These notes cover the second part of an introductory course on computational linguistics, also known as **MPRI 2-27-1: Logical and computational structures for linguistic modelling**. The course is subdivided into two parts: the first, taught this year by Éric Villemonte de la Clergerie, covers grammars and automata for syntax modelling, while the second part focuses on logical approaches to syntax and semantics. Among the prerequisites to the course are

- classical notions of formal language theory, in particular regular and context-free languages, and more generally the Chomsky hierarchy,
- a basic command of English and French morphology and syntax, in order to understand the examples;
- some acquaintance with logic and proof theory is also advisable.

These notes are based on numerous articles—and I have tried my best to provide stable hyperlinks to online versions in the references—, and on the excellent material of Benoît Crabbé, Éric Villemonte de la Clergerie, and Philippe de Groote who taught this course with me.

Several courses at MPRI provide an in-depth treatment of subjects we can only hint at. The interested student should consider attending

**MPRI 1-18: Tree automata and applications**: tree languages and term rewriting systems will be our basic tools in many models;

**MPRI 2-16: Finite automata modelisation**: only the basic theory of weighted automata is used in our course;

**MPRI 2-26-1: Web data management**: you might be surprised at how many concepts are similar, from automata and logics on trees for syntax to description logics for semantics.

## Contents

1. **Model-Theoretic Syntax**
   1.0.1 Model-Theoretic vs. Generative .......................... 5
   1.0.2 Tree Structures ........................................ 6
   1.1 Monadic Second-Order Logic ............................... 7
   1.1.1 Linguistic Analyses in wMSO ......................... 9
   1.1.2 wS2S .................................................. 12
   1.2 Propositional Dynamic Logic ............................ 13
   1.2.1 Model-Checking ..................................... 14
   1.2.2 Satisfiability ..................................... 14
   Fisher-Ladner Closure .................................. 15
   Reduced Formulae ...................................... 16
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.3</td>
<td>Expressiveness</td>
<td>18</td>
</tr>
<tr>
<td>1.3</td>
<td>Parsing as Intersection</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>First-Order Semantics</td>
<td>21</td>
</tr>
<tr>
<td>2.1</td>
<td>Formal Semantics</td>
<td>21</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Event Semantics</td>
<td>22</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Thematic Roles</td>
<td>23</td>
</tr>
<tr>
<td>2.2</td>
<td>A Dip into Description Logics</td>
<td>24</td>
</tr>
<tr>
<td>2.2.1</td>
<td>A Basic Description Logic</td>
<td>24</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Translation into First-Order Logic</td>
<td>25</td>
</tr>
<tr>
<td>2.3</td>
<td>Modal Semantics</td>
<td>26</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Background: Modal Logic</td>
<td>26</td>
</tr>
<tr>
<td>2.3.2</td>
<td>First-Order Modal Logic</td>
<td>29</td>
</tr>
<tr>
<td>2.4</td>
<td>Decidability</td>
<td>30</td>
</tr>
<tr>
<td>2.4.1</td>
<td>The Guarded Fragment</td>
<td>31</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Guarded Bisimulations</td>
<td>31</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Models of Bounded Treewidth</td>
<td>32</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Limitations &amp; Extensions</td>
<td>33</td>
</tr>
<tr>
<td>3</td>
<td>Tree Patterns</td>
<td>35</td>
</tr>
<tr>
<td>3.1</td>
<td>Background: Existential First-Order Logic</td>
<td>35</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Characterisations over Finite Models</td>
<td>36</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Tree Models</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>Meta-Grammars</td>
<td>38</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Diathesis Alternation</td>
<td>38</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Complexity</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>Underspecified Semantics</td>
<td>40</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Scope Ambiguities</td>
<td>40</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Hole Semantics</td>
<td>41</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Constructive Satisfiability</td>
<td>42</td>
</tr>
<tr>
<td>4</td>
<td>Higher-Order Semantics</td>
<td>47</td>
</tr>
<tr>
<td>4.1</td>
<td>Compositional Semantics</td>
<td>47</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Background: Simply Typed Lambda Calculus</td>
<td>48</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Ground Terms over Second-Order Signatures</td>
<td>49</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Higher-Order Homomorphisms</td>
<td>51</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Tree Transductions</td>
<td>52</td>
</tr>
<tr>
<td>4.2</td>
<td>Intensionality</td>
<td>54</td>
</tr>
<tr>
<td>4.3</td>
<td>Higher-Order Logic</td>
<td>56</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Background: Church’s Simple Theory of Types</td>
<td>56</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Type-Logical Semantics</td>
<td>57</td>
</tr>
<tr>
<td>5</td>
<td>References</td>
<td>61</td>
</tr>
</tbody>
</table>

### Notations

We use the following notations in this document. First, as is customary in linguistic texts, we prefix agrammatical or incorrect examples with an asterisk, like *ationhospitalmis or *sleep man to is the.
These notes also contain some exercises, and a difficulty appreciation is indicated as a number of asterisks in the margin next to each exercise—a single asterisk denotes a straightforward application of the definitions.

**Relations.** We only consider binary relations, i.e. subsets of $A \times B$ for some sets $A$ and $B$. The inverse of a relation $R$ is $R^{-1} = \{(b, a) \mid (a, b) \in R\}$, its domain is $R^{-1}(B)$ and its range is $R(A)$. Beyond the usual union, intersection and complement operations, we denote the composition of two relations $R_1 \subseteq A \times B$ and $R_2 \subseteq B \times C$ as $R_1 \circ R_2 = \{(a, c) \mid \exists b \in B, (a, b) \in R_1 \land (b, c) \in R_2\}$. The reflexive transitive closure of a relation is noted $R^* = \bigcup_i R^i$, where $R^0 = \text{Id}_A = \{(a, a) \mid a \in A\}$ is the identity over $A$, and $R^{i+1} = R \circ R^i$.

**Terms.** A ranked alphabet a pair $(\Sigma, r)$ where $\Sigma$ is a finite alphabet and $r : \Sigma \to \mathbb{N}$ gives the arity of symbols in $\Sigma$. The subset of symbols of arity $n$ is noted $\Sigma_n$.

Let $\mathcal{X}$ be a set of variables, each with arity 0, assumed distinct from $\Sigma$. We write $\mathcal{X}_n$ for a set of $n$ distinct variables taken from $\mathcal{X}$.

The set $T(\Sigma, \mathcal{X})$ of terms over $\Sigma$ and $\mathcal{X}$ is the smallest set s.t. $\Sigma_0 \subseteq T(\Sigma, \mathcal{X})$, $\mathcal{X} \subseteq T(\Sigma, \mathcal{X})$, and if $n > 0$, $f$ is in $\Sigma_n$, and $t_1, \ldots, t_n$ are terms in $T(\Sigma, \mathcal{X})$, then $f(t_1, \ldots, t_n)$ is a term in $T(\Sigma, \mathcal{X})$. The set of terms $T(\Sigma, \emptyset)$ is also noted $T(\Sigma)$ and is called the set of ground terms.

A term $t$ in $T(\Sigma, \mathcal{X})$ is linear if every variable of $\mathcal{X}$ occurs at most once in $t$. A linear term in $T(\Sigma, \mathcal{X}_n)$ is called a context, and the expression $C[t_1, \ldots, t_n]$ for $t_1, \ldots, t_n$ in $T(\Sigma)$ denotes the term in $T(\Sigma)$ obtained by substituting $t_i$ for $x_i$ for each $1 \leq i \leq n$, i.e. is a shorthand for $C\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$. We denote $C^n(\Sigma)$ the set of contexts with $n$ variables, and $C(\Sigma)$ that of contexts with a single variable—in which case we usually write $\Box$ for this unique variable.

**Trees.** By tree we mean a finite ordered ranked tree $t$ over some set of labels $\Sigma$, i.e. a partial function $t : \{0, \ldots, k\}^* \to \Sigma$ where $k$ is the maximal rank, associating to a finite sequence its label. The domain of $t$ is prefix-closed, i.e. if $ui \in \text{dom}(t)$ for $u \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $u \in \text{dom}(t)$, and predecessor-closed, i.e. if $ui \in \text{dom}(t)$ for $u \in \mathbb{N}^*$ and $i \in \mathbb{N}_{>0}$, then $u(i - 1) \in \text{dom}(t)$.

The set $\Sigma$ can be turned into a ranked alphabet simply by building $k+1$ copies of it, one for each possible rank in $\{0, \ldots, k\}$; we note $a^{(m)}$ for the copy of a label $a$ in $\Sigma$ with rank $m$. Because in linguistic applications tree node labels typically denote syntactic categories, which have no fixed arities, it is useful to work under the convention that $a$ denotes the “unranked” version of $a^{(m)}$. This also allows us to view trees as terms (over the ranked version of the alphabet), and conversely terms as trees (by erasing ranking information from labels)—we will not distinguish between the two concepts.

**Term Rewriting Systems.** A term rewriting system over some ranked alphabet $\Sigma$ is a set of rules $R \subseteq (T(\Sigma, \mathcal{X}))^2$, each noted $t \to t'$. Given a rule $r : t \to t'$ (also noted $t \xrightarrow{r} t'$), with $t, t'$ in $T(\Sigma, \mathcal{X}_n)$, the associated one-step rewrite relation over $T(\Sigma)$ is $\Rightarrow = \{[C[t\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}], C'[t'\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}] \mid C \in C(\Sigma), t_1, \ldots, t_n \in T(\Sigma)\}$. We write $\xrightarrow{\text{R}}$ for $\Rightarrow$, and $\xrightarrow{\text{R}}$ for $\bigcup_{r \in R} \xrightarrow{\text{R}}$. See Comon et al. (2007) for missing definitions and notations.
Chapter 1

Model-Theoretic Syntax

In contrast with the generative approaches of the first part of the course, we take here a different stance on how to formalise constituent-based syntax. Instead of a more or less operational description using some string or term rewrite system, the trees of our linguistic analyses are now models of logical formulæ.

1.0.1 Model-Theoretic vs. Generative

The connections between the classes of tree structures that can be singled out through logical formulæ on the one hand and context-free grammars or finite tree automata on the other hand are well-known, and we will survey some of these bridges. Thus the interest of a model theoretic approach does not reside so much in what can be expressed as in how it can be expressed.

Local vs. Global View  The model-theoretic approach simplifies the specification of global properties of syntactic analyses. Let us consider for instance the problem of finding the head of a constituent, which can be used to lexicalise CFGs. Remember that the solution there was to explicitly annotate each nonterminal with the head information of its subtree—which is the only way to percolate the head information up the trees in a context-free grammar. On the other hand, one can write a logic formula postulating the existence of a unique head word for each node of a tree (see (1.19) and (1.20)).

Gradience of Grammaticality  Agrammatical sentences can vary considerably in their degree of agrammaticality. Rather than a binary choice between grammatical and agrammatical, one would rather have a finer classification that would give increasing levels of agrammaticality to the following sentences:

* In a hole in the ground there lived a hobbit.
* In a hole in ground there lived a hobbit.
* Hobbit a ground in lived there a the hole in.

One way to achieve this finer granularity with generative syntax is to employ weights as a measure of grammaticality. Note that it is not quite what we obtained through probabilistic methods, because estimated probabilities are not grammaticality judgements per se, but occurrence-based (although smoothing techniques attempt to account for missing events).

A natural way to obtain a gradience of grammaticality using model theoretic methods is to structure formulæ as large conjunctions $\bigwedge \varphi_i$, where each conjunct...
\( \varphi_i \) implements a specific linguistic notion. A degree of grammaticality can be derived from (possibly weighted) counts of satisfied conjuncts.

**Open Lexicon** An underlying assumption of generative syntax is the presence of a finite lexicon \( \Sigma \). A specific treatment is required in automated systems in order to handle unknown words.

This limitation is at odds with the diachronic addition of new words to languages, and with the grammaticality of sentences containing **pseudo-words**, as for instance

- Could you hand over the salt, please?
- Could you smurf over the smurf, please?

Again, structuring formulæ in such a way that lexical information only further constrains the linguistic trees makes it easy to handle unknown or pseudo-words, which simply do not add any constraint.

**Infinite Sentences** A debatable point is whether natural language sentences should be limited to finite ones. An example illustrating why this question is not so clear-cut is an expression for “mutual belief” that starts with the following:

- Jones believes that iron rusts, and Smith believes that iron rusts, and Jones believes that Smith believes that iron rusts, and Smith believes that Jones believes that iron rusts, and Jones believes that Smith believes that Jones believes that iron rusts, and . . .

Dealing with infinite sequences and trees requires to extend the semantics of generative devices (CFGs, PDAs, etc.) and leads to complications. By contrast, logics are not **a priori** restricted to finite models, and in fact the two examples we will see are expressive enough to force the choice of either infinite or finite models. Of course, for practical applications one might want to restrict oneself to finite models.

**Algorithmic Costs** Formulæ in the logics considered in this chapter are provably more succinct than context-free grammars. The downfall is an algorithmic cost increased in the same proportion, e.g. parsing can require exponential time for PDL [Afanasiev et al., 2005], and non-elementary time for wMSO [Meyer, 1975; Reinhardt, 2002].

1.0.2 **Tree Structures**

Before we turn to the two logical languages that we consider for model-theoretic syntax, let us introduce the structures we will consider as possible models. Because we work with constituent analyses, these will be **labelled ordered trees**. Given a set \( A \) of labels, a tree structure is a tuple \( \mathcal{M} = (W, \downarrow, \rightarrow, (P_a)_{a \in A}) \) where \( W \) is a set of nodes, \( \downarrow \) and \( \rightarrow \) are respectively the child and next-sibling relations over \( W \), and each \( P_a \) for \( a \) in \( A \) is a unary labelling relation over \( W \). We take \( W \) to be isomorphic to some prefix-closed and predecessor-closed subset of \( \mathbb{N}^* \), where \( \downarrow \) and \( \rightarrow \) can then be defined by

\[
\downarrow \overset{\text{def}}{=} \{(w, w_i) \mid i \in \mathbb{N} \land w_i \in W\} \\
\rightarrow \overset{\text{def}}{=} \{(w_i, w(i+1)) \mid i \in \mathbb{N} \land w(i+1) \in W\}.
\]
Note that (a) we do not limit ourselves to a single label per node, i.e. we actually work on trees labelled by $\Sigma \overset{\text{def}}{=} 2^A$, (b) we do not bound the rank of our trees, and (c) we do not assume the set of labels to be finite.

**Binary Trees** One way to deal with unranked trees is to look at their encoding as “first child/next sibling” binary trees. Formally, given a tree structure $\mathcal{M} = \langle W, \downarrow, \rightarrow, (P_a)_{a \in A} \rangle$, we construct a **labelled binary tree** $t$, which is a partial function $\{0,1\}^* \rightarrow \Sigma$ with a prefix-closed domain. We define for this $\text{dom}(t) = \text{fcns}(W)$ and $t(w) = \{a \in A \mid P_a(\text{fcns}^{-1}(w))\}$ for all $w \in \text{dom}(t)$, where

$$\text{fcns}(\varepsilon) \overset{\text{def}}{=} \varepsilon \quad \text{fcns}(w0) \overset{\text{def}}{=} \text{fcns}(w) \quad \text{fcns}(w(i + 1)) \overset{\text{def}}{=} \text{fcns}(wi)$$

(1.3)

for all $w \in \mathbb{N}^*$ and $i \in \mathbb{N}$ and the corresponding inverse mapping is

$$\text{fcns}^{-1}(\varepsilon) \overset{\text{def}}{=} \varepsilon \quad \text{fcns}^{-1}(w0) \overset{\text{def}}{=} \text{fcns}^{-1}(w) 0 \quad \text{fcns}^{-1}(w1) \overset{\text{def}}{=} \text{fcns}^{-1}(w) + 1$$

(1.4)

for all $w \in \varepsilon \cup \{0,1\}^*$, under the understanding that $(wi) + 1 = w(i + 1)$ for all $w \in \mathbb{N}^*$ and $i \in \mathbb{N}$. Observe that binary trees $t$ produced by this encoding verify $\text{dom}(t) \subseteq \{0,1\}^*$.

The tree $t$ can be seen as a **binary structure** $\text{fcns}(\mathcal{M}) = \langle \text{dom}(t), \downarrow_0, \downarrow_1, (P_a)_{a \in A} \rangle$, defined by

$\downarrow_0 \overset{\text{def}}{=} \{(w, w0) \mid w0 \in \text{dom}(t)\}$

(1.5)

$\downarrow_1 \overset{\text{def}}{=} \{(w, w1) \mid w1 \in \text{dom}(t)\}$

(1.6)

$P_a \overset{\text{def}}{=} \{w \in \text{dom}(t) \mid a \in t(w)\}$.

(1.7)

The domains of our constructed binary trees are not necessarily predecessor-closed, which can be annoying. Let $\#$ be a fresh symbol not in $A$; given $t$ a labelled binary tree, its **closure** $\bar{t}$ is the tree with domain

$$\text{dom}(\bar{t}) \overset{\text{def}}{=} \{\varepsilon, 1\} \cup \{0w \mid w \in \text{dom}(t)\} \cup \{0wi \mid w \in \text{dom}(t) \land i \in \{0,1\}\}$$

(1.8)

and labels

$\bar{t}(w) \overset{\text{def}}{=} \begin{cases} t(w') & \text{if } w = 0w' \land w' \in \text{dom}(t) \\ \# & \text{otherwise.} \end{cases}$

(1.9)

Note that in $\bar{t}$, every node is either a node not labelled by $\#$ with exactly two children, or a $\#$-labelled leaf with no children, or a $\#$-labelled root with two children, thus $\bar{t}$ is a full (aka strict) binary tree.

**1.1 Monadic Second-Order Logic**

We consider the **weak monadic second-order logic** (wMSO), over tree structures $\mathcal{M} = \langle W, \downarrow, \rightarrow, (P_a)_{a \in A} \rangle$ and two infinite countable sets of first-order variables $X_1$ and second-order variables $X_2$. Its syntax is defined by

$$\psi ::= x = y \mid x \in X \mid x \downarrow y \mid x \rightarrow y \mid P_a(x) \mid \neg \psi \mid \psi \lor \psi \mid \exists x. \psi \mid \exists X. \psi$$

where $x, y$ range over $X_1$, $X$ over $X_2$, and $a$ over $A$. We write $\text{FV}(\psi)$ for the set of variables free in a formula $\psi$; a formula without free variables is called a **sentence**.
First-order variables are interpreted as nodes in $W$, while second-order variables are interpreted as finite subsets of $W$ (it would otherwise be the full second-order logic). Let $\nu : X_1 \to W$ and $\mu : X_2 \to \mathcal{P}_f(W)$ be two corresponding assignments; then the satisfaction relation is defined by

\[
\begin{align*}
\mathcal{M} \models_{\nu,\mu} x = y & \quad \text{if } \nu(x) = \nu(y) \\
\mathcal{M} \models_{\nu,\mu} x \in X & \quad \text{if } \nu(x) \in \mu(X) \\
\mathcal{M} \models_{\nu,\mu} x \downarrow y & \quad \text{if } \nu(x) \downarrow \nu(y) \\
\mathcal{M} \models_{\nu,\mu} P_a(x) & \quad \text{if } P_a(\nu(x)) \\
\mathcal{M} \models_{\nu,\mu} \neg \psi & \quad \text{if } \mathcal{M} \not\models_{\nu,\mu} \psi \\
\mathcal{M} \models_{\nu,\mu} \psi \lor \psi' & \quad \text{if } \mathcal{M} \models_{\nu,\mu} \psi \text{ or } \mathcal{M} \models_{\nu,\mu} \psi' \\
\mathcal{M} \models_{\nu,\mu} \exists x. \psi & \quad \text{if } \exists w \in W, \mathcal{M} \models_{\nu(x=w),\mu} \psi \\
\mathcal{M} \models_{\nu,\mu} \exists X. \psi & \quad \text{if } \exists U \subseteq W, U \text{ finite } \land \mathcal{M} \models_{\nu,\mu} \{ X ← U \} \psi .
\end{align*}
\]

As usual, we define conjunctions as $\psi \land \psi' \defeq \neg (\neg \psi \lor \neg \psi')$, implications as $\psi \supset \psi' \defeq \psi \lor \neg \psi'$, and equivalences as $\psi \equiv \psi' \defeq \psi \supset \psi' \land \psi' \supset \psi$.

Given a wMSO formula $\psi$, we are interested in two algorithmic problems: the **satisfiability** problem, which asks whether there exist $\mathcal{M}$ and $\nu$ and $\mu$ s.t. $\mathcal{M} \models_{\nu,\mu} \psi$, and the **model-checking** problem, which given $\mathcal{M}$ asks whether there exist $\nu$ and $\mu$ s.t. $\mathcal{M} \models_{\nu,\mu} \psi$. By modifying the vocabulary to have labels in $A \uplus \text{FV}(\psi)$, these questions can be rephrased on a wMSO sentence $\psi'$:

\[
\psi' \defeq \exists \text{FV}(\psi). \psi \land \left( \bigwedge_{x \in X_1 \cap \text{FV}(\psi)} P_a(x) \land \forall y. x \neq y \supset \neg P_a(y) \right) \land \\
\left( \bigwedge_{X \in X_2 \cap \text{FV}(\psi)} \forall y. y \in X \equiv P_X(y) \right).
\]

In practical applications of model-theoretic techniques we restrict ourselves to finite models for these questions.

**Example 1.1.** Here are a few useful wMSO formulæ: To allow any label in a finite set $B \subseteq A$:

\[
P_B(x) \defeq \bigvee_{a \in B} P_a(x)
\]

\[
P_B(X) \defeq \forall x. x \in X \supset P_B(x).
\]

To check whether we are at the root or a leaf or similar constraints:

\[
\text{root}(x) \defeq \neg \exists y. y \downarrow x
\]

\[
\text{leaf}(x) \defeq \neg \exists y. x \downarrow y
\]

\[
\text{internal}(x) \defeq \neg \text{leaf}(x)
\]

\[
\text{children}(x, X) \defeq \forall y. y \in X \equiv x \downarrow y
\]

\[
x \downarrow y \defeq x \downarrow y \land \exists z. z \rightarrow y.
\]
To use the monadic transitive closure of a formula \( \psi(u, v) \) with \( u, v \in \text{FV}(\psi) \): such a formula \( \psi(u, v) \) defines a binary relation over the model, and \([\text{TC}_{u,v} \psi(u, v)]\) then defines the transitive reflexive closure of the relation:

\[
x \ [\text{TC}_{u,v} \psi(u, v)] y \overset{\text{def}}{=} \forall X. (x \in X \land \forall u,v. (u \in X \land \psi(u, v) \supset v \in X) \supset y \in X)
\] (1.10)

For example,

\[
x \downarrow^* y \overset{\text{def}}{=} x \ [\text{TC}_{u,v} u \downarrow v] y
\]

\[
x \rightarrow^* y \overset{\text{def}}{=} x \ [\text{TC}_{u,v} u \rightarrow v] y
\]

### 1.1.1 Linguistic Analyses in wMSO

Let us illustrate how we can work out a constituent-based analysis using wMSO. Following the ideas on grammaticality expressed at the beginning of the chapter, we define large conjunctions of formulæ expressing various linguistic constraints.

