MPRI course 2-27-1, year 2011–2012

A Short Introduction to Formal Linguistics

Sylvain Schmitz
LSV, ENS Cachan & CNRS
November 30, 2011 (r1640M)

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 modeling*. Among their prerequisites 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 also is advisable.

Contents

Further Reading

Interested students will find a good general textbook on natural language processing in Jurafsky and Martin (2009). The present notes have a strong bias towards formal language theory—reference textbooks in this domain include (Harrison, 1978; Berstel, 1979; Sakarovitch, 2009; Comon et al., 2007)—, but this is hardly representative of the general field of natural language processing and computational linguistics. In particular, the overwhelming importance of statistical approaches in the current body of research makes the textbook of Manning and Schütze (1999) another recommended reference.

The main journal of natural language processing is *Computational Linguistics*. As often in computer science, the main conferences of the field have equivalent if not greater importance than journal outlets, and one will find among the major conferences *ACL* ("Annual Meeting of the Association for Computational Linguistics"), *EACL* ("European Chapter of the ACL"), *NAACL* ("North American Chapter of the ACL"), and *CoLing* ("International Conference on Computational Linguistics"). A very good point in favor of the ACL community is their early adoption of open access; one will find all the ACL publications online at `http://www.aclweb.org/anthology/`.

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 (although the treatment of e.g. rational relations in **??** can be generalized to n-ary relations). 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 $R_1 \fatsemi R_2 = \{(a, c) \mid \exists b \in B, (a, b) \in R_1 \wedge (b, c) \in R_2\}$. The **reflexive transitive closure** of a relation is noted $R^\star = \bigcup_i R^i$, where $R^0 = \mathrm{Id}_A = \{(a, a) \mid a \in A\}$ is the **identity** over A, and $R^{i+1} = R \fatsemi R^i$.

Monoids. A **monoid** $\langle \mathbb{M}, \cdot, 1_\mathbb{M} \rangle$ is a set of elements $\mathbb{M}$ along with an associative operation $\cdot$ and a neutral element $1_\mathbb{M} \in \mathbb{M}$. We are often dealing with the **free monoid** $\langle \Sigma^*, \cdot, \varepsilon \rangle$ generated by concatenation $\cdot$ of elements from a finite set Σ. A monoid is **commutative** if $a \cdot b = b \cdot a$ for all a, b in $\mathbb{M}$.

We lift $\cdot$ to subsets of $\mathbb{M}$ by $L_1 \cdot L_2 = \{m_1 \cdot m_2 \mid m_1 \in L_1, m_2 \in L_2\}$. Then for $L \subseteq \mathbb{M}$, $L^0 = \{1_\mathbb{M}\}$ and $L^{i+1} = L \cdot L^i$, and we define the **Kleene star** operator by $L^* = \bigcup_i L^i$.

Semirings. A **semiring** $\langle \mathbb{K}, \oplus, \odot, 0_\mathbb{K}, 1_\mathbb{K} \rangle$ is endowed with two binary operations, an addition $\oplus$ and a multiplication $\odot$ such that

- $\langle \mathbb{K}, \oplus, 0_\mathbb{K} \rangle$ is a commutative monoid for addition with $0_\mathbb{K}$ for neutral element,
- $\langle \mathbb{K}, \odot, 1_\mathbb{K} \rangle$ is a monoid for multiplication with $1_\mathbb{K}$ for neutral element,
- multiplication distributes over addition, i.e. $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$ and $(a \oplus b) \odot c = (a \odot c) \oplus (b \odot c)$ for all a, b, c in $\mathbb{K}$,
- $0_\mathbb{K}$ is a zero for multiplication, i.e. $a \odot 0_\mathbb{K} = 0_\mathbb{K} \odot a = 0_\mathbb{K}$ for all a in $\mathbb{K}$.

Among the semirings of interest are the

- boolean semiring $\langle \mathbb{B}, \vee, \wedge, 0, 1 \rangle$ where $\mathbb{B} = \{0, 1\}$,
- probabilistic semiring $\langle \mathbb{R}_+, +, \cdot, 0, 1 \rangle$ where $\mathbb{R}_+ = [0, +\infty)$ is the set of positive reals (sometimes restricted to $[0, 1]$ when in presence of a probability distribution),
- tropical semiring $\langle \mathbb{R}_+ \uplus \{+\infty\}, \min, +, +\infty, 0 \rangle$,
- rational semiring $\langle \mathrm{Rat}(\Delta^*), \cup, \cdot, \emptyset, \{\varepsilon\} \rangle$ where $\mathrm{Rat}(\Delta^*)$ is the set of rational sets over some alphabet Δ.

String Rewrite Systems. A **string rewrite system** or **semi-Thue systems** over an alphabet Σ is a relation $R \subseteq \Sigma^* \times \Sigma^*$. The elements (u, v) of R are called **string rewrite rules** and noted $u \to v$. The **one step derivation relation** generated by R, noted $\overset{R}{\Rightarrow}$, is the relation over Σ^* defined for all w, w' in Σ^* by $w \overset{R}{\Rightarrow} w'$ iff there exist x, y in Σ^* such that $w = xuy$, $w' = xvy$, and $u \to v$ is in R. The **derivation relation** is the reflexive transitive closure $\overset{R}{\Rightarrow}{}^\star$.

See also the monograph by Book and Otto (1993).

Prefixes. The **prefix ordering** $\leq_{\mathrm{pref}}$ over Σ^* is defined by $u \leq_{\mathrm{pref}} v$ iff there exists v' in Σ^* such that $v = uv'$. We note $\mathrm{Pref}(v) = \{u \mid u \leq_{\mathrm{pref}} v\}$ the set of prefixes of v, and $u \wedge v$ the longest common prefix of u and v.

See Comon et al. *(2007) for missing definitions and notations.*

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

Let $\mathcal{X}$ be a set of **variables**, each with arity 0, assumed distinct from Σ. 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 Σ 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 Σ_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 $\mathcal{C}^n(\Sigma)$ the set of contexts with n variables, and $\mathcal{C}(\Sigma)$ 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 Σ, 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 $ui \in \mathrm{dom}(t)$ for u in $\mathbb{N}^*$ and i in $\mathbb{N}$, then $u \in \mathrm{dom}(t)$, and **predecessor-closed**, i.e. if $ui \in \mathrm{dom}(t)$ for u in $\mathbb{N}^*$ and i in $\mathbb{N}_{>0}$, then $u(i-1) \in \mathrm{dom}(t)$.

The set Σ 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 Σ 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 Σ 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 $\overset{r}{\Rightarrow} = \{(C[t\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}], C[t'\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}]) \mid C \in \mathcal{C}(\Sigma), t_1, \ldots, t_n \in T(\Sigma)\}$. We write $\overset{r_1 r_2}{\Longrightarrow}$ for $\overset{r_1}{\Rightarrow} \fatsemi \overset{r_2}{\Rightarrow}$, and $\overset{R}{\Rightarrow}$ for $\bigcup_{r \in R} \overset{r}{\Rightarrow}$.

Chapter 1

Mildly Context-Sensitive Syntax

Recall that **context-sensitive languages** (aka **type-1 languages**) are defined by phrase structure grammars with rules of form $\lambda A \rho \to \lambda \alpha \rho$ with A in N, λ, ρ in V^*, and α in V^+. Their expressive power is equivalent to that of **linear bounded automata** (LBA), i.e. Turing machines working in linear space. Such grammars are not very useful from a computational viewpoint: membership is PSPACE-complete, and emptiness is undecidable.

Still, for the purposes of constituent analysis of syntax, one would like to use string- and tree-generating formalisms with greater expressive power than context-free grammars. The rationale is twofold:

- some natural language constructs are not context-free, the Swiss-German account by Shieber (1985) being the best known example. Such fragments typically involve so-called **limited cross-serial dependencies**, as in the languages $\{a^n b^m c^n d^m \mid n, m \geq 0\}$ or $\{ww \mid w \in \{a, b\}^*\}$.
- the class of regular tree languages is not rich enough to account for the desired linguistic analyses (e.g. Kroch and Santorini, 1991, for Dutch).

This second argument is actually the strongest: the class of tree structures and how they are combined—which ideally should relate to how semantics compose—in context-free grammars are not satisfactory from a linguistic modeling point of view.

Based on his experience with **tree-adjoining grammars** (TAGs) and weakly equivalent formalisms (head grammars, a version of combinatory categorial grammars, and linear indexed grammars; see Joshi et al., 1991), Joshi (1985) proposed an *informal* definition of which properties a class of formal languages should have for linguistic applications: **mildly context-sensitive languages** (MCSLs) were "roughly" defined as the extensions of context-free languages that accommodate

1. *limited cross-serial dependencies*, while preserving
2. constant growth—a requisite nowadays replaced by **semilinearity**, which demands the Parikh image of the language to be a semilinear subset of $\mathbb{N}^{|\Sigma|}$ (Parikh, 1966), and
3. *polynomial time recognition*.

A possible *formal* definition for MCSLs is the class of languages generated by **multiple context-free grammars** (MCFGs, Seki et al., 1991), or equivalently **linear context-free rewrite systems** (LCFRSs, Weir, 1992), **multi-component tree adjoining grammars** (MCTAGs), and quite a few more.

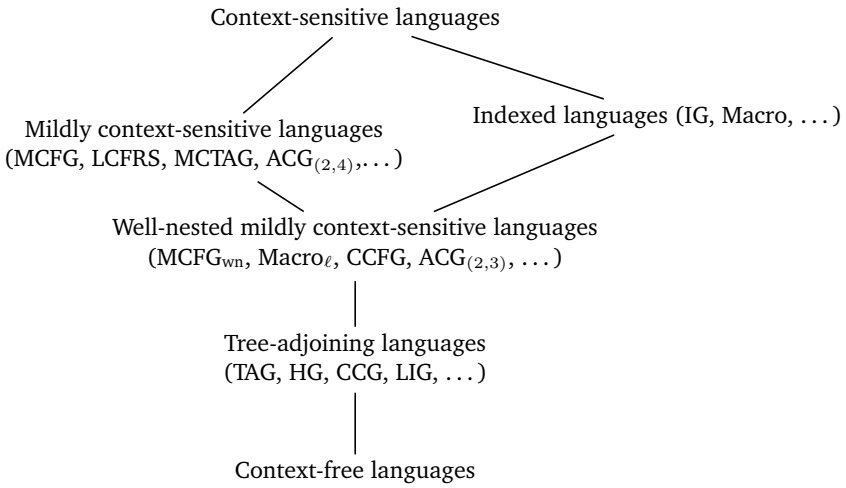

Figure 1.1: Hierarchies between context-free and full context-sensitive languages.

We will however concentrate on two strict subclasses: tree adjoining languages (TALs, Section 1.1) and well-nested MCSLs (wnMCSLs, Section 1.2); Figure 1.1 illustrates the relationship between these classes. As in **??** our main focus will be on the corresponding tree languages, representing linguistic constituency analyses and sentence composition.

1.1 Tree Adjoining Grammars

Tree-adjoining grammars are a restricted class of term rewrite systems (we will see later that they are more precisely a subclass of the linear monadic context-free tree grammars). They have first been defined by Joshi et al. (1975) and subsequentely extended in various ways; see Joshi and Schabes (1997) for the "standard" definitions.

Definition 1.1 (Tree Adjoining Grammars)**.** A **tree adjoining grammar** (TAG) is a tuple $\mathcal{G} = \langle N, \Sigma, T_\alpha, T_\beta, S\rangle$ where N is a finite *nonterminal* alphabet, Σ a finite *terminal* and $N \cap \Sigma = \emptyset$, T_α and T_β two finite sets of finite **initial** and **auxiliary** trees, where $T_\alpha \cup T_\beta$ is called the set of **elementary** trees, and S in N a *start symbol*.

Given the nonterminal alphabet N, define

- $N{\downarrow} \stackrel{\text{def}}{=} \{A{\downarrow} \mid A \in N\}$ the ranked alphabet of **substitution** labels, all with arity 0,
- $N^{\text{na}} \stackrel{\text{def}}{=} \{A^{\text{na}} \mid A \in N\}$ the unranked alphabet of **null adjunction** labels,
- $N_\star \stackrel{\text{def}}{=} \{A_\star \mid A \in N \cup N^{\text{na}}\}$ the ranked alphabet of **foot** variables, all with arity 0.

In order to work on ranked trees, we confuse N with $N_{>0}$, Σ with Σ_0, and N^{na} with $N^{\text{na}}_{>0}$ in the following. Then the set $T_\alpha \cup T_\beta$ of elementary trees is a set of trees of height at least one. They always have a root labeled by a symbol in $N \cup N^{\text{na}}$, and we define accordingly $\mathrm{rl}(t)$ of a tree t as its *unranked* root label modulo $^{\text{na}}$: $\mathrm{rl}(t) \stackrel{\text{def}}{=} A$ if there exists m in $\mathbb{N}_{>0}$, $t(\varepsilon) = A^{(m)}$ or $t(\varepsilon) = A^{\text{na}(m)}$. Then

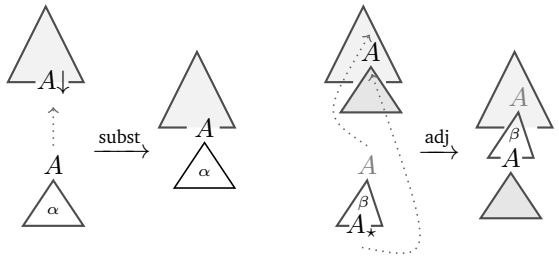

Figure 1.2: Schematics for the substitution and adjunction operations.

- $T_\alpha \subseteq T(N \cup N{\downarrow} \cup N^{\mathrm{na}} \cup \Sigma \cup \{\varepsilon^{(0)}\})$ is a finite set of finite trees α with nonterminal or null adjunction symbols as internal node labels, and terminal symbols or ε or substitution symbols as leaf labels;
- $T_\beta \subseteq T(N \cup N{\downarrow} \cup N^{\mathrm{na}} \cup \Sigma \cup \{\varepsilon^{(0)}\}, N_\star)$ trees $\beta[A_\star]$ are defined similarly, except for the additional condition that they should have *exactly* one leaf, called the **foot node**, labeled by a variable $A_\star$, which has to match the root label $A = \mathrm{rl}(\beta)$. The foot node $A_\star$ acts as a hole, and the auxiliary tree is basically a context.

The semantics of a TAG is that of a finite term rewrite system with rules (see Figure 1.2)

$$
\begin{aligned}
R_{\mathcal{G}} \stackrel{\text{def}}{=} & \{A{\downarrow} \to \alpha \mid \alpha \in T_\alpha \wedge \mathrm{rl}(\alpha) = A\} && \text{(substitution)}\\
\cup\ & \{A^{(m)}(x_1, \ldots, x_m) \to \beta[A^{(m)}(x_1, \ldots, x_m)] \mid m \in \mathbb{N}_{>0}, A^{(m)} \in N_m, \beta[A_\star] \in T_\beta\}\\
\cup\ & \{A^{(m)}(x_1, \ldots, x_m) \to \beta[A^{\mathrm{na}(m)}(x_1, \ldots, x_m)] \mid m \in \mathbb{N}_{>0}, A^{(m)} \in N_m, \beta[A^{\mathrm{na}}_\star] \in T_\beta\}\ . && \text{(adjunction)}
\end{aligned}
$$

A **derivation** starts with an initial tree in T_α and applies rules from $R_{\mathcal{G}}$ until no substitution node is left:

$$L_T(\mathcal{G}) \stackrel{\text{def}}{=} \{h(t) \mid \exists t \in T(N \cup \Sigma \cup \{\varepsilon^{(0)}\}), \exists \alpha \in T_\alpha, \mathrm{rl}(\alpha) = S \wedge \alpha \stackrel{R_{\mathcal{G}}}{\Longrightarrow}{}^\star\ t\}$$

is the **tree language** of $\mathcal{G}$, where the $^{\mathrm{na}}$ annotations are disposed of, thanks to an alphabetic tree homomorphism h generated by $h(A^{\mathrm{na}(m)}) \stackrel{\text{def}}{=} A^{(m)}$ for all $A^{\mathrm{na}(m)}$ of N^{na}, and $h(X) \stackrel{\text{def}}{=} X$ for all X in $N \cup \Sigma \cup \{\varepsilon^{(0)}\}$. The **string language** of $\mathcal{G}$ is

$$L(\mathcal{G}) \stackrel{\text{def}}{=} \mathrm{yield}(L_T(\mathcal{G}))$$

the set of yields of all its trees.

Example 1.2. Figure 1.3 presents a tree adjoining grammar with

$$
\begin{aligned}
N &= \{\mathsf{S}, \mathsf{NP}, \mathsf{VP}, \mathsf{VBZ}, \mathsf{NNP}, \mathsf{NNS}, \mathsf{RB}\}\ ,\\
\Sigma &= \{\mathit{likes}, \mathit{Bill}, \mathit{mushrooms}, \mathit{really}\}\ ,\\
T_\alpha &= \{\alpha_1, \alpha_2, \alpha_3\}\ ,\\
T_\beta &= \{\beta_1\}\ ,\\
S &= \mathsf{S}\ .
\end{aligned}
$$

Its sole S-rooted initial tree is α_1, on which one can substitute α_2 or α_3 in order to get *Bill likes mushrooms* or *mushrooms likes mushrooms*; the adjunction of β_1 on the

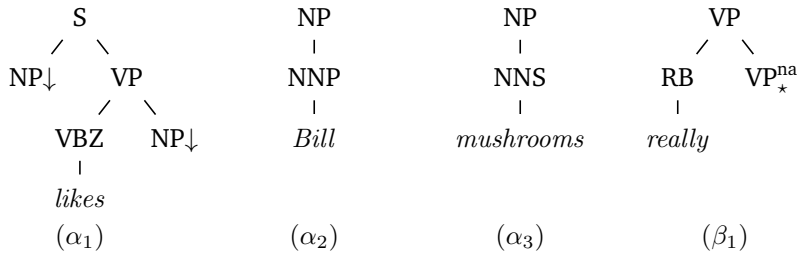

Figure 1.3: A tree adjoining grammar.

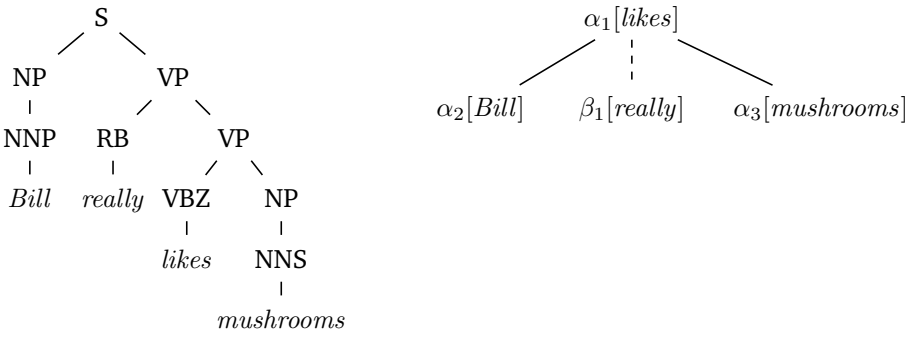

Figure 1.4: A derived tree and the corresponding derivation tree for the TAG of Example 1.2.

VP node of α_1 also yields *Bill really likes mushrooms* (see Figure 1.4) or *mushrooms really really really likes Bill*. In the TAG literature, a tree in $T(N \cup N^{\mathrm{na}} \cup \Sigma \cup \{\varepsilon^{(0)}\})$ obtained through the substitution and adjunction operations is called a **derived tree**, while a **derivation tree** records how the rewrites took place (see Figure 1.4 for an example; children of an elementary tree are shown in addressing order, with plain lines for substitutions and dashed lines for adjunctions).

Example 1.3 (Copy Language)**.** The **copy language** $L_{\text{copy}} \stackrel{\text{def}}{=} \{ww \mid w \in \{a,b\}^*\}$ is generated by the TAG of Figure 1.5 with $N = \{S\}$, $\Sigma = \{a,b\}$, $T_\alpha = \{\alpha_\varepsilon\}$, and $T_\beta = \{\beta_a, \beta_b\}$.

(∗) **Exercise 1.1.** Give a TAG for the language $\{a^n b^m c^n d^m \mid n, m \geq 0\}$.

