1\ \ ], [_0\ Fo((\mathbf{a}, x, \mathbf{book}x), Ty(e)]]], a10 & \texttt{a book} \\
\Rightarrow_C & [_a\ [_0\ [_1\ Fo(\lambda y \mathbf{read})(\mathbf{a}, x, \mathbf{book}x)(y), Ty(e \rightarrow t), [_1\ ], [_0\ ]]], a1 & \downarrow \\
\Rightarrow_C & [_a\ \mathbf{read}(\mathbf{a}, x, \mathbf{book}x)(\mathbf{John}), Ty(t), [_0\ ], [_1\ [_1\ ][_0\ ]]], a & \in \mathit{LoFo}
\end{array}$$

Figure 4: Pointed Partial Tree Models arising in the course of parsing "`John read a book.`" At every transition only the new aspects are highlighted.

THEN ELSE variety. A lexical transitions tests IF the some finite set of formulas (the *condition*) holds at the Node Description where the pointer is located — this may include the presence (or absence) of labels at that node, modal statements about decorations located at Descriptions 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., the condition does not hold) a second action is undertaken, usually an 'abort' action.
— A set P of **Pragmatic actions**. These add information to a Tree Description which is neither contained in the natural language string, nor follows from language specific principles. These are actions of two sorts, (i) substitution processes which "enrich" aspects of the tree structure description as it is projected, primarily by processes of substitution replacing some lexical

A *configuration* is now a pair

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

of a Pointed Partial Tree $\mathcal{T}n$ and a string s of lexical actions, i.e., $s \in L^*$. The basic *rewrite* relation of our parsing model is the binary relation on configurations,

$$(\mathcal{T}n, s) \Rightarrow_{LCP} (\mathcal{T}'n', s')$$

defined by, either $s = \alpha s'$ for $\alpha \in L$ and $\alpha(\mathcal{T}n) = \mathcal{T}'n'$, or $s = s'$ and there is some rule $\rho \in C \cup P$ such that $\rho(\mathcal{T}n) = \mathcal{T}'n'$.

As usual we let $\Rightarrow^*$ be the reflexive and transitive closure of $\Rightarrow$.

Definition 10 (Grammatical Strings) The set of *grammatical strings*, given C, L and P, can be defined as:

$\mathcal{L}(L, C, P) = \{s \in L^* \mid \exists \mathcal{T}n \in \mathit{Goal} : (AXIOM, s) \Rightarrow^*_{LCP} (\mathcal{T}n, e)\}$.

A natural language must be able to project any logical form in our representation language. That is, language $\mathcal{L}(C, L, P)$ is *expressively adequate* (EA) if for every $\mathcal{T}n \in$ *Goal* there is a $s \in L^*$: $(AXIOM, s) \Rightarrow^*_{LCP} (\mathcal{T}n, e)$.

The progress from Axiom to Goal is non-deterministic: at every state of the parse the word currently under consideration can generally be assigned more than one structural role in the Tree Description. That is, the path from Axiom to Goal is one through a space of *alternative parse courses*. For instance, an

$$\begin{array}{lll}
 & [_a\ ?Ty(t)], a & \mathit{Axiom} \\
\Rightarrow_C & [_a\ ?Ty(t), [_*\ ?Ty(e)]], a* & \downarrow \\
\Rightarrow_{LC} & [_a\ ?Ty(t), [_*\ (\mathbf{a}, x, \mathbf{book}x), Ty(e)]], a & \texttt{a book} \\
\Rightarrow_C & [_a\ [_0\ ?Ty(e)], [_1\ ], [_*\ ]], a0 & \downarrow \\
\Rightarrow_L & [_a\ [_0\ , Fo(\mathbf{John}), Ty(e)], [_1\ ?Ty(e \rightarrow t)], [_*\ ]], a1 & \texttt{John} \\
\Rightarrow_L & [_a\ [_0\ ], [_1\ [_1\ Fo(\lambda x \lambda y \mathbf{read}(x)(y)), Ty(e \rightarrow (e \rightarrow t))], [_0\ ?Ty(e)]][_*\ ]], a10 & \texttt{read} \\
\Rightarrow_C & [_a[_0\ ], [_1[_1\ \ ], [_0\ Fo(\mathbf{a}, x, \mathbf{book}x), Ty(e)]]], a10 & * := 10 \\
\Rightarrow_C & [_a\ [_0\ [_1\ Fo(\lambda y \mathbf{read})(\mathbf{a}, x, \mathbf{book}x)(y), Ty(e \rightarrow t), [_1\ ], [_0\ ]]], a1 & \downarrow \\
\Rightarrow_C & [_a\ \mathbf{read}(\mathbf{a}, x, \mathbf{book}x)(\mathbf{John}), Ty(t), [_0\ ], [_1\ [_1\ ][_0\ ]]], a & \in \mathit{LoFo}
\end{array}$$

Figure 5: Pointed Partial Tree Models arising in the course of parsing "`A book John read.`" At every transition only the new aspects are highlighted.

NP heading an NL string may end up as subject *John read a book*, it may be a fronted object, *A book John read*, or it may be a topic constituent, *(As for) a book John read it*. The first two possibilities are worked out in Figures 4 and 5 respectively.
Notice that parse of "*John read a book*" and "*A book John read*" end with the same logical form, but this form is reached starting from the Axiom along different routes. Thus the Axiom must be able to access a (finite!) number of alternative partial Tree Descriptions to accommodate the first NP of the sentence.

5 Conclusion

In papers and books cited in the references (especially the forthcoming [1999b]) applications of this model are found involving a wide variety of linguistic phenomena in a number of languages. Of course, the main difference between the approach to parsing suggested by the model in this paper and the more familiar approaches based on a Montagovian framework is the absence of a syntactic algebra and thus of a homomorphic relation between syntactic and semantic structure. A natural language string is seen as a sequence of PDL programs and any syntactic structure in this sequence derives from processing constraints: a word occurring in an NL string leaves the parser with a given partially constructed logical form together with a pointer; the syntactic correctness of a subsequent word then comes down to whether or not the current logical form satisfies the condition of that word.

References

[1994] P. Blackburn & W. Meyer Viol 1994, Linguistics, logic and finite trees, *Bulletin of IGPL*.

[1996] Kempson, R., W Meyer Viol and D. Gabbay: Language Understanding: a Procedural Perspective, in C. Retore (ed.), *Logical Aspects of Com-*

putational Linguistics, First International Conference, LACL 1996,228–247. Lecture Notes in Computer Science Vol 1328, Springer Verlag.

[1997] Meyer Viol, W., R. Kibble, R. Kempson and D. Gabbay (1997) 'Indefinites as Epsilon Terms: A Labelled Deduction Account,' in H. Bunt and R. Muskens (eds.) Computing Meaning: Current Issues in Computational Semantics. Kluwer Academic Publishers. Dordrecht and Boston.

[1999a] Kempson, R., W Meyer Viol and D. Gabbay, VP-ellipsis: towards a dynamic structural account. In S. Lappin, E. Benmamoun, 1999, *Fragments: studies in ellipsis and gapping*, OUP.

[1998] [5] Kempson, R., and Meyer Viol, W., Topic and Focus Structures: the Dynamics of Tree Growth, in *Proceedings of the 4d Formal Grammar Conference*, Saarbruecken, 1998.

[1999b] Kempson, R. Meyer-Viol, W. and Gabbay, D. *Dynamic Syntax*. Blackwell's, Oxford, 1999.