**Basic Grammatical Labels**

Let us fix two disjoint finite sets \( N \) of grammatical categories and \( \Theta \) of part-of-speech tags and distinguish a particular category \( S \in N \) standing for sentences, and let \( N \sqcup \Theta \subseteq A \) (we do not assume \( A \) to be finite).

Define the formula

\[
\text{labels}_{N,\Theta} \overset{\text{def}}{=} \forall x. \text{root}(x) \supset P_S(x),
\] (1.11)

which forces the root label to be \( S \);

\[
\land \forall x. \text{internal}(x) \supset \bigvee_{a \in N \sqcup \Theta} P_a(x) \land \bigwedge_{b \in N \sqcup \Theta \setminus \{a\}} \neg P_b(x)
\] (1.12)

checks that every internal node has exactly one label from \( N \sqcup \Theta \) (plus potentially others from \( A \setminus (N \sqcup \Theta) \));

\[
\land \forall x. \text{leaf}(x) \supset \neg P_{N \sqcup \Theta}(x)
\] (1.13)

forbids grammatical labels on leaves;

\[
\land \forall y. \text{leaf}(y) \supset \exists x. x \downarrow y \land P_\Theta(x)
\] (1.14)

expresses that leaves should have POS-labelled parents;

\[
\land \forall x. \exists y_0 y_1 y_2. x \downarrow^* y_0 \land y_0 \downarrow y_1 \land y_1 \downarrow y_2 \land \text{leaf}(y_2) \supset P_N(x)
\] (1.15)

verifies that internal nodes at distance at least two from some leaf should have labels drawn from \( N \), and are thus not POS-labelled by (1.12), and thus cannot have a leaf as a child by (1.13);

\[
\land \forall x. P_\Theta(x) \supset \neg \exists y z. y \neq z \land x \downarrow y \land x \downarrow z
\] (1.16)

discards trees where POS-labelled nodes have more than one child. The purpose of \( \text{labels}_{N,\Theta} \) is to restrict the possible models to trees with the particular shape we use in constituent-based analyses.

**Open Lexicon**

Let us assume that some finite part of the lexicon is known, as well as possible POS tags for each known word. One way to express this in an open-ended manner is to define a finite set \( L \subseteq A \) disjoint from \( N \) and \( \Theta \), and a relation \( \text{pos} \subseteq L \times \Theta \). Then the formula

\[
\text{lexicon}_{L,\text{pos}} \overset{\text{def}}{=} \forall x. \bigvee_{\ell \in L} \left( P_\ell(x) \supset \text{leaf}(x) \land \bigwedge_{\ell' \in L \setminus \{\ell\}} \neg P_{\ell'}(x) \land \forall y. y \downarrow x \supset P_{\text{pos}(\ell)}(y) \right)
\] (1.17)
makes sure that only leaves can be labelled by words, and that when a word is
known (i.e. if it appears in $L$), it should have one of its allowed POS tag as imme-
diate parent. If the current POS tagging information of our lexicon is incomplete,
then this particular constraint will not be satisfied. For an unknown word however,
any POS tag can be used.

**Context-Free Constraints** It is of course easy to enforce some local constraints
in trees. For instance, assume we are given a CFG $G = \langle N, \Theta, P, S \rangle$
describing the “usual” local constraints between grammatical categories and POS tags. Assume $\varepsilon$
belongs to $A$; then the formula

$$\text{grammar}_G \overset{\text{def}}{=} \forall x. (P_\varepsilon(x) \supset \neg P_{N \cup \Theta \cup L}(x)) \land \bigvee_{B \in N} P_B(x) \supset \bigvee_{B \rightarrow \beta \in P} \exists y. x \downarrow_0 y \land \text{rule}_\beta(y)$$

(1.18)

forces the tree to comply with the rules of the grammar, where

$$\text{rule}_X(\beta)(x) \overset{\text{def}}{=} P_X(x) \land \exists y. x \rightarrow y \land \text{rule}_\beta(y) \quad \text{(for } \beta \neq \varepsilon \text{ and } X \in N \cup \Theta)$$

$$\text{rule}_X(x) \overset{\text{def}}{=} P_X(x) \land \exists y. x \rightarrow y \quad \text{(for } X \in N \cup \Theta)$$

$$\text{rule}_\varepsilon(x) \overset{\text{def}}{=} P_\varepsilon(x) \land \text{leaf}(x).$$

Again, the idea is to provide a rather permissive set of local constraints, and to be
able to spot the cases where these constraints are not satisfied.

**Non-Local Dependencies** Implementing local constraints as provided by a CFG
is however far from ideal. A much more interesting approach would be to take
advantage of the ability to use long-distance constraints, and to model subcate-
gorisation frames and modifiers.

The following examples also show that some of the typical features used for
training statistical models can be formally expressed using wMSO. This means that
treebank annotations can be computed very efficiently once a tree automaton has
been computed for the wMSO formulæ, in time linear in the size of the treebank.

**Head Percolation.** The first step is to find which child is the head among its
siblings; several heuristics have been developed to this end, and a simple way to
describe such heuristics is to use a head percolation function $h : N \rightarrow \{l, r\} \times (N \cup \Theta)^*$
that describes for a given parent label $A$ a list of potential labels $X_1, \ldots, X_n$
in $N \cup \Theta$ in order of priority and a direction $d \in \{l, r\}$ standing for “leftmost” or
“rightmost”: such a value means that the leftmost (resp. rightmost) occurrence of
$X_1$ is the head, this unless $X_1$ is not among the children, in which case we should
try $X_2$ and so on, and if $X_n$ also fails simply choose the leftmost (resp. rightmost)
child (see e.g. [Collins, 1999](#)). For instance, the function

$$h(S) = (r, \text{TO IN VP S SBAR} \cdots)$$

$$h(VP) = (l, \text{VBD VBN VBZ VB VBG VP} \cdots)$$

$$h(NP) = (r, \text{NN NNP NNS NNPS JJR CD} \cdots)$$

$$h(PP) = (l, \text{IN TO VBG VBN} \cdots)$$

would result in the correct head annotations in [Figure 1.1](#).
Given such a head percolation function $h$, we can express the fact that a given node is a head:

$$\text{head}(x) \overset{\text{def}}{=} \text{leaf}(x) \lor \bigvee_{B \in N} \exists y. y \downarrow x \land \text{children}(y, Y) \land P_B(y) \land \text{head}_{h(B)}(x, Y)$$

(1.19)

$$\text{head}_{d,X}^\beta(x, Y) \overset{\text{def}}{=} \neg \text{priority}_{d,X}^\beta(x, Y) \supset (\text{head}_{d,X}^\beta(x, Y) \land \neg P_X(Y))$$

$$\text{head}_{l,X}^\beta(x, Y) \overset{\text{def}}{=} \forall y. y \in Y \supset x \rightarrow^* y$$

$$\text{head}_{r,X}^\beta(x, Y) \overset{\text{def}}{=} \forall y. y \in Y \supset y \rightarrow^* x$$

$$\text{priority}_{l,X}^\beta(x, Y) \overset{\text{def}}{=} P_X(x) \land \forall y. y \in Y \land y \rightarrow^* x \supset \neg P_X(y)$$

$$\text{priority}_{r,X}^\beta(x, Y) \overset{\text{def}}{=} P_X(x) \land \forall y. y \in Y \land x \rightarrow^* y \supset \neg P_X(y).$$

where $\beta$ is a sequence in $(N \uplus \Theta)^*$ and $X$ a symbol in $N \uplus \Theta$.

![Figure 1.1: A derivation tree refined with lexical and parent information.](image)

**Lexicalisation.** Using head information, we can also recover lexicalisation information:

$$\text{lexicalise}(x, y) \overset{\text{def}}{=} \text{leaf}(y) \land x [\text{TC}_{u,v}^u] u \downarrow v \land \text{head}(u)] y .$$

(1.20)

This formula recovers the lexical information in Figure 1.1.

**Exercise 1.1.** Propose wMSO formulae to recover the parent and lexical POS information in constituent trees, as illustrated in Figure 1.1.

**Modifiers.** Here is a first use of wMSO to extract information about a proposed constituent tree: try to find which word is modified by another word. For instance, for an adverb we could write something like

$$\text{modify}_{RB}(x, y) \overset{\text{def}}{=} \exists x'. y. z. z \downarrow x \land P_{RB}(z) \land \text{lexicalise}(x', x) \land y' \downarrow x' \land \neg \text{lexicalise}(y', x) \land \text{lexicalise}(y', y)$$

(1.21)

that finds a maximal head $x'$ and the lexical projection of its parent $y'$. This formula finds for instance that really modifies likes in Figure 1.2.
Figure 1.2: Derivation tree for Who does Bill think Bill really likes?

(*) Exercise 1.2. Modify (1.21) to make sure that any leaf with a parent tagged by the POS RB modifies either a verb or an adjective.

(**) Exercise 1.3. Consider the ε node in Figure 1.2: modify (1.20) to recover that who lexicalises the bottommost NP node.

1.1.2 wS2S

The classical logics for trees do not use the vocabulary of tree structures M, but rather that of binary structures \( \langle \text{dom}(t), \downarrow_0, \downarrow_1, (P_a)_{a \in A} \rangle \). The weak monadic second-order logic over this vocabulary is called the weak monadic second-order logic of two successors (wS2S). The semantics of wS2S should be clear.

The interest of considering wS2S at this point is that it is well-known to have a decidable satisfiability problem, and that for any wS2S sentence \( \psi \) one can construct a tree automaton \( A_\psi \)—with tower \( |\psi| \) as size—that recognises all the finite models of \( \psi \). More precisely, when working with finite binary trees and closed formulæ \( \psi \),

\[
L(A_\psi) = \{ t \in T(\Sigma \uplus \{\#\}) \mid t \text{ finite} \land t \models \psi \}.
\]

Now, it is easy to translate any wMSO sentence \( \psi \) into a wS2S sentence \( \psi' \) s.t. \( M \models \psi \iff \text{f CNS(M)} \models \psi' \). This formula simply has to interpret the \( \downarrow \) and \( \to \) relations into their binary encodings: let

\[
\psi' \mathrel{\overset{\text{def}}{=}} \psi \land \exists x.\neg(\exists z.z \downarrow_0 x \lor z \downarrow_1 x) \land \neg(\exists y.x \downarrow_1 y)
\]

where the conditions on \( x \) ensure it is at the root and does not have any right child, and where \( \psi \) uses the macros

\[
x \downarrow y \mathrel{\overset{\text{def}}{=}} \exists x_0.x \downarrow_0 x_0 \land (x_0 [\text{TC}_{u,v} u \downarrow_1 v] y)
\]

\[
x \to y \mathrel{\overset{\text{def}}{=}} x \downarrow_1 y.
\]

The conclusion of this construction is

**Theorem 1.2.** Satisfiability and model-checking for wMSO are decidable.
Exercise 1.4 (ω Successors). Show that the weak second-order logic of ω successors (ωSωS), i.e. with \( \downarrow_i = \{ (w, wi) \mid wi \in W \} \) defined for every \( i \in \mathbb{N} \), has decidable satisfiability and model-checking problems.

1.2 Propositional Dynamic Logic

An alternative take on model-theoretic syntax is to employ modal logics on tree structures. Several properties of modal logics make them interesting to this end: their decision problems are usually considerably simpler, and they allow to express rather naturally how to hop from one point of interest to another.

Propositional dynamic logic (Fischer and Ladner [1979]) is a two-sorted modal logic where the basic relations can be composed using regular operations: on tree structures \( \mathfrak{M} = (W, \downarrow, \rightarrow, (P_a)_{a \in A}) \), its terms follow the abstract syntax

\[
\begin{align*}
\pi &::= \downarrow | \rightarrow | \pi^{-1} | \pi; \pi | \pi + \pi | \pi^* | \varphi? \\
\varphi &::= a | T | \neg \varphi | \varphi \lor \varphi | \langle \pi \rangle \varphi
\end{align*}
\]

(path formulae) (node formulae)

where \( a \) ranges over \( A \).

The semantics of a node formula on a tree structure \( \mathfrak{M} = (W, \downarrow, \rightarrow, (P_a)_{a \in A}) \) is a set of tree nodes \( \llbracket \varphi \rrbracket = \{ w \in W \mid \mathfrak{M}, w \models \varphi \} \), while the semantics of a path formula is a binary relation over \( W \):

\[
\begin{align*}
\llbracket a \rrbracket &= \{ w \in W \mid P_a(w) \} \\
\llbracket \downarrow \rrbracket &= \downarrow \\
\llbracket \rightarrow \rrbracket &= \rightarrow \\
\llbracket \neg \varphi \rrbracket &= W \setminus \llbracket \varphi \rrbracket \\
\llbracket \varphi_1 \lor \varphi_2 \rrbracket &= \llbracket \varphi_1 \rrbracket \cup \llbracket \varphi_2 \rrbracket \\
\llbracket \langle \pi \rangle \varphi \rrbracket &= \llbracket \pi^{-1} \rrbracket(\llbracket \varphi \rrbracket) \\
\llbracket \pi^* \rrbracket &= \llbracket \pi \rrbracket^* \\
\llbracket \varphi? \rrbracket &= \text{Id}_{\llbracket \varphi \rrbracket}.
\end{align*}
\]

Finally, a tree \( \mathfrak{M} \) is a model for a PDL formula \( \varphi \) if its root is in \( \llbracket \varphi \rrbracket \), written \( \mathfrak{M}, \text{root} \models \varphi \).

We define the classical dual operators

\[
\begin{align*}
\bot &\equiv \neg T \\
\varphi_1 \triangleleft \varphi_2 &\equiv \neg (\neg \varphi_1 \lor \neg \varphi_2) \\
\langle \pi \rangle \varphi &\equiv \neg \langle \pi^{-1} \rangle \neg \varphi.
\end{align*}
\]

(1.26)

We also define

\[
\begin{align*}
\uparrow &\equiv \downarrow^{-1} \\
\leftarrow &\equiv \rightarrow^{-1} \\
\text{root} &\equiv [\uparrow] \bot \\
\text{leaf} &\equiv [\downarrow] \bot \\
\text{first} &\equiv [\leftarrow] \bot \\
\text{last} &\equiv [\rightarrow] \bot.
\end{align*}
\]

Exercise 1.5 (Converses). Prove the following equivalences:

\[
\begin{align*}
(p_1; p_2)^{-1} &\equiv p_2^{-1}; p_1^{-1} \quad (1.27) \\
(p_1 + p_2)^{-1} &\equiv p_1^{-1} + p_2^{-1} \quad (1.28) \\
(p^*)^{-1} &\equiv (p^{-1})^* \quad (1.29) \\
(\varphi?)^{-1} &\equiv \varphi? . \quad (1.30)
\end{align*}
\]
Exercise 1.6 (Reductions). Prove the following equivalences:
\[
\begin{align*}
\langle \pi_1; \pi_2 \rangle \varphi &\equiv \langle \pi_1 \rangle \langle \pi_2 \rangle \varphi & (1.31) \\
\langle \pi_1 + \pi_2 \rangle \varphi &\equiv (\langle \pi_1 \rangle \varphi) \lor (\langle \pi_2 \rangle \varphi) & (1.32) \\
\langle \pi^* \rangle \varphi &\equiv \varphi \lor \langle \pi; \pi^* \rangle \varphi & (1.33) \\
\langle \phi_1 \rangle \varphi_2 &\equiv \phi_1 \land \varphi_2 . & (1.34)
\end{align*}
\]

1.2.1 Model-Checking

The model-checking problem for PDL is rather easy to decide. Given a model \( \mathcal{M} = (W, \downarrow, \rightarrow, (P_p)_{p \in A}) \), we can compute inductively the satisfaction sets and relations using standard algorithms. This is a P algorithm.

1.2.2 Satisfiability

Unlike the model-checking problem, the satisfiability problem for PDL is rather demanding: it is EXPTIME-complete.

Theorem 1.3 (Fischer and Ladner, 1979). Satisfiability for PDL is EXPTIME-hard.

As with wMSO, it is more convenient to work on binary trees \( t \) of the form \( \langle \mathrm{dom}(t), \downarrow_0, \downarrow_1, (P_p)_{p \in \mathrm{At}(0,1)} \rangle \) that encode our tree structures. Compared with the wMSO case, we add two atomic predicates 0 and 1 that hold on left and right children respectively. The syntax of PDL over such models simply replaces \( w \) with \( w^{\downarrow_0} \) and \( w^{\downarrow_1} \); as with wMSO in Section 1.1.2 we can interpret these relations in PDL by
\[
\downarrow^{\downarrow_0}; \downarrow^{\downarrow_1} \defeq \downarrow^{\downarrow_0}; \downarrow^{\downarrow_1} \quad \text{and} \quad \varphi \defeq \varphi \\
\downarrow^{\downarrow_0}; \downarrow^{\downarrow_1} \quad \text{def} \quad \text{def} \\
\]
that checks that \( \varphi \) holds, that the 0 and 1 labels are correct, and verifies \( \mathcal{M}, w \models \varphi \) iff \( \text{fcns}(\mathcal{M}), \text{fcns}(w) \models \varphi' \). The conditions in (1.36) ensure that the tree we are considering is the image of some tree structure by \( \text{fcns} \): we first go back to the root by the path \( \uparrow^*; \text{root}^? \), and then verify that the root does not have a right child.

Normal Form. Let us write
\[
\uparrow_0 \defeq \downarrow^{-1} \quad \text{and} \quad \uparrow_1 \defeq \downarrow^{\downarrow_0} ;
\]
then using the equivalences of Exercise 1.5 we can reason on PDL with a restricted path syntax
\[
\alpha \defeq \downarrow_0 \uparrow_0 \uparrow_1 \quad \text{(atomic relations)} \\
\pi \defeq \alpha \mid \alpha \uparrow_1 \mid \uparrow_1 \mid \pi \mid \pi + \pi \mid \pi^* \mid \varphi ? \quad \text{(path formulæ)}
\]
and using the dualities of (1.26), we can restrict node formulæ to be of form
\[
\varphi \defeq a \mid \neg a \mid T \mid \bot \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \langle \pi \rangle \varphi \mid [\pi] \varphi . \quad \text{(node formulæ)}
\]

Lemma 1.4. For any PDL formula \( \varphi \), we can construct an equivalent formula \( \varphi' \) in normal form with \( |\varphi'| = O(|\varphi|) \).

Proof sketch. The normal form is obtained by “pushing” negations and converses as far towards the leaves as possible, and can result in the worst-case in doubling the size of \( \varphi \) due to the extra \( \neg \) and \( \downarrow^* \) at the leaves.
Fisher-Ladner Closure

The equivalences found in Exercise 1.6 and their duals allow to simplify PDL formula into a reduced normal form we will soon see, which is a form of disjunctive normal form with atomic propositions and atomic modalities for literals. In order to obtain algorithmic complexity results, it will be important to be able to bound the number of possible such literals, which we do now.

The Fisher-Ladner closure of a PDL formula in normal form \( \varphi \) is the smallest set \( S \) of formulæ in normal form s.t.

1. \( \varphi \in S \),
2. if \( \varphi_1 \lor \varphi_2 \in S \) or \( \varphi_1 \land \varphi_2 \in S \) then \( \varphi_1 \in S \) and \( \varphi_2 \in S \),
3. if \( \langle \pi \rangle \varphi' \in S \) or \( [\pi] \varphi' \in S \) then \( \varphi' \in S \),
4. if \( \langle \pi_1; \pi_2 \rangle \varphi' \in S \) then \( \langle \pi_1 \rangle \langle \pi_2 \rangle \varphi' \in S \),
5. if \( [\pi_1; \pi_2] \varphi' \in S \) then \( [\pi_1] [\pi_2] \varphi' \in S \),
6. if \( \langle \pi_1 + \pi_2 \rangle \varphi' \in S \) then \( \langle \pi_1 \rangle \varphi' \in S \) and \( \langle \pi_2 \rangle \varphi' \in S \),
7. if \( \langle \pi_1 + \pi_2 \rangle \varphi' \in S \) then \( [\pi_1] \varphi' \in S \) and \( [\pi_2] \varphi' \in S \),
8. if \( \langle \pi^* \rangle \varphi' \in S \) then \( \langle \pi \rangle \langle \pi^* \rangle \varphi' \in S \),
9. if \( [\pi^*] \varphi' \in S \) then \( [\pi] [\pi^*] \varphi' \in S \),
10. if \( \langle \varphi_1 ? \rangle \varphi_2 \in S \) or \( [\varphi_1 ?] \varphi_2 \in S \) then \( \varphi_1 \in S \).

We write \( FL(\varphi) \) for the Fisher-Ladner closure of \( \varphi \).

**Lemma 1.5.** Let \( \varphi \) be a PDL formula in normal form. Its Fisher-Ladner closure is of size \( |FL(\varphi)| \leq |\varphi| \).

**Proof.** We construct a surjection \( \sigma \) between positions \( p \) in the term \( \varphi \) and the formulæ in \( S \):

- for positions \( p \) spanning a node subformula \( \text{span}(p) = \varphi_1 \), we can map to \( \varphi_1 \) (this corresponds to cases 1–3 and 10 on subformulae of \( \varphi' \));
• for positions $p$ spanning a path subformula $\text{span}(p) = \pi$, we find the closest ancestor spanning a node subformula (thus of form $\langle \pi' \rangle \varphi_1$ or $[\pi']\varphi_1$). If $\pi = \pi'$ we map $p$ to the same $\langle \pi' \rangle \varphi_1$ or $[\pi']\varphi_1$. Otherwise we consider the parent position $p'$ of $p$, which is mapped to some formula $\sigma(p')$, and distinguish several cases:

- for $\sigma(p') = \langle \pi_1; \pi_2 \rangle \varphi_2$ we map $p$ to $\langle \pi_1 \rangle \langle \pi_2 \rangle \varphi_2$ if $\text{span}(p) = \pi_1$ and to $\langle \pi_2 \rangle \varphi_2$ if $\text{span}(p) = \pi_2$ (this matches case 4 and the further application of 3);

- for $\sigma(p') = [\pi_1; \pi_2] \varphi_2$ we map $p$ to $[\pi_1] [\pi_2] \varphi_2$ if $\text{span}(p) = \pi_1$ and to $[\pi_2] \varphi_2$ if $\text{span}(p) = \pi_2$ (this matches case 5 and the further application of 3);

- for $\sigma(p') = \langle \pi_1 + \pi_2 \rangle \varphi_2$ and $\text{span}(p) = \pi_i$ with $i \in \{1, 2\}$, we map $p$ to $\langle \pi_i \rangle \varphi_2$ (this matches case 6);

- for $\sigma(p') = [\pi_1 + \pi_2] \varphi$ and $\text{span}(p) = \pi_i$ with $i \in \{1, 2\}$, we map $p$ to $[\pi_i] \varphi$ (this matches case 7);

- for $\sigma(p') = \langle \pi^* \rangle \varphi_2$, $\text{span}(p) = \pi$ and we map $p$ to $\langle \pi \rangle \langle \pi^* \rangle \varphi_2$ (this matches case 8);

- for $\sigma(p') = [\pi^*] \varphi_2$, $\text{span}(p) = \pi$ and we map $p$ to $[\pi] [\pi^*] \varphi_2$ (this matches case 9).