1.1.1 Linguistic Analyses Using TAGs

Starting in particular with Kroch and Joshi (1985)'s work, the body of literature on linguistic analyses using TAGs and their variants is quite large. As significant evidence of the practical interest of TAGs, the XTAG project (XTAG Research Group, 2001) has published a large TAG for English, with a few more than 1,000 elementary unanchored trees. This particular variant of TAGs, a **lexicalized**, **feature-based** TAG, uses finite **feature structures** and **lexical anchors**. We will briefly survey the architecture of this grammar, and give a short account of it how treats some long-distance dependencies in English.

Lexicalized Grammar

A TAG is **lexicalized** if all its elementary trees have at least one terminal symbol as a leaf. In linguistic modeling, it will actually have one distinguished terminal symbol, called the **anchor**, plus possibly some other terminal symbols, called

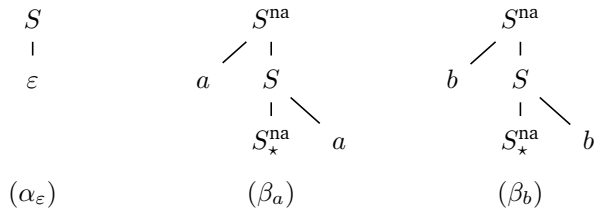

Figure 1.5: A TAG for L_{copy}.

coanchors. An anchor serves as head word for at least a part of the elementary tree, as *likes* for α_1 in Figure 1.3. Coanchors serve for particles, prepositions, etc., whose use is mandatory in the syntactic phenomenon modeled by the elementary tree, as *by* for α_5 in Figure 1.6.

Subcategorization Frames Each elementary tree then instantiates a **subcategorization frame** for its anchor, i.e. specifications of the number and categories of the arguments of a word. For instance, *to like* is a **transitive verb** taking a NP subject and a NP complement, as instantiated by α_1 in Figure 1.3; similarly, *to think* takes a clausal S complement, as instantiated by β_2 in Figure 1.6. These first two examples are **canonical** instantiations of the subcategorization frames of *to like* and *to think*, but there are other possible instantiations, for instance **interrogative** with α_4 or **passive** with α_5 for *to like*.

*A more principled organization of the trees for subcategorization frames and their various instantiations can be obtained thanks to a **meta grammar** describing the set of elementary trees (see e.g. Crabbé, 2005).*

Example 1.4. Extend the TAG of Figure 1.3 with the trees of Figure 1.6. This new grammar is now able to generate

> mushrooms are liked by Bill
> mushrooms think Bill likes Bill
> who does Bill really think Bill really likes

In a feature-based grammar, both the obligatory adjunction of a single β_3 on the S node of α_4, and that of a single β_4 on the VP node of α_5 are controlled through the feature structures, and there is no overgeneration from this simple grammar.

Syntactic Lexicon In practice, elementary trees as the ones of Figure 1.3 are not present as such in the XTAG grammar. It rather contains **unanchored** versions of these trees, with a specific marker $\diamond$ for the anchor position. For instance, α_2 in Figure 1.3 would be stored as a context NP(NNP($\diamond$)) and enough information to know that *Bill* anchors this tree.

The anchoring information is stored in a **syntactic lexicon** associating with each lexical entry classes of trees that it anchors. The XTAG project has developed a naming ontology for these classes based on subcategorization frame and type of construction (e.g. canonical, passive, ...).

Long-Distance Dependencies

Let us focus on α_4 in Figure 1.6. The "move" of the object NP argument of *likes* into sentence-first position as a WhNP is called a **long-distance dependency**. Observe that a CFG analysis would be difficult to come with, as this "move" crosses through the VP subtree of *think*—see the dotted dependency in the derived tree of Figure 1.7. We leave the question of syntax/semantics interfaces using derivation trees to later chapters.

See Schabes and Shieber (1994) for an alternative definition of adjunction, which yields more natural derivation trees. Among the possible interfaces to semantics, let us mention the use of feature structures (Gardent and Kallmeyer, 2003; Kallmeyer and Romero, 2004), or better a mapping from the derivation structures to logical ones (de Groote, 2001).

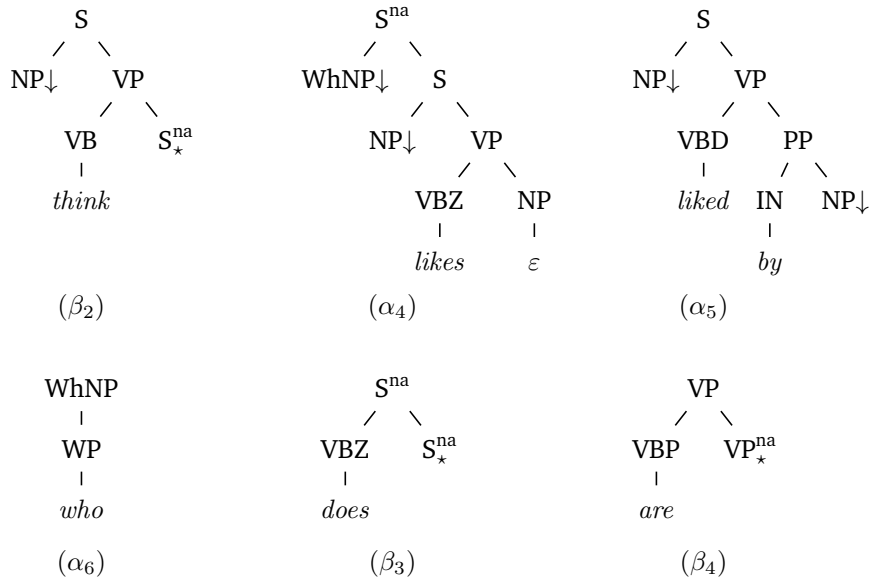

Figure 1.6: More elementary trees for the tree adjoining grammar of Example 1.2.

1.1.2 *Background:* Context-Free Tree Grammars

Context-free tree languages are an extension of regular tree languages proposed by Rounds (1970):

See Gécseg and Steinby (1997, Section 15) and Comon et al. *(2007, Section 2.5). Regarding string languages, the set* $\mathrm{yield}(L(\mathcal{G}))$ *of CFTGs characterizes the class of* ***indexed languages*** *(Aho, 1968; Fischer, 1968). Context-free tree languages are also defined through* ***top-down pushdown tree automata*** *(Guessarian, 1983).*

Definition 1.5 (Context-Free Tree Grammars)**.** A **context-free tree grammar** (CFTG) is a tuple $\mathcal{G} = \langle N, \Sigma, S, R\rangle$ consisting of a ranked *nonterminal* alphabet N, a ranked *terminal* alphabet Σ, an *axiom* $S^{(0)}$ in N_0, and a finite set of rules R of form $A^{(n)}(y_1, \ldots, y_n) \to e$ with $e \in T(N \cup \Sigma, \mathcal{Y}_n)$ where $\mathcal{Y}$ is an infinite countable set of *parameters*. The *language* of $\mathcal{G}$ is defined as

$$L(\mathcal{G}) \stackrel{\text{def}}{=} \{t \in T(\Sigma) \mid S^{(0)} \stackrel{R}{\Rightarrow}{}^{\star}\, t\}.$$

Observe that a **regular tree grammar** is simply a CFTG where every nonterminal is of arity 0.

Example 1.6 (Squares)**.** The CFTG with rules

$$S \to A(a, f(a, f(a, a)))$$
$$A(y_1, y_2) \to A(f(y_1, y_2), f(y_2, f(a, a))) \mid y_2$$

has $\{a^{n^2} \mid n \geq 1\}$ for $\mathrm{yield}(L(\mathcal{G}))$: Note that

$$\sum_{i=0}^{n-1} 2i + 1 = n + 2\sum_{i=0}^{n-1} i = n^2 \tag{1.1}$$

and that if $S \Rightarrow^n A(t_1, t_2)$, then $\mathrm{yield}(t_1) = a^{n^2}$ and $\mathrm{yield}(t_2) = 2n + 1$.

Example 1.7 (Non-primes)**.** The CFTG with rules

$$S \to A(f(a, a))$$
$$A(y) \to A(f(y, a)) \mid B(y)$$
$$B(y) \to f(y, B(y)) \mid f(y, y)$$

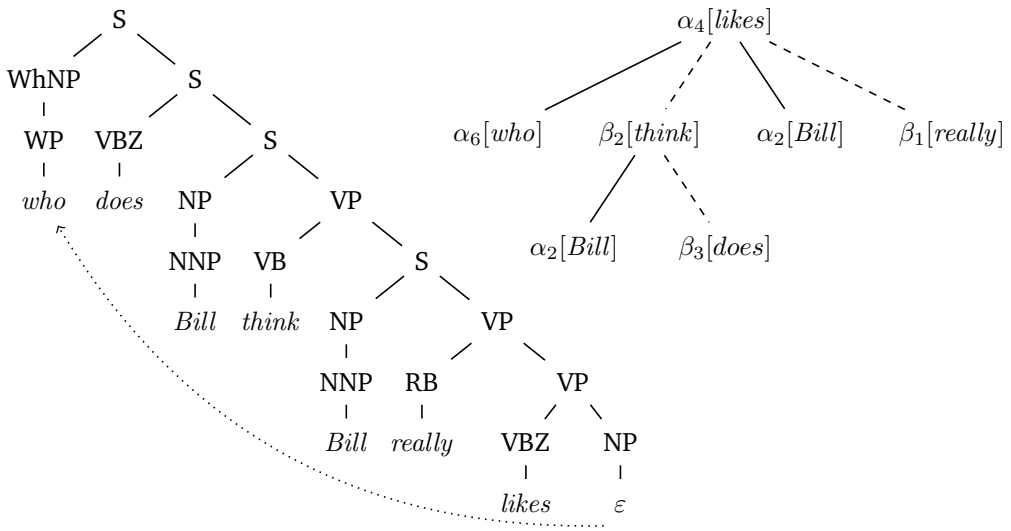

Figure 1.7: Derived and derivation trees for *Who does Bill think Bill really likes?* using the TAG of Figures 1.3 and 1.6.

has $\{a^n \mid n \geq 2 \text{ is not a prime}\}$ for $\text{yield}(L(\mathcal{G}))$: in a derivation

$$S \Rightarrow A(f(a,a)) \Rightarrow^m A(t) \Rightarrow B(t) \Rightarrow^n C[B(t)] \Rightarrow t'$$

with t' in $T(\Sigma)$, we have $\text{yield}(t) = a^{2+m}$, $\text{yield}(C[B(t)]) = a^{(2+m)n}$, and finally $\text{yield}(t') = a^{(2+m)(n+1)}$.

Exercise 1.2 (Powers of 2)**.** Give a CFTG with $\text{yield}(L(\mathcal{G})) = \{a^n b a^{2^n} \mid n \geq 1\}$. **(∗)**

Exercise 1.3 (Normal Form)**.** Show that any CFTG can be put in a normal form where every rule in R is either of form $A^{(n)}(y_1, \ldots, y_n) \to a^{(n)}(y_1, \ldots, y_n)$ with a in Σ_n or of form $A^{(n)}(y_1, \ldots, y_n) \to e$ with e in $T(N, \mathcal{Y}_n)$. **(∗)**

IO and OI Derivations

If we see derivations in a CFTG as evaluation in a recursive program with nonterminals are functions, a natural way to define the semantics of a nonterminal $A^{(n)}$ is for them to take fully derived trees in $T(\Sigma)$ as parameters, i.e. to use *call-by-value* semantics, or equivalently inside-out (**IO**) evaluation of the rewrite rules, i.e. evaluation starting from the innermost nonterminals. The dual possibility is to consider outside-in (**OI**) evaluation, which corresponds to *call-by-name* semantics. Formally, for a set of rewrite rules R,

See Fischer (1968).

$$\overset{\text{IO}}{\Longrightarrow} \overset{\text{def}}{=} \overset{R}{\Longrightarrow} \cap \{(C[A^{(n)}(t_1, \ldots, t_n)], C[t]) \mid C \in \mathcal{C}(N \cup \Sigma), A^{(n)} \in N_n, t_1, \ldots, t_n \in T(\Sigma)\}$$

$$\overset{\text{OI}}{\Longrightarrow} \overset{\text{def}}{=} \overset{R}{\Longrightarrow} \cap \{(C[A^{(n)}(t_1, \ldots, t_n), t_{n+1}, t_{n+m-1}], C[t, t_{n+1}, \ldots, t_{n+m-1}])$$
$$\mid m \geq 1, C \in \mathcal{C}^m(\Sigma), A^{(n)} \in N_n, t_1, \ldots t_{n+m-1} \in T(N \cup \Sigma)\} .$$

Example 1.8 (IO vs. OI)**.** Consider the CFTG with rules

$$S \to A(B) \qquad A(y) \to f(y,y)$$
$$B \to g(B) \qquad B \to a .$$

Then OI derivations are all of form

$$S \overset{\text{OI}}{\Longrightarrow} A(B) \underset{\text{OI}}{\overset{f}{\Longrightarrow}} (B, B) \overset{\text{OI}}{\Longrightarrow}^{n+m} f(g^m(a), g^n(a))$$

for some m, n in $\mathbb{N}$, whereas the IO derivations are all of form

$$S \overset{\text{IO}}{\Longrightarrow} A(B) \overset{\text{IO}}{\Longrightarrow}^{n} A(g^n(a)) \overset{\text{IO}}{\Longrightarrow} f(g^n(a), g^n(a)) \ .$$

The two modes of derivation give rise to two tree languages $L_{\text{OI}}(\mathcal{G})$ and $L_{\text{IO}}(\mathcal{G})$, both obviously included in $L(\mathcal{G})$.

Theorem 1.9 (Fischer, 1968)**.** *For any CFTG $\mathcal{G}$, $L_{\text{IO}}(\mathcal{G}) \subseteq L_{\text{OI}}(\mathcal{G}) = L(\mathcal{G})$.*

As seen with Example 1.8, the case $L_{\text{IO}}(\mathcal{G}) \subsetneq L_{\text{OI}}(\mathcal{G})$ can occur. Theorem 1.9 shows that can assume OI derivations whenever it suits us; for instance, a basic observation is that OI derivations on different subtrees are independent:

Lemma 1.10. *Let $\mathcal{G} = \langle N, \Sigma, S, R\rangle$. If $t_1, \dots, t_n$ are trees in $T(N \cup \Sigma)$, C is a context in $\mathcal{C}^n(\Sigma)$, and $t = C[t_1, \dots, t_n] \overset{R}{\Longrightarrow}^m t'$ for some m, then there exist $m_1, \dots, m_n$ in $\mathbb{N}$ and $t'_1, \dots, t'_n$ in $T(N \cup \Sigma)$ s.t. $t_i \overset{R}{\Longrightarrow}^{m_i} t'_i$, $m = m_1 + \cdots + m_n$, and $t' = C[t'_1, \dots, t'_n]$.*

Proof. Let us proceed by induction on m. For the base case, the lemma holds immediately for $m = 0$ by choosing $m_i = 0$ and $t'_i = t_i$ for each $1 \leq i \leq n$. For the induction step, consider a derivation $t = C[t_1, \dots, t_n] \overset{R}{\Longrightarrow}^m t' \overset{R}{\Longrightarrow} t''$. By induction hypothesis, we find $m_1, \dots, m_n$ and $t'_1, \dots, t'_n$ with $t_i \overset{R}{\Longrightarrow}^{m_i} t'_i$, $m = \sum_{i=1}^n m_i$, and $t' = C[t'_1, \dots, t'_n] \overset{R}{\Longrightarrow} t''$. Since $C \in \mathcal{C}^n(\Sigma)$ is a linear term devoid of nonterminal symbols, the latter derivation step stems from a rewrite occurring in some t'_i subtree. Thus $t_i \overset{R}{\Longrightarrow}^{m_i+1} t''_i$ for some t''_i s.t. $t'' = C[t'_1, \dots, t''_i, \dots, t'_n]$. □

In contrast with Theorem 1.9, if we consider the *classes* of tree languages that can be described by CFTGs using IO and OI derivations, we obtain incomparable classes (Fischer, 1968).

1.1.3 TAGs as Context-Free Tree Grammars

Tree adjoining grammars can be seen as a special case of **context-free tree grammars** with a few restrictions on the form of its rewrite rules. This is a folklore result, which was stated (at least) by Mönnich (1997), Fujiyoshi and Kasai (2000), and Kepser and Rogers (2011), and which is made even more obvious with the "rewriting"-flavoured definition we gave for TAGs.

Translation from TAGs to CFTGs Given a TAG $\mathcal{G} = \langle N, \Sigma, T_\alpha, T_\beta, S\rangle$, we construct a CFTG $\mathcal{G}' = \langle N', \Sigma', S{\downarrow}, R \cup R'\rangle$ with

$$\begin{aligned}
N' &\overset{\text{def}}{=} N{\downarrow} \cup \{\bar{A}^{(1)} \mid A \in N\} \\
\Sigma' &\overset{\text{def}}{=} \Sigma_0 \cup \{\varepsilon^{(0)}\} \cup N_{>0} \\
R &\overset{\text{def}}{=} \{A{\downarrow} \to \tau(\alpha) \mid \alpha \in T_\alpha \wedge \mathrm{rl}(\alpha) = A\} \\
&\quad \cup \{\bar{A}^{(1)}(y) \to \tau(\beta)[\bar{A}^{(1)}(y)] \mid \beta[A_\star] \in T_\beta\} \\
&\quad \cup \{\bar{A}^{(1)}(y) \to \tau(\beta)[y] \mid \beta[A^{\text{na}}_\star] \in T_\beta\} \\
R' &\overset{\text{def}}{=} \{\bar{A}^{(1)}(y) \to y \mid \bar{A}^{(1)} \in \bar{N}\}
\end{aligned}$$

where $\tau : T(\Delta \cup \{\Box\}) \to T(\Delta' \cup \{\Box\})$ for $\Delta \stackrel{\text{def}}{=} N{\downarrow} \cup N^{\text{na}}_{>0} \cup \Sigma'$ and $\Delta' \stackrel{\text{def}}{=} N' \cup \Sigma'$ is a tree homomorphism generated by

$$\tau(A^{(m)}(x_1, \ldots, x_m)) \stackrel{\text{def}}{=} \bar{A}^{(1)}(A^{(m)}(x_1, \ldots, x_m))$$
$$\tau(A^{\text{na}(m)}) \stackrel{\text{def}}{=} A^{(m)}(x_1, \ldots, x_m)$$

and the identity for the other cases (i.e. for symbols in $N{\downarrow} \cup \Sigma_0 \cup \{\varepsilon, \Box\}$).

Example 1.11. Consider again the TAG of Figure 1.5 for the copy language: we obtain $\mathcal{G}' = \langle N', \Sigma', S{\downarrow}, R \cup R' \rangle$ with $N' = \{S{\downarrow}, \bar{S}\}$, $\Sigma' = \{S, a, b, \varepsilon\}$, and rules

$$\begin{aligned} R = \{ & S{\downarrow} \to \bar{S}(S(\varepsilon)), && \text{(corresponding to } \alpha_\varepsilon\text{)} \\ & \bar{S}(y) \to S(a, \bar{S}(S(y, a))), && \text{(corresponding to } \beta_a\text{)} \\ & \bar{S}(y) \to S(b, \bar{S}(S(y, b)))\} && \text{(corresponding to } \beta_b\text{)} \\ R' = \{ & \bar{S}(y) \to y\} \ . \end{aligned}$$

Proposition 1.12. $L_T(\mathcal{G}) = L(\mathcal{G}')$.

Proof of $L_T(\mathcal{G}) \subseteq L(\mathcal{G}')$. We first prove by induction on the length of derivations:

Claim 1.12.1. For all trees t in $T(\Delta)$, $t \stackrel{R_\mathcal{G}}{\Longrightarrow}{}^\star\ t'$ implies t' is in $T(\Delta)$ and $\tau(t) \stackrel{R}{\Longrightarrow}{}^\star\ \tau(t')$.

Proof of Claim 1.12.1. That $T(\Delta)$ is closed under $R_\mathcal{G}$ is immediate. For the second part of the claim, we only need to consider the case of a single derivation step:

For a substitution $C[A{\downarrow}] \stackrel{R_\mathcal{G}}{\Longrightarrow} C[\alpha]$ occurs iff α is in T_α with $\text{rl}(\alpha) = A$, which implies $\tau(C[A{\downarrow}]) = \tau(C)[\tau(A{\downarrow})] = \tau(C)[A{\downarrow}] \stackrel{R}{\Longrightarrow} \tau(C)[\tau(\alpha)] = \tau(C[\alpha])$.

For an adjunction $C[A^{(m)}(t_1, \ldots, t_m)] \stackrel{R_\mathcal{G}}{\Longrightarrow} C[\beta[A^{(m)}(t_1, \ldots, t_m)]]$ occurs iff $\beta[A_\star]$ is in T_β, implying

$$\begin{aligned} \tau(C[A^{(m)}(t_1, \ldots, t_m)]) &= \tau(C)[\bar{A}^{(1)}(A^{(m)}(\tau(t_1), \ldots, \tau(t_m)))] \\ &\stackrel{R}{\Longrightarrow} \tau(C)[\tau(\beta)[\bar{A}^{(1)}(A^{(m)}(\tau(t_1), \ldots, \tau(t_m)))]] \\ &= \tau(C[\beta[A^{(m)}(t_1, \ldots, t_m)]]) \ . \end{aligned}$$

The case of a tree $\beta[A^{\text{na}}_\star]$ is similar. [1.12.1]

Claim 1.12.2. If t is a tree in $T(N^{\text{na}} \cup \Sigma')$, then there exists a derivation $\tau(t) \stackrel{R'}{\Longrightarrow}{}^\star\ h(t)$ in $\mathcal{G}'$.

Proof of Claim 1.12.2. We proceed by induction on t:

For a tree rooted by $A^{(m)}$:

$$\begin{aligned} \tau(A^{(m)}(t_1, \ldots, t_m)) &= \bar{A}^{(1)}(A^{(m)}(\tau(t_1), \ldots, \tau(t_m))) \\ &\stackrel{R'}{\Longrightarrow} A^{(m)}(\tau(t_1), \ldots, \tau(t_m)) \\ &\stackrel{R'}{\Longrightarrow}{}^\star\ A^{(m)}(h(t_1), \ldots, h(t_m)) && \text{(by ind. hyp.)} \\ &= h(A^{(m)}(t_1, \ldots, t_m)) \ . \end{aligned}$$

For a tree rooted by $A^{\mathrm{na}(m)}$:

$$\begin{aligned}\tau(A^{\mathrm{na}(m)}(t_1,\ldots,t_m)) &= A^{(m)}(\tau(t_1),\ldots,\tau(t_m))\\ &\overset{R'}{\Longrightarrow}{}^{\star}\ A^{(m)}(h(t_1),\ldots,h(t_m)) && \text{(by ind. hyp.)}\\ &= h(A^{\mathrm{na}(m)}(t_1,\ldots,t_m))\ .\end{aligned}$$

The case of a tree rooted by a in $\Sigma\cup\{\varepsilon\}$ is trivial. [1.12.2]

For the main proof: Let t be a tree in $L_T(\mathcal{G})$; there exist t' in $T(N^{\mathrm{na}}\cup\Sigma')$ and α in T_α with $\mathrm{rl}(\alpha)=S$ s.t. $\alpha \overset{R_{\mathcal{G}}}{\Longrightarrow}{}^{\star}\ t'$ and $t=h(t')$. Then $S{\downarrow}\overset{R}{\Rightarrow}\tau(\alpha)\overset{R}{\Rightarrow}{}^{\star}\ \tau(t')$ according to Claim 1.12.1, and then $\tau(t')\overset{R'}{\Longrightarrow}{}^{\star}\ t$ removes all its nonterminals according to Claim 1.12.2. □

Proof of $L(\mathcal{G}')\subseteq L_T(\mathcal{G})$. We proceed similarly for the converse proof. We first need to restrict ourselves to *well-formed* trees (and contexts): we define the set $L\subseteq T(\Delta'\cup\{\Box\})$ as the language of all trees and contexts where every node labeled $\bar{A}^{(1)}$ in $\bar{N}$ has $A^{(m)}$ in N as the label of its daughter—L is defined formally in the proof of the following claim:

Claim 1.12.3. The homomorphism τ is a bijection from $T(\Delta\cup\{\Box\})$ to L.

Proof of Claim 1.12.3. It should be clear that τ is injective and has a range included in L. We can define τ^{-1} as a deterministic top-down tree transduction from $T(\Delta'\cup\{\Box\})$ into $T(\Delta\cup\{\Box\})$ with L for domain, thus proving surjectivity: Let $\mathcal{T}=\langle\{q\}\cup\{q_A\mid A\in N\},\Delta'\cup\{\Box\},\Delta\cup\{\Box\},\rho,\{q\}\rangle$ with rules

$$\begin{aligned}\rho=\ &\{q(A^{(1)}(x))\to q_A(x)\mid \bar{A}^{(1)}\in\bar{N}\}\\ &\cup\{q_A(A^{(m)}(x_1,\ldots,x_m))\to A^{(m)}(q(x_1),\ldots,q(x_m))\mid A^{(m)}\in N\}\\ &\cup\{q(A^{(m)}(x_1,\ldots,x_m))\to A^{\mathrm{na}(m)}(q(x_1),\ldots,q(x_m))\mid A^{(m)}\in N\}\\ &\cup\{q(a^{(m)}(x_1,\ldots,x_m)\to a^{(m)}(q(x_1),\ldots,q(x_m))\mid a^{(m)}\in N{\downarrow}\cup\Sigma\cup\{\varepsilon^{(0)},\Box^{(0)}\}\}\ .\end{aligned}$$

We see immediately that $[\![\mathcal{T}]\!](t)=\tau^{-1}(t)$ for all t in L. [1.12.3]

Thanks to Claim 1.12.3, we can use τ^{-1} in our proofs. We obtain claims mirroring Claim 1.12.1 and Claim 1.12.2 using the same types of arguments:

Claim 1.12.4. For all trees t in L, $t\overset{R}{\Rightarrow}{}^{\star}\ t'$ implies t' in L and $\tau^{-1}(t)\overset{R_{\mathcal{G}}}{\Longrightarrow}{}^{\star}\ \tau^{-1}(t')$.

Claim 1.12.5. If t is a tree in $L\cap T(\bar{N}\cup\Sigma')$, t' a tree in $T(\Sigma)$, and $t\overset{R'}{\Longrightarrow}{}^{\star}\ t'$, then $h(\tau^{-1}(t'))=\tau^{-1}(t)$.

For the main proof, consider a derivation $S{\downarrow}\overset{R}{\Rightarrow}{}^{\star}\ t$ with $t\in T(\Sigma')$ of $\mathcal{G}$. We can reorder this derivation so that $S{\downarrow}\overset{R}{\Rightarrow}\tau(\alpha)\overset{R}{\Rightarrow}{}^{\star}\ \tau(t')\overset{R'}{\Longrightarrow}{}^{\star}\ t$ for some α in T_α with $\mathrm{rl}(\alpha)=S$ and t' in $L\cap T(\bar{N}\cup\Sigma')$ (i.e. t' does not contain any symbol from $N{\downarrow}$). By Claim 1.12.4, $\alpha\overset{R_{\mathcal{G}}}{\Longrightarrow}{}^{\star}\ t'$ and by Claim 1.12.5 $h(t')=\tau^{-1}(t)$. Since t belongs to $T(\Sigma')$, $\tau^{-1}(t)=t$, which shows that t belongs to $L_T(\mathcal{G})$. □

From CFTGs to TAGs The converse direction is more involved, because TAGs as usually defined have *locality* restrictions (in a sense comparable to that of CFGs generating only *local* tree languages) caused by their label-based selection mechanisms for the substitution and adjunction rules. This prompted the definition of **non-strict** definitions for TAGs, where root and foot labels of auxiliary trees do not have to match, where tree selection for substitution and adjunction is made through *selection lists* attached to each substitution node or adjunction site, and where elementary trees can be reduced to a leaf or a foot node (which does not make much sense for strict TAGs due to the selection mechanism); see Kepser and Rogers (2011).

Putting these considerations aside, the essential fact to remember is that TAGs are "almost" equivalent to **linear**, **monadic** CFTGs as far as tree languages are concerned, and *exactly* for string languages: a CFTG is called

- **linear** if, for every rule $A^{(n)}(y_1, \ldots, y_n) \to e$ in R, the right-hand side e is linear,
- **monadic** if the maximal rank of a non-terminal is 1.

Exercise 1.4 (Non-Strict TAGs)**.** Definition 1.1 is a **strict** definition of TAGs. **(∗∗∗)**

1. Read the definition of non-strict TAGs given by Kepser and Rogers (2011). Show that strict and non-strict TAGs derive the same string languages.
2. Give a non-strict TAG for the regular tree language
$$S((A(a, \Box))^* \cdot b, (A(\Box, a))^* \cdot b) \ . \tag{1.2}$$
3. Can you give a strict TAG for it? There are more trivial tree languages lying beyond the reach of strict TAGs: prove that the two following finite languages are not TAG tree languages:
$$\{A(a), B(a)\} \tag{1.3}$$
$$\{a\} \tag{1.4}$$
Note that allowing distinct foot and root labels in auxiliary trees is useless for these examples.

1.2 Well-Nested MCSLs

The class of **well-nested MCSLs** is at the junction of different extensions of context-free languages that still lie below full context-sensitive ones Figure 1.1. This provides characterizations both in terms of

- **well-nested multiple context-free grammars** (or equivalently well-nested linear context-free rewrite systems) (Kanazawa, 2009), and in terms of
- **linear macro grammars** (Seki and Kato, 2008), a subclass of the macro grammars of Fischer (1968), also characterized via linear context-free tree grammars (Rounds, 1970) or linear macro tree transducers (Engelfriet and Vogler, 1985).

We concentrate on this second view.

1.2.1 Linear CFTGs

As already seen with tree adjoining grammars, the case of **linear** CFTGs is of particular interest. Intuitively, the relevance of linearity for linguistic modeling is that *arguments* in a subcategorization frame have a linear behaviour: they should appear exactly the stated number of times (by contrast, *modifiers* can be added freely).

See Kepser and Mönnich (2006).

Linear CFTGs enjoy a number of properties. For instance, unlike the general case, for linear CFTGs the distinction between IO and OI derivations is irrelevant:

Proposition 1.13. *Let $\mathcal{G} = \langle N, \Sigma, S, R\rangle$ be a linear CFTG. Then $L_{\mathrm{IO}}(\mathcal{G}) = L_{\mathrm{OI}}(\mathcal{G})$.*

Proof. Consider a derivation $S \overset{R}{\Longrightarrow}{}^{\star} t$ in a linear CFTG. Thanks to Theorem 1.9, we can assume this derivation to be OI. Let us pick the last non-IO step within this OI derivation:

$$\begin{aligned} S &\overset{\mathrm{OI}}{\Longrightarrow}{}^{\star} C[A^{(n)}(e_1, \ldots, e_n)] \\ &\overset{r_A}{\Longrightarrow} C[e_A\{y_1 \leftarrow e_1, \ldots, y_n \leftarrow e_n\}] \\ &\overset{\mathrm{IO}}{\Longrightarrow}{}^{\star} t \end{aligned}$$

using some rule $r_A : A^{(n)}(y_1, \ldots, y_n) \to e_A$, where an e_i contains a nonterminal. By Lemma 1.10, we can "pull" all the independent rewrites occurring after this $\overset{r_A}{\Longrightarrow}$ so that they occur before the $\overset{r_A}{\Longrightarrow}$ rewrite, so that the next rewrite occurs within the context C. Since everything after this $\overset{r_A}{\Longrightarrow}$ is IO, this rewrite has to involve an innermost nonterminal, thus a nonterminal that was not introduced in e_A, but one that already appeared in some e_i: in the context C:

$$\begin{aligned} &e_A\{y_1 \leftarrow e_1, \ldots, y_i \leftarrow C'[B^{(m)}(e'_1, \ldots, e'_m)], \ldots, y_n \leftarrow e_n\} \\ &\overset{r_B}{\Longrightarrow} e_A\{y_1 \leftarrow e_1, \ldots, y_i \leftarrow C'[e_B\{x_1 \leftarrow e'_1, \ldots, x_m \leftarrow e'_m\}], \ldots, y_n \leftarrow e_n\} \end{aligned}$$

which is possible *thanks to linearity*: in general, there is no way to force the various copies of e_i to use the same rewrite for $B^{(m)}$. Now this sequence is easily swapped: in the context C:

$$\begin{aligned} &A^{(n)}(e_1, \ldots, C'[B^{(m)}(e'_1, \ldots, e'_m)], \ldots, e_n) \\ &\overset{r_B}{\Longrightarrow} A^{(n)}(e_1, \ldots, C'[e_B\{x_1 \leftarrow e'_1, \ldots, x_m \leftarrow e'_m\}], \ldots, e_n) \\ &\overset{r_A}{\Longrightarrow} e_A\{y_1 \leftarrow e_1, \ldots, y_i \leftarrow C'[e_B\{x_1 \leftarrow e'_1, \ldots, x_m \leftarrow e'_m\}], \ldots, y_n \leftarrow e_n\}\ . \end{aligned}$$

Repeating this operation for every nonterminal that occurred in the e_i's yields a derivation of the same length for $S \overset{R}{\Longrightarrow}{}^{\star} t$ with a shorter OI prefix and a longer IO suffix. Repeating the argument at this level yields a full IO derivation. □

Proposition 1.13 allows to apply several results pertaining to IO derivations to linear CFTGs. A simple one is an alternative semantics for IO derivations in a CFTG $\mathcal{G} = \langle N, \Sigma, S, R\rangle$: the semantics of a nonterminal $A^{(n)}$ can be recast as a subset of the relation $[\![A^{(n)}]\!] \subseteq (T(\Sigma))^{n+1}$:

$$[\![A^{(n)}]\!](t_1, \ldots, t_n) \overset{\text{def}}{=} \bigcup_{(A^{(n)}(y_1,\ldots,y_n)\to e)\in R} [\![e]\!](t_1, \ldots, t_n)$$

where $[\![e]\!] \subseteq (T(\Sigma))^{n+1}$ is defined inductively for all subterms e in rule right-hand sides—with n variables in the corresponding *full* term—by

$$[\![a^{(m)}(e_1,\ldots,e_m)]\!](t_1,\ldots,t_n) \stackrel{\text{def}}{=} \{a^{(m)}(t'_1,\ldots,t'_m) \mid \forall 1 \leq i \leq m.t'_i \in [\![e_i]\!](t_1,\ldots,t_n)\}$$

$$[\![B^{(m)}(e_1,\ldots,e_m)]\!](t_1,\ldots,t_n) \stackrel{\text{def}}{=} \{[\![B^{(m)}]\!](t'_1,\ldots,t'_m) \mid \forall 1 \leq i \leq m.t'_i \in [\![e_i]\!](t_1,\ldots,t_n)\}$$

$$[\![y_i]\!](t_1,\ldots,t_n) \stackrel{\text{def}}{=} \{t_i\} .$$

The consequence of this definition is

$$L_{\text{IO}}(\mathcal{G}) = [\![S^{(0)}]\!] .$$

This semantics will be easier to employ in the following proofs concerned with IO derivations (and thus applicable to linear CFTGs).

This section relies heavily on Maneth et al. *(2007).*

Parsing as Intersection Let us look into more algorithmic issues and consider the parsing problem for linear CFTGs. In order to apply the parsing as intersection paradigm, we need two main ingredients: the first is emptiness testing (Proposition 1.14), the second is closure under intersection with regular sets (Proposition 1.15). We actually prove these results for IO derivations in CFTGs rather than for linear CFTGs solely.

Proposition 1.14 (Emptiness)**.** *Given a CFTG $\mathcal{G}$, one can decide whether $L_{\text{IO}}(\mathcal{G}) = \emptyset$ in $O(|\mathcal{G}|)$.*

Proof sketch. Given $\mathcal{G} = \langle N, \Sigma, S, R\rangle$, we construct a context-free grammar $\mathcal{G}' = \langle N', \emptyset, P, S\rangle$ s.t. $L_{\text{IO}}(\mathcal{G}) = \emptyset$ iff $L(\mathcal{G}') = \emptyset$ and $|\mathcal{G}'| = O(|\mathcal{G}|)$. Since emptiness of CFGs can be tested in linear time, this will yield the result. We define for this

$$N' \stackrel{\text{def}}{=} N \cup \bigcup_{A^{(m)}(y_1,\ldots,y_m)\to e \in R} \text{Sub}(e) ,$$

i.e. we consider both nonterminals and positions inside rule right hand sides as nonterminals of $\mathcal{G}'$, and

$$\begin{aligned} P' \stackrel{\text{def}}{=} & \{A \to e \mid A^{(m)}(y_1,\ldots,y_m) \to e \in R\} && \text{(rules)} \\ & \cup \{a^{(m)}(e_1,\ldots,e_m) \to e_1 \cdots e_m \mid a \in \Sigma \cup \mathcal{Y}\} && (\Sigma\text{- or } \mathcal{Y}\text{-labeled positions}) \\ & \cup \{A^{(m)}(e_1,\ldots,e_m) \to A e_1 \cdots e_m\} . && (N\text{-labeled positions}) \end{aligned}$$

We note N-labeled positions with arity information and nonterminal symbols without in order to be able to distinguish them. Note that terminal- or variable-labeled positions with arity 0 give rise to empty rules, whereas for nonterminal-labeled positions of arity 0 we obtain unit rules.

The constructed grammar is clearly of linear size; we leave the fixpoint induction proof of $X \stackrel{\mathcal{G}'}{\Longrightarrow}{}^{\star} \varepsilon$ iff $[\![X]\!] \neq \emptyset$ to the reader. □

Proposition 1.15 (Closure under Intersection with Regular Tree Languages)**.** *Let $\mathcal{G}$ be a (linear) CFTG with maximal nonterminal rank M and maximal number of nonterminals in a right-hand side D, and $\mathcal{A}$ a DTA with $|Q|$ states. Then we can construct a (linear) CFTG $\mathcal{G}'$ with $L_{\text{IO}}(\mathcal{G}') = L_{\text{IO}}(\mathcal{G}) \cap L$ and $|\mathcal{G}'| = O(|\mathcal{G}| \cdot |Q|^{M+D+1})$.*

Proof. Let $\mathcal{G} = \langle N, \Sigma, S, R\rangle$ and $\mathcal{A} = \langle Q, \Sigma, \delta, F\rangle$. We define $\mathcal{G}' = \langle N', \Sigma, S', R'\rangle$ where

$$N' \stackrel{\text{def}}{=} \{S'\} \cup \bigcup_{m \leq M} N_m \times Q^{m+1},$$

i.e. we add a new axiom and otherwise consider tuples of form $\langle A^{(m)}, q_0, q_1, \ldots, q_m\rangle$ as nonterminals of rank m,

$$\begin{aligned} R' \stackrel{\text{def}}{=} &\{S' \to \langle S, q_f\rangle \mid q_f \in F\} \\ &\cup \{\langle A, q_0, \ldots, q_m\rangle^{(m)}(y_1, \ldots, y_m) \to e' \\ &\qquad \mid A^{(m)}(y_1, \ldots, y_m) \to e \in R \wedge e' \in \theta_{q_0 q_1 \cdots q_m}(e)\}, \end{aligned}$$

where each $\theta_{q_0 q_1 \cdots q_m}$ is a nondeterministic translation of right-hand sides, under the understanding that variable y_i should hold a tree recognized by state q_i and the root should be recognized by q_0:

$$\theta_{q_0 q_1 \cdots q_m}(a^{(m)}(e_1, \ldots, e_m)) \stackrel{\text{def}}{=} \{a^{(m)}(e'_1, \ldots, e'_m) \mid \exists (q_0, a, q'_1, \ldots, q'_m) \in \delta, \forall 1 \leq i \leq m, e'_i \in \theta_{q'_i q_1 \cdots q_m}(e_i)\}$$

$$\theta_{q_0 q_1 \cdots q_m}(B^{(m)}(e_1, \ldots, e_m)) \stackrel{\text{def}}{=} \{\langle B, q_0, q'_1, \ldots, q'_m\rangle(e'_1, \ldots, e'_m) \mid \forall 1 \leq i \leq m, q'_i \in Q \wedge e'_i \in \theta_{q'_i q_1 \cdots q_m}(e_i)\}$$

$$\theta_{q_0 q_1 \cdots q_m}(y_i) \stackrel{\text{def}}{=} \{y_i\} .$$

The intuition behind this definition is that $\mathcal{G}'$ guesses that the trees passed as y_i parameters will be recognized by state q_i of $\mathcal{A}$, leading to a tree generated by $A^{(m)}$ and recognized by q_0. A computationally expensive point is the translation of nonterminals in the right-hand side, where we actually guess an assignment of states for its parameters.