The function $\sigma$ we just defined is indeed surjective: we have covered every formula produced by every rule. Figure 1.3 presents an example term and its mapping. □

Reduced Formulae

Reduced Normal Form. We try now to reduce formulæ into a form where any modal subformula is under the scope of some atomic modality $\langle \alpha \rangle$ or $[\alpha]$. Given a formula $\varphi$ in normal form, this is obtained by using the equivalences of Exercise 1.6 and their duals, and by putting the formula into disjunctive normal form, i.e.

$$\varphi \equiv \bigvee_i \bigwedge_j \chi_{i,j}$$

(1.37)

where each $\chi_{i,j}$ is of form

$$\chi ::= a \mid \neg a \mid \langle \alpha \rangle \varphi' \mid [\alpha] \varphi'$$

(reduced formulæ)

Observe that all the equivalences we used can be found among the rules of the Fisher-Ladner closure of $\varphi$:

**Lemma 1.6.** Given a PDL formula $\varphi$ in normal form, we can construct an equivalent formula $\bigvee_i \bigwedge_j \chi_{i,j}$ where each $\chi_{i,j}$ is a reduced formula in $\text{FL}(\varphi)$.

Two-Way Alternating Tree Automaton

The presentation follows mostly [Calvanese et al.](2009). We finally turn to the construction of a tree automaton that recognises the models of a normal form formula $\varphi$. To simplify matters, we use a powerful model for this automaton: a two-way alternating tree automaton (2ATA) over finite ranked trees.
Definition 1.7. A two-way alternating tree automaton (2ATA) is a tuple \( A = \langle Q, \Sigma, q_i, F, \delta \rangle \) where \( Q \) is a finite set of states, \( \Sigma \) is a ranked alphabet with maximal rank \( k \), \( q_i \in Q \) is the initial state, and \( \delta \) is a transition function from pairs of states and symbols \((q, a)\) in \( Q \times \Sigma \) to positive Boolean formulae \( f \) in \( \mathcal{B}_+(\{-1, \ldots, k\} \times Q) \), defined by the abstract syntax

\[
 f ::= (d, q) \mid f \lor f \mid f \land f \mid \top \mid \bot,
\]

where \( d \) ranges over \( \{-1, \ldots, k\} \) and \( q \) over \( Q \). For a set \( J \subseteq \{-1, \ldots, k\} \times Q \) and a formula \( f \), we say that \( J \) satisfies \( f \) if assigning \( \top \) to elements of \( J \) and \( \bot \) to those in \( \{-1, \ldots, k\} \times Q \setminus J \) makes \( f \) true. A 2ATA is able to send copies of itself to a parent node (using the direction \(-1\)), to the same node (using direction \(0\)), or to a child (using directions in \( \{1, \ldots, k\} \)).

Given a labelled ranked ordered tree \( t \) over \( \Sigma \), a run of \( A \) is a tree \( \rho \) labelled by \( \text{dom}(t) \times Q \) satisfying

1. \( \varepsilon \) is in \( \text{dom}(\rho) \) with \( \rho(\varepsilon) = (\varepsilon, q_i) \),
2. if \( w \) is in \( \text{dom}(\rho) \), \( \rho(w) = (u, q) \) and \( \delta(q, t(u)) = f \), then there exists \( J \subseteq \{-1, \ldots, k\} \times Q \) of form \( J = \{(d_0, q_0), \ldots, (d_n, q_n)\} \) s.t. \( J \models f \) and for all \( 0 \leq i \leq n \) we have

\[
 w_i \in \text{dom}(\rho) \quad \rho(w_i) = (u'_i, q_i) \quad u'_i = \begin{cases} u(d_i - 1) & \text{if } d_i > 0 \\ u & \text{if } d_i = 0 \\ u' \text{ where } u = u'_j & \text{otherwise} \end{cases}
\]

with each \( u'_i \in \text{dom}(t) \).

A tree is accepted if there exists a run for it.

Theorem 1.8 (Vardi 1998). Given a 2ATA \( A = \langle Q, \Sigma, q_i, F, \delta \rangle \), deciding the emptiness of \( L(A) \) can be done in deterministic time \( |\Sigma| \cdot 2^{O(k|Q|^3)} \).

Automaton of a Formula Let \( \varphi \) be a formula in normal form. We want to construct a 2ATA \( A_\varphi = \langle Q, \Sigma, q_i, \delta \rangle \) that recognises exactly the closed models of \( \varphi \), so that we can test the satisfiability of \( \varphi \) by Theorem 1.8. We assume wlog. that \( A \subseteq \text{Sub}(\varphi) \). We define

\[
 Q \overset{\text{def}}{=} \text{FL}(\varphi) \cup \{q_i, q_\varphi, q_\#\} \\
 \Sigma \overset{\text{def}}{=} \{\#^{(0)}, \#^{(2)}\} \cup \{a^{(2)} \mid a \subseteq A \cup \{0, 1\}\}.
\]

The transitions of \( A_\varphi \) are based on formula reductions. Let \( \varphi' \) be a formula in \( \text{FL}(\varphi) \) which is not reduced: then we can find an equivalent formula \( \bigvee_i \bigwedge_j \chi_{i,j} \) where each \( \chi_{i,j} \) is reduced. We define accordingly

\[
 \delta(\varphi', a) \overset{\text{def}}{=} \bigvee_i \bigwedge_j (0, \chi_{i,j})
\]
for all such $\varphi'$ and all $a \subseteq A$, thereby staying in place and checking the various $\chi_{i,j}$. For a reduced formula $\chi$ in $\FL(\varphi)$, we set for all $a \subseteq A \cup \{0, 1\}$

$$
\delta(p, a) \overset{\text{def}}{=} \begin{cases} 
\top & \text{if } p \in a \\
\bot & \text{otherwise}
\end{cases}
$$

$$
\delta(\neg p, a) \overset{\text{def}}{=} \begin{cases} 
\bot & \text{if } p \in a \\
\top & \text{otherwise}
\end{cases}
$$

$$
\delta(\langle \downarrow 0 \rangle \varphi', a) \overset{\text{def}}{=} \langle 1, \varphi' \rangle \\
\delta(\langle \downarrow 1 \rangle \varphi', a) \overset{\text{def}}{=} \langle 2, \varphi' \rangle \\
\delta(\langle \uparrow 0 \rangle \varphi', a) \overset{\text{def}}{=} \langle -1, \varphi' \rangle \wedge \langle 0, 0 \rangle \\
\delta(\langle \uparrow 1 \rangle \varphi', a) \overset{\text{def}}{=} \langle -1, \varphi' \rangle \wedge \langle 0, 1 \rangle \\
\delta(\langle \uparrow 0 \rangle \varphi', a) \overset{\text{def}}{=} \langle -1, \varphi' \rangle \wedge \langle 0, 0 \rangle \\
\delta(\langle \uparrow 1 \rangle \varphi', a) \overset{\text{def}}{=} \langle -1, \varphi' \rangle \wedge \langle 0, 1 \rangle \\
\delta(\langle \uparrow 1 \rangle \varphi', a) \overset{\text{def}}{=} \langle -1, \varphi' \rangle \wedge \langle 0, 0 \rangle
$$

where the subformulæ 0 and 1 are used to check that the node we are coming from was a left or a right son and $q_\#$ checks that the node label is $\#$:

$$
\delta(q_\#, \#) \overset{\text{def}}{=} \top \\
\delta(q_\#, a) \overset{\text{def}}{=} \bot
$$

The initial state $q_i$ checks that the root is labelled $\#$ and has $\varphi$ for left son and another $\#$ for right son:

$$
\delta(q_i, \#) \overset{\text{def}}{=} \langle 1, \varphi \rangle \wedge \langle 2, q_\# \rangle \\
\delta(q_i, a) \overset{\text{def}}{=} \bot
$$

For any state $q$ beside $q_i$ and $q_\#$

$$
\delta(q, \#) \overset{\text{def}}{=} \bot
$$

**Corollary 1.9.** Satisfiability of PDL can be decided in $\text{EXPTIME}$.

**Proof sketch.** Given a PDL formula $\varphi$, by [Lemma 1.4](#) construct an equivalent formula in normal form $\varphi'$ with $|\varphi'| = O(|\varphi|)$. We then construct $A_{\varphi'}$ with $O(|\varphi|)$ states by [Lemma 1.5](#) and an alphabet of size at most $2^{O(|\varphi|)}$, s.t. $\vec{t}$ is accepted by $A_{\varphi'}$ iff $t, \text{root} \models \varphi$. By [Theorem 1.8](#) we can decide the existence of such a tree $\vec{t}$ in time $2^{O(|\varphi|^3)}$. The proof carries to satisfiability on tree structures rather than binary trees. 

### 1.2.3 Expressiveness

**Monadic Transitive Closure** PDL can be expressed in $\text{FO[TC}^1\text{]}$ the *first-order logic with monadic transitive closure*. The translation can be expressed by induction, yielding formulæ $\text{ST}_x(\varphi)$ with one free variable $x$ for node formulæ and $\text{ST}_{x,y}(\pi)$ with two free variables for path formulæ, such that $M \models_{x \rightarrow y} \text{ST}_x(\varphi)$ iff

See [ten Cate and Segoufin](#).
$w \in \llbracket \varphi \rrbracket_{\mathcal{M}}$ and $\mathcal{M} \models x \rightarrow_{u, y} v \ ST_{x, y}(\pi)$ iff $u \llbracket \pi \rrbracket_{\mathcal{M}} v$:

\[
\begin{align*}
ST_x(a) & \overset{\text{def}}{=} P_a(x) \\
ST_x(\top) & \overset{\text{def}}{=} (x = x) \\
ST_x(\neg \varphi) & \overset{\text{def}}{=} \neg ST_x(\varphi) \\
ST_x(\varphi_1 \lor \varphi_2) & \overset{\text{def}}{=} ST_x(\varphi_1) \lor ST_x(\varphi_2) \\
ST_x(\langle x \rangle \varphi) & \overset{\text{def}}{=} \exists y. (\exists y. ST_{x,y}(\pi)) \land ST_y(\varphi) \\
ST_{x,y}(\downarrow) & \overset{\text{def}}{=} x \downarrow y \\
ST_{x,y}(\rightarrow) & \overset{\text{def}}{=} x \rightarrow y \\
ST_{x,y}(\pi^{-1}) & \overset{\text{def}}{=} \forall x. (\forall x. ST_x(\pi)) \land ST_{x,y}(\varphi) \\
ST_{x,y}(\pi_1; \pi_2) & \overset{\text{def}}{=} \exists z. ST_{x,z}(\pi_1) \land ST_{z,y}(\pi_2) \\
ST_{x,y}(\pi_1 + \pi_2) & \overset{\text{def}}{=} ST_{x,y}(\pi_1) \lor ST_{x,y}(\pi_2) \\
ST_{x,y}(\varphi^*) & \overset{\text{def}}{=} \forall \zeta. \bigwedge_{u,v} \bigwedge_{x,y} \bigwedge_{z} (\forall \zeta. ST_{x,y}(\pi)(x,y)) \land ST_x(\varphi) .
\end{align*}
\]

It is known that wMSO is strictly more expressive than $\text{FO}[\text{TC}^1]$ \cite{tenCateSegoufin2010} (Theorem 2). Ten Cate and Segoufin also provide an extension of PDL with a “within” modality that extracts the subtree at the current node; they show that this extension is exactly as expressive as $\text{FO}[\text{TC}^1]$. It is open whether $\text{FO}[\text{TC}^1]$ is strictly more expressive than PDL without this extension.

**Exercise 1.7 (Within modality).** Let $\mathcal{M} = (W, \downarrow, \rightarrow, (P_a)_{a \in A})$ be a tree structure and $p$ be a point in $\mathcal{M}$. We define the substructure at $p$, noted $\mathcal{M} \upharpoonright p$, as the substructure induced by $W \upharpoonright p = \{ w \in W \mid p \downarrow w \}$. The semantics of a PDLW formula $W \varphi$ is defined by $\mathcal{M}, w \models W \varphi$ iff $\mathcal{M} \upharpoonright w, w \models \varphi$.

Propose a translation of PDLW formulæ into $\text{FO}[\text{TC}^1]$. (***)

**Conditional PDL** A particular fragment of PDL called **conditional PDL** (cPDL) is equivalent to $\text{FO}[^* \rightarrow *]$:

\[
\begin{align*}
\pi & ::= \alpha \mid \alpha^* \mid \pi; \pi \mid \pi + \pi \mid (\alpha; \varphi^*)^* \mid \varphi? \quad \text{(conditional paths)}
\end{align*}
\]

The translation to $\text{FO}[^* \rightarrow *]$ is as above, with

\[
\begin{align*}
ST_{x,y}(\downarrow) & \overset{\text{def}}{=} x \downarrow^* y \land x \neq y \land \forall z. x \downarrow^* z \land x \neq z \lor y \downarrow^* z \\
ST_{x,y}(\rightarrow) & \overset{\text{def}}{=} x \rightarrow^* y \\
ST_{x,y}((\alpha; \varphi)^*) & \overset{\text{def}}{=} \forall \zeta. (ST_{x,z}(\alpha^*) \land ST_{z,y}(\alpha^*)) \lor ST_z(\varphi) .
\end{align*}
\]

An example of a PDL formula that is not first-order definable, and thus not definable in cPDL, is $[(\downarrow \downarrow)^*]a$, which ensures that all the nodes situated at an even distance from the root are labelled by $a$.

**Exercise 1.8.** Express the formulæ $\text{(1.12)}$–$\text{(1.21)}$ in cPDL. (*)
1.3 Parsing as Intersection

The parsing as intersection framework readily applies to model-theoretic syntax. Indeed, in both the wMSO and the PDL cases, given a formula \( \varphi \), we can effectively construct a non-deterministic tree automaton \( A_\varphi \) that recognises the exactly closed trees that satisfy \( \varphi \). Given a sentence \( w \) to parse, it remains to intersect this tree language \( L(A_\varphi) \) with the set of closed binary trees with \( w \) as yield to recover the set of parses of \( w \):

\[ \text{Exercise 1.9.} \] Fix a finite word \( w \) and a finite alphabet \( \Gamma \) of internal nodes. Define a non-deterministic tree automaton that recognises the set of closed binary trees with \( w \) as yield—the yield should here be understood with the ‘#’ symbols ignored.
Chapter 2

First-Order Semantics

In this chapter and the next two chapters, we survey a few aspects of computational semantics. Many formalisms can be used to define meaning representations of linguistic expressions. Here we focus on first-order representations, along with a few related ones.

2.1 Formal Semantics

Concrete applications of computational semantics include for instance weeding out syntactic representations that map to unsatisfiable sentences, checking whether some form of entailment holds between two sentences (for instance for summarisation tasks), or querying databases with natural language interfaces (think airline reservation or weather forecasts), etc. The algorithmic aspects of these applications turn around the usual decision problems in model-theoretic aspects of logic: satisfiability, model-checking (i.e. satisfiability in presence of a database), and querying (an existing database).

Here by “database” we simply mean a (not necessarily finite) relational structure \( \mathcal{R} = (W, (R_i)_{i}) \) where \( W \) is a domain of the various possible entities, and \((R_i^{(k_i)})_{i}\) is a vocabulary, where each \( R_i^{(k_i)} \) is interpreted as a \( k_i \)-ary relation \( R_i \) over \( W \), \( k_i > 0 \). We also allow for constants and denote them using nullary symbols like \( R^{(0)} \); they are interpreted as single points in \( W \). The first-order language thus allows to reason about truths regarding entities and their relations.

Example 2.1. For instance, assume our vocabulary includes \( \text{John}^{(0)} \) as a constant denoting John, along with \( \text{apple}^{(1)} \), \( \text{red}^{(1)} \), and \( \text{eat}^{(2)} \), we can associate the sentence

\[
\exists x. \text{apple}^{(1)}(x) \land \text{red}^{(1)}(x) \land \text{eat}^{(2)}(\text{John}^{(0)}, x)
\]

to the sentence John eats a red apple. Our interpretation might be s.t.

\[
a, j \in W \quad a \in \text{red} \quad a \in \text{apple} \\
 j = \text{John} \quad (j, a) \in \text{eat} 
\]
in which case the sentence is satisfiable using the assignment \( \{x \mapsto a\} \).

An interesting consequence of this analysis is that paraphrases are typically associated with the same semantics: (2.1) could for instance be the formalisation of

John eats a red apple.
A red apple is eaten by John.
An apple that John eats is red.
2.1.1 Event Semantics

The kind of modelling that underlies Example 2.1 is a rather straightforward one: named entities (e.g. John, or the President) are interpreted as constants, properties (e.g. red, apple) as unary relations, and verbs as relations with an arity equal to the number of arguments present in their subcategorisation frames.

This however leads to some issues when determining the number of arguments for a particular instance of a verb, and drawing the appropriate inferences from our representations. Consider for instance the sentences

John eats.
John eats a red apple.
John eats an apple in a park.
John eats in a park.
John slowly eats a red apple in a park.

Using the approach of Example 2.1, we need to introduce several relations $eat^{(i)}$ largely beyond the simple choice between the intransitive $eat^{(1)}$ and transitive $eat^{(2)}$ forms of $eat$:

\[
eat^{(1)} (\text{John}^{(0)})
\]
\[
\exists x. eat^{(2)} (\text{John}^{(0)}, x) \land \text{red}^{(1)}(x) \land \text{apple}^{(1)}(x) \tag{2.2}
\]
\[
\exists xy. eat^{(3)} (\text{John}^{(0)}, x, y) \land \text{apple}^{(1)}(x) \land \text{park}^{(1)}(y) \tag{2.3}
\]
\[
\exists y. eat^{(2)} (\text{John}^{(0)}, y) \land \text{park}^{(1)}(y) \tag{2.4}
\]
\[
\exists xy. eat^{(4)} (\text{John}^{(0)}, x, y, \text{slowly}^{(0)}) \land \text{red}^{(1)}(x) \land \text{apple}^{(1)}(x) \land \text{park}^{(1)}(y) \tag{2.5}
\]

where basically any extra modifier also necessitates a new variant of $eat$.

How can we relate all the variations of $eat$ so that e.g. (2.6) entails each of (2.2–2.5)? One possibility is to add explicit meaning postulates like

\[
\forall jxy. eat^{(3)} (j, x, y) \supset eat^{(2)} (j, x) \tag{2.7}
\]
\[
\forall jx. eat^{(2)} (j, x, y) \supset eat^{(1)} (j) \tag{2.8}
\]
\[
\vdots
\]
\[
\forall Py. \text{in}^{(2)} (P, y) \supset P \tag{2.11}
\]
\[
\forall P. \text{slowly}^{(1)}(P) \supset P \tag{2.12}
\]

where $P$ ranges over formulæ. Of course there is no particular reason not to choose

\[
\exists xy. \text{slowly}^{(1)} (\text{location}^{(2)} (\text{eat}^{(2)} (\text{John}^{(0)}, x), y)) \land \text{red}^{(1)}(x) \land \text{apple}^{(1)}(x) \land \text{park}^{(1)}(y) \tag{2.13}
\]
instead, and proving the equivalence of (2.10) and (2.13) would require yet more machinery. (We will however return to modal operators later in Section 2.3.)

As we can see, this solution scales rather poorly. Another possibility is to pick a very general version of eat, like eat₅, and express the simpler versions with existentially quantified arguments:

\[
eat^1_1(j) \equiv \exists y a \cdot \eat^4_5(j, x, y, a) \tag{2.14}
\]

\[
eat^2_2(j, x) \equiv \exists y a \cdot \eat^4_5(j, x, y, a) \tag{2.15}
\]

\[
eat^3_3(j, x, y) \equiv \exists a \cdot \eat^4_5(j, x, y, a) \tag{2.16}
\]

\[
eat^2_4(j, y) \equiv \exists y a \cdot \eat^4_5(j, x, y, a) \tag{2.17}
\]

However, while it seems reasonable that the event denoted by John eats has an implicit object and location, there is no particular reason for it to be performed slowly or quickly, and it could also occur at noon or at dawn, necessitating yet another argument slot.

A solution is to use a two-sorted domain that differentiates between events and entities, and to add an explicit event argument to verbs:

\[
\exists e. \eat^2_1(e, \text{John}(0)) \tag{2.18}
\]

\[
\exists x. \eat^2_2(e, \text{John}(0), x) \land \text{red}^1(x) \land \text{apple}^1(x) \tag{2.19}
\]

\[
\exists y. \eat^2_3(e, \text{John}(0), x) \land \text{apple}^1(x) \land \text{park}^1(y) \land \text{location}^2(e, y) \tag{2.20}
\]

\[
\exists y. \eat^2_4(e, \text{John}(0)) \land \text{park}^1(y) \land \text{location}^2(e, y) \tag{2.21}
\]

\[
\exists y. \eat^2_5(e, \text{John}(0), x) \land \text{red}^1(x) \land \text{apple}^1(x) \land \text{park}^1(y) \land \text{location}^2(e, y) \land \text{slowly}^1(e) \tag{2.22}
\]

This Davidsonian analysis succeeds in reducing the variations to the two main forms of eat. It also yields a rather more natural way of handling time and aspects modifiers like slowly. Note that the distinction between intransitive and transitive forms of verbs are better motivated than the ones between say (2.2) and (2.5): contrast for instance

I sank the Bismark.
I sank.

where the transitive usage does not imply the intransitive one.

### 2.1.2 Thematic Roles

The Davidsonian analysis can be further refined by employing thematic roles: instead of seeing the intransitive form eat²¹ and the transitive one eat²³ as two wholly different relations, we can further refine them using a fixed set of thematic relations between events and entities:

\[
\exists e. \text{eat}^1(e) \land \text{agent}^2(e, \text{John}(0)) \tag{2.23}
\]

\[
\exists x. \text{eat}^1(e) \land \text{agent}^2(e, \text{John}(0)) \land \text{patient}^2(e, x) \land \text{apple}^1(x) \tag{2.24}
\]

correspond to the two sentences John eats and John eats an apple respectively. The earlier issue with sank is avoided by changing the nature of the relation between
Table 2.1: A basic set of thematic roles.

<table>
<thead>
<tr>
<th>Role</th>
<th>Typical use</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent</td>
<td>John eats</td>
</tr>
<tr>
<td>patient</td>
<td>John eats an apple</td>
</tr>
<tr>
<td>experiencer</td>
<td>John regrets his actions.</td>
</tr>
<tr>
<td>cause</td>
<td>The crisis worries John.</td>
</tr>
<tr>
<td>theme</td>
<td>John asks a question.</td>
</tr>
<tr>
<td>beneficiary</td>
<td>John gives Mary a kiss.</td>
</tr>
</tbody>
</table>

The definition of a fixed set of thematic roles and how to classify the different uses are of course problematic; Table 2.1 proposes a very simple account.