We can already check that $\mathcal{G}'$ is constructed through at most $|R| \cdot |Q|^{M+1}$ calls to θ translations, each allowing at most $|Q|^D$ choices for the nonterminals in the argument right-hand side. In fine, each rule of $\mathcal{G}$ is duplicated at most $|Q|^{M+D+1}$ times.

For a tuple of states $q_1, \ldots, q_m$ in Q^m, let us define the relation $[\![q_1 \cdots q_m]\!] \subseteq (T(\Sigma))^m$ as the cartesian product of the sets $[\![q_i]\!] \stackrel{\text{def}}{=} \{t \in T(\Sigma) \mid q_i \stackrel{R_T}{\Longrightarrow}{}^{\star}\ t\}$. We can check that, for all $m \leq M$, all states $q_0, q_1, \ldots, q_m$ of Q, and all nonterminals $A^{(m)}$ of N,

$$[\![\langle A, q_0, q_1, \ldots, q_m\rangle]\!]([\![q_1 \cdots q_m]\!]) = [\![A^{(m)}]\!] \cap [\![q_0]\!] .$$

This last equality proves the correctness of the construction. □

Note to self: D can be made equal to $\max(M, K)$ where K is the maximal terminal rank for any IO grammar; could this be improved thanks to linearity—ideally to $D = 1$? No, after a bit of thought you can't. Gómez-Rodríguez, Kuhlmann and Satta propose an $O(|w|^{2(M+2)})$ upper bound for wnMCSLs in their ACL 2010 paper.

In order to use these results for *string* parsing, we merely need to construct, given a string w and a ranked alphabet Σ, the universal DTA with w as yield—it has $O(|w|^2)$ states, thus we can obtain an $O(|\mathcal{G}| \cdot |w|^{2(M+D+1)})$ upper bound for IO parsing with CFTGs *even in the non linear case*.

1.2.2 Two-Level Syntax

One of the original propositions of Chomsky's *Syntactic Structures* is a distinction between "deep" and "surface" syntactic structures, which could be related by transformations. In the light of the derivation vs. derived tree distinction with TAGs, it would be revealing to try to apply this dichotomy more widely. This paradigm is sometimes called **two-level syntax**, relating two (or more!) syntactic levels by tree transformations, for instance tree transductions (Shieber, 2006) or typed morphisms on λ-terms (de Groote, 2001). We consider here one such class of tree transformations that generates exactly the context-free tree languages.

Background: Macro Tree Transducers

Definition 1.16 (Macro Tree Transducers)**.** A **macro tree transducer** (MTT) is a tuple $\mathcal{T} = \langle Q, \Sigma, \Delta, I, R\rangle$ consisting of three finite ranked alphabets Q, Σ, and Δ of *states*, *input*, and *output* symbols, a set $I \subseteq Q_1$ of initial states, all with arity 1, and a set R of rewrite rules over $T(Q \cup \Sigma \cup \Delta, \mathcal{X} \cup \mathcal{Y})$ for $\mathcal{X}, \mathcal{Y}$ two infinite countable sets of input variables and parameters, each rule being of form

$$q^{(n+1)}(a^{(m)}(x_1, \ldots, x_m), y_1, \ldots, y_n) \to t$$

where $q^{(n+1)}$ is in Q_{n+1}, $a^{(m)}$ in Σ_m, the input variables x_i in $\mathcal{X}_m$, and the parameters y_j in $\mathcal{Y}_n$, and t is a tree in $\mathrm{RHS}(Q, \Delta, m, n)$ defined by the abstract syntax

$$t ::= y_j \mid a^{(r)}(t, \ldots, t) \mid q^{(p+1)}(x_i, t, \ldots, t)$$

where y_j is in $\mathcal{Y}_n$, $a^{(r)}$ in Δ_r, $q^{(p+1)}$ in Q_{p+1}, and x_i in $\mathcal{X}_m$.

The semantics $[\![\mathcal{T}]\!]$ of a MTT is a relation in $T(\Sigma) \times T(\Delta)$ defined by

$$[\![\mathcal{T}]\!] \stackrel{\text{def}}{=} \{(t, t') \in T(\Sigma) \times T(\Delta) \mid \exists q_i^{(1)} \in I, q_i^{(1)}(t) \stackrel{R}{\Rightarrow}{}^{\star}\, t'\} \,.$$

Chapter 2

Model-Theoretic Syntax

In contrast with the generative approaches of the previous chapters, we take here a different stance on how to formalize 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 *models* of logical formulæ.

2.0.1 Model-Theoretic vs. Generative

Most of this discussion is inspired by Pullum and Scholz (2001).

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 but rather 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 was used in **??** to lexicalize PCFGs. 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 (2.19) and (2.20)).

Practical aspects of the notion of grammaticality gradience have been investigated in the context of ***property grammars****, see e.g. Duchier* et al. *(2009).*

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 in the ground there lived a hobbit.
*In a hole in 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 the probabilistic methods of **??**, because estimated probabilities are not grammaticality judgments per se, but merely occurrence-based. In particular, even with smoothing techniques, missing events often receive essentially the same probability.

A natural way to obtain a gradience of grammaticality using model theoretic methods is to structure formulæ as large conjunctions $\bigwedge_i \varphi_i$, where each conjunct φ_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 Σ. 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 infinite models or finite ones. Of course, for practical applications one might want to restrict oneself to finite models.

2.0.2 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: these will be **labeled ordered trees**. Given a set A of labels, a **tree structure** is a tuple $\mathfrak{M} = \langle W, \downarrow, \rightarrow, (P_a)_{a \in A} \rangle$ 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 labeling 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

$$\downarrow \stackrel{\text{def}}{=} \{(w, wi) \mid i \in \mathbb{N} \wedge wi \in W\} \tag{2.1}$$

$$\rightarrow \stackrel{\text{def}}{=} \{(wi, w(i+1)) \mid i \in \mathbb{N} \wedge w(i+1) \in W\} \,. \tag{2.2}$$

Note that (a) we do not limit ourselves to a single label per node, i.e. we actually work on trees labeled by $\Sigma \stackrel{\text{def}}{=} 2^A$, (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 $\mathfrak{M} = \langle W, \downarrow, \rightarrow, (P_a)_{a\in A}\rangle$, we construct a **labeled binary tree** t, which is a partial function $\{0,1\}^* \to \Sigma$ with a prefix-closed domain. We define for this $\mathrm{dom}(t) = \mathrm{fcns}(W)$ and $t(w) = \{a \in A \mid P_a(\mathrm{fcns}^{-1}(w))\}$ for all $w \in \mathrm{dom}(t)$, where

See Comon et al. *(2007, Section 8.3.1).*

$$\mathrm{fcns}(\varepsilon) \stackrel{\text{def}}{=} \varepsilon \qquad \mathrm{fcns}(w0) \stackrel{\text{def}}{=} \mathrm{fcns}(w)0 \qquad \mathrm{fcns}(w(i+1)) \stackrel{\text{def}}{=} \mathrm{fcns}(wi)1 \tag{2.3}$$

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

$$\mathrm{fcns}^{-1}(\varepsilon) \stackrel{\text{def}}{=} \varepsilon \quad \mathrm{fcns}^{-1}(w0) \stackrel{\text{def}}{=} \mathrm{fcns}^{-1}(w)0 \qquad \mathrm{fcns}^{-1}(w1) \stackrel{\text{def}}{=} \mathrm{fcns}^{-1}(w) + 1 \tag{2.4}$$

for all w in $\varepsilon \cup 0\{0,1\}^*$, 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 $\mathrm{dom}(t) \subseteq 0\{0,1\}^*$.

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

$$\downarrow_0 \stackrel{\text{def}}{=} \{(w, w0) \mid w0 \in \mathrm{dom}(t)\} \tag{2.5}$$

$$\downarrow_1 \stackrel{\text{def}}{=} \{(w, w1) \mid w1 \in \mathrm{dom}(t)\} \tag{2.6}$$

$$P_a \stackrel{\text{def}}{=} \{w \in \mathrm{dom}(t) \mid a \in t(w)\}\ . \tag{2.7}$$

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

$$\mathrm{dom}(\bar{t}) \stackrel{\text{def}}{=} \{0w \mid w \in \mathrm{dom}(t)\} \cup \{iwj \mid w \in \mathrm{dom}(t) \wedge i, j \in \{0,1\}\} \tag{2.8}$$

and labels

$$\bar{t}(w) \stackrel{\text{def}}{=} \begin{cases} t(w') & \text{if } w = 0w' \wedge w' \in \mathrm{dom}(t) \\ \{\#\} & \text{otherwise.} \end{cases} \tag{2.9}$$

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

2.1 Monadic Second-Order Logic

We consider the **weak monadic second-order logic** (wMSO), over tree structures $\mathfrak{M} = \langle W, \downarrow, \rightarrow, (P_a)_{a\in A}\rangle$ and two infinite countable sets of first-order variables $\mathcal{X}_1$ and second-order variables $\mathcal{X}_2$. Its syntax is defined by

See Comon et al. *(2007, Section 8.4).*

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

where x, y range over $\mathcal{X}_1$, X over $\mathcal{X}_2$, and a over A. We write $\mathrm{FV}(\psi)$ for the set of variables free in a formula ψ; 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 $\nu : \mathcal{X}_1 \to W$ and $\mu : \mathcal{X}_2 \to \mathcal{P}_f(W)$ be two corresponding assignments; then the satisfaction relation is defined by

$$\begin{aligned}
\mathfrak{M} &\models_{\nu,\mu} x = y && \text{if } \nu(x) = \nu(y)\\
\mathfrak{M} &\models_{\nu,\mu} x \in X && \text{if } \nu(x) \in \mu(X)\\
\mathfrak{M} &\models_{\nu,\mu} x \downarrow y && \text{if } \nu(x) \downarrow \nu(y)\\
\mathfrak{M} &\models_{\nu,\mu} x \to y && \text{if } \nu(x) \to \nu(y)\\
\mathfrak{M} &\models_{\nu,\mu} P_a(x) && \text{if } P_a(\nu(x))\\
\mathfrak{M} &\models_{\nu,\mu} \neg\psi && \text{if } \mathfrak{M} \not\models_{\nu,\mu} \psi\\
\mathfrak{M} &\models_{\nu,\mu} \psi \vee \psi' && \text{if } \mathfrak{M} \models_{\nu,\mu} \psi \text{ or } \mathfrak{M} \models_{\nu,\mu} \psi'\\
\mathfrak{M} &\models_{\nu,\mu} \exists x.\psi && \text{if } \exists w \in W, \mathfrak{M} \models_{\nu\{x\leftarrow w\},\mu} \psi\\
\mathfrak{M} &\models_{\nu,\mu} \exists X.\psi && \text{if } \exists U \subseteq W, U \text{ finite} \wedge \mathfrak{M} \models_{\nu,\mu\{X\leftarrow U\}} \psi\ .
\end{aligned}$$

Given a wMSO formula ψ, we are interested in two algorithmic problems: the **satisfiability** problem, which asks whether there exist $\mathfrak{M}$ and ν and μ s.t. $\mathfrak{M} \models_{\nu,\mu} \psi$, and the **model-checking** problem, which given $\mathfrak{M}$ asks whether there exist ν and μ s.t. $\mathfrak{M} \models_{\nu,\mu} \psi$. By modifying the vocabulary to have labels in $A \uplus \mathrm{FV}(\psi)$, these questions can be rephrased on a wMSO *sentence*

$$\exists \mathrm{FV}(\psi).\psi \wedge \left(\bigwedge_{x \in \mathcal{X}_1 \cap \mathrm{FV}(\psi)} P_x(x) \wedge \forall y.x \neq y \supset \neg P_x(y) \right) \wedge \left(\bigwedge_{X \in \mathcal{X}_2 \cap \mathrm{FV}(\psi)} \forall y.y \in X \equiv P_X(y) \right)\ .$$

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

Example 2.1. Here are a few useful wMSO formulæ: To allow any label in a finite set $B \subseteq A$:

$$\begin{aligned}
P_B(x) &\stackrel{\text{def}}{=} \bigvee_{a \in B} P_a(x)\\
P_B(X) &\stackrel{\text{def}}{=} \forall x.x \in X \supset P_B(x)\ .
\end{aligned}$$

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

$$\begin{aligned}
\mathrm{root}(x) &\stackrel{\text{def}}{=} \neg\exists y.y \downarrow x\\
\mathrm{leaf}(x) &\stackrel{\text{def}}{=} \neg\exists y.x \downarrow y\\
\mathrm{internal}(x) &\stackrel{\text{def}}{=} \neg\mathrm{leaf}(x)\\
\mathrm{children}(x, X) &\stackrel{\text{def}}{=} \forall y.y \in X \equiv x \downarrow y\\
x \downarrow_0 y &\stackrel{\text{def}}{=} x \downarrow y \wedge \neg\exists z.z \to y\ .
\end{aligned}$$

To use the **monadic transitive closure** of a formula $\psi(u, v)$ with $u, v \in \mathrm{FV}(\psi)$:

$$x\ [\mathrm{TC}_{u,v}\ \psi(u, v)]\ y \stackrel{\text{def}}{=} \forall X.(x \in X \wedge \forall uv.(u \in X \wedge \psi(u, v) \supset v \in X) \supset y \in X) \tag{2.10}$$

$$\begin{aligned}
x \downarrow^\star y &\stackrel{\text{def}}{=} x\ [\mathrm{TC}_{u,v}\ u \downarrow v]\ y\\
x \to^\star y &\stackrel{\text{def}}{=} x\ [\mathrm{TC}_{u,v}\ u \to v]\ y\ .
\end{aligned}$$

2.1.1 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.

See Rogers (1998) for a complete analysis using wMSO. Monadic second-order logic can also be applied to queries *in treebanks (Kepser, 2004; Maryns and Kepser, 2009).*

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

Define the formula

$$\mathrm{labels}_{N,\Theta} \stackrel{\text{def}}{=} \forall x.\mathrm{root}(x) \supset P_S(x) \tag{2.11}$$

forces the root label to be S;

$$\wedge\, \forall x.\mathrm{internal}(x) \supset \bigvee_{a \in N \uplus \Theta} P_a(x) \wedge \bigwedge_{b \in N \uplus \Theta \setminus \{a\}} \neg P_b(x) \tag{2.12}$$

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

$$\wedge\, \forall x.\mathrm{leaf}(x) \supset \neg P_{N \uplus \Theta}(x) \tag{2.13}$$

forbids grammatical labels on leaves;

$$\wedge\, \forall y.\mathrm{leaf}(y) \supset \exists x.x \downarrow y \wedge P_\Theta(x) \tag{2.14}$$

expresses that leaves should have POS-labeled parents;

$$\wedge\, \forall x.\exists y_0 y_1 y_2.x \downarrow^\star y_0 \wedge y_0 \downarrow y_1 \wedge y_1 \downarrow y_2 \wedge \mathrm{leaf}(y_2) \supset P_N(x) \tag{2.15}$$

verifies that internal nodes at distance at least two from some leaf should have labels drawn from N, and are thus not POS-labeled by (2.12), and thus cannot have a leaf as a child by (2.13);

$$\wedge\, \forall x.P_\Theta(x) \supset \neg\exists yz.y \neq z \wedge x \downarrow y \wedge x \downarrow z \tag{2.16}$$

discards trees where POS-labeled nodes have more than one child. The purpose of $\mathrm{labels}_{N,\Theta}$ 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 manner is to define a finite set $L \subseteq A$ disjoint from N and Θ, and a relation $\mathrm{pos} \subseteq L \times \Theta$. Then the formula

$$\mathrm{lexicon}_{L,\mathrm{pos}} \stackrel{\text{def}}{=} \forall x. \bigvee_{\ell \in L} \left(P_\ell(x) \supset \mathrm{leaf}(x) \wedge \bigwedge_{\ell' \in L \setminus \{\ell\}} \neg P_{\ell'}(x) \wedge \forall y.y \downarrow x \supset P_{\mathrm{pos}(\ell)}(y) \right) \tag{2.17}$$

makes sure that only leaves can be labeled 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 immediate 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 $\mathcal{G} = \langle N, \Theta, P, S\rangle$ describing the "usual" local constraints between grammatical categories and POS tags. Assume ε belongs to A; then the formula

$$\mathrm{grammar}_{\mathcal{G}} \stackrel{\text{def}}{=} \forall x.(P_\varepsilon(x) \supset \neg P_{N\uplus\Theta\uplus L}(x)) \wedge \bigvee_{B\in N} P_B(x) \supset \bigvee_{B\to\beta\in P} \exists y.x \downarrow_0 y \wedge \mathrm{rule}_\beta(y) \tag{2.18}$$

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

$$\begin{aligned}
\mathrm{rule}_{X\beta}(x) &\stackrel{\text{def}}{=} P_X(x) \wedge \exists y.x \to y \wedge \mathrm{rule}_\beta(y) && \text{(for } \beta \neq \varepsilon \text{ and } X \in N \uplus \Theta\text{)}\\
\mathrm{rule}_X(x) &\stackrel{\text{def}}{=} P_X(x) \wedge \neg\exists y.x \to y && \text{(for } X \in N \uplus \Theta\text{)}\\
\mathrm{rule}_\varepsilon(x) &\stackrel{\text{def}}{=} P_\varepsilon(x) \wedge \mathrm{leaf}(x)\ .
\end{aligned}$$

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 subcategorization frames (recall Section 1.1.1) and modifiers.

Head Percolation. The first step is to provide find which child is the **head** among its sisters; several heuristics have been developed to this end, and a simple way to describe such heuristics is to use a **head percolation** function $h : N \to \{l, r\}\times(N\uplus\Theta)^*$ that describes for a given parent label A a list of potential labels $X_1, \ldots, X_n$ in $N \uplus \Theta$ 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, and 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, Appendix A). For instance, the function

$$\begin{aligned}
h(\mathsf{S}) &= (r, \mathsf{TO\ IN\ VP\ S\ SBAR} \cdots)\\
h(\mathsf{VP}) &= (l, \mathsf{VBD\ VBN\ VBZ\ VB\ VBG\ VP} \cdots)\\
h(\mathsf{NP}) &= (r, \mathsf{NN\ NNP\ NNS\ NNPS\ JJR\ CD} \cdots)\\
h(\mathsf{PP}) &= (l, \mathsf{IN\ TO\ VBG\ VBN} \cdots)
\end{aligned}$$

would result in the correct head annotations in Figure 2.1.

Given such a head percolation function h, we can express the fact that a given node is a head:

$$\mathrm{head}(x) \stackrel{\text{def}}{=} \mathrm{leaf}(x) \vee \bigvee_{B\in N} \exists yY.y \downarrow x \wedge \mathrm{children}(y, Y) \wedge P_B(y) \wedge \mathrm{head}_{h(B)}(x, Y) \tag{2.19}$$

$$\begin{aligned}
\mathrm{head}_{d,X\beta}(x, Y) &\stackrel{\text{def}}{=} \neg\mathrm{priority}_{d,X}(x, Y) \supset (\mathrm{head}_{d,\beta}(x, Y) \wedge \neg P_X(Y))\\
\mathrm{head}_{l,\varepsilon}(x, Y) &\stackrel{\text{def}}{=} \forall y.y \in Y \supset x \to^\star y\\
\mathrm{head}_{r,\varepsilon}(x, Y) &\stackrel{\text{def}}{=} \forall y.y \in Y \supset y \to^\star x\\
\mathrm{priority}_{l,X}(x, Y) &\stackrel{\text{def}}{=} P_X(x) \wedge \forall y.y \in Y \wedge y \to^\star x \supset \neg P_X(y)\\
\mathrm{priority}_{r,X}(x, Y) &\stackrel{\text{def}}{=} P_X(x) \wedge \forall y.y \in Y \wedge x \to^\star y \supset \neg P_X(y)\ .
\end{aligned}$$

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

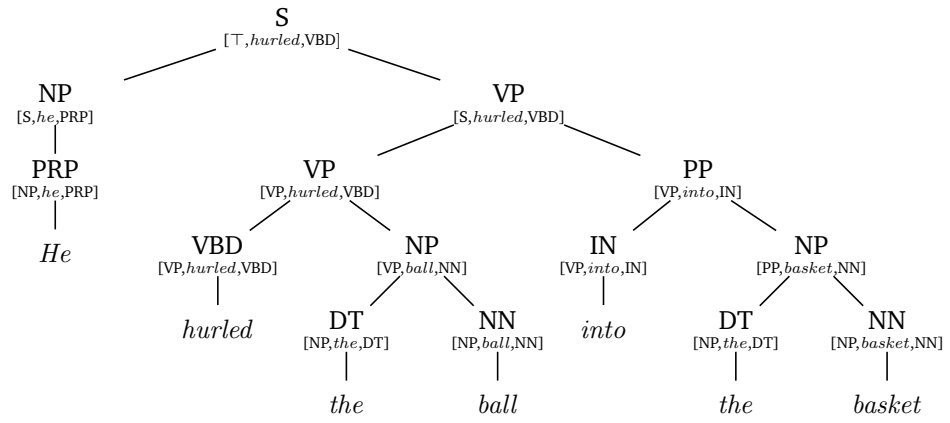

Figure 2.1: A derivation tree refined with lexical and parent information.