For the sake of simplicity, we will not explicitly use event semantics and thematic roles in the remainder of the notes; the reader might convince herself that it is always possible.

### 2.2 A Dip into Description Logics

Let us make a short detour through a family of logics primarily developed for knowledge representation. Basic description logics, similarly to the modal logics we will see in Section 2.3, can be translated into first-order logic, so their use does not yield any additional expressive power. Their interest is rather that they force us into well-behaved fragments of FO, where we are able to draw inferences and reason automatically.

#### 2.2.1 A Basic Description Logic

We will confine our interest to one of the most basic logics: $\mathcal{ALC}$ the “attributive concept language with complements.” We describe the models of $\mathcal{ALC}$ as structures $\mathfrak{M} = \langle W, A, R \rangle$ where $W$ is a domain, $A$ is a finite set of atomic concepts $a \subseteq W$, and $R$ is a finite set of roles $r \subseteq W^2$.

An $\mathcal{ALC}$ concept definition $C$ is defined by the syntax

$$C ::= \top \mid a \mid C \sqcap C \mid \neg C \mid \exists r.C$$

where $a$ ranges over $A$ and $r$ over $R$. This syntax can be enriched by $\bot \overset{\text{def}}{=} \neg \top$, $C \sqcup D \overset{\text{def}}{=} \neg(\neg C \sqcap \neg D)$, and $\forall r.j.C \overset{\text{def}}{=} \neg \exists r.j.\neg C$. A concept defines a subset $\llbracket C \rrbracket^\mathfrak{M}$ of a model $\mathfrak{M}$:

$$\llbracket \top \rrbracket^\mathfrak{M} = W, \quad \llbracket a \rrbracket^\mathfrak{M} = a, \quad \llbracket \neg C \rrbracket^\mathfrak{M} = W \setminus \llbracket C \rrbracket^\mathfrak{M},$$

$$\llbracket C \sqcap D \rrbracket^\mathfrak{M} = \llbracket C \rrbracket^\mathfrak{M} \cap \llbracket D \rrbracket^\mathfrak{M}, \quad \llbracket \exists r.C \rrbracket^\mathfrak{M} = r^{-1}(\llbracket C \rrbracket^\mathfrak{M}).$$
The basic questions one might ask on concepts are consistency ones, i.e. whether there exists a model $\mathfrak{M}$ such that $[C]^{\mathfrak{M}}$ is non-empty. An especially useful case is that of an inclusion $C \subseteq D$, i.e. the inconsistency of $C \cap \neg D$.

Examples Consider the sentence *Every man loves a woman*. Its most common semantic reading can be formalised in first-order logic as

$$\forall y. \text{man}^{(1)}(y) \supset \exists x. \text{woman}^{(1)}(x) \land \text{love}^{(2)}(y, x)$$  \hspace{1cm} (2.27)

It can also be formalised as a consistency question in $\mathcal{ALC}$:

$$\text{Man} \subseteq \exists \text{love}. \text{Woman}$$  \hspace{1cm} (2.28)

where the binary relation $\text{love}^{(2)}$ is translated as a role, and the unary predicates $\text{man}^{(1)}$ and $\text{woman}^{(1)}$ as atomic concepts. The sentence *A man eats an apple* is captured by the consistency of

$$\text{Man} \cap \exists \text{eat}. \text{Apple}$$  \hspace{1cm} (2.29)

Extensions There are many extensions of $\mathcal{ALC}$ in the literature. For instance, description logics often allow for names in the form of nominals $i$, which are atomic concepts interpreted as singleton sets in the model. The syntax of concept definitions is then extended to allow $\{i\}$.

For instance, the sentence *John eats a red apple* can be checked by

$$\{\text{John}\} \subseteq \exists \text{eat}. (\text{Apple} \cap \text{Red})$$  \hspace{1cm} (2.30)

and the sentence *Helen of Troy is loved by every man in Greece* by

$$(\text{Man} \cap \exists \text{inhabit}. \{\text{Greece}\}) \subseteq \exists \text{love}. \{\text{Helen of Troy}\}$$  \hspace{1cm} (2.31)

2.2.2 Translation into First-Order Logic

As hinted by the first-order and $\mathcal{ALC}$ formalisations in (2.27)–(2.28), there is a translation of $\mathcal{ALC}$ into first-order logic. Every nominal $i$ is associated with a constant symbol $i^{(0)}$, every atomic concept $a$ with a unary predicate $a^{(1)}$, and every role $r$ with a binary relation $r^{(2)}$. Then, a concept definition $C$ is translated into a first-order formula $ST_x(C)$ with a single free variable $x$:

$$ST_x(\top) \equiv x = x$$
$$ST_x(\{i\}) \equiv x = i^{(0)}$$
$$ST_x(-C) \equiv \neg ST_x(C)$$
$$ST_x(C \cap D) \equiv ST_x(C) \land ST_x(D)$$
$$ST_x(\exists r.C) \equiv \exists y. r^{(2)}(x, y) \land ST_y(C)$$

This satisfies $[C]^{\mathfrak{M}} = \{ w \in W \mid \mathfrak{M} \models x \mapsto w \ ST_x(C) \}$. Consistency questions are then translated into first-order sentences:

$$ST(C) \equiv \exists x. ST_x(C) \quad ST(C \subseteq D) \equiv \forall x. ST_x(C) \supset ST_x(D)$$

These definitions result for instance in the following first-order semantics for (2.31):

$$\forall y.(\text{man}^{(1)}(y) \land \text{inhabit}^{(2)}(y, \text{Greece}^{(0)})) \supset \text{love}^{(2)}(y, \text{Helen of Troy}^{(0)})$$  \hspace{1cm} (2.32)

Two important remarks can be made regarding this translation:

1. it only requires two distinct variables, and
2. every first-order quantifier is guarded by a binary relation symbol (corresponding to the $\mathcal{ALC}$ role).

Each of these conditions is enough to yield decidability of $\mathcal{ALC}$; see Section 2.4.
2.3 Modal Semantics

Modalities are a means of qualifying truth judgements. Modal operators capture the linguistic concepts of tense, mood, and aspect, and more generally modifiers:

John is ___ happy.

we can insert instead of the blank any of necessarily, possibly, known by me to be, now, then, ... Modal logic offers a unified framework to study such modifiers.

2.3.1 Background: Modal Logic

A frame is a couple $\mathfrak F = \langle W, R \rangle$ where $W$ is a non-empty set of worlds and $R$ a binary relation over $W$. A model is a couple $\mathfrak M = \langle \mathfrak F, V \rangle = \langle W, R, V \rangle$ where $\mathfrak F$ is a frame and $V$ is a valuation from a set of atomic propositions $A$ to subsets of $W$.

Basic Modal Language Given a set $A$ of atomic propositions, a (basic) modal formula $\varphi$ is defined by the syntax

$$\varphi ::= p \mid \top \mid \neg \varphi \mid \varphi \lor \varphi' \mid \lozenge \varphi$$

where $p$ ranges over $A$. The $\Box$ modality is defined as the dual of $\lozenge$:

$$\Box \varphi \overset{\text{def}}{=} \neg \lozenge \neg \varphi.$$

A formula satisfies a model $\mathfrak M$ in a world $w$ of $W$, written $\mathfrak M, w \models \varphi$, in the following inductive cases:

- $\mathfrak M, w \models \top$ always
- $\mathfrak M, w \models p$ iff $w \in V(p)$
- $\mathfrak M, w \models \neg \varphi$ iff $\mathfrak M, w \not\models \varphi$
- $\mathfrak M, w \models \varphi \lor \varphi'$ iff $\mathfrak M, w \models \varphi$ or $\mathfrak M, w \models \varphi'$
- $\mathfrak M, w \models \lozenge \varphi$ iff $\exists w', w R w'$ and $\mathfrak M, w' \models \varphi$.

Logics The diamond $\lozenge$ and box $\Box$ modalities can take many different interpretations. For instance,

- in alethic logic, we reason about possible truths: $\lozenge \varphi$ denotes that “possibly $\varphi$” and $\Box \varphi$ “necessarily $\varphi$”. If we follow Leibniz and imagine multiple “possible worlds” in an universe $W$, something “possible” is one holding in at least one possible world, and something “necessary” holds in all possible worlds. In order to obtain such semantics, we should work on total frames where $w R w'$ for all $w, w'$ in $W$.

- In epistemic logic, we reason about knowledge of agents (mind the difference with beliefs): instead of writing $\Box \varphi$ to denote the fact that “the agent knows $\varphi$”, we write $K \varphi$. Epistemic logic is typically interpreted over transitive, symmetric, and reflexive frames, i.e. where $R$ is an equivalence relation. If the knowledge of several agents is to be modelled, we can introduce multiple relations $R_a$ and modalities $K_a$, one for each agent $a$. 

See Blackburn et al. [2001].
• In the basic temporal logic, \( \Diamond \varphi \) denotes that “at some future point, \( \varphi \) holds”, written \( P\varphi \). Its dual \( G\varphi \) means that in all future points, \( \varphi \) holds. Its converse \( P \) allows to reason about the past, and is defined by \( \mathfrak{M}, w \models P\varphi \) if there exists \( w' R w \) s.t. \( \mathfrak{M}, w' \models \varphi \), with dual \( H \). One expects \( R \) to be a transitive, irreflexive relation. An important distinction arises between linear time and branching time frames: in the first case, there is a unique possible future, while in the second case there exist multiple different futures.

Exercise 2.1 (Basic Axiom). Show that \( K : \Box (\varphi \lor \psi) \lor (\Box \varphi \lor \Box \psi) \) is valid, i.e. for any model \( \mathfrak{M} \) and any world \( w \) of \( W \), \( \mathfrak{M}, w \models K \). (*)

Exercise 2.2 (Transitive Frames). Show that, if \( R \) is transitive, then \( 4 : \Diamond \Diamond \varphi \lor \Diamond \varphi \) is valid. (*)

Exercise 2.3 (Epistemic Frames). Prove the following implications for all modal formulæ \( \varphi \) when \( R \) is an equivalence relation:

\[ T : \Box \varphi \lor \varphi \text{— in epistemic logic, if indeed an agent really knows something, then it must be true—,} \]

\[ 4 : \Box \varphi \lor \Box \Box \varphi \text{— in epistemic logic again, an agent has introspection about its own knowledge—,} \]

\[ B : \varphi \lor \Box \Diamond \varphi \text{— in epistemic logic again, a truth is known by the agent as possibility compatible with her knowledge.} \]

Modal Languages As seen with our examples, the basic modal language can be extended to multiple modalities and underlying relations; in particular PDL defined in Section 1.2 is a modal language with an unbounded number of binary relations. A modal similarity type \( O \) is a ranked alphabet of modal operators \( \Delta \) of arity \( r(\Delta) \). A modal formula is then defined as

\[ \varphi ::= p \mid T \mid \neg \varphi \mid \varphi \lor \varphi \mid \Delta(\varphi_1, \ldots, \varphi_{r(\Delta)}) \]

where \( p \) ranges over \( A \) and \( \Delta \) over \( O \). Its semantics are defined over \( O \)-frames \( \mathfrak{F} = (W, (R_\Delta)_{\Delta \in O}) \) where each \( R_\Delta \) relation is of arity \( r(\Delta) + 1 \), by

\[ \mathfrak{M}, w \models \Delta(\varphi_1, \ldots, \varphi_{r(\Delta)}) \text{ iff } \exists w_1, \ldots, w_{r(\Delta)} \in W(w, w_1, \ldots, w_{r(\Delta)}) \in R_\Delta \]

\[ \text{and } \forall 1 \leq i \leq r(\Delta), \mathfrak{M}, w_i \models \varphi_i. \]

Exercise 2.4 (\( \text{ALC} \) as a Modal Language). Provide a consistency-preserving translation from \( \text{ALC} \) concepts into modal formulæ.

Standard Translation Modal languages have a standard translation into first-order logic over the vocabulary \( \langle (R_\Delta)_{\Delta \in O}, (P_p)_{p \in A} \rangle \) where \( P_p = V(p) \):

\[ \text{ST}_x(p) \overset{\text{def}}{=} P_p(x) \]

\[ \text{ST}_x(T) \overset{\text{def}}{=} (x = x) \]

\[ \text{ST}_x(\neg \varphi) \overset{\text{def}}{=} \neg \text{ST}_x(\varphi) \]

\[ \text{ST}_x(\varphi \lor \varphi') \overset{\text{def}}{=} \text{ST}_x(\varphi) \lor \text{ST}_x(\varphi') \]

\[ \text{ST}_x(\Delta(\varphi_1, \ldots, \varphi_{r(\Delta)})) \overset{\text{def}}{=} \exists x_1 \ldots x_{r(\Delta)}. R_\Delta(x, x_1, \ldots, x_{r(\Delta)}) \land \bigwedge_{i=1}^{r(\Delta)} \text{ST}_{x_i}(\varphi_i) \]

In branching frames, the \( \Diamond \) modality becomes similar to the \( EF \) modality of CTL (thus \( \Box \) is similar to \( AG \)). A similar distinction between linear past and branching past can be made (Kupferman et al. 2012).
is a FO formula with a free variable $x$ equivalent to $\varphi$: $M, w \models \varphi$ iff $M \models \varphi_{x \mapsto w}$.

See Blackburn et al. (2001, Chapter 2).

**Bisimulations and Modal Invariance**

**Definition 2.2** (Bisimulations). Let $O$ be a modal similarity type and let $M = \langle W, (R_\Delta)_{\Delta \in O}, V \rangle$ and $M' = \langle W', (R'_\Delta)_{\Delta \in O}, V' \rangle$ be two $O$-models. A non-empty relation $Z \subseteq W \times W'$ is a **bisimulation** between $M$ and $M'$ if for all $w, w'$ s.t. $w \mathrel{Z} w'$,

1. \( \{ p \in A \mid w \in V(p) \} = \{ p' \in A \mid w' \in V'(p') \} \),
2. if \((w, w_1, \ldots, w_{r(\Delta)}) \in R_\Delta\), then there are $w'_1, \ldots, w'_{r(\Delta)}$ in $W'$ s.t. $w \mathrel{Z} w'_i$ for all $1 \leq i \leq r(\Delta)$ and $(w', w'_1, \ldots, w'_{r(\Delta)}) \in R'_\Delta$, and
3. if $(w', w'_1, \ldots, w'_{r(\Delta)}) \in R'_\Delta$, then there are $w_1, \ldots, w_{r(\Delta)}$ in $W$ s.t. $w_i \mathrel{Z} w'_i$ for all $1 \leq i \leq r(\Delta)$ and $(w, w_1, \ldots, w_{r(\Delta)}) \in R_\Delta$.

We say that $w$ and $w'$ are **bisimilar**, noted $w \leftrightarrow w'$, if there exists a bisimulation $Z$ s.t. $w \mathrel{Z} w'$.

**Proposition 2.3** (Invariance for Bisimulation). Let $O$ be a modal similarity type, and $M$ and $M'$ be $O$-models. Then, for every $w$ in $W$ and $w'$ in $W'$ with $w \leftrightarrow w'$, and every modal formula $\varphi$, $M, w \models \varphi$ iff $M', w' \models \varphi$.

**Proof.** The proof proceeds by induction on $\varphi$. The case where $\varphi$ is an atomic proposition is a consequence of (1) in Definition 2.2. The case where $\varphi$ is $\top$ is trivial, and the cases of Boolean connectives follow from the induction hypothesis. For a formula of form $\Delta(\varphi_1, \ldots, \varphi_{r(\Delta)})$:

$M, w \models \Delta(\varphi_1, \ldots, \varphi_{r(\Delta)})$
implies $\exists w_1, w_{r(\Delta)} \in W. (w, w_1, \ldots, w_{r(\Delta)}) \in R_\Delta \land \forall 1 \leq i \leq r(\Delta). M, w_i \models \varphi_i$
implies $\exists w'_1, w'_{r(\Delta)} \in W'. (w', w'_1, \ldots, w'_{r(\Delta)}) \in R'_\Delta \land \forall 1 \leq i \leq r(\Delta). M', w'_i \models \varphi_i$
(by ind. hyp. and (2))
implies $M', w' \models \Delta(\varphi_1, \ldots, \varphi_{r(\Delta)})$,

and the converse implication holds symmetrically thanks to (3) and the induction hypothesis.

It is worth mentioning that the converse does not hold in general: there exist models which are undistinguishable by modal formulae but not bisimilar. In the case of models with **finite image** however, where for every $R_\Delta$ and $w$

$\{(w_1, \ldots, w_{r(\Delta)}) \mid (w, w_1, \ldots, w_{r(\Delta)}) \in R_\Delta\}$
is finite, the converse holds: let us define the **modal equivalence** relation $w \leftrightarrow w'$ as holding iff $w$ and $w'$ are indistinguishable, i.e.

$\{ \varphi \mid M, w \models \varphi \} = \{ \varphi' \mid M', w' \models \varphi' \}$.

**Theorem 2.4** (Hennessy-Milner Theorem). Let $O$ be a modal similarity type, and $M$ and $M'$ be $O$-models with finite image. If $w \leftrightarrow w'$, then $w \leftrightarrow w'$.
Proof. Let us prove that modal equivalence is a bisimulation relation. Condition (1) holds since a difference in labelling would be witnessed by propositional formulæ. For condition (2), assume \( w \leftrightarrow w' \) and \((w, w_1, \ldots, w_{r(\Delta)}) \in R_{\Delta}\), and assume that there do not exist \( w_1', \ldots, w_{r(\Delta)}' \) satisfying (2). The image set \( S' = \{ (w_1', \ldots, w_{r(\Delta)}') \mid (w', w_1', \ldots, w_{r(\Delta)}') \in R_{\Delta}' \} \) is finite, and non empty since otherwise \( \mathfrak{M}, w \models \Delta, \top \) but \( \mathfrak{M}', w' \not\models \Delta, \top \). Thus \( S' \) is a finite set \( \{ (w_1', \ldots, w_{r(\Delta)}'), \ldots, (w_{n1}', \ldots, w_{nr(\Delta)}') \} \) where, by assumption, for every \( 1 \leq j \leq n \), there exists \( 1 \leq i \leq r(\Delta) \) s.t. \( w_i \leftrightarrow w_j' \), i.e. there exists a formula \( \varphi_{j,i} \), s.t. \( \mathfrak{M}, w_i \models \varphi_{j,i} \) but \( \mathfrak{M}', w_j' \not\models \varphi_{j,i} \). But then

\[
\mathfrak{M}, w \models \Delta \left( \bigwedge_{1 \leq j \leq n} \varphi_{j,1}, \ldots, \bigwedge_{1 \leq j \leq n} \varphi_{j,r(\Delta)} \right),
\]

\[
\mathfrak{M}', w' \not\models \Delta \left( \bigwedge_{1 \leq j \leq n} \varphi_{j,1}, \ldots, \bigwedge_{1 \leq j \leq n} \varphi_{j,r(\Delta)} \right),
\]

in contradiction with \( w \leftrightarrow w' \). The argument for condition (3) is symmetric. \( \square \)

The van Benthem Characterisation Theorem We saw earlier that any modal formula has a standard translation into first-order. A converse statement holds for a semantically restricted class of first-order formulæ.

Let us say that a first-order formula \( \psi(x) \) in \( \text{FO}(\text{FO}_\Delta) \cup \text{FO}_p \) with one free variable \( x \) is invariant for bisimulation if for all models \( \mathfrak{M} \) and \( \mathfrak{M}' \), all states \( w \) in \( \mathfrak{M} \) and \( w' \) in \( \mathfrak{M}' \) in bisimulation, we have \( \mathfrak{M} \models \psi(x) \) iff \( \mathfrak{M} \models \psi(x) \).

Decision Problems Many classes of frames yield modal logics with decidable satisfiability and model-checking problems, even when the corresponding first-order theory is undecidable, or suffers from much larger decision complexities. Many logics have NP-complete satisfaction problems, while the basic modal language is PSPACE-complete. Model-checking of finite models is usually P-complete.

2.3.2 First-Order Modal Logic

In order to work with both modal operators and first-order semantics as in Section 2.1, we introduce a mixed logic, first-order modal logic (FOML). For simplicity we give the definitions for the basic modal operator and not the fully general modal logic. The syntax of the logic over a vocabulary \( \langle R_i \rangle_i \) of \( k_i \)-ary symbols is

\[
\varphi ::= x = y \mid R_i(x_1, \ldots, x_{k_i}) \mid \neg \varphi \mid \varphi \land \varphi \mid \diamond \varphi \mid \exists x. \varphi
\]

with \( x, x_1, \ldots, x_{k_i}, y \) ranging over an infinite countable set of variables \( \mathcal{X} \).

We consider structures \( \mathfrak{M} = \langle W, R, D, I \rangle \) where \( \langle W, R \rangle \) is a frame, \( D \) is a domain function from \( W \) to non-empty sets, and \( I \) is an interpretation function mapping each \( R_i \) with arity \( k_i > 0 \) and world \( w \) from \( W \) into a \( k_i \)-ary relation \( I(R_i)(w) \) over \( D(w) \) (constants are handled similarly). The domain of the model is \( \mathcal{D} = \bigcup_{w \in W} D(w) \). A valuation is a partial mapping from variables in \( \mathcal{X} \) to the domain
The satisfaction of a formula by a model $\mathcal{M}$ at a world $w$ for a valuation $\nu$ is defined inductively by

\[
\mathcal{M}, w \models \nu x = y \iff \nu(x) = \nu(y)
\]

\[
\mathcal{M}, w \models \nu \phi \iff \mathcal{M}, w \not\models \nu \phi
\]

\[
\mathcal{M}, w \models \nu \phi \land \phi' \iff \mathcal{M}, w \models \nu \phi \text{ and } \mathcal{M}, w \models \nu \phi'
\]

\[
\mathcal{M}, w \models \nu \Box \phi \iff \exists w' \in W. w R w' \text{ and } \mathcal{M}, w' \models \nu \phi
\]

\[
\mathcal{M}, w \models \nu \exists x. \phi
\]

The domain $D(w)$ denotes the set of objects in the world $w$; this set is allowed to vary from world to world, i.e. the semantics allows a varying domain. Because we restrict the domain of quantified variables to the current domain, we take an actualist quantification. A constant domain semantics instead considers $D(w) = \mathcal{D}$ for all $w$ in $W$; the resulting semantics is also called possibilist quantification.

Unlike the domain, valuations are rigid in this semantics: the value of a variable does not depend on the current world. In the case of varying domains, it can potentially refer to an object from another world but not existing in the current one (but cannot do much with it). In the following we will use constant domains.

**Example 2.6** (First-order temporal logic). Let us consider some very simple examples in the temporal extension of first-order logic: we can model the meaning of the following sentence

John will eat an apple.

as

\[
\exists a. \text{apple}^{(1)}(a) \land F(\text{eat}^{(2)}_2(\text{John}^{(0)}(0), a)).
\]  

(2.33)

Observe however that, in an actualist view, this reading implies the existence of the apple John will eventually eat in the current instant; the formula might not be satisfied by the model if no appropriate object $a$ on which $\text{apple}(a)$ holds can be found. Another reading would be

\[
F(\exists a. \text{apple}^{(1)}(a) \land \text{eat}^{(2)}_2(\text{John}^{(0)}(0), a)).
\]

(2.34)

### 2.4 Decidability

In modern terms, the Entscheidungsproblem or classical decision problem of Hilbert asks, given a first-order formula $\psi$, whether it is satisfiable. Church and Turing famously proved in the 1930s that the problem is undecidable, and a long line of research has established the decidability status of many fragments of first-order logic. Notably, the decidability status is known for all the prefix classes for formulæ in prenex normal form.

For instance, the semantic reading

\[
\exists x. \text{woman}^{(1)}(x) \land \forall y. \text{man}^{(1)}(y) \supset \text{love}^{(2)}(y, x)
\]

(2.35)

for Every man loves a woman—to be contrasted with (2.27)—belongs to the $\exists^* \forall^*$ class shown decidable by Bernays and Schönfinkel and $\text{NExpTime}$-complete by Lewis (1980). It also belongs to the two-variable fragment $\text{FO}^2$, which was shown decidable by Mortimer and $\text{NExpTime}$-complete by Grädel, Kolaitis, and Vardi (1997). The standard translations of $\mathcal{ALC}$ and of basic modal logic also yield $\text{FO}^2$ formulæ, and they are therefore decidable (they are actually $\text{PSPACE}$-complete).
2.4.1 The Guarded Fragment

We are going to look more closely at one of the decidable fragments of first-order logic, called the \( k \)-variable guarded fragment (GFO\(^k\)). The satisfiability problem in GFO\(^k\) is \( \text{ExpTime} \)-complete (Grädel and Walukiewicz, 1999); in fact this complexity also holds for the fixed-point extension of GFO\(^k\).

Let \( \lambda \) be the set of variables. A guarded formula over a vocabulary \( (R_i^{(k_i)})_i \) is defined syntactically by

\[
\psi := x = y \mid R_i^{(k_i)}(z) \mid \neg \psi \mid \psi \land \psi \mid \exists y. \alpha(x, y). \psi(y)
\]

where \( x, y \) are variables in \( \lambda \), \( R_i^{(k_i)} \) is a relation symbol of arity \( k_i \), \( z \) is a \( k_i \)-tuple of variables in \( \lambda \), and \( x, y \) denote tuples of variables in \( \lambda \), \( \alpha(x, y) \) a positive atomic formula, and \( \psi(x, y) \) a GFO\(^k\) formula with \( \text{FV}(\psi) \subseteq \text{FV}(\alpha) = x \cup y \). Guarded universal quantification \( \forall y. \alpha(x, y) \supset \psi(x, y) \) is defined by duality.

For example, the formula \( (2.27) \) is in GFO\(^2\): \( \text{man}^{(1)}(y) \) guards the universal quantification and \( \text{love}^{(2)}(y, x) \) guards the existential quantification. By contrast, \( (2.35) \) is not in GFO\(^2\): the universal quantification \( \forall y. \text{man}^{(1)}(y) \supset \text{love}^{(2)}(y, x) \) is not guarded. Observe more generally that the standard translations of ALC or basic modal formulae are in GFO\(^2\).

Guarded Bisimulations

Let \( \mathcal{M} = (W, (R_i)_i) \) be a relational structure. A set \( X = \{w_1, \ldots, w_n\} \subseteq W \) is guarded in \( \mathcal{M} \) if there exists a positive atomic formula \( \alpha(x_1, \ldots, x_n) \) such that \( \mathcal{M} \models x_1 \rightarrow w_1 \ldots x_n \rightarrow w_n \alpha(x_1, \ldots, x_n) \). In particular, every singleton \( \{w\} \) is guarded by \( x = x \) and every hyperedge \( \langle w_1, \ldots, w_{ki} \rangle \) in the relation \( R_i \) is guarded by \( R_i^{(k_i)}(x_1, \ldots, x_{ki}) \).

A guarded-\( k \)-bisimulation between two structures \( \mathcal{M} \) and \( \mathcal{M}' \) is a non-empty set \( I \) of partial isomorphisms \( f: X \rightarrow X' \) from \( \mathcal{M} \) to \( \mathcal{M}' \), where \( X \subseteq W \) and \( X' \subseteq W' \) are guarded sets of cardinal at most \( k \), such that the following condition is satisfied: for every \( f: X \rightarrow X' \) in \( I \),

1. for every guarded set \( Y \subseteq W \) in \( \mathcal{M} \) of size at most \( k \), there exists \( g: Y \rightarrow Y' \) in \( I \) such that \( f \) and \( g \) agree on \( X \cap Y \), and

2. for every guarded set \( Y' \subseteq W' \) in \( \mathcal{M}' \) of size at most \( k \), there exists \( g: Y \rightarrow Y' \) in \( I \) such that \( f^{-1} \) and \( g^{-1} \) agree on \( X' \cap Y' \).

As in the modal case, we write \( \mathcal{M} \sim_k \mathcal{M}' \) if there exists a guarded-\( k \)-bisimulation between \( \mathcal{M} \) and \( \mathcal{M}' \). We also write \( \mathcal{M} \sim_k \mathcal{M}' \) if for all GFO\(^k\) sentences \( \psi \), \( \mathcal{M} \models \psi \) iff \( \mathcal{M}' \models \psi \). Proposition 2.3 can be extended to the case of guarded-\( k \)-bisimilarity:

**Proposition 2.7.** Let \( \mathcal{M} \) and \( \mathcal{M}' \) be two relational structures over the vocabulary \( (R_i)_i \). If \( \mathcal{M} \sim_k \mathcal{M}' \), then \( \mathcal{M} \sim_{k'} \mathcal{M}' \).

**Proof.** Let \( I \) be a guarded-\( k \)-bisimulation between \( \mathcal{M} \) and \( \mathcal{M}' \). We show by induction on \( \psi \) in GFO\(^k\) that, if \( \psi(x) \) has \( n \) free variables and there exist two \( n \)-tuples \( a \) in \( \mathcal{M} \) and \( a' \) in \( \mathcal{M}' \) such that \( \mathcal{M} \models_{x \rightarrow a} \psi(x) \) but \( \mathcal{M}' \not\models_{x \rightarrow a'} \psi(x) \), then there is no partial isomorphism \( f \) in \( I \) with \( f: a \rightarrow a' \). This will entail that \( I \) is empty when \( n = 0 \), i.e. in the case of a sentence \( \psi \) in GFO\(^k\), thus contradicting \( \mathcal{M} \sim_k \mathcal{M}' \).

For an atomic formula \( \psi(x) = \alpha(x) \) where \( \mathcal{M} \models_{x \rightarrow a} \alpha(x) \) but \( \mathcal{M}' \not\models_{x \rightarrow a'} \alpha(x) \), assume that there exists \( f \) in \( I \) mapping \( a \) to \( a' \). Then by condition (1), there must
exist $g$ in $I$ with domain $a$ that agrees with $f$ on $a$, i.e. $g \colon a \to a'$. This would entail $\mathcal{M}' \not\models_{x \to a'} \alpha(x)$, a contradiction.

For a conjunction $\psi(x_1, x_2) = \psi_1(x_1) \land \psi_2(x_2)$ where $\mathcal{M} \models_{x_1 \to a_1, x_2 \to a_2} \psi_1(x_1, x_2)$ but $\mathcal{M}' \not\models_{x_1 \to a_1', x_2 \to a_2'} \psi_1(x_1, x_2)$, for some $j \in \{1, 2\}$, $\mathcal{M}' \not\models_{x_j \to a_j'} \psi_j(x_j)$ and by induction hypothesis there is no $f_j$ in $I$ that maps $a_j$ to $a_j'$. Then $f$ in $I$ that maps $a_j$ to $a_j'$ for all $j \in \{1, 2\}$. The case of a negated formula is similarly immediate by induction hypothesis.

The interesting case is that of an existential quantification $\psi(x) = \exists y. \alpha(x, y) \land \varphi(x, y)$. Since $\mathcal{M} \models_{x \to a} \psi(x)$, there exists $b$ in $\mathcal{M}$ such that $\mathcal{M} \models_{x \to a, y \to b} \alpha(x, y) \land \varphi(x, y)$. Suppose toward a contradiction that there exists $f$ in $I$ that maps $a$ to $a'$.

By condition (1), since $a \cup b$ is guarded by $\alpha(x, y)$, there exists $g$ in $I$ that maps $a$ to $a'$ and $b$ to $b'$. Then $\mathcal{M}' \models_{x \to a', y \to b'} \alpha(x, y)$ since is a partial isomorphism, which entails that $\mathcal{M}' \not\models_{x \to a', y \to b'} \varphi(x, y)$, which together with the existence of $g$ contradicts the induction hypothesis on $\varphi$.

\[\square\]

Models of Bounded Treewidth

An important model-theoretic property of $\mathcal{AC}$ and the basic modal language is that they enjoy the tree model property: if a formula is satisfiable, then it has a tree model. In the case of GFO$^k$, we can generalise this idea to models of treewidth bounded by $k - 1$, see Proposition 2.8. In the case where $k = 2$ (which is the case of $\mathcal{AC}$ and the basic modal logic), we find again the tree model property.

On an intuitive level, the treewidth of a structure tells how close to a tree the structure looks like. Trees and forests have treewidth 1, cycles have treewidth 2, etc. An example of a class of structures with unbounded treewidth is the class of $n \times n$ grids, each with treewidth $n$. Formally, the treewidth of a structure $\mathcal{M} = \langle W, (R_i^{(k_i)})_i \rangle$ is the minimal $k$ such that there exists a tree $t$ labelled by bags in $\{X \subseteq W \mid |X| \leq k + 1\}$, such that

1. for every guarded set $X$ in $\mathcal{M}$ there exists a position $u$ in $\text{dom} \ t$ with $X \subseteq t(u)$, and

2. for every element $a$ in $\mathcal{M}$, the set of nodes $\{u \in \text{dom} \ t \mid b \in t(u)\}$ is connected in $t$ using the child relation $\downarrow$.

For each $u$ in $\text{dom} \ t$, $t(u)$ induces a substructure $\mathcal{S}(u) \subseteq \mathcal{M}$ of cardinality at most $k + 1$. The tree $t$ is called a tree decomposition of $\mathcal{M} = \bigcup_{u \in \text{dom} \ t} \mathcal{S}(u)$.

Consider a structure $\mathcal{M}$. We are going to construct a guarded-$k$-bisimilar unravelling $\mathcal{M}'$ with treewidth at most $k - 1$. We construct for this two trees $t$ and $t'$ with the same domain $\text{dom} \ t = \text{dom} \ t'$ such that for each position $u$, $t(u)$ induces a guarded substructure $\mathcal{S}(u) \subseteq \mathcal{M}$ and $t'(u)$ a substructure $\mathcal{S}'(u) \subseteq \mathcal{M}'$ isomorphic to $\mathcal{S}(u)$; then $t'$ will be a tree decomposition of $\mathcal{M}'$.

The root $\emptyset$ is labelled $\emptyset$ in both $t$ and $t'$. Inductively, given a position $u$ with $t(u) = \{a_1, \ldots, a_r\}$ and $t'_u = \{a'_1, \ldots, a'_s\}$, we create for every guarded set $\{b_1, \ldots, b_s\}$ of size $s \leq k$ in $\mathcal{M}$ a child node $v$ of $u$ such that $t(v) = \{b_1, \ldots, b_s\}$ and $t'(v) = \{b'_1, \ldots, b'_s\}$ defined for all $1 \leq i \leq s$ by $b'_i = a'_j$ if $b_i = a_j$ for some $1 \leq j \leq r$ and $b'_i$ is a fresh element otherwise. Define accordingly the induced substructure $\mathcal{S}(v)$ to be isomorphic to the induced substructure $\mathcal{S}(v)$, giving rise to a partial isomorphism $f_u \colon t(v) \to t'(v)$ when setting $f_u(b_i) \overset{\text{def}}{=} b'_i$. Finally, let $\mathcal{M}' = \bigcup_{u \in \text{dom} \ t} \mathcal{S}(u)$.

Observe that the tree $t'$ is a tree decomposition of $\mathcal{M}'$. This entails that $\mathcal{M}'$ has treewidth at most $k - 1$. Furthermore, $\{f_u \mid u \in \text{dom} \ t\}$ is a non-empty (note that the root $\emptyset$ gives rise to the empty isomorphism) set of partial isomorphisms.

See Robertson and Seymour (1986).
between \( \mathcal{M} \) and \( \mathcal{M}' \), which satisfies the conditions of a guarded-\( k \)-bisimulation: \( \mathcal{M} \leftrightarrow_k \mathcal{M}' \). Hence, by Proposition 2.7:

**Proposition 2.8.** If a sentence \( \psi \) in GFO\( ^k \) has a model, then it has a model of treewidth at most \( k - 1 \).

Proposition 2.8 is instrumental in the proof of Grädel and Walukiewicz (1999) that the satisfiability problem for GFO\( ^k \) is in \( \text{ExpTime} \). More precisely, the idea is to reduce the problem to a modal \( \mu \)-calculus satisfiability question over (infinite, countable) trees: given a GFO\( ^k \) formula \( \psi \), one can construct a modal \( \mu \)-calculus formula \( \varphi \) which describes a tree decomposition of a model of \( \psi \) of treewidth at most \( k - 1 \). The complexity then follows by adapting the results of Vardi (1998) on the emptiness problem for 2ATAs over infinite trees.

**Limitations & Extensions**

Although the guarded fragment includes many formulæ of interest in formal semantics, it is not comprehensive: (2.35) is an example of an unguarded formula. We can furthermore show that there is no equivalent formula in GFO. Observe that the two structures depicted in Figure 2.1 are guarded-2-bisimilar for the following set \( I \) of partial isomorphisms

\[
\begin{align*}
    f_{ab} &: a \mapsto a', b \mapsto b' \\
    f_{cd} &: c \mapsto a', d \mapsto b'
\end{align*}
\]

Because every guarded set in \( \mathcal{M} \) is in the domain of one of the partial isomorphisms in \( I \), every guarded set in \( \mathcal{M}' \) is in the range of at least one of the partial isomorphisms in \( I \), and all the partial isomorphisms in \( I \) agree, this is indeed a guarded-2-bisimulation. Therefore by Proposition 2.7 \( \mathcal{M} \) and \( \mathcal{M}' \) are undistinguishable through guarded formulæ over the vocabulary \{ \text{man}^{(1)}, \text{woman}^{(1)}, \text{love}^{(2)} \}. However, \( \mathcal{M} \not\models \exists x.\text{woman}^{(1)}(x) \land (\forall y.\text{man}^{(1)}(y) \supset \text{love}^{(2)}(y,x)) \) but \( \mathcal{M}' \models \exists x.\text{woman}^{(1)}(x) \land (\forall y.\text{man}^{(1)}(y) \supset \text{love}^{(2)}(y,x)) \). In particular, no ALC formula can express (2.35).

Another issue with the guarded fragment is that the axiom for transitivity of a binary relation \( R \), which can be expressed by

\[
\forall xyz.R^{(2)}(x,y) \land R^{(2)}(y,z) \supset R^{(2)}(x,z)
\]

or by

\[
\forall xy.R^{(2)}(x,y) \supset (\forall z.R^{(2)}(y,z) \supset R^{(2)}(x,z))
\]

is not guarded—nor in FO\( ^2 \). In fact, the two-variable guarded fragment without equality and only a handful of transitive relations is already undecidable.
(Ganzinger et al., 1999). This is an issue when considering epistemic or temporal modal logics, where transitivity is assumed; thankfully, decidability can be recovered when restricting transitive relations to occur solely in guards (e.g. Ganzinger et al., 1999; Michaliszyn, 2009).
Chapter 3

Tree Patterns

In this chapter, we consider formulæ called patterns from severely restricted fragments of first-order logic over trees. These provide concise means to define tree languages while avoiding the non-elementary complexity of full first-order logic over finite trees (e.g., Reinhardt, 2002). More precisely, we use patterns to define finite tree languages, which are then used as elementary trees in a grammar (Section 3.2) or as possible semantic readings in ambiguous sentences (Section 3.3).

3.1 Background: Existential First-Order Logic

When describing finite structures, existential sentences of first-order logic pop-up naturally: given a structure \( M = \langle W, (R_i)_i \rangle \) over a finite domain \( W = \{ w_1, \ldots, w_n \} \) and a finite relational vocabulary \((R_i)_i\) (with no constants), the canonical sentence associated with \( M \) is

\[
\varphi_M \overset{\text{def}}{=} \exists x_1 \ldots x_n. \chi_M^+(x_1, \ldots, x_n) \tag{3.1}
\]

where the formula \( \chi_M^+ \) is its positive diagram and consists of the conjunction of all the positive relational atomic formulæ true of \( M \):

\[
\chi_M^+(x_1, \ldots, x_n) \overset{\text{def}}{=} \bigwedge_i \bigwedge_{(w_{i_1}, \ldots, w_{i_k}) \in R_i} R_i^{(i_k)}(x_{i_1}, \ldots, x_{i_k}) \tag{3.2}
\]

Observe that \( M \models \varphi_M \), and more precisely \( M \models \nu \chi_M^+(x_1, \ldots, x_n) \) using the valuation \( \nu: x_i \mapsto w_i \). The canonical sentence \( \varphi_M \) only uses existential quantification and conjunction.

EFO and its Fragments

More generally, existential first-order logic (EFO) over a vocabulary \( \sigma \) is defined syntactically by

\[
\begin{align*}
\alpha &::= x = y \mid R^{(k)}(x_1, \ldots, x_k) \\
\varphi &::= \alpha \mid \neg \alpha \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \exists x. \varphi
\end{align*}
\]

(atomic formulæ)

( existential formulæ)

where \( x, y, x_1, \ldots, x_k \) range over \( \mathcal{X} \) the set of variables, and \( R \) over the vocabulary \( \sigma \).

- If both negated atoms \( \neg \varphi \) and disjunctions \( \varphi \lor \varphi \) are forbidden, we obtain primitive positive formulæ (E\(^+\)CFO), which are equivalent to conjunctive queries used in the database literature.
• If negated atoms \( \neg \varphi \) are forbidden, we obtain existential positive formulæ \((E^+ \text{FO})\), which are equivalent to unions of conjunctive queries used in the database literature.

• Finally, if disjunctions \( \varphi \lor \varphi \) are forbidden, we obtain existential conjunctive formulæ \((ECFO)\).

**Normal Forms** When putting an existential formula \( \varphi \) in disjunctive normal form, we see that it is equivalent to a finite disjunction of existential conjunctive formulæ \( \psi_i \)

\[
\varphi \equiv \bigvee_i \psi_i
\]  
(3.3)

where in turn each existential conjunctive formula \( \psi \) can be put in prenex form

\[
\psi \equiv \exists x. \bigwedge_j \beta_j(x_j)
\]  
(3.4)

where the \( \beta_j \)'s are atoms or negated atoms and \( x_j \) is a subvector of \( x \). (If additionally \( \varphi \) was positive, then each \( \psi_i \) is primitive positive and the \( \beta_j \)'s are atoms.)

Observe finally that any atom of the form \( x = y \) in some \( \psi \) can be eliminated by identifying the two variables \( x \) and \( y \) in \( \psi \):

\[
\exists x_1x_2\ldots x_n. \chi \wedge x_1 = x_2 \equiv \exists x_2\ldots x_n. \{x_1 \leftarrow x_2\}
\]  
(3.5)

so that the \( \beta_j \)'s are necessarily relational or of the form \( x \neq y \).

**Small Models** Given an existential conjunctive sentence \( \psi = \exists x_1\ldots x_n. \chi \), we can look at its models with at most \( n \) elements:

\[
\text{Mod}_{\leq n}(\psi) \overset{\text{def}}{=} \{ \mathcal{M} = (W, \sigma) \mid |W| \leq n \land \mathcal{M} \models \psi \}.
\]  
(3.6)

If \( \psi \) is positive, and positive equality atoms of the form \( x = y \) have been eliminated as explained just before (thus only positive relational atoms appear in \( \psi \)), then \( \psi \) has a canonical model \( \mathcal{M}_\psi \) with domain \( \{w_1, \ldots, w_n\} \) and a tuple \((w_{i1}, \ldots, w_{ik})\) in a \( k \)-ary relation \( R \) iff \( R^{(k)}(x_{i1}, \ldots, x_{ik}) \) is an atom in \( \psi \). Clearly, \( \mathcal{M}_\psi \models \psi \), and furthermore the canonical sentence associated with \( \mathcal{M}_\psi \) is \( \psi \) itself.

\[\text{(*)} \]

**Exercise 3.1** (Canonical Model). Given an existential conjunctive sentence \( \psi \) without positive equality atoms (but possibly with some negated atom of the form \( \neg R^{(k)}(x_{i1}, \ldots, x_{ik}) \) or \( x_i \neq x_j \)), we distinguish its positive part \( \psi^+ \), which contains only the positive relational atoms of \( \psi \). Show that, if \( \psi \) is satisfiable, then \( \mathcal{M}_{\psi^+} \models \psi \).

### 3.1.1 Characterisations over Finite Models

Fix some finite vocabulary \( \sigma = (R_i)_i \). Given two structures \( \mathcal{M} = (W, (R_i)_i) \) and \( \mathcal{M}' = (W', (R'_i)_i) \), \( \mathcal{M} \) is an induced substructure of \( \mathcal{M}' \) if \( W \subseteq W' \) and \( R_i = R'_i \cap W'^k \) for each \( k \)-ary relation. In that case, we also say that \( \mathcal{M}' \) is an extension of \( \mathcal{M} \) and write \( \mathcal{M} \subseteq \mathcal{M}' \). A sentence \( \varphi \) in FO is preserved under extensions if \( \mathcal{M} \models \varphi \) and \( \mathcal{M} \subseteq \mathcal{M}' \) together imply \( \mathcal{M}' \models \varphi \).

The Loś-Tarski theorem states that a first-order sentence is preserved under extensions over the class of all (finite and infinite) structures if and only if it is equivalent to an existential sentence. If we work on a particular class of structures, the theorem might fail, but one direction remains correct:
**Proposition 3.1.** Let $C$ be a class of structures. If $\varphi$ is equivalent to an existential sentence over $C$, then it is preserved under extensions over $C$.

**Proof.** Let $\mathcal{M} = \langle W, (R_i)_{i} \rangle$ and $\mathcal{M}' = \langle W', (R'_i)_{i} \rangle$ be two structures in $C$ with $\mathcal{M} \models \varphi$ and $\mathcal{M} \subseteq \mathcal{M}'$. Write $\varphi$ as a finite disjunction of ECFO sentences as in (3.3): there exists a disjunct $\psi$ such that $\mathcal{M} \models \psi$. More precisely, $\psi$ can be put in prenex normal form as $\psi = \exists x_1 \ldots x_n, \chi$ where $\chi$ is a conjunction of atoms and negated atoms, and $\mathcal{M} \models \chi$ for some valuation $\nu : \{x_1, \ldots, x_n\} \to W$. Consider the substructure $\mathcal{M}_{\nu}$ (not necessarily in $C$) induced by the subset $\nu(\{x_1, \ldots, x_n\}) \subseteq W$ in $\mathcal{M}$: then $\mathcal{M}_{\nu} \models \chi$ and $\mathcal{M}_{\nu} \subseteq \mathcal{M} \subseteq \mathcal{M}'$. We can easily check that $\mathcal{M}' \models \chi$ and the result follows. \hfill \Box

In our applications, we will be especially interested in the (induced-)minimal models of existential sentences: given a class $C$ of structures and a first-order sentence $\varphi$, $\mathcal{M}$ in $C$ is a minimal model of $\varphi$ if $\mathcal{M} \models \varphi$ and, if $\mathcal{M}' \subseteq \mathcal{M}$, then $\mathcal{M}' \not\models \varphi$.