Lexicalization. Using head information, we can also recover lexicalization information:

$$\text{lexicalize}(x, y) \stackrel{\text{def}}{=} \text{leaf}(y) \wedge [\text{TC}_{u,v}\, u \downarrow v \wedge \text{head}(v)] \,. \tag{2.20}$$

This formula recovers the lexical information in Figure 2.1.

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

$$\begin{aligned}\text{modify}(x, y) \stackrel{\text{def}}{=} \exists x' y' z.\, & z \downarrow x \wedge P_{\text{RB}}(z) \wedge \text{lexicalize}(x', x) \wedge y' \downarrow x' \\ & \wedge \neg\text{lexicalize}(y', x) \wedge \text{lexicalize}(y', y)\end{aligned} \tag{2.21}$$

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 1.7.

Exercise 2.1. Modify (2.21) to make sure that any leaf with a parent tagged by the POS RB modifies either a verb or an adjective. **(∗)**

2.1.2 wS2S

The classical logics for trees do not use the vocabulary of tree structures $\mathfrak{M}$, 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.

See (Doner, 1970; Thatcher and Wright, 1968; Rabin, 1969; Meyer, 1975) for classical results on wS2S, and more recently (Rogers, 1996, 2003) for linguistic applications.

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 ψ one can construct a tree automaton $\mathcal{A}_\psi$—of size non-elementary in that of ψ—that recognizes all the finite models of ψ. More precisely, when working with finite binary trees and closed formulæ ψ,

$$L(\mathcal{A}_\psi) = \{\bar{t} \in T(\Sigma \uplus \{\{\#\}\}) \mid t \text{ finite} \wedge t \models \psi\} \,. \tag{2.22}$$

See Comon et al. *(2007, Section 3.3)—their construction is easily extended to handle labeled trees. Using automata over infinite trees, these can also be handled (Rabin, 1969; Weyer, 2002).*

Now, it is easy to translate any wMSO sentence ψ into a wS2S sentence ψ' s.t. $\mathfrak{M} \models \psi$ iff $\text{fcns}(\mathfrak{M}) \models \psi'$. This formula simply has to *interpret* the $\downarrow$ and $\rightarrow$

relations into their binary encodings: let

$$\psi' \stackrel{\text{def}}{=} \psi \wedge \exists x.\neg(\exists z.z \downarrow_0 x \vee z \downarrow_1 x) \wedge \neg(\exists y.x \downarrow_1 y) \tag{2.23}$$

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

$$x \downarrow y \stackrel{\text{def}}{=} \exists X.\exists x_0.x \downarrow_0 x_0 \wedge x_0 \in X \wedge y \in X \\ \wedge \forall z.(z \in X \wedge z \neq y \supset \exists z'.z' \in X \wedge z \downarrow_1 z') \tag{2.24}$$

$$x \to y \stackrel{\text{def}}{=} x \downarrow_1 y\ . \tag{2.25}$$

The conclusion of this construction is

Theorem 2.2. *Satisfiability and model-checking for wMSO are decidable.*

(∗) **Exercise 2.2** (ω Successors)**.** Show that the weak second-order logic of ω successors (**wSωS**), i.e. with $\downarrow_i \stackrel{\text{def}}{=} \{(w, wi) \mid wi \in W\}$ defined for every $i \in \mathbb{N}$, has decidable satisfiability and model-checking problems.

2.2 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 on ordered trees was first defined by Kracht (1995). The name of PDL on trees *is due to Afanasiev* et al. *(2005); this logic is also known as **Regular XPath** in the XML processing community Marx (2005). Various fragments have been considered through the years; see for instance Blackburn* et al. *(1993, 1996); Palm (1999); Marx and de Rijke (2005).*

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 $\mathfrak{M} = \langle W, \downarrow, \to, (P_a)_{a \in A} \rangle$, its terms follow the abstract syntax

$$\pi ::= \downarrow \mid \to \mid \pi^{-1} \mid \pi;\pi \mid \pi + \pi \mid \pi^* \mid \varphi? \qquad \text{(path formulæ)}$$
$$\varphi ::= a \mid \top \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\varphi \qquad \text{(node formulæ)}$$

where a ranges over A.

The **semantics** of a node formula on a tree structure $\mathfrak{M} = \langle W, \downarrow, \to, (P_a)_{a \in A} \rangle$ is a set of tree nodes $[\![\varphi]\!] = \{w \in W \mid \mathfrak{M}, w \models \varphi\}$, while the semantics of a path formula is a binary relation over W:

$$[\![a]\!] \stackrel{\text{def}}{=} \{w \in W \mid P_a(w)\} \qquad [\![\downarrow]\!] \stackrel{\text{def}}{=} \downarrow$$
$$[\![\top]\!] \stackrel{\text{def}}{=} W \qquad [\![\to]\!] \stackrel{\text{def}}{=} \to$$
$$[\![\neg\varphi]\!] \stackrel{\text{def}}{=} W \backslash [\![\varphi]\!] \qquad [\![\pi^{-1}]\!] \stackrel{\text{def}}{=} [\![\pi]\!]^{-1}$$
$$[\![\varphi_1 \vee \varphi_2]\!] \stackrel{\text{def}}{=} [\![\varphi_1]\!] \cup [\![\varphi_2]\!] \qquad [\![\pi_1;\pi_2]\!] \stackrel{\text{def}}{=} [\![\pi_1]\!] \fatsemi [\![\pi_2]\!]$$
$$[\![\langle\pi\rangle\varphi]\!] \stackrel{\text{def}}{=} [\![\pi]\!]^{-1}([\![\varphi]\!]) \qquad [\![\pi_1 + \pi_2]\!] \stackrel{\text{def}}{=} [\![\pi_1]\!] \cup [\![\pi_2]\!]$$
$$[\![\pi^*]\!] \stackrel{\text{def}}{=} [\![\pi]\!]^\star$$
$$[\![\varphi?]\!] \stackrel{\text{def}}{=} \mathrm{Id}_{[\![\varphi]\!]}\ .$$

Finally, a tree $\mathfrak{M}$ is a **model** for a PDL formula φ if its root is in $[\![\varphi]\!]$, written $\mathfrak{M}, \mathrm{root} \models \varphi$.

We define the classical dual operators

$$\bot \stackrel{\text{def}}{=} \neg\top \qquad \varphi_1 \wedge \varphi_2 \stackrel{\text{def}}{=} \neg(\neg\varphi_1 \vee \neg\varphi_2) \qquad [\pi]\varphi \stackrel{\text{def}}{=} \neg\langle\pi\rangle\neg\varphi\ . \tag{2.26}$$

We also define

$$\uparrow \stackrel{\text{def}}{=} \downarrow^{-1} \qquad \leftarrow \stackrel{\text{def}}{=} \rightarrow^{-1}$$
$$\mathrm{root} \stackrel{\text{def}}{=} [\uparrow]\bot \qquad \mathrm{leaf} \stackrel{\text{def}}{=} [\downarrow]\bot$$
$$\mathrm{first} \stackrel{\text{def}}{=} [\leftarrow]\bot \qquad \mathrm{last} \stackrel{\text{def}}{=} [\rightarrow]\bot\ .$$

Exercise 2.3 (Converses)**.** Prove the following equivalences: **(∗)**

$$(\pi_1;\pi_2)^{-1} \equiv \pi_2^{-1};\pi_1^{-1} \tag{2.27}$$
$$(\pi_1+\pi_2)^{-1} \equiv \pi_1^{-1}+\pi_2^{-1} \tag{2.28}$$
$$(\pi^*)^{-1} \equiv (\pi^{-1})^* \tag{2.29}$$
$$(\varphi?)^{-1} \equiv \varphi?\ . \tag{2.30}$$

Exercise 2.4 (Reductions)**.** Prove the following equivalences: **(∗)**

$$\langle\pi_1;\pi_2\rangle\varphi \equiv \langle\pi_1\rangle\langle\pi_2\rangle\varphi \tag{2.31}$$
$$\langle\pi_1+\pi_2\rangle\varphi \equiv (\langle\pi_1\rangle\varphi) \vee (\langle\pi_2\rangle\varphi) \tag{2.32}$$
$$\langle\pi^*\rangle\varphi \equiv \varphi \vee \langle\pi;\pi^*\rangle\varphi \tag{2.33}$$
$$\langle\varphi_1?\rangle\varphi_2 \equiv \varphi_1 \wedge \varphi_2\ . \tag{2.34}$$

2.2.1 Model-Checking

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

As with MSO, the main application of PDL on trees is to query treebanks (see e.g. Lai and Bird, 2010).

2.2.2 Satisfiability

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

See also (Blackburn et al.*, 2001, Section 6.8) for a reduction from a tiling problem and (Harel* et al.*, 2000, Chapter 8) for a reduction from alternating Turing machines.*

Theorem 2.3 (Fischer and Ladner, 1979)**.** *Satisfiability for PDL is* EXPTIME*-hard.*

As with wMSO, it is more convenient to work on binary trees $t = \langle \mathrm{dom}(t), \downarrow_0, \downarrow_1, (P_a)_{a\in A}\rangle$ that encode our tree structures. The syntax of PDL over such models simply replaces $\downarrow$ and $\rightarrow$ by $\downarrow_0$ and $\downarrow_1$; as with wMSO in Section 2.1.2 we can *interpret* these relations in PDL by

$$\downarrow \stackrel{\text{def}}{=} \downarrow_0;\downarrow_1^* \qquad\qquad \rightarrow \stackrel{\text{def}}{=} \downarrow_1 \tag{2.35}$$

and translate any PDL formula φ into a formula

$$\varphi' \stackrel{\text{def}}{=} \varphi \wedge [\uparrow^*;\mathrm{root}?;\downarrow_1]\bot \tag{2.36}$$

that checks that φ holds and verifies $\mathfrak{M}, w \models \varphi$ iff $\mathrm{fcns}(\mathfrak{M}), \mathrm{fcns}(w) \models \varphi'$. The conditions in (2.36) ensure that the tree we are considering is the image of some tree structure by fcns: we first go back to the root by the path $\uparrow^*;\mathrm{root}?$, and then verify that the root does not have a right child.

Normal Form. Let us write

$$\uparrow_0 \stackrel{\text{def}}{=} \downarrow_0^{-1} \qquad\qquad \uparrow_1 \stackrel{\text{def}}{=} \downarrow_1^{-1} ;$$

then using the equivalences of Exercise 2.3 we can reason on PDL with a restricted path syntax

$$\alpha ::= \downarrow_0 \mid \uparrow_0 \mid \downarrow_1 \mid \uparrow_1 \qquad \text{(atomic relations)}$$
$$\pi ::= \alpha \mid \pi;\pi \mid \pi + \pi \mid \pi^* \mid \varphi? \qquad \text{(path formulæ)}$$

and using the dualities of (2.26), we can restrict node formulæ to be of form

$$\varphi ::= a \mid \neg a \mid \top \mid \bot \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle\pi\rangle\varphi \mid [\pi]\varphi \; . \qquad \text{(node formulæ)}$$

Lemma 2.4. *For any PDL formula φ, we can construct an equivalent formula φ' 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 only result in doubling the size of φ due to the extra $\neg$ and $^{-1}$ at the leaves. □

Fisher-Ladner Closure

The equivalences found in Exercise 2.4 and their duals allow to simplify PDL formulæ into a reduced normal form we will see soon, 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 φ is the smallest set S of formulæ in normal form s.t.

1. $\varphi \in S$,
2. if $\varphi_1 \vee \varphi_2 \in S$ or $\varphi_1 \wedge \varphi_2 \in S$ 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 $[\pi_1 + \pi_2]\varphi' \in S$ 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 $\mathrm{FL}(\varphi)$ for the Fisher-Ladner closure of φ.

Lemma 2.5. *Let φ be a PDL formula in normal form. Its Fisher-Ladner closure is of size $|\mathrm{FL}(\varphi)| \leq |\varphi|$.*

Proof. We construct a surjection σ between positions p in the term φ and the formulæ in S:

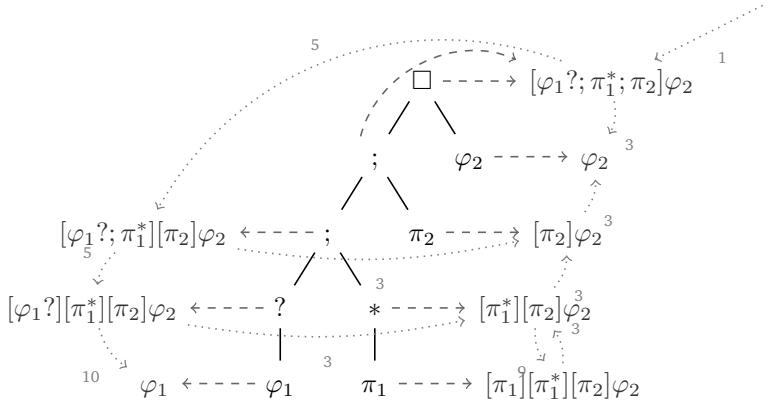

Figure 2.2: The surjection σ from positions in $\varphi \stackrel{\text{def}}{=} [\varphi_1?;\pi_1^*;\pi_2]\varphi_2$ to $\mathrm{FL}(\varphi)$ (dashed), and the rules used to construct $\mathrm{FL}(\varphi)$ (dotted).

- for positions p spanning a node subformula $\mathrm{span}(p) = \varphi_1$, we can map to φ_1 (this corresponds to cases 1—3 and 10 on subformulæ of φ');
- for positions p spanning a path subformula $\mathrm{span}(p) = \pi$, 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 $\mathrm{span}(p) = \pi_1$ and to $\langle\pi_2\rangle\varphi_2$ if $\mathrm{span}(p) = \pi_2$ (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 $\mathrm{span}(p) = \pi_1$ and to $[\pi_2]\varphi_2$ if $\mathrm{span}(p) = \pi_2$ (this matches case 5 and the further application of 3);
  - for $\sigma(p') = \langle\pi_1 + \pi_2\rangle\varphi_2$ and $\mathrm{span}(p) = \pi_i$ 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 $\mathrm{span}(p) = \pi_i$ with $i \in \{1,2\}$, we map p to $[\pi_i]\varphi_2$ (this matches case 7);
  - for $\sigma(p') = \langle\pi^*\rangle\varphi_2$, $\mathrm{span}(p) = \pi$ and we map p to $\langle\pi\rangle\langle\pi^*\rangle\varphi_2$ (this matches case 8);
  - for $\sigma(p') = [\pi^*]\varphi_2$, $\mathrm{span}(p) = \pi$ and we map p to $[\pi][\pi^*]\varphi_2$ (this matches case 9).

The function σ we just defined is indeed surjective: we have covered every formula produced by every rule. Figure 2.2 presents an example term and its mapping. □

Reduced Formulæ

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 φ in normal form, this is obtained by using the equivalences of Exercise 2.4

and their duals, and by putting the formula into disjunctive normal form, i.e.

$$\varphi \equiv \bigvee_i \bigwedge_j \chi_{i,j} \tag{2.37}$$

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

$$\chi ::= a \mid \neg a \mid \langle\alpha\rangle\varphi' \mid [\alpha]\varphi' \ . \qquad \text{(reduced formulæ)}$$

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

Lemma 2.6. *Given a PDL formula φ 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* $\mathrm{FL}(\varphi)$.

Two-Way Alternating Tree Automaton

The presentation follows mostly Calvanese et al. *(2009).*

We finally turn to the construction of a tree automaton that recognizes the models of a normal form formula φ. To simplify matters, we use a powerful model for this automaton: a **two-way alternating tree automaton** (2ATA) over finite ranked trees.

Definition 2.7. A **two-way alternating tree automaton** (2ATA) is a tuple $\mathcal{A} = \langle Q, \Sigma, q_i, F, \delta\rangle$ where Q is a finite set of states, Σ is a ranked alphabet with maximal rank k, $q_i \in Q$ is the initial state, and δ is a transition function from pairs of states and symbols (q, a) in $Q \times \Sigma$ to *positive boolean formulæ* f in $\mathcal{B}_+(\{-1, \ldots, k\} \times Q)$, defined by the abstract syntax

$$f ::= (d, q) \mid f \vee f \mid f \wedge f \mid \top \mid \bot \ ,$$

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 $\{-1, \ldots, k\} \times Q \backslash J$ 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 labeled ranked ordered tree t over Σ, a **run** of $\mathcal{A}$ is a tree ρ labeled by $\mathrm{dom}(t) \times Q$ satisfying

1. ε is in $\mathrm{dom}(\rho)$ with $\rho(\varepsilon) = (\varepsilon, q_i)$,
2. if w is in $\mathrm{dom}(\rho)$, $\rho(w) = (u, q)$ and $\delta(q, t(u)) = f$, then there exists $J \subseteq \{-1, \ldots, k\} \times Q$ of form $J = \{(d_1, q_1), \ldots, (d_n, q_n)\}$ s.t. $J \models f$ and for all $1 \leq i \leq n$ we have

$$w(i-1) \in \mathrm{dom}(\rho) \qquad u' = \begin{cases} u(d_i - 1) & \text{if } d_i > 0 \\ u & \text{if } d_i = 0 \\ u' \text{ where } u = u'j & \text{otherwise} \end{cases}$$

$u' \in \mathrm{dom}(t)$ and $\rho(wi) = (u', q_i)$.

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

Theorem 2.8 (Vardi, 1998)**.** *Given a 2ATA $\mathcal{A} = \langle Q, \Sigma, q_i, F, \delta\rangle$, deciding the emptiness of $L(\mathcal{A})$ can be done in deterministic time $|\Sigma| \cdot 2^{O(k|Q|^3)}$.*

Automaton of a Formula Let φ be a formula in normal form. We want to construct a 2ATA $\mathcal{A}_\varphi = \langle Q, \Sigma, q_i, \delta\rangle$ that recognizes exactly the closed models of φ, so that we can test the satisfiability of φ by Theorem 2.8. We assume wlog. that $A \subseteq \mathrm{Sub}(\varphi)$. We define

$$Q \stackrel{\text{def}}{=} \mathrm{FL}(\varphi) \uplus \{q_i, q_\varphi, q_0, q_1, q_\#, q_\top\}$$
$$\Sigma \stackrel{\text{def}}{=} \{\#^{(0)}, \#^{(2)}\} \cup \{a^{(2)} \mid a \subseteq A\} .$$

The transitions of $\mathcal{A}_\varphi$ are based on formula reductions. Let φ' be a formula in $\mathrm{FL}(\varphi)$ 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

$$\delta(\varphi', a) \stackrel{\text{def}}{=} \bigvee_i \bigwedge_j (0, \chi_{i,j})$$

for all such φ' and all $a \subseteq A$, thereby staying in place and checking the various $\chi_{i,j}$. For a reduced formula χ in $\mathrm{FL}(\varphi)$, we set for all $a \subseteq A$

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

$$\delta(\langle\downarrow_0\rangle\varphi', a) \stackrel{\text{def}}{=} (1, \varphi') \qquad \delta([\downarrow_0]\varphi', a) \stackrel{\text{def}}{=} (1, \varphi') \vee (1, q_\#)$$
$$\delta(\langle\downarrow_1\rangle\varphi', a) \stackrel{\text{def}}{=} (2, \varphi') \qquad \delta([\downarrow_1]\varphi', a) \stackrel{\text{def}}{=} (2, \varphi') \vee (2, q_\#)$$
$$\delta(\langle\uparrow_0\rangle\varphi', a) \stackrel{\text{def}}{=} (-1, \varphi') \wedge (-1, q_1) \qquad \delta([\uparrow_0]\varphi', a) \stackrel{\text{def}}{=} ((-1, \varphi') \wedge (-1, q_1)) \vee (-1, q_\#)$$
$$\delta(\langle\uparrow_1\rangle\varphi', a) \stackrel{\text{def}}{=} (-1, \varphi') \wedge (-1, q_0) \qquad \delta([\uparrow_1]\varphi', a) \stackrel{\text{def}}{=} ((-1, \varphi') \wedge (-1, q_0)) \vee (-1, q_\#)$$

where the states q_0 and q_1 are used to check that the node we are coming from was a right or a left son:

$$\delta(q_0, a) \stackrel{\text{def}}{=} (1, q_\top) \vee (1, q_\#) \qquad \delta(q_1, a) \stackrel{\text{def}}{=} (2, q_\top) \vee (2, q_\#)$$

and $q_\#$ checks that the node label is $\#$:

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

while $q_\top$ does not enforce any condition besides being labeled by $a \subseteq A$:

$$\delta(q_\top, a) \stackrel{\text{def}}{=} \top .$$

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

$$\delta(q_i, \#) \stackrel{\text{def}}{=} (1, q_\varphi) \wedge (2, q_\#) \qquad \delta(q_i, a) \stackrel{\text{def}}{=} \bot$$
$$\delta(q_\varphi, a) \stackrel{\text{def}}{=} \delta(\varphi, a) \wedge (2, q_\#) .$$

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

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

Corollary 2.9. *Satisfiability of PDL can be decided in* ExpTime.

Proof sketch. Given a PDL formula φ, by Lemma 2.4 construct an equivalent formula in normal form φ' with $|\varphi'| = O(|\varphi|)$. We then construct $\mathcal{A}_{\varphi'}$ with $O(|\varphi|)$ states by Lemma 2.5 and an alphabet of size at most $|\varphi|$, s.t. $\bar{t}$ is accepted by $\mathcal{A}_{\varphi'}$ iff $t, \mathrm{root} \models \varphi$. By Theorem 2.8 we can decide the existence of such a tree $\bar{t}$ in time $2^{O(|\varphi|^3)}$. The proof carries to satisfiability on tree structures rather than binary trees. □

2.2.3 Expressiveness

A few quick notes:

See Cate and Segoufin (2010).

PDL can be expressed in FO[TC^1] the **first-order logic with monadic transitive closure**. Translation into FO[TC^1]:

$$\begin{aligned}
\mathrm{ST}_x(a) &\stackrel{\text{def}}{=} P_a(x) \\
\mathrm{ST}_x(\top) &\stackrel{\text{def}}{=} (x = x) \\
\mathrm{ST}_x(\neg\varphi) &\stackrel{\text{def}}{=} \neg\mathrm{ST}_x(\varphi) \\
\mathrm{ST}_x(\varphi_1 \vee \varphi_2) &\stackrel{\text{def}}{=} \mathrm{ST}_x(\varphi_1) \vee \mathrm{ST}_x(\varphi_2) \\
\mathrm{ST}_x(\langle\pi\rangle\varphi) &\stackrel{\text{def}}{=} \exists y.\mathrm{ST}_{x,y}(\pi) \wedge \mathrm{ST}_y(\varphi) \\
\mathrm{ST}_{x,y}(\downarrow) &\stackrel{\text{def}}{=} x \downarrow y \\
\mathrm{ST}_{x,y}(\rightarrow) &\stackrel{\text{def}}{=} x \rightarrow y \\
\mathrm{ST}_{x,y}(\pi^{-1}) &\stackrel{\text{def}}{=} \mathrm{ST}_{y,x}(\pi) \\
\mathrm{ST}_{x,y}(\pi_1;\pi_2) &\stackrel{\text{def}}{=} \exists z.\mathrm{ST}_{x,z}(\pi_1) \wedge \mathrm{ST}_{z,y}(\pi_2) \\
\mathrm{ST}_{x,y}(\pi_1 + \pi_2) &\stackrel{\text{def}}{=} \mathrm{ST}_{x,y}(\pi_1) \vee \mathrm{ST}_{x,y}(\pi_2) \\
\mathrm{ST}_{x,y}(\pi^*) &\stackrel{\text{def}}{=} [\mathrm{TC}_{u,v}\,\mathrm{ST}_{u,v}(\pi)](x,y) \\
\mathrm{ST}_{x,y}(\varphi?) &\stackrel{\text{def}}{=} (x = y) \wedge \mathrm{ST}_x(\varphi)\ .
\end{aligned}$$

It is known that wMSO is strictly more expressive than FO[TC^1] (Cate and Segoufin, 2010, Theorem 2). 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 FO[TC^1]. It is open whether FO[TC^1] is strictly more expressive than PDL without this extension.

See Marx (2005).

A particular fragment called **conditional PDL** is equivalent to FO[$\downarrow^\star, \rightarrow^\star$]:

$$\pi ::= \alpha \mid \alpha^* \mid \pi;\pi \mid \pi + \pi \mid (\alpha;\varphi?)^* \mid \varphi? \qquad \text{(conditional paths)}$$

The translation to FO[$\downarrow^\star, \rightarrow^\star$] is as above, with

$$\begin{aligned}
\mathrm{ST}_{x,y}(\downarrow^*) &\stackrel{\text{def}}{=} x \downarrow^\star \\
\mathrm{ST}_{x,y}(\uparrow^*) &\stackrel{\text{def}}{=} y \downarrow^\star x \\
\mathrm{ST}_{x,y}(\rightarrow^*) &\stackrel{\text{def}}{=} x \rightarrow^\star y \\
\mathrm{ST}_{x,y}(\leftarrow^*) &\stackrel{\text{def}}{=} y \rightarrow^\star x \\
\mathrm{ST}_{x,y}((\alpha;\varphi?)^*) &\stackrel{\text{def}}{=} \forall z.(\mathrm{ST}_{x,z}(\alpha^*) \wedge \mathrm{ST}_{z,y}(\alpha^*)) \supset \mathrm{ST}_z(\varphi)\ .
\end{aligned}$$

Chapter 3

Model-Theoretic Semantics

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

See Chapter 17 of Jurafsky and Martin (2009) for more examples of meaning representations.

3.1 First-Order 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 **summarization** 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 $\mathfrak{M} = \langle D, (R_i)_i \rangle$ where D 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 D. The first-order language thus allows to reason about **truths** regarding entities and their relations.

Example 3.1. For instance, assume our vocabulary includes $John^{(0)}$ as a constant denoting *John*, along with $apple^{(1)}$, $red^{(1)}$, and $eat^{(2)}$, we can associate the sentence

$$\exists x. apple^{(1)}(x) \wedge red^{(1)}(x) \wedge eat^{(2)}(John^{(0)}, x) \tag{3.1}$$

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

$$\begin{array}{lll} a, j \in D & a \in red & a \in apple \\ j = John & (j, a) \in eat\ , & \end{array}$$

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: (3.1) could for instance be the formalization of

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

3.1.1 Event Semantics

The kind of modelling that underlies Example 3.1 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 **subcategorization 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 3.1, we need to introduce several relations $eat^{(i)}$ largely beyond the simple choice between the intransitive $eat_1^{(1)}$ and transitive $eat_2^{(2)}$ forms of *eat*:

$$eat_1^{(1)}(John^{(0)}) \tag{3.2}$$

$$\exists x.eat_2^{(2)}(John^{(0)}, x) \wedge red^{(1)}(x) \wedge apple^{(1)}(x) \tag{3.3}$$

$$\exists xy.eat_3^{(3)}(John^{(0)}, x, y) \wedge apple^{(1)}(x) \wedge park^{(1)}(y) \tag{3.4}$$

$$\exists y.eat_4^{(2)}(John^{(0)}, y) \wedge park^{(1)}(y) \tag{3.5}$$

$$\exists xy.eat_5^{(4)}(John^{(0)}, x, y, slowly^{(0)}) \wedge red^{(1)}(x) \wedge apple^{(1)}(x) \wedge park^{(1)}(y) \tag{3.6}$$

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. (3.6) entails each of (3.2–3.5)? One possibility is to add explicit meaning postulates like

$$\forall jxy.eat_3^{(3)}(j, x, y) \supset eat_2^{(2)}(j, x) \tag{3.7}$$

$$\forall jx.eat_2^{(2)}(j, x, y) \supset eat_1^{(1)}(j) \tag{3.8}$$

$$\ldots \tag{3.9}$$

Similarly, we could treat *slowly* and the locative *in* as modal operators and rewrite (3.6) as

$$\exists xy.location^{(2)}(slowly^{(1)}(eat_2^{(2)}(John^{(0)}, x), y) \wedge red^{(1)}(x) \wedge apple^{(1)}(x) \wedge park^{(1)}(y) \tag{3.10}$$

along with the schemata

$$\forall Py.in^{(2)}(P, y) \supset P \tag{3.11}$$

$$\forall P.slowly^{(1)}(P) \supset P \tag{3.12}$$

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

$$\exists xy.slowly^{(1)}(location^{(2)}(eat_2^{(2)}(John^{(0)}, x), y) \wedge red^{(1)}(x) \wedge apple^{(1)}(x) \wedge park^{(1)}(y) \tag{3.13}$$

instead, and proving the equivalence of (3.10) and (3.13) would require yet more machinery. (We will however return to modal operators later in Section 3.4.)

As we can see, this solution scales rather poorly. Another possibility is to pick a very general version of *eat*, like eat_5, and express the simpler versions with existentially quantified arguments:

$$eat_1^{(1)}(j) \stackrel{\text{def}}{=} \exists xya.eat_5^{(4)}(j,x,y,a) \tag{3.14}$$

$$eat_2^{(2)}(j,x) \stackrel{\text{def}}{=} \exists ya.eat_5^{(4)}(j,x,y,a) \tag{3.15}$$

$$eat_3^{(3)}(j,x,y) \stackrel{\text{def}}{=} \exists a.eat_5^{(4)}(j,x,y,a) \tag{3.16}$$

$$eat_4^{(2)}(j,y) \stackrel{\text{def}}{=} \exists ya.eat_5^{(4)}(j,x,y,a)\ . \tag{3.17}$$

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:

$$\exists e.eat_1^{(2)}(e, John^{(0)}) \tag{3.18}$$

$$\exists ex.eat_2^{(3)}(e, John^{(0)}, x) \wedge red^{(1)}(x) \wedge apple^{(1)}(x) \tag{3.19}$$

$$\exists exy.eat_2^{(3)}(e, John^{(0)}, x) \wedge apple^{(1)}(x) \wedge park^{(1)}(y) \wedge location^{(2)}(e,y) \tag{3.20}$$

$$\exists ey.eat_1^{(2)}(e, John^{(0)}) \wedge park^{(1)}(y) \wedge location^{(2)}(e,y) \tag{3.21}$$

$$\exists exy.eat_2^{(3)}(e, John^{(0)}, x) \wedge red^{(1)}(x) \wedge apple^{(1)}(x) \wedge park^{(1)}(y) \wedge location^{(2)}(e,y) \wedge slowly^{(1)}(e) \tag{3.22}$$

See Davidson (1967).

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 (3.2) and (3.5): contrast for instance

I sank the Bismark.
I sank.

where the transitive usage does not imply the intransitive one.

3.1.2 Thematic Roles

This is known as a ***neo-Davidsonian*** *analysis (Parsons, 1990).*

The Davidsonian analysis can be further refined by employing **thematic roles**: instead of seeing the intransitive form $eat_1^{(2)}$ and the transitive one $eat_2^{(3)}$ as two wholly different relations, we can further refine them using a fixed set of thematic relations between events and entities:

$$\exists e.eat^{(1)}(e) \wedge agent^{(2)}(e, John^{(0)}) \tag{3.23}$$

$$\exists ex.eat^{(1)}(e) \wedge agent^{(2)}(e, John^{(0)}) \wedge patient^{(2)}(e,x) \wedge apple^{(1)}(x) \tag{3.24}$$

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 the subject and the verb:

$$\exists e.sink^{(1)}(e) \wedge agent^{(2)}(e, I^{(0)}) \wedge patient^{(2)}(e, Bismark^{(0)}) \tag{3.25}$$

$$\exists e.sink^{(1)}(e) \wedge patient^{(2)}(e, I^{(0)}) \tag{3.26}$$

Role	Typical use
agent	*John eats*
patient	*John eats an apple.*
experiencer	*John regrets his actions.*
	The crisis worries John.
cause	*The crisis worries John.*
	John regrets his behaviour.
theme	*John asks a question.*
	John gives Mary a kiss.
beneficiary	*John gives Mary a kiss.*

Table 3.1: A basic set of thematic roles.

The definition of a fixed set of thematic roles and how to classify the different uses are of course problematic; Table 3.1 proposes a very simple account.

3.2 Syntax/Semantics Interface

We have presented several possible first-order analyses for simple sentences in the previous section, 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**:

See Janssen (1997) and the compositionality article of the Stanford Encyclopedia of Philosophy *for extensive discussions 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., Chapter 13)

Let us illustrate this principle on Example 3.1: by associating a semantic representation to each meaningful word in the sentence, i.e. if we define $[\![\textit{John}]\!]$, $[\![\textit{eats}]\!]$ 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 process (otherwise *John loves Mary* and *Mary loves John* would not be distinguishable).

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 $[\![\textit{John}]\!] = John^{(0)}$ and $[\![\textit{a red apple}]\!] = \exists x.apple^{(1)}(x) \wedge red^{(1)}(x)$, it is not so clear how one should combine everything and obtain (3.1) or more involved representations like (3.24). Moreover any solution will be dependent on the specific syntactic analysis.

3.2.1 *Background:* Simply Typed Lambda Calculus

See e.g. Hindley (1997).

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 3.5),

we favour a presentation that uses the **simply typed λ-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 **λ-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 λ operator is a *binding* with the usual associated notion of free variables. A λ-term L is a **λI-term** if in every subterm $\lambda x.M$, $x \in \mathrm{FV}(M)$. 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 λ-term**; we let $\Lambda_\ell(C)$ denote the set of linear λ-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 α, β, and η reductions:

$$\begin{aligned} \lambda x.L &\to_\alpha \lambda y.(L\{x \leftarrow y\}) \\ (\lambda x.L)M &\to_\beta L\{x \leftarrow N\} \\ \lambda x.Lx &\to_\eta L \end{aligned}$$

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

Types Assume we are provided with some non-empty countable set of *atomic* types A; then **types** in $\mathcal{T}_A$ 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 τ is defined inductively as

$$\mathrm{ord}(a) = 1 \qquad \mathrm{ord}(\sigma \to \tau) = \max(\mathrm{ord}(\sigma) + 1, \mathrm{ord}(\tau)) \ .$$

A **higher-order signature** is a triple $\Sigma = \langle A, C, t\rangle$ where A is a set of atomic types, C a countable set of *constants* and $t : C \to \mathcal{T}_A$ a *typing* of the constants. Given a higher-order signature, each λI-term of $\Lambda(C)$ can be assigned a type in $\mathcal{T}_A$ by the deduction rules

$$\frac{t(c) = \tau}{\vdash_\Sigma c : \tau}\ (\mathsf{Cons}) \qquad \frac{}{x : \tau \vdash_\Sigma x : \tau}\ (\mathsf{Var}) \qquad \frac{\Gamma, x : \sigma \vdash_\Sigma L : \tau}{\Gamma \vdash_\Sigma \lambda x.L : \sigma \to \tau}\ (\to\mathsf{I})$$

$$\frac{\Gamma \vdash_\Sigma L : \sigma \to \tau \quad \Delta \vdash_\Sigma M : \sigma \quad \Gamma, \Delta \text{ compatible}}{\Gamma, \Delta \vdash_\Sigma LM}\ (\to\mathsf{E})$$

where the **type contexts** Γ, Δ are type assignments from free variables to $\mathcal{T}_A$; in $(\to\mathsf{E})$ the two assignments have to be *compatible*, i.e. to assign the same types to common variables. For *linear* lambda terms, this compatibility requirement is useless as $\mathrm{FV}(L) \cap \mathrm{FV}(M) = \emptyset$. We can extend the typing system to any λ-term instead of λI-terms if we allow $(\to\mathsf{I})$ on the premise $\Gamma \vdash_\Sigma L : \tau$ where $x \notin \mathrm{FV}(L)$ (nor in the domain of Γ).

Encoding Trees For C a ranked alphabet, we can encode terms $c^{(n)}(t_1, \ldots, t_n)$ in $T(C)$ as typed λ-terms $c^{(n)} t_1 \cdots t_n$ in $\Lambda(C)$: let $o \in A$ denote the type of trees in $T(C)$, then we define the type of $c^{(n)}$ as $\overbrace{o \to \cdots \to o}^{n \text{ times}} \to o = o^n \to o$; this way only ground λ-terms encoding trees in $T(C)$ can be typed.

We write $\Sigma_C = \langle \{o\}, C, c^{(n)} \mapsto o^n \to o \rangle$ for the **tree signature** over the ranked alphabet C, and ℓ_C for the bijection between $T(C)$ and ground λ-terms in $\Lambda(C)$ with type o.

(∗) **Exercise 3.1** (Types for Tree Languages)**.**

1. Given a **deterministic local tree language** L—i.e. the language of a deterministic bottom-up tree automaton $\mathcal{A} = \langle Q, C, \delta, F \rangle$ where for each $c^{(n)}$ in C_n, $|\{q \in Q \mid \exists q_1, \ldots, q_n.(q, c^{(n)}, q_1, \ldots, q_n) \in \delta\}| \leq 1$—, define a signature $\Sigma = \langle A, C, t \rangle$ and a subset F of A such that a ground λ-term can be typed in F iff it encodes a tree in L.

2. Show that any regular tree language is the image of a deterministic local tree language by an alphabetic tree homomorphism.

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

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

Proposition 3.2 (Subject Reduction)**.** *If* $\Gamma \vdash_\Sigma L : \tau$ *and* $L \Rightarrow^\star_{\beta\eta} M$ *then* $\Gamma \vdash_\Sigma M : \tau$.

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

Proposition 3.3 (Subject Expansion)**.** *If* τ *is a linear* λ*-term,* $\Gamma \vdash_\Sigma L : \tau$*, and* $M \Rightarrow^\star_\beta L$*, then* $\Gamma \vdash_\Sigma M$.

The second main result about typed λ-terms is that reduction is **strongly normalizing**: every sequence of rewrites eventually terminates to a term in normal form.

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

Theorem 3.4 (Strong Normalization)**.** *If* L *is a typable* λ*-term, then every* $\beta\eta$*-reduction starting at* L *is finite.*

It is worth mentioning that every linear λ-term *is* typable.

3.2.2 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 λ-term over the input signature into a λ-term over the output signature and then to $\beta\eta$-reduce it to a λ-term in normal form.

*This idea is now pretty common, and lies at the heart of (second-order) **abstract categorial grammars** (ACG de Groote, 2001); see also the **context-free λ-term grammar** (CFLG) formulation of Kanazawa (2007) or the simple presentation of Blackburn and Bos (2005, Chapter 2).*

Definition 3.5 (Higher-Order Homomorphism)**.** A **higher-order homomorphism** from a set of constants C to a set of constants C' is generated by a function $[\![.]\!]$ mapping constants in C to closed λ-terms in $\Lambda(C')$. We lift $[\![.]\!]$ to a homomorphism from $\Lambda(C)$ to $\Lambda(C')$ by $[\![x]\!] = x$, $[\![LM]\!] = [\![L]\!][\![M]\!]$, and $[\![\lambda x.L]\!] = \lambda x.[\![L]\!]$.

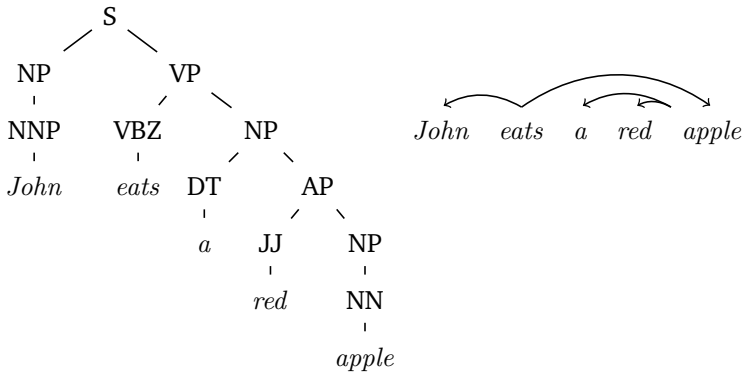

Figure 3.1: Constituent and dependency analyses for *John eats a red apple*.