**Lemma 3.2.** Let $C$ be a class of finite structures closed under induced substructures. If $\varphi$ is equivalent to an existential sentence over $C$, then $\varphi$ has finitely many minimal models in $C$.

**Proof.** Using again the disjunctive normal form equivalent to $\varphi$, it suffices to show that there are finitely many minimal models for a disjunct $\psi$ in ECFO. Let $\psi \equiv \exists x_1 \ldots x_n, \chi$, $\mathcal{M}$ be a minimal model of $\psi$, and $\nu$ be a valuation such that $\mathcal{M} \models \chi$. Then $\nu$ induces as in the proof of [Proposition 3.1] a substructure $\mathcal{M}_{\nu} \subseteq \mathcal{M}$ with $\mathcal{M}_{\nu} \models \chi$. Because $C$ is closed under induced substructures, $\mathcal{M}_{\nu}$ also belongs to $C$, and because $\mathcal{M}$ was assumed minimal, this in turn entails that $\mathcal{M}_{\nu}$ and $\mathcal{M}$ are isomorphic, and thus that $\mathcal{M}$ has at most $n$ elements.

In other words, if $\mathcal{M}$ is a minimal model in $C$, then

$$\mathcal{M} \in \text{Mod}_{\leq n}(\psi). \tag{3.7}$$

(Note that this is not directly implied by Exercise 3.1, because $\mathcal{M}_{\psi^+}$ might not be in $C$.) We conclude by noting that $\text{Mod}_{\leq n}(\psi)$ is finite for every $n$, and that $n$ itself is bounded by the quantifier depth of $\varphi$. \hfill \Box

**Exercise 3.2** (Diagrams). Let $\mathcal{M} = \langle W, (R_i)_{i} \rangle$ be a finite structure with $W = \{w_1, \ldots, w_n\}$. We define its diagram as the conjunction of the atomic and negated atomic formulae it satisfies under the valuation $\nu : x_j \mapsto w_j$:

$$\chi_{\mathcal{M}} \overset{\text{def}}{=} \bigwedge_{1 \leq k \leq n} x_j \neq x_k \wedge \left( \bigwedge_{(w_{i_1}, \ldots, w_{i_k}) \in R_i} R_{i}^{(i_k)}(x_{i_1}, \ldots, x_{i_k}) \wedge \bigwedge_{(w_{i_1}, \ldots, w_{i_k}) \notin R_i} \neg R_{i}^{(i_k)}(x_{i_1}, \ldots, x_{i_k}) \right) \tag{3.8}$$

Show that, for any structure $\mathcal{M}'$, $\mathcal{M}' \models \exists x_1 \ldots x_n, \chi_{\mathcal{M}}$ iff $\mathcal{M} \subseteq \mathcal{M}'$ (up to isomorphism).

**Exercise 3.3** (Converse of Lemma 3.2). Let $C$ be a class of finite structures and let $\varphi$ be a first-order sentence preserved under extensions on $C$. Show that, if $\varphi$ has finitely many minimal models in $C$, then it is equivalent to an existential sentence over $C$.

Somewhat similar ideas can be worked out for existential positive sentences (instead of existential sentences) and homomorphisms between structures (instead of induced substructures), see Asterias et al. (2006); Rossman (2008); Dawar (2010).
3.1.2 Tree Models

Unranked Trees  Let us consider finite ordered unranked trees, with labels taken from some finite set $A$; note that for our applications we assume that each tree position is labelled by a single symbol from $A$. Because first-order logic cannot express transitive closures, we explicitly add the transitive reflexive closures $\downarrow^*$ of $\downarrow$ and $\rightarrow^*$ of $\rightarrow$ to our signature. In other words, we work over the relational signature $\langle \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, (P_a)_{a \in A} \rangle$, and our class of models is restricted to trees, where the interpretation of $\downarrow^*$ (resp. $\rightarrow^*$) must coincide with the transitive reflexive closure of the interpretation of $\downarrow$ (resp. $\rightarrow$).

An issue with the class of trees is that it is not closed under induced substructures. For instance, the proof of Lemma 3.2 is incorrect for trees, e.g. the sentence

$$\exists xyz. P_a(x) \land P_b(y) \land P_c(z) \land x \downarrow^* y \land x \not\downarrow y \land x \downarrow^* z \land x \not\downarrow z \quad (3.9)$$

has minimal models of size 4 of the following form, for any label $\$ \in A:

```
   a
  / \  \\
\$  /  c
  /  
```

Ranked Trees  Another vocabulary of interest is $\langle (\downarrow_i)_{i \leq k}, \downarrow^*, (P_a)_{a \in A} \rangle$ where $A$ is a finite ranked alphabet and $k$ is the maximal arity in $A$. Again, the class of ranked trees is not closed under induced substructures.

Theorem 3.3 ([Koller et al., 2001]). Satisfiability of $ECFO((\downarrow_i)_{i \leq k}, \downarrow^*, (P_a)_{a \in A})$ sentences is $NP$-complete.

3.2 Meta-Grammars

In order to cope with the difficulty of hand-writing grammars with an adequate coverage of a natural language, it turns out to be quite convenient to see the grammar itself as the result of a compilation from a higher-level formalism. There exist many ways to define such a meta-grammar. Here we will focus on a simple formalism where the low-level grammar is the set of minimal models of an existential first-order formula on trees.

3.2.1 Diathesis Alternation

One of the difficulties in competence grammars is to account for the many possible subcategorisation frames each lemma might allow. For instance, a transitive verb like $eat$ allows for the sentences

- John eats an apple.
- Who eats an apple?
- What does John eat?
- An apple is eaten by John.

This not only leads to an explosion in the number of elementary tree structures in a context-free or tree-adjoining grammar, but also makes the semantic mapping (with adequate thematic roles) more cumbersome.
By allowing to factor some common patterns in elementary trees, we gain in succinctness. Moreover, by identifying linguistically-motivated atomic constructions, we obtain a more readable, easier to maintain description of the syntax. For instance, various elementary trees for transitive verbs can be described by the formulæ (number agreement could be handled through feature structures):

\[
\text{TransitiveVerb} \overset{\text{def}}{=} \text{ActiveTransitiveVerb} \lor \text{PassiveTransitiveVerb}
\]

\[
\text{ActiveTransitiveVerb} \overset{\text{def}}{=} \text{Subject} \land \text{ActiveVerb} \land (\text{CanonicalObject} \lor \text{Wh-NP-Object})
\]

\[
\text{PassiveTransitiveVerb} \overset{\text{def}}{=} \text{CanonicalSubject} \land \text{PassiveVerb} \land \text{CanonicalByObject}
\]

\[
\text{Subject} \overset{\text{def}}{=} \text{CanonicalSubject} \lor \text{Wh-NP-Subject}
\]

where each of the basic formulæ ActiveVerb, PassiveVerb, etc. is the canonical positive primitive formula of the corresponding tree in Figure 3.1. For instance, the conjunction

\[
\text{CanonicalSubject} \land \text{ActiveVerb} \land \text{Wh-NP-Object}
\]

(3.10)
gives rise to the unique minimal model of Figure 3.2.

3.2.2 Complexity

Observe that we only used the ↓, →, and →⁺ axes in our examples in Section 3.2. One might hope that this fragment of $E^+\text{FO}$ would have a polynomial-time sat-
is satisfiability problem, but it turns out to be NP-hard already for primitive positive sentences with only \( \rightarrow \) and \( \rightarrow^+ \):

**Proposition 3.4.** Satisfiability of \( E^+ \text{CFO}(\rightarrow, \rightarrow^+, (P_a)_a) \) sentences is NP-complete.

**Proof.** By **??**, satisfiability is in NP, thus we only need to prove hardness.

We reduce for this from the Shortest Common Supersequence Problem (SSSP), c.f. **Räihä and Ukkonen [1981]**. An instance of SSSP is an integer \( k \) \( \in \mathbb{N} \) and a set of strings \( S = \{ s_i = a_{i_1} \cdots a_{i_{\ell_i}} \}_{1 \leq i \leq p} \) over some finite alphabet \( \Sigma \). The instance is positive if there exists a string \( s \) of length at most \( k \), which is simultaneously a supersequence of every string in \( S \), i.e. for every \( i \), there exist strings \( s'_0, \ldots, s'_{\ell_i+1} \) s.t. \( s = s'_0 a_{i_1} s'_1 a_{i_2} \cdots a_{i_{\ell_i}} s'_{\ell_i+1} \). Importantly, if \( s \) is such a witness, then any supersequence of \( s \) over some alphabet that includes \( \Sigma \) and of length exactly \( k \) is also a witness.

Given an instance \((k, S)\) of SSSP, we build an existential positive sentence \( \varphi \), which is satisfiable iff the instance is positive. The idea is to find a sequence of children that spells out a witness \( s \) for the SSSP instance. In order to isolate this sequence, we add a fresh symbol \( \# \) to \( \Sigma \) and make sure that we work between two nodes labelled with \( \# \):

\[
\varphi \overset{\text{def}}{=} \exists z z'. P_{\#}(z) \land P_{\#}(z') \land \varphi_{=k}(z, z') \land \varphi_S(z, z')
\]

On the one hand, our intention is for \( \varphi_{=k} \) to make sure that the segment between \( z \) and \( z' \) is of length exactly \( k \):

\[
\varphi_{=k}(z, z') \overset{\text{def}}{=} \exists x_1 \ldots x_k. z \rightarrow x_1 \land x_k \rightarrow z' \land \bigwedge_{1 \leq j < k} x_j \rightarrow x_{j+1}.
\]

On the other hand, \( \varphi_S(z, z') \) should ensure that the segment between \( z \) and \( z' \) is indeed a supersequence of every \( s_i = a_{i_1} \cdots a_{i_{\ell_i}} \):

\[
\varphi_S(z, z') \overset{\text{def}}{=} \bigwedge_{1 \leq i \leq p} \exists y_1 \ldots y_{\ell_i}. z \rightarrow^+ y_1 \land y_{\ell_i} \rightarrow^+ z' \land \bigwedge_{1 \leq j \leq \ell_i} P_{a_{i_j}}(y_j) \land \bigwedge_{1 \leq r < \ell_i} y_r \rightarrow^+ y_{j+1}.
\]

\[\square\]

### 3.3 Underspecified Semantics

#### 3.3.1 Scope Ambiguities

An pervasive issue in semantic representations is related to **scope ambiguities**. Linguistic expressions are often semantically ambiguous (i.e. they have several possible readings that are mapped to different meaning representations) but fail to reflect this ambiguity syntactically (e.g. they have a single syntactic analysis). For instance, the sentence **Every man loves a woman** accepts two readings

\[
\exists y. \text{woman}(y) \land \forall x. \text{man}(x) \supset \exists e. \text{love}(e) \land \text{agent}(e, x) \land \text{patient}(e, y) \quad (3.11)
\]

\[
\forall x. \text{man}(x) \supset \exists y. \text{woman}(y) \land \exists e. \text{love}(e) \land \text{agent}(e, x) \land \text{patient}(e, y) \quad (3.12)
\]

depending on whether we are talking about one single woman or not; there is no clear reason why we should provide the sentence with different syntactic analyses. Assuming we view meaning construction as a relation from one syntactic representation to several semantic ones, the number of readings can grow exponentially
with the number of scope-bearing operators (quantifiers, modal operators, etc.), and simply enumerating the possible readings quickly turns impossible.

For instance, the sentence

*A politician can fool most voters on most issues most of the time, but no politician can fool every voter on every issue all of the time.*

is reputed as having several thousand readings. Arguably, not all these readings are born equal: some might be implied by others (just like (3.11) implies (3.12)), and some downright impossible. However there can still remain a considerable number of incomparable readings. A naive approach to counting the number of possible readings is to consider all the permutations of quantifiers in a sentence: for a sentence with \( n \) quantifiers this will yield \( n! \) different readings. Hobbs and Shieber (1987) for instance refine this approach and show how the sentence

*Every representative of a company saw most samples.*

has actually 5 distinct readings instead of \( 3! = 6 \): they argue that the reading where “for each representative there is a group of most samples which he saw, and furthermore, for each sample he saw, there was a company he was a representative of” is impossible.

A broadly adopted solution to the problems raised by scope ambiguities is to employ underspecified representations for semantics, which allow to represent several readings with a single representation. One might think such a trick, while computationally useful, defeats the very purpose of compositionality, but it does not if we view the underspecified representation as the actual meaning of the sentence...

There exist several such formalisms (e.g. Bos 1996, Egg et al. 2001, Althaus et al. 2003, Copestake et al. 2005) but we will focus on one in particular: the hole semantics of Bos. The idea of hole semantics is to take as a semantic representation language (SRL) the logic we use for semantic representation (in our case FO) and build on top of it an underspecified representation language (URL), which describes the set of desired SRLs. As the latter are terms, the URL can be a formula s.t. the SRLs are its ranked tree models, i.e. we can reuse classical model-theoretic methods.

### 3.3.2 Hole Semantics

The syntax of hole formulæ is a restricted fragment of ECFO\((↓, \downarrow^*, (P_a)_{a \in A})\). We distinguish between two sorts of variables: labels \( l \) in \( \mathcal{L} \) and holes \( h \) in \( \mathcal{H} \) so that dominance relations \( \downarrow^* \) can only go from holes to labels, and holes can only appear as unlabeled leaves; furthermore, immediate children relations and labelling predicates \( P_a \) are combined in a construct \( l : a^{(r)}(x_1, \ldots, x_r) \) that enforces the correct arity of \( a \):

\[
\gamma ::= l : a^{(r)}(x_1, \ldots, x_r) \mid h \downarrow^* l \mid \gamma \land \gamma \mid \exists x.\gamma \text{ (hole formulæ)}
\]

where \( l \) ranges over \( \mathcal{L}, a^{(r)} \) over \( A_r, x, x_1, \ldots, x_r \) over \( \mathcal{L} \cup \mathcal{H} \), and \( h \) over \( \mathcal{H} \). As with ECFO formulæ, hole formulæ \( \gamma \) can be put in prenex normal form

\[
\gamma \equiv \exists l_1 \ldots l_n h_1 \ldots h_m \bigwedge_p \gamma_p . \quad (3.13)
\]
Hole formulæ $\gamma$ are interpreted in ECFO $((\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a \in A})$ by associating a formula $[\gamma]$

$$[\gamma] = \exists l_1 \ldots l_n h_1 \ldots h_m. \bigwedge_{1 \leq i < j \leq n} l_i \neq l_j \land \bigwedge_p \gamma_p$$

(3.14)

where we interpret

$$l : a^{(r)}(x_1, \ldots, x_r) \overset{\text{def}}{=} P_a(l) \land \bigwedge_{i=1}^r l \downarrow_{i-1} x_i .$$

(3.15)

A variable $x$ in a hole formula is a root if there does not exist $x_0, \ldots, x_r$ and $a^{(r)}$ s.t. $x_0 : a^{(r)}(x_1, \ldots, x_r)$ is a subformula of $\gamma$ where $x = x_j$ for some $1 \leq j \leq r$. A hole formula is normal if

1. in every $h \downarrow^* l$ subformula, $l$ is a root of $\gamma$,
2. every hole appears exactly once as a child of a $l : a^{(r)}(x_1, \ldots, x_r)$ subformula, and thus cannot be a root,
3. every label should appear at most once as a parent and at most once as a child in a $l : a^{(r)}(x_1, \ldots, x_r)$ subformula. This excludes for instance $l' : f^{(2)}(l_1, l_2) \land l : f^{(2)}(l_1', l_2')$, or $l_1 : g^{(1)}(l) \land l_2 : g^{(1)}(l)$.

Normal hole formulæ with this interpretation into ECFO give rise to normal dominance constraints, which are known to be efficiently testable for satisfiability:

Theorem 3.5 (Althaus et al., 2003). Satisfiability of normal hole formulæ is in $P$.

Constructive Satisfiability

The issue with our interpretation of hole formulæ into ECFO is that not every model $\mathcal{M}$ over $A$ is suitable as a SRL formula. For instance, there could be extra points in the model not constrained by $\gamma$, or conversely several labels could be mapped to a single node. An alternative notion of model is needed in practice.

Consider a hole formula in prenex conjunctive normal form as in (3.13). Then a plugging $P$ is an injective function from holes $\{h_1, \ldots, h_m\}$ to labels $\{l_1, \ldots, l_n\}$. A model $\mathcal{M} = \langle \text{dom}(t), (\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a \in A} \rangle$ of $\gamma$ is a plugged model for a plugging $P$ if its domain is in bijection with the set of labels (we write $\text{dom}(t) = \{l_1, \ldots, l_n\}$) and $\mathcal{M} \models \nu \gamma$ where the valuation $\nu$ is defined by

$$\nu(x) \overset{\text{def}}{=} \begin{cases} \hat{x} & \text{if } x \in L \\ \hat{P}(x) & \text{if } x \in H. \end{cases}$$

(3.16)

The structure $\mathcal{M}$ is a constructive model for $\gamma$ if there exists a plugging $P$ s.t. it is a plugged model for $P$.

**Example 3.6.** Let us extend the syntax of hole formulæ by allowing larger tree segments:

$$\gamma ::= l : a^{(r)}(\theta_1, \ldots, \theta_r) \mid h \downarrow^* l \mid \gamma \land \gamma \mid \exists x. \gamma \quad \text{(hole formulæ)}$$

$$\theta ::= a^{(r)}(\theta_1, \ldots, \theta_r) \mid h \quad \text{(tree formulæ)}$$
Figure 3.3: Underspecified formula for (3.11) and (3.12). Dominance relations are indicated through dotted arrows and holes by boxes.

and translating back into hole formulæ by defining

\[
\begin{align*}
x_{\theta} & \overset{\text{def}}{=} \begin{cases} 
h & \text{if } \theta = h \\
l_\theta & \in \mathcal{L} \text{ a fresh label for each } \theta \\
\end{cases} \\
l : a^{(r)}(\theta_1, \ldots, \theta_r) & \overset{\text{def}}{=} l : a^{(r)}(x_{\theta_1}, \ldots, x_{\theta_r}) \\
a^{(r)}(\theta_1, \ldots, \theta_r) & \overset{\text{def}}{=} \exists \theta, l_\theta : a^{(r)}(x_{\theta_1}, \ldots, x_{\theta_r}).
\end{align*}
\]

A hole semantic formula that models the two readings (3.11) and (3.12) is the following (see also Figure 3.3):

\[
\begin{align*}
\exists l_1 l_2 l_3 h_1 h_2 l_1 : \forall^{(2)}(x^{(0)}, \text{man}^{(1)}(x^{(0)}) \supset^{(2)} h_1) \land l_2 : \exists^{(2)}(y^{(0)}, \text{woman}^{(1)}(y^{(0)}) \land^{(2)} h_2) \\
\land l_3 : \text{love}^{(2)}(x^{(0)}, y^{(0)}) \land h_1 \downarrow^\ast l_3 \land h_2 \downarrow^\ast l_3 .
\end{align*}
\]

Constructive satisfiability puts a higher toll on computations than basic satisfiability:

**Theorem 3.7.** Constructive satisfiability of normal hole formulæ is NP-complete.

**Proof.** For the NP upper bound, deciding whether a formula \( \gamma \) has a constructive model can be checked by

1. guessing both a plugging \( P \) and the corresponding model

\[
\mathfrak{M} = \{\{\hat{t}_1, \ldots, \hat{t}_n\}, \{\hat{t}_i\}_{i<k}, (P_a)_{a \in A}\};
\]

this model is of polynomial size in \( |\gamma| \),

2. computing the dominance relation \( (\bigcup_{i<k} \downarrow^\ast)^\ast \) over \( \mathfrak{M} \) (this is in P) to obtain a model

\[
\mathfrak{M}' = \{\{\hat{t}_1, \ldots, \hat{t}_n\}, \{\hat{t}_i\}_{i<k}, \downarrow^\ast, (P_a)_{a \in A}\}
\]

still of polynomial size, and

3. verifying that \( \mathfrak{M}' \) is a model of the existentially conjunctive formula \( [\gamma] \) for the assignment \( \nu \) defined in (3.16) (this is in P).

For the NP lower bound, we exhibit a reduction from the 3-Partition Problem. An instance of this problem is given by a finite multiset \( A = \{a_1, \ldots, a_{3m}\} \) of integers and a bound \( B \), all in \( \mathbb{N} \) and encoded in unary, such that \( \frac{B}{3} < a_i < \frac{2B}{3} \) for all \( i \) and \( \sum_{i=1}^{3m} a_i = mB \). The instance is positive if there exists a partition \( A_1 \cup A_2 \cup \cdots \cup A_m \)
of \( A \) s.t. for all \( j \), \(|A_j| = 3\) and \( \sum_{a \in A_j} a = B \). We can assume \( B > 0 \) (or \( a_i = 0 \) for all \( i \)).

We construct from an instance \((A, B)\) a hole formula over the ranked alphabet \( \{f^{(a_i + 1)}, g^{(m)} | 1 \leq i \leq 3m\} \):

\[
\exists ! l_{f_1} \ldots l_{f_{3m}} g^{(B+1)} b_1 \ldots B \lor l_{f_1}^{(a_i + 1)} h^{(m)} h_{f_1}^{(m)} h_{f_2}^{(a_i + 1)} h_{f_3}^{(a_i + 1)} h_{f_{3m}}^{(a_i + 1)}.
\]

\[
\forall \sum_{i < 3m} l_{f_i} : f^{(a_i + 1)}(h^{(1)} f_1, \ldots, h^{(a_i + 1)} f_{i+1})
\]

\[
\land g : g^{(m)}(h^{(1)} g_1, \ldots, h^{(m)} g_m) \land \bigwedge_{1 \leq j \leq m} \bigwedge_{1 \leq k \leq B + 1} j_{f_b} \downarrow^* l_{g_i} \land \bigwedge_{1 \leq j \leq m} \bigwedge_{1 \leq k \leq B + 1} h_{g_j} \downarrow^* l_{j_{f_b}}.
\]

Assume first that there exists a partition \( A_1 \cup \ldots \cup A_m \) of \( A \): we plug \( h_g \) with \( l_g \), and for each class \( A_j = \{a_x, a_y, a_z\} \) with \( a_x + a_y + a_z = B \), we plug \( h_{g_j} \) with \( l_{f_x} \), \( h_{g_j}^{a_x + 1} \) with \( l_{f_y} \), and \( h_{g_j}^{a_y + 1} \) with \( l_{f_z} \), and the remaining \( B + 1 \) holes \( h_{g_j}, h_{f_x}, h_{g_j}^{a_x + 1}, h_{f_y}, h_{f_x}^{a_y + 1}, h_{f_z}, h_{g_j}^{a_z + 1} \) by the labels \( j_{f_b} \) for \( 1 \leq k \leq B + 1 \).