Example 3.6. Continuing with Example 3.1, Figure 3.1 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 3.1, we have

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

and

$$C' = \{John^{(0)}, \wedge^{(2)}, \exists^{(2)}, \dots\} .$$

We assign the semantics

$$\begin{aligned}
[\![\mathit{John}^{(0)}]\!] &= \lambda x.x\ John^{(0)} \\
[\![\mathit{apple}^{(0)}]\!] &= \lambda x.apple^{(1)}\ x \\
[\![\mathit{red}^{(0)}]\!] &= \lambda x.red^{(1)}\ x \\
[\![\mathrm{AP}^{(2)}]\!] &= \lambda x_1 x_2 x.(x_1\ x) \wedge (x_2\ x) \\
[\![\mathit{a}^{(0)}]\!] &= \lambda xy.\exists u.(x\ u) \wedge (y\ u) \\
[\![\mathrm{NP}^{(2)}]\!] &= \lambda x_1 x_2 x.x_1\ x_2\ x \\
[\![\mathit{eats}^{(0)}]\!] &= \lambda xy.\exists e.(eat^{(1)}\ e) \wedge x(\lambda a.agent^{(2)}\ e\ a) \\
&\qquad \wedge y(\lambda p.patient^{(2)}\ e\ p) \\
[\![\mathrm{VP}^{(2)}]\!] &= \lambda x_1 x_2 x.x_1\ x\ x_2 \\
[\![\mathrm{S}^{(2)}]\!] &= \lambda x_1 x_2.x_2\ x_1
\end{aligned}$$

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

$$\begin{aligned}
[\![\mathrm{AP}\ \mathit{red}\ \mathit{apple}]\!] &\Rightarrow^\star_\beta \lambda x.(red^{(1)}\ x) \wedge (apple^{(1)}\ x) \\
[\![\mathrm{NP}\ \mathit{a}\ \mathrm{AP}\ \mathit{red}\ \mathit{apple}]\!] &\Rightarrow^\star_\beta \lambda x.\exists u.(red^{(1)}\ u) \wedge (apple^{(1)}\ u) \wedge (x\ u) \\
[\![\mathrm{VP}\ \mathit{eats}\ \mathrm{NP}\ \mathit{a}\ \mathrm{AP}\ \mathit{red}\ \mathit{apple}]\!] &\Rightarrow^\star_\beta \lambda x.\exists e.(eat^{(1)}\ e) \wedge x(\lambda a.agent^{(2)}\ e\ a) \\
&\qquad \wedge \exists u.(red^{(1)}\ u) \wedge (apple^{(1)}\ u) \wedge (patient^{(2)}\ e\ u) \\
[\![\mathrm{S}\dots]\!] &\Rightarrow^\star_\beta \exists e.(eat^{(1)}\ e) \wedge (agent^{(2)}\ e\ John^{(0)}) \\
&\qquad \wedge \exists u.(red^{(1)}\ u) \wedge (apple^{(1)}\ u) \wedge (patient^{(2)}\ e\ u) ,
\end{aligned}$$

which is the λ-term encoding of (3.24).

3.2.3 Tree Transductions

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

Definition 3.7 (Typed Homomorphism)**.** A **typed homomorphism** between two signatures $\Sigma = \langle A, C, t\rangle$ and $\Sigma' = \langle A', C', t'\rangle$ 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. $\vdash_{\Sigma'} [\![c]\!] : [\![t(c)]\!]$ is a valid typing judgement for all c in C.

This makes it easier to focus on *trees*:

Definition 3.8 (Higher-Order Tree Functions)**.** Let $\Sigma = \langle A, C, t\rangle$ and $\Sigma'_{C'}$ be two signatures where C and C' are two ranked alphabets, $[\![.]\!]$ be a typed homomorphism between Σ and Σ', and $s \in A$ a distinguished input atomic type with $[\![s]\!] = o$. We define the corresponding (partial) **higher-order tree function** $\mathscr{T} : T(C) \to T(C')$ by

$$\mathscr{T}(t_1) = t_2 \text{ iff } \vdash_\Sigma \ell_C(t_1) : s \wedge [\![\ell_C(t_1)]\!] \Rightarrow^\star_{\beta\eta} \ell_{C'}(t_2)\ . \tag{3.27}$$

Because $\ell^{-1}_{C'}$ is only defined on ground λ-terms, we see in particular that $\ell_{C'}(t_2)$ must be in $\beta\eta$-normal form.

Example 3.9. The semantic construction of Example 3.6 is a higher-order tree function when setting Σ_C and $\Sigma_{C'}$ as input and output types 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 $[\![c]\!]$ is linear for every c in C.

See de Groote (2001).

Definition 3.10 (Abstract Categorial Grammar)**.** An **abstract categorial grammar** (ACG) is a tuple $\mathcal{G} = \langle \Sigma, \Sigma', [\![.]\!], s\rangle$ where $\Sigma = \langle A, C, t\rangle$ and $\Sigma' = \langle A', C', t'\rangle$ are two signatures, $[\![.]\!]$ is a *linear* typed homomorphism, and s in A is a distinguished type. The **abstract language** $\mathscr{A}(\mathcal{G})$ of $\mathcal{G}$ is

$$\mathscr{A}(\mathcal{G}) = \{L \in \Lambda_\ell(C) \mid \vdash_\Sigma L : s\}$$

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

$$\mathscr{O}(\mathcal{G}) = [\![\mathscr{A}(\mathcal{G})]\!]$$

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

A second-order ACG with output signature $\Sigma_{C'}$ defines a *linear* higher-order tree function from its abstract language to its object language. The expressiveness of such tree functions has been studied by Kanazawa (2010): the object languages of such ACGs 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). Thus we could also implement such transformations as more classical tree transductions, although that would be at the expense of the ability to view the translation as one into higher-order semantics as in Section 3.5.

3.3 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

$$\exists y.woman(y) \wedge \forall x.man(x) \supset \exists e.love(e) \wedge agent(e,x) \wedge patient(e,y) \quad (3.28)$$

$$\forall x.man(x) \supset \exists y.woman(y) \wedge \exists e.love(e) \wedge agent(e,x) \wedge patient(e,y)\ . \quad (3.29)$$

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.

If we make the choice of compositional semantics and implement compositionality through homomorphisms, then we are facing a serious issue: how can we map a single syntactic representation to several semantic ones? Scope ambiguities are however a more general issue: even if 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.
>
> (Poesio, 1994)

is reputed as having several thousand readings. Arguably, not all these readings are born equal: some might be implied by others (just like (3.28) implies (3.29)), 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.

3.3.1 *Background:* Conjunctive Queries over Trees

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 tree models, i.e. we can reuse the model-theoretic methods of Chapter 2.

A URL formula is essentially a set of *constraints* describing admissible formulæ. Our constraints will be expressed in **existential conjunctive FO** (ECFO) over the vocabulary $\langle(\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a\in A}\rangle$ where A is the *finite* ranked alphabet used to write terms in the SRL and k is the maximal arity in A. Let $\mathcal{X}$ be an infinite countable set of variables, then we consider formulæ φ defined by

See Koller et al. *(2001) for an early proof of* NPTIME*-completeness of ECFO$((\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a\in A})$ over trees, and Hidders (2004); Björklund* et al. *(2007) for related results on XPath fragments.*

$$\alpha ::= x = y \mid x \downarrow_i y \mid x \downarrow^* y \mid P_a(x) \mid \neg\alpha \qquad \text{(atoms)}$$
$$\varphi ::= \alpha \mid \varphi \wedge \varphi \mid \exists x.\varphi\ , \qquad \text{(formulæ)}$$

where x, y are in $\mathcal{X}$, $i < k$, and a is in A. We interpret formulæ over ranked trees t which are prefix-closed and predecessor-closed partial maps $\{0, \ldots, k-1\}^* \to A$ s.t. if $t(u) = a^{(i)}$ then $u(i-1)$ is in $\mathrm{dom}(t)$ but ui is not. Such a tree defines a relational structure $\mathfrak{M} = \langle \mathrm{dom}(t), (\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a\in A}\rangle$ where

$$\downarrow_i \stackrel{\text{def}}{=} \{(u, ui) \in \mathrm{dom}(t)^2\}$$
$$\downarrow^* \stackrel{\text{def}}{=} \left(\bigcup_{i<k} \downarrow_i\right)^\star$$
$$P_a \stackrel{\text{def}}{=} \{u \in \mathrm{dom}(t) \mid t(u) = a\}\ .$$

Any ECFO formula φ can be put in **prenex conjunctive normal form**

$$\varphi \equiv \exists x_1 \ldots x_n . \bigwedge_p \alpha_p\ .$$

Theorem 3.11 (Koller et al., 2001)**.** *Satisfiability of closed ECFO$((\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a\in A})$ formulæ is* NPTIME*-complete.*

3.3.2 Hole Semantics

Our ECFO presentation of hole semantics follows Blackburn and Bos (2005, Chapter 3) rather than the original definition of Bos (1996).

The syntax of **hole formulæ** is a restricted fragment of ECFO$((\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a\in A})$. 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:

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

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

$$\gamma \equiv \exists l_1 \ldots l_n h_1 \ldots h_m . \bigwedge_p \gamma_p\ . \qquad (3.30)$$

Hole formulæ γ are *interpreted* in ECFO$((\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a\in A})$ by associating a formula $[\gamma]$

$$[\gamma] = \exists l_1 \dots l_n h_1 \dots h_m . \bigwedge_{1\leq i<j\leq n} l_i \neq l_j \wedge \bigwedge_p \gamma_p \tag{3.31}$$

where we interpret

$$l : a^{(r)}(x_1, \dots, x_r) \stackrel{\text{def}}{=} P_a(l) \wedge \bigwedge_{i=1}^{r} l \downarrow_{i-1} x_i \ . \tag{3.32}$$

A variable x in a hole formula is a **root** if there does not exist $x_0, \dots, x_r$ and $a^{(r)}$ s.t. $x_0 : a^{(r)}(x_1, \dots, x_r)$ is a subformula of γ where $x = x_r$. A hole formula is **normal** if

1. in every $h \downarrow^* l$ subformula, l is a root of γ,
2. every hole appears exactly once as a child of a $l : a^{(r)}(x_1, \dots, x_r)$ 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 $l : a^{(r)}(x_1, \dots, x_r)$ subformula. This excludes for instance $l' : f^{(2)}(l, l)$, $l : f^{(2)}(l_1, l_2) \wedge l : f^{(2)}(l'_1, l'_2)$, or $l_1 : g^{(1)}(l) \wedge l_2 : g^{(1)}(l)$.

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

Theorem 3.12 (Althaus et al., 2003)**.** *Satisfiability of normal hole formulæ is in* PTIME.

Constructive Satisfiability

The issue with our interpretation of hole formulæ into ECFO is that not every model $\mathfrak{M}$ over A is suitable as a SRL formula. For instance, there could be extra points in the model not constrained by γ, 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 (3.30). Then a **plugging** P is an injective function from holes $\{h_1, \dots, h_m\}$ to labels $\{l_1, \dots, l_n\}$. A model $\mathfrak{M} = \langle \mathrm{dom}(t), (\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a\in A}\rangle$ of γ is a **plugged model** for a plugging P if its domain is in bijection with the set of labels (we write $\mathrm{dom}(t) = \{\hat{l}_1, \dots, \hat{l}_n\}$) and $\mathfrak{M} \models_\nu \gamma$ where the valuation ν is defined by

$$\nu(x) \stackrel{\text{def}}{=} \begin{cases} \hat{x} & \text{if } x \in \mathcal{L} \\ \widehat{P(x)} & \text{if } x \in \mathcal{H} \ . \end{cases} \tag{3.33}$$

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

Example 3.13. Let us extend the syntax of hole formulæ by allowing larger tree segments:

$$\gamma ::= l : a^{(r)}(\theta_1, \dots, \theta_r) \mid h \downarrow^* l \mid \gamma \wedge \gamma \mid \exists x.\gamma \qquad \text{(hole formulæ)}$$
$$\theta ::= a^{(r)}(\theta_1, \dots, \theta_r) \mid h \qquad \text{(tree formulæ)}$$

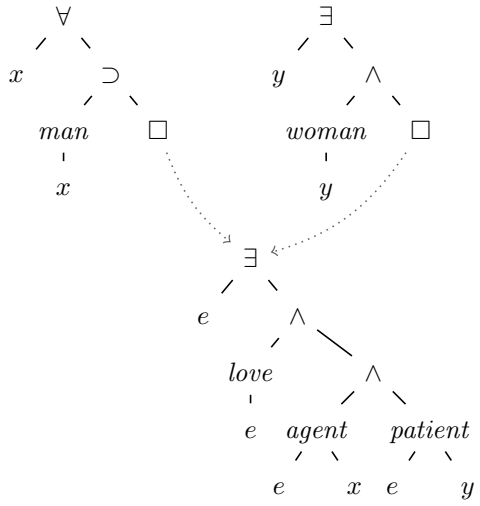

Figure 3.2: Underspecified formula for (3.28) and (3.29). Dominance relations are indicated through dotted arrows and holes by boxes.

and translating back into hole formulæ by defining

$$x_\theta \stackrel{\text{def}}{=} \begin{cases} h & \text{if } \theta = h \\ l_\theta \in \mathcal{L} & \text{a fresh label for each } \theta \text{ otherwise} \end{cases}$$

$$l : a^{(r)}(\theta_1, \ldots, \theta_r) \stackrel{\text{def}}{=} l : a^{(r)}(x_{\theta_1}, \ldots, x_{\theta_r})$$

$$a^{(r)}(\theta_1, \ldots, \theta_r) \stackrel{\text{def}}{=} \exists l_\theta . l_\theta : a^{(r)}(x_{\theta_1}, \ldots, x_{\theta_r}) \; .$$

A hole semantic formula that models the two readings (3.28) and (3.29) is the following (see also Figure 3.2):

$$\exists l_1 l_2 l_3 h_1 h_2 . l_1 : (\forall^{(2)} x^{(0)} . man^{(1)}(x^{(0)}) \supset^{(2)} h_1) \wedge l_2 : (\exists^{(2)} y^{(0)} . woman^{(1)}(y^{(0)}) \wedge^{(2)} h_2)$$
$$\wedge l_3 : (\exists^{(2)} e^{(0)} . love(1)(e^{(0)}) \wedge^{(2)} agent^{(2)}(e^{(0)}, x^{(0)}) \wedge^{(2)} patient^{(1)}(e^{(0)}, x^{(0)}))$$
$$\wedge h_1 \downarrow^* l_3 \wedge h_2 \downarrow^* l_3 \; .$$

Polynomial-time processing can be recovered if we further restrict hole formulæ; see Koller et al. *(2003).*

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

Theorem 3.14. *Constructive satisfiability of normal hole formulæ is* NPTIME*-complete.*

Proof. For the NPTIME upper bound, deciding whether a formula γ has a constructive model can be checked by

1. guessing a plugging P and constructing the corresponding model

$$\mathfrak{M} = \langle \{\hat{l}_1, \ldots, \hat{l}_n\}, (\downarrow_i)_{i<k}, (P_a)_{a \in A} \rangle \; ; \tag{3.34}$$

   this model is of polynomial size in $|\gamma|$,

2. computing the dominance relation $(\bigcup_{i<k} \downarrow_i)^\star$ over $\mathfrak{M}$ (this is in PTIME) to obtain a model

$$\mathfrak{M}' = \langle \{\hat{l}_1, \ldots, \hat{l}_n\}, (\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a \in A} \rangle \tag{3.35}$$

   still of polynomial size, and

3. verifying that $\mathfrak{M}'$ is a model of the existentially conjunctive formula $[\gamma]$ for the assignment ν defined in (3.33) (this is in PTIME).

For the NPTIME lower bound, we exhibit a reduction from the 3-partition problem: TODO □

This lower bound was given in (Althaus et al.*, 2003, Theorem 10.1).*

Exercise 3.2 (Tree Automata for Hole Formulæ)**.** The set of constructive models of a constraint is clearly a regular tree language. Provide a construction for a regular tree automaton $\mathcal{A}_\gamma$ that recognizes exactly the constructive models of a normal hole formula γ. **(****)**

Hint: I would use $2^{\{l_1,\ldots,l_n\}} \times \{l_1,\ldots,l_n\} \times 2^{\{h_1,\ldots,h_m\}}$ as state set, although there certainly are better ways; see for instance Koller et al. *(2008).*

The size of the automaton constructed in Exercise 3.2 is exponential in the size of the formula. This is unavoidable, as there exist normal formulæ γ_n of size $O(n)$ s.t. any automaton recognizing the set of plugged models of γ_n requires at least 2^n states: let

$$A_n \stackrel{\text{def}}{=} \{a^{(0)}, g_1^{(1)}, \ldots, g_n^{(1)}\} \tag{3.36}$$

$$\gamma_n \stackrel{\text{def}}{=} \exists l l_1 \ldots l_n h_1 \ldots h_n . l : a^{(0)} \wedge \bigwedge_{i=1}^{n} l_i : g_i^{(1)}(h_i) \wedge h_i \downarrow^* l \ . \tag{3.37}$$

The normal formula γ_n has $n!$ different models, corresponding to the possible orderings of its n components $g_i(\Box)$: its set of plugged models is

$$L_n = \{g_{\pi(1)}(\Box) \cdot g_{\pi(2)}(\Box) \cdots g_{\pi(n)}(a) \mid \pi \text{ a permutation of } \{1,\ldots,n\}\} \ . \tag{3.38}$$

Lemma 3.15. *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

$$C_K \stackrel{\text{def}}{=} g_{i_1}(\Box) \cdots g_{i_{|K|}}(\Box) \tag{3.39}$$

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

$$t_K \stackrel{\text{def}}{=} C_{\bar{K}} \cdot C_K \cdot a \tag{3.40}$$

is in L_n.

Let Q_K be the set of states q of an automaton $\mathcal{A}_n$ for L_n s.t.

$$C_{\bar{K}} \cdot C_K \cdot a \Rightarrow^\star C_{\bar{K}} \cdot q \Rightarrow^\star q_f \tag{3.41}$$

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 $K \backslash K'$ and $q \in Q_K \cap Q_{K'}$. Then i belongs to $\bar{K}'$ and

$$C_{\bar{K}'} \cdot C_K \cdot a \Rightarrow^\star C_{\bar{K}'} \cdot q \Rightarrow^\star q_f \tag{3.42}$$

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 $\mathcal{A}_n$ 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 3.15 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 3.2.3 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.

3.4 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: in

> John is ____ happy.

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

3.4.1 *Background:* Modal Logic

See (Blackburn et al.*, 2001).*

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* P to subsets of W.

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

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi \vee \varphi \mid \Diamond\varphi$$

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

$$\Box\varphi \stackrel{\text{def}}{=} \neg\Diamond\neg\varphi \ .$$

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 $\mathfrak{M}, w \not\models \varphi$
$\mathfrak{M}, w \models \varphi \vee \varphi'$	iff $\mathfrak{M}, w \models \varphi$ or $\mathfrak{M}, w \models \varphi'$
$\mathfrak{M}, w \models \Diamond\varphi$	iff $\exists w', w\,R\,w'$ and $M, w' \models \varphi$.

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

- in **alethic logic**, we reason about possible truths: $\Diamond\varphi$ denotes that "*possibly* φ" and $\Box\varphi$ "*necessarily* φ". If we follow Leibniz and imagine multiple "possible worlds" 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* φ", 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 modeled, we can introduce multiple relations R_a and modalities K_a, one for each agent a.

- In the **basic temporal logic**, $\Diamond\varphi$ denotes that "at some *future* point, φ holds", written $F\varphi$. Its dual $G\varphi$ means that in all future points, φ holds. Its **converse** P allows to reason about the past, and is defined by $\mathfrak{M}, w \models P\varphi$ iff 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.