Conversely, assume there is a plugging \( P \) from holes to labels and let \( \forall \) be the corresponding plugged model using valuation \( \nu \). For every \( 1 \leq j \leq m \), consider the set \( A_j \) of integers \( a_i \) such that \( f_i \)-rooted fragments are plugged below \( h_{g_j} \), i.e.

\[
A_j \overset{\text{def}}{=} \{a_i | \forall \nu \models h_{g_j}^{a_i + 1} l_{f_i}\}.
\]

Note that \( A_1 \cup \ldots \cup A_m \) forms a partition of \( A \). Because a plugging is injective from holes to labels, each \( f_i \)-rooted fragment requires \( a_i + 1 \) labels, \( h_{g_j} \) requires one, and \( |A_j| + B + 1 \) are available using the \( f_i \)- and \( b \)-rooted fragments, we get that \( 1 + |A_j| + \sum_{a \in A_j} a_i \leq |A_j| + B + 1 \), hence \( \sum_{a \in A_j} a_i \leq B \) for every \( 1 \leq j \leq m \). Because \( \sum_{a \in A} a = mB \), there is no choice and \( \sum_{a \in A_j} a = B \).

Furthermore, \(|A_j| \geq 3\):

- \(|A_j| \neq 0\) since \( B > 0 \), and \( B + 1 \) fragments rooted by \( b \) must be plugged somewhere below the single hole \( h_g \);
- \(|A_j| \neq 1\) since a single \( f_i \)-rooted fragment provides \( a_i + 1 < \frac{B}{2} < B + 1 \) holes,
- \(|A_j| \neq 2\) since a pair \( \{a_x, a_y\} \) provides \( a_x + a_y + 2 < B + 1 \) holes.

Thus every \( A_j \) is of cardinality at least 3, and because \( 3m f_i \)-rooted fragments are available in total, this means that \(|A_j| = 3\) for all \( j \).

(***)

**Exercise 3.4 (Tree Automata for Hole Formulae).** The set of constructive models of a constraint is clearly a regular tree language. Provide a construction for a regular tree automaton \( \mathcal{A} \), that recognizes exactly the constructive models of a normal hole formula \( \gamma \).

**Hint:** I would use \( 2^{l_1 \ldots l_n} \times \{l_1, \ldots, l_n\} \times 2^{l_1 \ldots l_m} \) as state set, although there certainly are better ways; see for instance [Koller et al. (2008)].

The size of the automaton constructed in Exercise 3.4 is exponential in the size of the formula. This is unavoidable, as there exist normal formulæ \( \gamma_n \) of size \( O(n) \) s.t. any automaton recognizing the set of plugged models of \( \gamma_n \) requires at least \( 2^n \) states: let

\[
A_n \overset{\text{def}}{=} \{a^{(0)}, g_1^{(1)}, \ldots, g_n^{(1)}\}
\]

\[
\gamma_n \overset{\text{def}}{=} \exists l_1 \ldots l_n h_1 \ldots h_n l : a^{(0)} \land \bigwedge_{i=1}^n l_i : g_i^{(1)}(h_i) \land h_i \downarrow^* l.
\]
The normal formula $\gamma_n$ has $n!$ different models, corresponding to the possible orderings of its $n$ components $g_i(\Box)$: its set of plugged models is

$$L_n = \{ g_{\pi(1)}(\Box) \cdot g_{\pi(2)}(\Box) \cdots g_{\pi(n)}(a) \mid \pi \text{ a permutation of } \{1, \ldots, n\} \}.$$  

(3.21)

Lemma 3.8. Any finite tree automaton for $L_n$ requires at least $2^n$ states.

Proof. Define for every subset $K = \{i_1, \ldots, i_{|K|}\}$ of $\{1, \ldots, n\}$ (where $i_j < i_{j+1}$) the context

$$C_K \overset{\text{def}}{=} g_{i_1}(\Box) \cdots g_{i_{|K|}}(\Box)$$  

(3.22)

and let $\overline{K} = \{1, \ldots, n\} \setminus K$. Then the tree

$$t_K \overset{\text{def}}{=} C_{\overline{K}} \cdot C_K \cdot a$$  

(3.23)

is in $L_n$.

Let $Q_K$ be the set of states $q$ of an automaton $A_n$ for $L_n$ s.t.

$$C_K \cdot C_K \cdot a \Rightarrow^* C_{\overline{K}} \cdot q \Rightarrow^* q_f$$  

(3.24)

for some final state $q_f$. Since $t_K$ is in $L_n$, $Q_K \neq \emptyset$. Suppose there exist $K \neq K'$ s.t. $Q_K \cap Q_{K'} \neq \emptyset$, i.e. there exists $i$ in $K \setminus K'$ and $q \in Q_K \cap Q_{K'}$. Then $i$ belongs to $K'$ and

$$C_{K'} \cdot C_K \cdot a \Rightarrow^* C_{\overline{K'}} \cdot q \Rightarrow^* q_f$$  

(3.25)

recognizes a tree not in $L_n$ (the pattern $g_i(\Box)$ appears twice). Hence the non-empty sets $Q_K$ must be disjoint for different sets $K$, thus $A_n$ has at least $2^n$ states.

Note that the tree automaton $\langle 2^{\{1,\ldots,n\}}, A, \delta, \{\emptyset\} \rangle$ with $\delta = \{(q\backslash\{i\}, g_i, q) \mid i \in q \} \cup \{(\{1, \ldots, n\}, b)\}$ recognizes $L_n$, so this bound is optimal.

Lemma 3.8 shows that there might be exponential succinctness gains from the use of hole formulæ rather than tree automata for the description of semantic representations. One might object that the classes of tree languages obtained at the output of the linear higher-order tree functions of Section 4.1.4 are context-free tree languages and not necessarily regular ones, with potential exponential gains in succinctness. However, note that $L_n$ is basically a string language, and the exponential lower bounds on the size of any context-free string grammar for permutation languages (see e.g. [Filmus, 2011]) also apply to CFTGs for $L_n$. 


Chapter 4

Higher-Order Semantics

In this last chapter, we consider the use of higher-order functions in natural language semantics. We first motivate the need for such functions in Section 4.1 in order to define the interface between syntax and semantics. We then observe that, more generally, ‘increasing the order’ allows for elegant solutions to some difficulties like intensionality phenomena and many-world semantics.

4.1 Compositional Semantics

We have presented several possible first-order analyses for simple sentences in the previous chapters, but we have not touched yet the subject of how to obtain such semantic representations from syntactic analyses. A key concept in this regard is that of compositionality:

The meaning of a compound expression is a function of the meanings of its parts and of the syntactic rule by which they are combined.


(Partee et al., 1990, Chapter 13)

Let us illustrate this principle on Example [2.1] by associating a semantic representation to each meaningful word in the sentence, i.e. if we define [[John]], [[eats]] and so on, then the semantics of each intermediate structure like a red apple or John eats a red apple can be systematically computed as a function of its parts, based on the syntactic structures. Note that these structures play a crucial role, as otherwise John loves Mary and Mary loves John would not be distinguishable as naive ‘functions of their parts.’

You are probably familiar with this principle from programming language semantics. Typical arguments in favour of this principle for natural language hinge on productivity and systematicity of semantic construction: we are able to understand new linguistic expressions, and to understand similar expressions built from the same blocks and syntactic processes.

Leaving these questions aside and adopting a modelling viewpoint, compositionality is a rather strenuous requirement: for instance, assuming [[John]] = John\(^{(0)}\) and [[a red apple]] = \(\exists x.\text{apple}^{(1)}(x) \land \text{red}^{(1)}(x)\), it is not so clear how one should combine everything and obtain (2.1) or more involved representations like (2.24). Moreover any solution will be dependent on the specific syntactic analysis.
4.1.1 Background: Simply Typed Lambda Calculus

One of the best-studied ways to implement compositional semantics for natural languages is to use lambda expressions as semantic values associated with each component (Montague 1970, 1973). As Church's simple theory of types provides an elegant setting for model-theoretic higher-order semantics (see Section 4.3), we favour a presentation that uses the simply typed \( \lambda \)-calculus over the untyped one.

**Lambda Terms** Given an infinite countable set \( \mathcal{X} \) of variables, and \( C \) a countable set of constants, the set \( \Lambda(C) \) of \( \lambda \)-terms is defined by

\[
L ::= c \mid x \mid LL \mid \lambda x.L
\]

where \( c \) is a constant in \( C \) and \( x \) a variable in \( \mathcal{X} \).

The \( \lambda \) operator is a binding with the usual associated notion of free variables. We draw a distinction between closed terms, which have no free variables, and ground terms, which have no variables at all.

A \( \lambda \)-term \( L \) is a \( \lambda I \)-term if in every subterm \( \lambda x.M \), \( x \in \text{FV}(M) \). If furthermore \( x \) appears free in \( M \) exactly once, and each free variable \( y \) of \( L \) has at most one free occurrence in \( L \), then \( L \) is a linear \( \lambda \)-term; we let \( \Lambda^\ell(C) \) denote the set of linear \( \lambda \)-terms over \( C \). We write by convention \( \lambda xy.L \) for \( \lambda x.\lambda y.L \) and \( LMN \) for \( (LM)N \) (i.e. we treat application as left associative).

We assume the usual definitions for \( \alpha \), \( \beta \), and \( \eta \) reductions:

\[
\begin{aligned}
\lambda x.L & \to_\alpha \lambda y.\left(L\{x \leftarrow y\}\right) \\
(\lambda x.L)M & \to_\beta L\{x \leftarrow M\} \\
\lambda x.(Lx) & \to_\eta L
\end{aligned}
\]

(where substitutions have to avoid name clashes and \( x \notin \text{FV}(L) \) for \( \eta \)-reductions), and recall that \( \beta\eta \)-reductions are Church-Rosser: if \( L \Rightarrow^* \beta\eta M \) and \( L \Rightarrow^* \beta\eta N \), then there exists \( L' \) s.t. \( M \Rightarrow^* \beta\eta L' \) and \( N \Rightarrow^* \beta\eta L' \), which implies that \( \beta\eta \) reductions define unique normal forms, noted \( \Downarrow^*_{\beta\eta} \).

**Types** Assume we are provided with some non-empty countable set of atomic types \( A \); then types in \( T_A \) are terms defined inductively by

\[
\tau ::= a \mid \tau \to \tau
\]

where \( a \) ranges over \( A \). By convention we consider \( \to \) to be right-associative, i.e. we write \( \rho \to \sigma \to \tau \) for \( \rho \to (\sigma \to \tau) \). The order of a type \( \tau \) is defined inductively as

\[
\text{ord}(a) = 1 \quad \text{ord}(\tau \to \sigma) = \max(\text{ord}(\sigma) + 1, \text{ord}(\tau)).
\]

A higher-order signature is a triple \( \Sigma = \langle A, C, \tau \rangle \) where \( A \) is a set of atomic types, \( C \) a countable set of constants and \( \tau : C \to T_A \) a typing of the constants. Given a higher-order signature, each \( \lambda I \)-term of \( \Lambda(C) \) can be assigned a type in \( T_A \) by the deduction rules

\[
\frac{}{\Gamma \vdash_\Sigma c : \tau(c)} \quad \frac{}{\Gamma \vdash_\Sigma x : \tau} \quad \frac{}{\Gamma, x : \sigma \vdash_\Sigma L : \tau} \quad \frac{}{\Gamma \vdash_\Sigma \lambda x.L : \sigma \to \tau} (\to I)
\]

\[
\frac{}{\Gamma, \Delta \vdash_\Sigma LM : \tau} (\to E)
\]

See e.g. Hindley (1997).
where the type contexts $\Gamma, \Delta$ are type assignments from free variables to $\mathcal{T}_A$; in $\rightarrow_E$ the two assignments have to be compatible, i.e. assign the same types to common variables. For linear lambda terms, this compatibility requirement is useless as $\text{FV}(L) \cap \text{FV}(M) = \emptyset$. We can extend the typing system to any $\lambda$-term instead of $\lambda I$-terms if we additionally allow $\rightarrow_N$ to work on the premise $\Gamma \vdash_\Sigma L : \tau$ where $x$ is not among $\text{FV}(L)$ nor in the domain of $\Gamma$.

**Example 4.1** (B combinator). Define $B \triangleq \lambda x y z . x (y z)$. It can be typed by:

\[
\begin{array}{c}
x : a \rightarrow b \vdash_\Sigma x : a \rightarrow b \\
y : c \rightarrow a \vdash_\Sigma y : c \rightarrow a \\
z : c \rightarrow \vdash_\Sigma z : c \\
\hline
x : a \rightarrow b, y : c \rightarrow a, z : c \vdash_\Sigma x (yz) : b \\
y : c \rightarrow a, z : c \vdash_\Sigma y z : a \\
\hline
y : c \rightarrow a, z : c \vdash_\Sigma \lambda z . x (yz) : c \rightarrow b \\
\hline
x : a \rightarrow b \vdash_\Sigma \lambda x z . x (yz) : c \rightarrow b \\
\hline
\end{array}
\]

**Properties** Let us end this quick survey with a few important properties of the simply typed $\lambda$ calculus: The first two show that types are preserved by reductions:

**Proposition 4.2** (Subject Reduction). If $\Gamma \vdash_\Sigma L : \tau$ and $L \Rightarrow^*_\beta M$ then $\Gamma \vdash_\Sigma M : \tau$.

The converse holds for linear terms (and more generally for reductions that do not exercise non linear variables):

**Proposition 4.3** (Subject Expansion). If $\tau$ is a linear $\lambda$-term, $\Gamma \vdash_\Sigma L : \tau$, and $M \Rightarrow^*_\beta L$, then $\Gamma \vdash_\Sigma M : \tau$.

**Exercise 4.1.** Prove Proposition 4.2 and Proposition 4.3

The second main result about typed $\lambda$-terms is that reduction is strongly normalising: every sequence of rewrites eventually terminates to a term in normal form:

**Theorem 4.4** (Strong Normalisation). If $L$ is a typable $\lambda$-term, then every $\beta\eta$-reduction starting at $L$ is finite.

Remember that not every $\lambda$-term is typable; the typical example of a non-typable term being $\lambda x . x x$. However, every linear $\lambda$-term is typable. A related question is the type inhabitation problem: given a simple type $\tau$, does there exist a closed $\lambda$-term $L$ with type $\tau$? This is usually formulated over an empty set of constants $C = \emptyset$. By the Curry-Howard isomorphism (see e.g. [Hindley, 1997, Chapter 6]), the type inhabitation problem is the same as provability in intuitionistic propositional logic:

**Theorem 4.5** ([Statman, 1979b]). Simple type inhabitation is PSPACE-complete.

### 4.1.2 Ground Terms over Second-Order Signatures

Because we are typically interested in tree structures, it is worth investigating how they can be represented in the simply-typed $\lambda$-calculus. To this end, we restrict ourselves to second-order signatures $\Sigma = \langle A, C, \tau \rangle$, i.e. signatures such that the type of any constant $c$ is of form

$$\tau(c) = a_1 \rightarrow \cdots \rightarrow a_n \rightarrow a_0$$

for atomic $a_i$’s in $A$. 

See e.g. ([Hindley, 1997 Chapter 2]).

The length of $\beta\eta$ reductions can be non elementary in the size of the starting term (see [Statman, 1979a; Schwichtenberg, 1991]).

The type inhabitation problem becomes $2\text{EXP}$-complete for $\lambda I$-terms ([Schmitz, 2014]).
(**) **Exercise 4.2** (Normalised Typing System). Consider the normalised typing system with a single rule
\[
\frac{\tau(c) = a_1 \rightarrow \cdots \rightarrow a_n \rightarrow a_0 \quad \vdash_S t_1 : a_1 \ldots \vdash_S t_n : a_n}{\vdash_S ct_1 \cdots t_n : a_0} \quad \text{(App)}
\]
We want to show that, for all ground terms \( t \) and atomic types \( a \), \( \vdash_S t : a \) if and only if \( \vdash_S t : a \).

1. Show that, if \( \tau(c) = a_1 \rightarrow \cdots \rightarrow a_n \rightarrow a_0 \), \( 0 \leq i \leq n \), and \( \vdash_S t_j : a_j \) for all \( 0 < j \leq i \), then \( \vdash_S ct_1 \cdots t_i : a_{i+1} \rightarrow \cdots \rightarrow a_n \rightarrow a_0 \). Deduce that \( \vdash_S t : a \) implies \( \vdash_S t : a \) if \( t \) is ground and \( a \) atomic.

2. Show that, if \( \vdash_S t : a \) for a ground term \( t \) and type \( a \), then \( t = ct_1 \cdots t_i \) for some constant \( c \) with \( \tau(c) = a_1 \rightarrow \cdots \rightarrow a_n \rightarrow a_0 \), some \( 0 \leq i \leq n \), and some ground terms \( t_1, \ldots, t_i \) such that \( a = a_{i+1} \rightarrow \cdots \rightarrow a_n \rightarrow a_0 \) and \( \vdash_S t_j : a_j \) for \( 0 < j \leq i \) for some atomic types \( a_j \).

3. Deduce that \( \vdash_S t : a \) implies \( \vdash_S t : a \) whenever \( t \) is a ground term and \( a \) an atomic type.

For a second-order constant \( c \) with type \( \tau(c) = a_1 \rightarrow \cdots \rightarrow a_n \rightarrow a_0 \), we call \( n \) its arity (and thus can see \( C \) as a ranked alphabet) and associate to the ground lambda term \( t = ct_1 \cdots t_n \) with atomic type \( a_0 \) the unique tree \( \bar{t} = c^{(n)}(\bar{t}_1, \ldots, \bar{t}_n) \).

Given a second-order signature \( \Sigma \) and a distinguished atomic type \( s \), we define the ground tree language
\[
\mathcal{G}(\Sigma, s) \stackrel{\text{def}}{=} \{ \bar{t} \in T(C) \mid \vdash_S t : s \text{ where } t \text{ is ground} \} .
\]

**Example 4.6.** Consider the second-order signature \( \Sigma_0 \) with atomic types \( A_0 = \{ np, s, c \} \), constants \( C_0 = \{ \text{Alice, believe, left, someone, that} \} \), and typing
\[
\begin{align*}
\tau_0(\text{Alice}) &= np \\
\tau_0(\text{believe}) &= c \rightarrow np \rightarrow s \\
\tau_0(\text{left}) &= np \rightarrow s \\
\tau_0(\text{someone}) &= np \\
\tau_0(\text{that}) &= s \rightarrow c
\end{align*}
\]

The corresponding ranked alphabet is \( \mathcal{F}_0 = \{ \text{Alice}^{(0)}, \text{believe}^{(2)}, \text{left}^{(1)}, \text{someone}^{(0)}, \text{that}^{(1)} \} \).

Then the set of trees in \( \mathcal{G}(\Sigma_0, s) \) is recognised by a tree automaton \( A = \langle Q, \mathcal{F}_0, \delta, \emptyset \rangle \) with \( Q = A_0, I = \{ s \} \), and rules
\[
\delta = \{ (np, Alice^{(0)}), \\
(s, believe^{(2)}, c, np) \\
(s, left^{(1)}, np) \\
(np, someone^{(0)}) \\
(c, that^{(1)}, s) \} .
\]

(**) **Exercise 4.3** (Local Tree Automata). Let \( \mathcal{F} \) be a ranked alphabet. A deterministic top-down tree automaton \( A = \langle Q, \mathcal{F}, \delta, \{ q_0 \} \rangle \) is **local** if there exists a function \( \ell : \mathcal{F} \rightarrow Q \) such that the rules in \( \delta \) are all of the form \( \ell(f^{(n)}), f^{(n)}, q_1, \ldots, q_n \).

1. Show that, if \( L \) is recognized by a local deterministic top-down tree automaton, then there is a second order signature \( \Sigma \) and a distinguished atomic type \( s \) such that \( L = \mathcal{G}(\Sigma, s) \).
2. Show that, conversely, given a second-order signature $\Sigma$ and a distinguished atomic type $s$, there exists a local top-down deterministic tree automaton $A$ such that $L(A) = G(\Sigma, s)$.

By the previous exercise, not every regular tree language can be expressed as the ground tree language of a second-order signature, e.g. the language $L = \{ f(g(a), g(b)) \}$ is not local.

### 4.1.3 Higher-Order Homomorphisms

One of the main legacies of Richard Montague’s work is the idea that semantic representations can be obtained through the application of a homomorphism on the syntactic structure. However tree homomorphisms are clearly too weak for the kind of tree transductions we want to define; following Montague we use instead higher-order homomorphisms. The idea of these homomorphisms is to translate a syntactic tree representation (e.g. a derivation tree or a dependency tree), seen as a typed $\lambda$-term over the input signature, into a $\lambda$-term over the output signature and then to $\beta\eta$-reduce it to a $\lambda$-term in normal form.

**Definition 4.7 (Higher-Order Homomorphism).** A higher-order homomorphism from a set of constants $C$ to a set of constants $C'$ is generated by a function $J$ mapping constants in $C$ to closed $\lambda$-terms in $\Lambda(C')$. We lift $J$ to a homomorphism from $\Lambda(C)$ to $\Lambda(C')$ by $[x] = x$, $[LM] = [L][M]$, and $[\lambda x.L] = \lambda x.[L]$.

**Example 4.8.** Continuing with Example 2.1, Figure 4.1 presents two syntactic analyses (the dependency one could for instance be obtained from the constituent one through head percolation analysis or as the derivation tree of a TAG). For the constituent analysis of Figure 4.1, we have

$$C = \{\text{John}^{(0)}, \text{apple}^{(0)}, \ldots, \text{AP}^{(2)}, \text{NP}^{(2)}, \text{JJ}^{(1)}, \ldots, \text{S}^{(2)}\}$$

and

$$C' = \{\text{John}^{(0)}, \wedge^{(2)}, \exists^{(2)}, \ldots\}.$$
We assign the semantics:

\[
\begin{align*}
[John^{(0)}] &= \lambda x. x \; John^{(0)} \\
[apple^{(0)}] &= \lambda x. apple^{(1)} \; x \\
[red^{(0)}] &= \lambda x. red^{(1)} \; x \\
[AP^{(2)}] &= \lambda x_1 x_2 x. (x_1 \; x) \land (x_2 \; x) \\
[a^{(-)}] &= \lambda xy. \exists u. (x \; u) \land (y \; u) \\
[NP^{(2)}] &= \lambda x_1 x_2 x_1 \; x_2 \; x \\
[eats^{(0)}] &= \lambda xy. \exists e. (eat^{(1)} \; e) \land x (\lambda a. agent^{(2)} \; e \; a) \\
& \quad \land y (\lambda p. \; patient^{(2)} \; e \; p)
\end{align*}
\]

(ignoring tree nodes with a single child, for which we set e.g. \([NN^{(1)}] = \lambda x_1. x_1\). The first-order variables \(u\) and \(e\) could be considered as constants of arity 0 in \(C'\), but this causes some naming issues; an alternative would be treat \(\exists x. \varphi\) as \(\exists \lambda x. \varphi\). This definition results successively in

\[
\begin{align*}
[AP \; red \; apple] & \Rightarrow^* \lambda x. (red^{(1)} \; x) \land (apple^{(1)} \; x) \\
[NP \; a \; AP \; red \; apple] & \Rightarrow^* \lambda x. \exists u. (red^{(1)} \; u) \land (apple^{(1)} \; u) \land (x \; u) \\
[VP \; eats \; NP \; a \; AP \; red \; apple] & \Rightarrow^* \lambda x. \exists e. (eat^{(1)} \; e) \land x (\lambda a. agent^{(2)} \; e \; a) \\
& \quad \land \exists u. (red^{(1)} \; u) \land (apple^{(1)} \; u) \land (patient^{(2)} \; e \; u) \\
[S. . . ] & \Rightarrow^* \exists e. (eat^{(1)} \; e) \land (agent^{(2)} \; e \; John^{(0)}) \\
& \quad \land \exists u. (red^{(1)} \; u) \land (apple^{(1)} \; u) \land (patient^{(2)} \; e \; u)
\end{align*}
\]

which is the \(\lambda\)-term encoding of (2.24).

(* ) Exercise 4.4. Propose similarly a higher-order homomorphism from the dependency structure of Figure 4.1 into its semantics.

4.1.4 Tree Transductions

The definition we provided for higher-order homomorphisms does not use types explicitly; this is easy to remedy:

**Definition 4.9** (Typed Homomorphism). A **typed homomorphism** between two signatures \(\Sigma = \langle A, C, \tau\rangle\) and \(\Sigma' = \langle A', C', \tau'\rangle\) extends a higher-order homomorphism \([\ ]\) between \(C\) and \(C'\) by mapping each atomic type of \(A\) into a type of \(\mathcal{T}_{A'}\) s.t. \(\vdash_{\Sigma'} \; [c] : [\tau(c)]\) is a valid typing judgement for all \(c\) in \(C\).

**Example 4.10** (Higher-Order Tree Functions). Let us see how this definition can be exercised to define tree transductions. We define the generic tree signature over a ranked alphabet \(\mathcal{F}\) as \(\Sigma_{\mathcal{F}} \overset{\text{def}}{=} \langle \{ o \}, \mathcal{F}, \tau_{\mathcal{F}} \rangle\) where for every \(f^{(u)}\) in \(\mathcal{F}\), \(\tau_{\mathcal{F}}(f^{(u)}) \overset{\text{def}}{=} \underbrace{o \rightarrow \cdots \rightarrow o}_{n \text{ times}} \rightarrow o = o^n \rightarrow o\).

Let \(\Sigma_C\) and \(\Sigma_{C'}\) be two generic tree signatures over the ranked alphabets \(C\) and \(C'\), and let \([\ ]\) be a typed homomorphism between \(\Sigma_C\) and \(\Sigma_{C'}\), and \(s \in A\).
be a distinguished input atomic type with $[s] = o$. We define the corresponding (partial) higher-order tree function $T: T(C) \to T(C')$ by

$$T(\bar{t}_1) = \bar{t}_2 \text{ iff } \vdash_{\Sigma} t_1 : s \land [t_1] \Rightarrow^*_{\beta\eta} t_2.$$  \hfill (4.1)

Note that in this definition, because the bijection $\tau$ between $\lambda$-terms and trees is only defined for ground $\lambda$-terms, $t_2$ must be in $\beta\eta$-normal form.

The semantic construction of Example 4.8 is a higher-order tree function when setting $\Sigma_C$ and $\Sigma_C'$ as input and output signatures and if we consider $e$ and $v$ as nullary constants in $C'$.

**Linear Higher-Order Tree Functions** As often in linguistic applications, a case of particular interest is the linear one: a higher-order homomorphism between $C$ and $C'$ is linear if $[\bar{c}]$ is a linear term for every $c$ in $C$.

**Definition 4.11** (Abstract Categorial Grammar). An abstract categorial grammar (ACG) is a tuple $\mathcal{G} = (\Sigma, \Sigma', [\cdot], s)$ where $\Sigma = \langle A, C, \tau \rangle$ and $\Sigma' = \langle A', C', \tau' \rangle$ are two signatures, $[\cdot]$ is a linear typed homomorphism, and $s$ in $A$ is a distinguished atomic type. The abstract language $\mathcal{A}(\mathcal{G})$ of $\mathcal{G}$ is

$$\mathcal{A}(\mathcal{G}) \overset{\text{def}}{=} \{ L \in \Lambda_{\Sigma}(C) \mid \vdash_{\Sigma} L : s \}$$

the set of closed linear $\lambda$-terms typed by $s$ in the input signature, while its object language $\mathcal{O}(\mathcal{G})$ is

$$\mathcal{O}(\mathcal{G}) \overset{\text{def}}{=} [\mathcal{A}(\mathcal{G})]$$

the set of linear $\lambda$-terms obtained through the application of the homomorphism $[\cdot]$ to abstract terms.

A second-order ACG is an ACG with a second-order abstract signature $\Sigma$. Such ACGs are arguably the most relevant for the linguistic applications. Note that our objects of interest are usually the normal forms found in the object language: these turn out to be exactly the normal forms of the images of the ground terms in $\mathcal{A}(\mathcal{G})$:

$$\downarrow_{\beta\eta} \mathcal{O}(\mathcal{G}) = \downarrow_{\beta\eta} \{ [t] \in \Lambda_{\Sigma'}(C') \mid t \text{ ground } \in \mathcal{A}(\mathcal{G}) \}.$$  \hfill (4.2)

This follows from $\downarrow_{\beta\eta} [L] = \downarrow_{\beta\eta} \downarrow_{\beta\eta} [L]$ since $[\cdot]$ is a higher-order homomorphism, and the fact that a closed term $L$ in normal form is of atomic type $s$ iff it is ground (on a second-order signature).

Therefore, if the object signature is a generic tree signature $\Sigma_{C'}$, then a second-order ACG can be understood as defining a linear higher-order tree function from a local tree language (its abstract language) into the set of trees over $C'$ (its object language). The following exercise examines the simplest such situation, where the homomorphic images of atomic types in the abstract signature are mapped to tree types $o$ in the object signature:

**Exercise 4.5** (Tree Languages of ACG$_{2,1}$). Given an ACG $\mathcal{G} = (\Sigma, \Sigma_F, [\cdot], s)$ with a second-order abstract signature $\Sigma = \langle A, C, \tau \rangle$ and a generic tree signature $\Sigma_F$ over some ranked alphabet $F$ as object signature, we define its tree language as

$$\mathcal{F}(\mathcal{G}) \overset{\text{def}}{=} \{ \bar{t} \in T(F) \mid t \text{ ground } \in \mathcal{O}(\mathcal{G}) \}.$$  \hfill (4.3)

Assume that $\max_{a \in A} \text{ord}([a]) = 1$. Show that such ACGs generate exactly the set of regular tree languages.
More generally, the expressiveness of second-order ACGs has been studied by Kanazawa (2010): their object languages correspond to the tree languages of context-free hyperedge replacement grammars, which are also equivalent to attributed context-free grammars (Engelfriet and Heyker, 1992) and outputs of restricted forms of MTTs (Engelfriet and Maneth, 2000). This means that we could also implement the tree transformations defined by second-order ACGs using more classical tree transductions. However, this would be at the expense of the ability to view the translation as one into higher-order semantics, as we will do in Section 4.3. In that situation, we will no longer work with ground object terms.

4.2 Intensionality

Intensional Phenomena deal with the difference between a meaning and its denotation. A classical example given by Frege is concerned about equality in mathematics: if \( a \) and \( b \) designate the same object, and equality is about objects and not about their names, then there is no difference between “\( a = b \)" and “\( a = a \)". There is however a difference in informational content: the truth of these assertions depends on the context, and there exist contexts that differentiate between the two, namely those where \( a \) and \( b \) do not denote the same object.

Considering an example with more linguistic content, the sentence John knows that the morning star is the evening star might have different truth values depending on the extent of the knowledge of John, but if morning star and evening star are always mapped to the same object, namely Venus, we cannot model the case where John is not aware of their identity. Similar intensional phenomena can occur in relation with temporal modalities instead of epistemic ones: The King of England was the head of the Church of England holds true after King Henry VIII separated the Church from Rome in 1534, thus in worlds after 1534 where the King of England denotes Henry VIII or one of his successors; again an intensional reading should be preferred. A last classical example of Montague contrasts John finds a unicorn with John seeks a unicorn. These are structurally similar, but the first one implies that there exists a unicorn, while the second allows both readings: the so-called de dicto reading which does not imply the existence of unicorns, and the de re reading from which existence of unicorns follows. These two readings could be modelled using different scopes for the modal seeks.

Intensional Logic This reveals an issue with FOML: there is no way to map variables to different objects depending on the world under consideration. The solution adopted in first-order intensional logic (FOIL) is to use two sorts of variables, intensional and extensional ones. Intensions might denote different objects in different worlds: for instance if \( f \) is an intension and \( w \) is a world, then \( f(w) \) would be the extension of \( f \) in \( w \).

There is an issue with this account of intensionality. If \( f \) is an intension and \( P \) a unary predicate, then \( P(f) \) could mean that the extension of \( f \) verifies \( P \) (de re reading), or that the intension \( f \) itself verifies \( P \) (de dicto reading). For instance, The morning star is the evening star would use a de re reading, but The morning star is the last star seen in the morning would be true regardless of the actual object denoted by the morning star. If we consider alethic modalities, \( \Diamond P(f) \) might either mean that in some possible world \( w \), \( P(f(w)) \) holds, or that in some possible world \( w' \), \( P(f) \) holds. In order to distinguish between these alternatives, the de re reading is noted \( [\lambda x.\Diamond P(x)](f) \) and the de dicto one \( \Diamond [\lambda x.P(x)](f) \).
Given an infinite countable set of object variables $O$ and an infinite countable set of intension variables $I$, FOIL formulæ follow the syntax

$$\varphi ::= \lambda x.\varphi(f) \mid \neg\varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \exists y.\varphi$$

where $x, x'$ range over $O$, $f$ over $I$, $y, y_1, \ldots, y_k$ over $I \cup O$, $R_i$ is a $k_i$-ary relational symbol, and $\varphi$ is a formula with a free object variable $x$, so that $[\lambda x.\varphi](f)$ denotes $\varphi(x \leftarrow f)$. We write $[\lambda xx'.\varphi](f, f')$ for $[\lambda x.[\lambda x'.\varphi](g)](f)$. This last construction is a form of abstraction limited to first-order.

**Intensional models** for FOIL are of form $\mathfrak{M} = (W, R, D_O, D_I, I)$ where a distinction is drawn between the object domain $D_O$, which is a non-empty set in our constant semantics, and the intension domain $D_I$, which is a non-empty set of functions from $W$ to $D_O$, and $I$ maps a relational symbols $R_i$ with arity $k_i$ to a mapping $I(R_i)$ from $W$ to relations over $(D_O \cup D_I)^{k_i}$. A valuation is now a mapping assigning members of $D_O$ to object variables and members of $D_I$ to intension variables. The satisfaction relation is similar to that of FOML, with

$$\mathfrak{M}, w \models \nu f.\varphi \quad \text{iff } \exists i \in D_I(w).\mathfrak{M}, w \models [\nu f_{\leftarrow i}]\varphi$$

$$\mathfrak{M}, w \models \nu [\lambda x.\varphi](f) \quad \text{iff } \mathfrak{M}, w \models [\nu [x \leftarrow \nu(f)(w)]](f).$$

**Example 4.12 (Morning Star).** Let us consider again the sentence *The morning star is the evening star* and associate $f$ to the intension *the morning star* and $g$ to the intension *the evening star*. Then $[\lambda xx'.x = x'](f, g)$ is correct in the real world $w$, where $f$ and $g$ are associated to the same object $\nu(f)(w) = \nu(g)(w)$ in $D_O$, namely Venus. In an epistemic setting, the de dicto reading $K[\lambda xx'.x = x'](f, g)$ can be falsified if we find another state of knowledge $w'$ compatible with the real world $w$ where this information is missing, i.e. where $\nu(f)(w') \neq \nu(g)(w')$—this could be the case in the sentence *John knows that the morning star is the evening star* if John is unaware of their both being Venus. By contrast, the de re reading $[\lambda xx'.K(x = x')(f, g)$ is always satisfied in $w$ because in any state of knowledge compatible with the real world, $f$ and $g$ have received the same extension $\nu(f)(w) = \nu(g)(w)$.

**Example 4.13 (King of England).** The treatment of the sentence *The King of England was the head of the Church of England* is similar: consider the intensions $f$ for the *King of England*, $g$ for the *head of the Church of England*, and a point in time $w$. Then $P[\lambda xx'.x = x'](f, g)$ could be invalidated if there is no past time $w' < w$ where the denotations $\nu(f)(w')$ and $\nu(g)(w')$ were the same—i.e. before the 1538 secession from the Roman Church, but is valid in time points $w$ after the secession. The de re reading does not make any sense: $[\lambda xx'.P(x = x')(f, g)$ holds iff $\nu(f)(w) = \nu(g)(w)$ at the time of interest, regardless of past times where equality is evaluated.

**Total Intensionality** Let $D(f, x)$ stand for $[\lambda xx'.x = x'](f)$ where $x$ and $x'$ are distinct object variables. Then $\mathfrak{M}, w \models \nu D(f, x)$ holds iff $\nu(f)(w) = \nu(x)$.

The formula $\forall f \exists x. D(f, x)$ is valid in intensional models as defined so far, since $\nu(f)$ is a total function from $W$ to $D_O$. There is however no requirement for every object to be designated by some intension, i.e. for

$$\forall x. \exists f. D(f, x) \quad (4.4)$$

to hold. This is however a reasonable restriction; let us check for instance the following equivalence under the hypothesis of (4.4):

$$\exists x. \varphi \equiv \exists f.[\lambda x.\varphi](f) . \quad (4.5)$$
Indeed, for all \( \mathcal{M}, w, \nu \) and \( \varphi \),
\[
\mathcal{M}, w \models \nu \exists x.\varphi(\nu)(f)\text{ iff } \exists i \in D_I.\mathcal{M}, w \models \nu[\lambda x.\varphi(f)](i)\varphi
\]
(by (4.4) when choosing \( i(w) = e \))
\[
\text{iff } \exists i \in D_I.\mathcal{M}, w \models \nu[\lambda x.\varphi(f←i)](i)\varphi
\]
\[
\text{iff } \exists i \in D_I.\mathcal{M}, w \models \nu[\lambda x.\varphi(f←i,x←i(w))](i)\varphi
\]
\[
\text{iff } \exists e \in D_O.\mathcal{M}, w \models \nu[\lambda x.\varphi(x←e)](e)\text{ (by (4.4) when choosing } i(w) = e)\)
\]
and if the interpretation of \( \text{unicorn}^{(1)} \) is the same in all worlds accessible through the \( \text{TRY} \) modality,
\[
\equiv \exists x.\,\text{unicorn}^{(1)}(x) \wedge \text{TRY}(\mathcal{M}, w, \lambda x.\text{unicorn}^{(1)}(x))
\]
\[
\equiv \exists x.\text{unicorn}^{(1)}(x) \wedge \text{TRY}(\mathcal{M}, w, \lambda x.\text{unicorn}^{(1)}(x))\text{ (by (4.5))}
\]

### Exercise 4.6.
Show the following equivalence when (4.4) holds:
\[
\exists f.\Diamond[\lambda x.\varphi(f)](f) \equiv \Diamond(\exists x.\varphi).
\]

### Example 4.14 (Unicorn).
The sentence \( \text{John finds a unicorn} \) could be associated with the semantics
\[
\exists e.\text{find}^{(1)}(e) \wedge \text{agent}^{(2)}(e, \text{John}^{(0)}) \wedge \text{patient}^{(2)}(e, x) \wedge \text{unicorn}^{(1)}(x)
\]
but it is better to treat \( \text{unicorn} \) as an intension in the formula
\[
\exists u.\lambda x.\exists e.\text{find}^{(1)}(e) \wedge \text{agent}^{(2)}(e, \text{John}^{(0)}) \wedge \text{patient}^{(2)}(e, x) \wedge \text{unicorn}^{(1)}(x)(u)
\]
equivalent to (4.7) in totally intensional models according to (4.5). Then we better see the connection with the sentence \( \text{John seeks a unicorn} \) : its de dicto semantics would be
\[
\exists u.\lambda x.\text{TRY}(\text{John}^{(0)}, \exists e.\text{find}^{(1)}(e) \wedge \text{patient}^{(2)}(e, x) \wedge \text{unicorn}^{(1)}(x))(u)
\]
\[
\equiv \text{TRY}(\text{John}^{(0)}, \exists e.\text{find}^{(1)}(e) \wedge \text{patient}^{(2)}(e, x) \wedge \text{unicorn}^{(1)}(x))\text{ (by (4.6))}
\]
and its de re semantics
\[
\equiv \exists x.\text{TRY}(\text{John}^{(0)}, \exists e.\text{find}^{(1)}(e) \wedge \text{patient}^{(2)}(e, x) \wedge \text{unicorn}^{(1)}(x))\text{ (by (4.5))}
\]
and if the interpretation of \( \text{unicorn}^{(1)} \) is the same in all worlds accessible through the \( \text{TRY} \) modality,
\[
\equiv \exists x.\text{unicorn}^{(1)}(x) \wedge \text{TRY}(\text{John}^{(0)}, \exists e.\text{find}^{(1)}(e) \wedge \text{patient}^{(2)}(e, x))
\]

### 4.3 Higher-Order Logic

Most of the discussion on semantic representations can be recast in the framework of higher-order logic. This allows in particular to view the higher-order operations of Section 4.1 not as a technical means to generate \( \lambda \)-terms viewed as trees (which in turn can be interpreted in some logic), but instead to interpret these terms directly in the higher-order logic. They become the semantics of the sentences under consideration, with associated models.

#### 4.3.1 Background: Church’s Simple Theory of Types

Higher-order semantics are typically expressed in simply typed lambda calculus as defined in Section 4.1.1. As we want not just to manipulate typed \( \lambda \)-terms, but also to be able to infer truths, we need to introduce a set of logical constants and the associated logical rules.
Higher-Order Signature  In Church’s simple theory of types, we use a signature \( \Sigma = \langle A, C, t \rangle \) where \( A = \{ \iota, o \} \) is set of atomic types, where \( \iota \) denotes entities and \( o \) truths. The logical constants are \( C = \{ \bot, \supset, (\forall_\tau \in T(A)) \} \) with types \( t(\bot) = o \), \( t(\supset) = o \to o \to o \), and \( (\forall_\tau) = (\tau \to o) \to o \) for each type \( \tau \) in \( T(A) \).

We write as usual \( L \supset M \) for \( \supset LM \) and \( \forall_\tau x. L \) for \( \forall_\tau (\lambda x. L) \). The other logical connectives are defined as usual: \( \neg L \defeq L \supset \bot \), \( L \lor M \defeq (\neg L) \supset M \), \( L \land M \defeq (\neg (\neg L) \lor (\neg M)) \), etc. Equality is defined in the Leibnizian way as \( L = M \defeq \forall x. xL \supset xM \), i.e. equality is defined as having \( L \) and \( M \) agree on all possible properties \( x \).

Logical and Conversion Rules  The formal system needs two types of rules: logical rules for the logical constants, and conversion rules for the \( \lambda \)-terms. In natural deduction sequent style,

- \( \Gamma, L \vdash L \) (Ax)  
- \( \Gamma, \neg L \vdash \bot \) (\( \bot \)E)
- \( \Gamma, L \vdash M \)  
  \( \Gamma \vdash L \supset M \) (\( \supset \)I)  
  \( \Gamma \vdash L \supset M \)  
  \( \Gamma \vdash L \)  
  \( \Gamma \vdash M \) (\( \supset \)E)
- \( \Gamma \vdash L \)  
  \( x \notin \text{FV}(\Gamma) \)  
  \( \Gamma \vdash \forall_\tau x. L \) (\( \forall \)I)
- \( \Gamma \vdash \forall_\tau L \)  
  \( \Delta \vdash \Sigma : \tau \) (\( \forall \)E)
- \( \Gamma \vdash L \)  
  \( L =_\beta M \)  
  \( \Gamma \vdash M \) (\( \beta \))

The deduction system also often includes the extensionality axioms:

- \( \Gamma \vdash (\forall_\tau x. Lx =_M Mx) \supset (L = M) \) (\( \lambda X \))
- \( \Gamma \vdash (L \equiv M) \supset (L = M) \) (\( \equiv X \))

As their name indicates, the extensionality axioms make the simple theory of types unable to deal with intensional phenomena directly; a solution we will see in Section 4.3.2 will be to introduce an new atomic type \( s \) ranging over worlds.

Higher-order logic can express a form of set theory: view the set comprehension \( \{ x \mid P \} \) as \( \lambda x. P \), or \( e \in E \) as \( Ee \). In fact, Church (1940) shows how to implement Peano’s arithmetic in the simple theory of types, from which we can deduce the incompleteness of higher-order logic.

Standard Models  Higher-order logic comes with a very natural model theory. For each \( \tau \) in \( T(A) \), let \( D_\tau \) be the domain of expressions of type \( \tau \). Let \( D_o = \{ \bot, \supset, \} \) and \( D_\iota \) be some set of entities; then \( D_{\iota \to o} \) denotes the set of functions from \( D_\iota \) to \( D_o \), so that e.g. \( D_{\iota \to o} \) is the type of first-order predicates.

4.3.2 Type-Logical Semantics

Many classical modellings of natural language semantics in higher-order logic posit an additional type \( s \) of worlds in order to account for modalities and intensionality phenomena. The idea is to always treat truth values (of type \( o \)) as relativized with respect to a possible world of evaluation. Thus we will consider a
In order to obtain the appropriate type, a possibility is to set \( t(\text{walk}) = \tau \rightarrow s \rightarrow o \) and \( t(\text{John}) = \tau \). Looking at more complex examples (for instance Example 4.8), we arrive at the types of Table 4.1. The semantics of a sentence can then be computed by a higher-order homomorphism as in Section 4.1, but there will be no need to translate back from \( \lambda \)-terms to first-order terms in order to reason about the semantics; the \( \lambda \)-term is a meaning representation with full-fledged model theory. See Table 4.2 for some examples of semantic values.

In this table, the semantics of alethic and epistemic modal logics have been implemented directly using the \( R \), \( K \), and \( B \) constants with types \( s \rightarrow s \rightarrow o \),
ι → s → s → o, and ι → s → s → o respectively. The desired properties of these
relations can also be enforced; for instance ∀s,ww′. Rww′ forces R to be total.
Chapter 5

References