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 (Kupfermana et al.*).*

Exercise 3.3 (Basic Axiom)**.** Show that $\mathbf{K} : \Box(\varphi \supset \psi) \supset (\Box\varphi \supset \Box\psi)$ is **valid**, i.e. for any model $\mathfrak{M}$ and any world w of W, $\mathfrak{M}, w \models \mathbf{K}$. **(∗)**

Exercise 3.4 (Transitive Frames)**.** Show that, if R is transitive, then $\mathbf{4} : \Diamond\Diamond\varphi \supset \Diamond\varphi$ is valid. **(∗)**

Exercise 3.5 (Epistemic Frames)**.** Prove the following implications for all modal formulæ φ when R is an equivalence relation: **(∗)**

- $\mathbf{T} : \Box\varphi \supset \varphi$—in epistemic logic, if indeed an agent *really* knows something, then it must be true—,
- $\mathbf{4} : \Box\varphi \supset \Box\Box\varphi$—in epistemic logic again, an agent has *introspection* about its own knowledge—,
- $\mathbf{B} : \varphi \supset \Box\Diamond\varphi$—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 2.2 is a modal language with an unbounded number of binary relations. A **modal similarity type** O is a ranked alphabet of modal operators $\triangle$ of arity $r(\triangle)$. A **modal formula** is then defined as

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi \vee \varphi \mid \triangle(\varphi_1, \ldots, \varphi_{r(\triangle)})$$

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

$$\mathfrak{M}, w \models \triangle(\varphi_1, \ldots, \varphi_{r(\triangle)}) \quad \text{iff } \exists w_1, \ldots, w_{r(\triangle)} \in W.(w, w_1, \ldots, w_{r(\triangle)}) \in R_\triangle \text{ and } \forall 1 \leq i \leq r(\triangle).\mathfrak{M}, w_i \models \varphi_i\ .$$

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

$$\begin{aligned}
\mathrm{ST}_x(p) &\stackrel{\text{def}}{=} P_p(x) \\
\mathrm{ST}_x(\top) &\stackrel{\text{def}}{=} (x = x) \\
\mathrm{ST}_x(\neg\varphi) &\stackrel{\text{def}}{=} \neg \mathrm{ST}_x(\varphi) \\
\mathrm{ST}_x(\varphi \vee \varphi') &\stackrel{\text{def}}{=} \mathrm{ST}_x(\varphi) \vee \mathrm{ST}_x(\varphi') \\
\mathrm{ST}_x(\triangle(\varphi_1, \ldots, \varphi_{r(\triangle)})) &\stackrel{\text{def}}{=} \exists x_1 \ldots x_{r(\triangle)}. R_\triangle(x, x_1, \ldots, x_{r(\triangle)}) \wedge \bigwedge_{i=1}^{r(\triangle)} \mathrm{ST}_{x_i}(\varphi_i)
\end{aligned}$$

is a FO formula with a free variable x equivalent to φ: $\mathfrak{M}, w \models \varphi$ iff $\mathfrak{M} \models_{x \mapsto w} \mathrm{ST}_x(\varphi)$. By reusing variables in the standard translation, we can use only $(n+1)$ first-order variables if $\max_{\triangle \in O}(r(\triangle)) = n$.

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

Bisimulations and Modal Invariance

Definition 3.16 (Bisimulations)**.** Let O be a modal similarity type and let $\mathfrak{M} = \langle W, (R_\triangle)_{\triangle \in O}, V \rangle$ and $\mathfrak{M}' = \langle W, (R'_\triangle)_{\triangle \in O}, V' \rangle$ be two O-models. A non-empty relation $Z \subseteq W \times W'$ is a **bisimulation** between $\mathfrak{M}$ and $\mathfrak{M}'$ if for all w, w' s.t. $w \, Z \, w'$,

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

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

Proposition 3.17 (Invariance for Bisimulation)**.** *Let O be a modal similarity type, and $\mathfrak{M}$ and $\mathfrak{M}'$ be O-models. Then, for every w in W and w' in W' with $w \leftrightarrows w'$, and every modal formula φ, $\mathfrak{M}, w \models \varphi$ iff $\mathfrak{M}, w' \models \varphi$.*

Proof. The proof proceeds by induction on φ. The case where φ is an atomic proposition is a consequence of (1) in Definition 3.16, the case where φ is $\top$ is trivial, and the cases of boolean connectives follow from the induction hypothesis. For a formula of form $\triangle(\varphi_1, \ldots, \varphi_{r(\triangle)})$:

$$\begin{aligned}
&\mathfrak{M}, w \models \triangle(\varphi_1, \ldots, \varphi_{r(\triangle)}) \\
\text{implies } &\exists w_1, \ldots, w_{r(\triangle)} \in W. (w, w_1, \ldots, w_{r(\triangle)}) \in R_\triangle \wedge \forall 1 \leq i \leq r(\triangle). \mathfrak{M}, w_i \models \varphi_i \\
\text{implies } &\exists w'_1, \ldots, w'_{r(\triangle)} \in W'. \forall 1 \leq i \leq r(\triangle). \mathfrak{M}', w'_i \models \varphi_i \quad \text{(by ind. hyp. and (2))} \\
\text{implies } &\mathfrak{M}', w' \models \triangle(\varphi_1, \ldots, \varphi_{r(\triangle)}) \, ,
\end{aligned}$$

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

It is worth mentionning that the converse does not hold in general: there exist models which are undistinguishable by modal formulæ but not bisimilar. In the case of models with **finite image** however, where for every $R_\triangle$ and w $\{(w_1, \ldots, w_{r(\triangle)}) \mid (w, w_1, \ldots, w_{r(\triangle)}) \in R_\triangle\}$ is finite, the converse holds: let us define the **modal equivalence** relation $w \leftrightsquigarrow w'$ as holding iff w and w' are indistinguishable, i.e. $\{\varphi \mid \mathfrak{M}, w \models \varphi\} = \{\varphi' \mid \mathfrak{M}', w' \models \varphi'\}$.

Theorem 3.18 (Hennessy-Milner Theorem)**.** *Let O be a modal similarity type, and $\mathfrak{M}$ and $\mathfrak{M}'$ be O-models with finite image. If $w \leftrightsquigarrow w'$, then $w \underline{\leftrightarrow} w'$.*

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 $w \leftrightsquigarrow w'$ and $(w, w_1, \ldots, w_{r(\triangle)}) \in R_\triangle$, and assume that there do not exist $w'_1, \ldots, w'_{r(\triangle)}$ satisfying (2). The image set $S' = \{(w'_1, \ldots, w'_{r(\triangle)}) \mid (w', w'_1, \ldots, w'_{r(\triangle)}) \in R'_\triangle\}$ is finite, and non empty since otherwise $\mathfrak{M}, w \models \triangle(\top, \ldots, \top)$ but $\mathfrak{M}', w' \not\models \triangle(\top, \ldots, \top)$. Thus S' is a finite set $\{(w'_{1,1}, \ldots, w'_{1,r(\triangle)}), \ldots, (w'_{n,1}, \ldots, w'_{n,r(\triangle)})\}$ where, by assumption, for every $1 \leq j \leq n$, there exists $1 \leq i \leq r(\triangle)$ s.t. $w_i \not\leftrightsquigarrow w'_{j,i}$, i.e. there exists a formula $\varphi_{j,i}$ s.t. $\mathfrak{M}, w_i \models \varphi_{j,i}$ but $\mathfrak{M}', w'_{j,i} \not\models \varphi_{j,i}$. But then

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

in contradiction with $w \leftrightsquigarrow w'$. The argument for condition (3) is symmetric. □

The van Benthem Characterization 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 $\mathrm{FO}((R_\triangle)_{\triangle\in O}, (P_p)_{p\in A})$ 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 $\mathfrak{M} \models_{x\mapsto w} \psi(x)$ iff $\mathfrak{M} \models_{x\mapsto w'} \psi(x)$.

Theorem 3.19 (van Benthem Characterization Theorem)**.** *Let $\psi(x)$ be a first-order formula in FO$((R_\triangle)_{\triangle\in O}, (P_p)_{p\in A})$ with one free variable x. Then $\psi(x)$ is invariant for bisimulation iff it is equivalent to the standard translation of a modal formula.*

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 NPTIME-complete satisfaction problems, while the basic modal language is PSPACE-complete. Model-checking of finite models is usually PTIME-complete.

See Blackburn et al. *(2001, Chapter 6).*

3.4.2 First-Order Modal Logic

In order to work with both modal operators and first-order semantics as in Section 3.1, 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 $\langle (R_i)_i \rangle$ of k_i-ary symbols is

$$\varphi ::= x = y \mid R_i(x_1, \ldots, x_{k_i}) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi \mid \exists x.\varphi$$

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

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

$$\begin{array}{ll}
\mathfrak{M}, w \models_\nu x = y & \text{iff } \nu(x) = \nu(y) \\
\mathfrak{M}, w \models_\nu R_i(x_1, \ldots, x_{k_i}) & \text{iff } (\nu(x_1), \ldots, \nu(x_n)) \in I(R_i)(w) \\
\mathfrak{M}, w \models_\nu \neg\varphi & \text{iff } \mathfrak{M}, w \not\models_\nu \varphi \\
\mathfrak{M}, w \models_\nu \varphi \wedge \varphi' & \text{iff } \mathfrak{M}, w \models_\nu \varphi \text{ and } \mathfrak{M}, w \models_\nu \varphi' \\
\mathfrak{M}, w \models_\nu \Diamond\varphi & \text{iff } \exists w' \in W. w\, R\, w' \text{ and } \mathfrak{M}, w' \models_\nu \varphi \\
\mathfrak{M}, w \models_\nu \exists x.\varphi & \text{iff } \exists e \in D_{\mathcal{O}}(w). \mathfrak{M}, w \models_{\nu[x \leftarrow e]} \varphi\ .
\end{array}$$

See also the entry on actualism in the Stanford Encyclopedia of Philosophy.

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 $D_{\mathcal{O}}(w) = \mathfrak{D}$ 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 3.20 (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

$$\exists a.\mathit{apple}^{(1)}(a) \wedge P(\exists e.\mathit{eat}^{(1)}(e) \wedge \mathit{agent}^{(2)}(e, \mathit{John}^{(0)}) \wedge \mathit{patient}^{(2)}(e, a))\ . \quad (3.43)$$

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 $\mathit{apple}(a)$ holds can be found. Another reading would be

$$P(\exists a.\mathit{apple}^{(1)}(a) \wedge \exists e.\mathit{eat}^{(1)}(e) \wedge \mathit{agent}^{(2)}(e, \mathit{John}^{(0)}) \wedge \mathit{patient}^{(2)}(e, a))\ . \quad (3.44)$$

Recall that we can deal with such scope ambiguities as in Section 3.3 and associate a unique hole formula to the sentence.

This section is based on (Fitting, 2004) and the entry on intensional logic in the Stanford Encyclopedia of Philosophy.

3.4.3 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 $\mathcal{O}$ and an infinite countable set of intension variables $\mathcal{I}$, FOIL formulæ follow the syntax

$$\varphi ::= x = x' \mid R_i(y_1, \ldots, y_{k_i}) \mid [\lambda x.\varphi](f) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi \mid \exists y.\varphi$$

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

Fitting (2004) also adds a typing discipline to the relations R_i to better differentiate between intensional and extensional arguments.

Intensional models for FOIL are of form $\mathfrak{M} = \langle W, R, D_{\mathcal{O}}, D_{\mathcal{I}}, I\rangle$ where a distinction is drawn between the *object domain* $D_{\mathcal{O}}$, which is a non-empty set in our

constant semantics, and the *intension domain* $D_{\mathcal{I}}$, which is a non-empty set of functions from W to $D_{\mathcal{O}}$, and I maps a relational symbols R_i with arity k_i to a mapping $I(R_i)$ from W to relations over $(D_{\mathcal{O}} \cup D_{\mathcal{I}})^{k_i}$. A *valuation* is now a mapping assigning members of $D_{\mathcal{O}}$ to object variables and members of $D_{\mathcal{I}}$ to intension variables. The satisfiability relation is similar to that of FOML, with

$$\begin{aligned} &\mathfrak{M}, w \models_\nu \exists f.\varphi && \text{iff } \exists i \in D_{\mathcal{I}}(w).\mathfrak{M}, w \models_{\nu[f \leftarrow i]} \varphi \\ &\mathfrak{M}, w \models_\nu [\lambda x.\varphi](f) && \text{iff } \mathfrak{M}, w \models_{\nu[x \leftarrow \nu(f)(w)]} \varphi\ . \end{aligned}$$

Example 3.21 (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 $[\lambda x x'.x = x'](f, g)$ is correct in the real word w, where f and g are associated to the same object $\nu(f)(w) = \nu(g)(w)$ in $D_{\mathcal{O}}$, namely Venus. In an epistemic setting, the de dicto reading $K[\lambda x x'.x = x'](f, g)$ 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 $[\lambda x x'.K(x = x')](f, g)$ 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 3.22 (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 $P[\lambda x x'.x = x'](f, g)$ 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: $[\lambda x x'.P(x = x')](f, g)$ 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 $[\lambda x'.x = x'](f)$ 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 $D_{\mathcal{O}}$. There is however no requirement for every object to be designated by some intension, i.e. for

$$\forall x.\exists f.D(f, x) \tag{3.45}$$

to hold. This is however a reasonable restriction; let us check for instance the following equivalence under the hypothesis of (3.45):

$$\exists x.\varphi \equiv \exists f.[\lambda x.\varphi](f)\ . \tag{3.46}$$

Indeed, for all $\mathfrak{M}$, w, ν and φ,

$$\begin{aligned} &\mathfrak{M}, w \models_\nu \exists f.[\lambda x.\varphi](f) \\ \text{iff } &\exists i \in D_{\mathcal{I}}.\mathfrak{M}, w \models_{\nu[f \leftarrow i]} [\lambda x.\varphi](f) \\ \text{iff } &\exists i \in D_{\mathcal{I}}.\mathfrak{M}, w \models_{\nu[f \leftarrow i, x \leftarrow i(w)]} \varphi \\ \text{iff } &\exists e \in D_{\mathcal{O}}.\mathfrak{M}, w \models_{\nu[x \leftarrow e]} \varphi && \text{(by (3.45) when choosing } i(w) = e) \\ \text{iff } &\mathfrak{M}, w \models_\nu \exists x.\varphi\ . \end{aligned}$$

Exercise 3.6. Show the following equivalence when (3.45) holds: **(∗)**

$$\exists f.\Diamond[\lambda x.\varphi](f) \equiv \Diamond(\exists x.\varphi)\ . \tag{3.47}$$

Example 3.23 (Unicorn)**.** The sentence *John finds a unicorn* could be associated with the semantics

$$\exists ex.\mathit{find}^{(1)}(e) \wedge \mathit{agent}^{(2)}(e, \mathit{John}^{(0)}) \wedge \mathit{patient}^{(2)}(e, x) \wedge \mathit{unicorn}^{(1)}(x) \tag{3.48}$$

but it is better to treat *unicorn* as an intension in the formula

$$\exists u.[\lambda x.\exists e.\mathit{find}^{(1)}(e) \wedge \mathit{agent}^{(2)}(e, \mathit{John}^{(0)}) \wedge \mathit{patient}^{(2)}(e, x) \wedge \mathit{unicorn}^{(1)}(x)](u)\ , \tag{3.49}$$

equivalent to (3.48) in totally intensional models according to (3.46). Then we better see the connection with the sentence *John seeks a unicorn*: its de dicto semantics would be

$$\exists u.\mathrm{TRY}(\mathit{John}^{(0)}, [\lambda x.\exists e.\mathit{find}^{(1)}(e) \wedge \mathit{patient}^{(2)}(e, x) \wedge \mathit{unicorn}^{(1)}(x)](u)) \tag{3.50}$$

$$\equiv \mathrm{TRY}(\mathit{John}^{(0)}, \exists ex.\mathit{find}^{(1)}(e) \wedge \mathit{patient}^{(2)}(e, x) \wedge \mathit{unicorn}^{(1)}(x)) \quad \text{(by (3.47))}$$

and its de re semantics

$$\exists u.[\lambda x.\mathrm{TRY}(\mathit{John}^{(0)}, \exists e.\mathit{find}^{(1)}(e) \wedge \mathit{patient}^{(2)}(e, x) \wedge \mathit{unicorn}^{(1)}(x)](u) \tag{3.51}$$

$$\equiv \exists x.\mathrm{TRY}(\mathit{John}^{(0)}, \exists e.\mathit{find}^{(1)}(e) \wedge \mathit{patient}^{(2)}(e, x) \wedge \mathit{unicorn}^{(1)}(x)) \quad \text{(by (3.46))}$$

and if the interpretation of $\mathit{unicorn}^{(1)}$ is the same in all worlds accessible through the TRY modality,

$$\equiv \exists x.\mathit{unicorn}^{(1)}(x) \wedge \mathrm{TRY}(\mathit{John}^{(0)}, \exists e.\mathit{find}^{(1)}(e) \wedge \mathit{patient}^{(2)}(e, x))\ .$$

3.5 Higher-Order Semantics

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 3.2 not as a technical means to generate trees, but as the true semantics of the sentences under consideration.

3.5.1 *Background:* Church's Simple Theory of Types

See Church (1940) and the entry in the Stanford Encyclopedia of Philosophy.

Higher-order semantics are typically expressed in simply typed lambda calculus as defined in Section 3.2.1. As we want not just to manipulate typed λ-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 ι denotes *entities* and o *truths*. The logical constants are $C = \{\bot, \supset, (\forall_\tau)_{\tau\in\mathcal{T}(A)}\}$ with types $t(\bot) = o$, $t(\supset) = o \to o \to o$, and $(\forall_\tau) = (\tau \to o) \to o$ for each type τ in $\mathcal{T}(A)$.

We write as usual $L \supset M$ for $\supset\, L\, M$ and $\forall_\tau x.L$ for $\forall_\tau(\lambda x.L)$. The other logical connectives are defined classically: $\neg L \stackrel{\text{def}}{=} L \supset \bot$, $L \vee M \stackrel{\text{def}}{=} (\neg L) \supset M$, $L \wedge M \stackrel{\text{def}}{=} \neg((\neg L) \vee (\neg M))$, etc. Equality is defined in the Leibnizian way as $L = M \stackrel{\text{def}}{=} \forall x.x\, L \supset x\, M$, 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 λ-terms. In natural deduction sequent style,

$$\frac{}{\Gamma, L \Vdash L}\ (\mathsf{Ax}) \qquad \frac{\Gamma, \neg L \Vdash \bot}{\Gamma \Vdash L}\ (\bot\mathsf{E})$$

$$\frac{\Gamma, L \Vdash M}{\Gamma \Vdash L \supset M}\ (\supset\mathsf{I}) \qquad \frac{\Gamma \Vdash L \supset M \quad \Gamma \Vdash L}{\Gamma \Vdash M}\ (\supset\mathsf{E})$$

$$\frac{\Gamma \Vdash L \quad x \notin \mathrm{FV}(\Gamma)}{\Gamma \Vdash \forall_\tau x.L}\ (\forall\mathsf{I}) \qquad \frac{\Gamma \Vdash \forall_\tau L \quad \Delta \vdash_\Sigma M : \tau}{\Gamma \Vdash L\, M}\ (\forall\mathsf{E})$$

$$\frac{\Gamma \Vdash L \quad L =_\beta M}{\Gamma \Vdash M}\ (\beta)$$

The deduction system also often includes the **extensionality axioms**:

$$\frac{}{\Gamma \Vdash (\forall_\tau x.L\, x = M\, x) \supset (L = M)}\ (\lambda\mathsf{X}) \qquad \frac{}{\Gamma \Vdash (L \equiv M) \supset (L = M)}\ (\equiv\mathsf{X})$$

More axioms are used in the simple theory of types; see Church (1940).

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 3.5.2 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 $E\, e$. In fact, Church (1940) shows how to implement Peano's arithmetic in the simple theory of types, from which we can deduce incompleteness of higher-order logic.

See also Henkin (1950).

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

3.5.2 Type-Logical Semantics

We follow Muskens (2011) for this section, itself based on Gallin (1975). See also the entry on Montague semantics in the Stanford Encyclopedia of Philosophy.

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 higher-order signature $\Sigma = \langle A, \{\bot, \supset, (\forall_\tau)_{\tau\in\mathcal{T}(A)}\} \cup C, t\rangle$ as in the simple theory of types, where $A = \{s, \iota, o\}$ and C denotes additional non-logical constants. To simplify matters, we avoid explicit events from Section 3.1.2.

Due to the relativization wrt. worlds, a simple sentence like *John walks* is expected to be of type $s \to o$ and to be associated to a logical representation like

$$walks\ John\ . \tag{3.52}$$

*Observe that we introduced an explicit type for worlds in the logic: this can be avoided if we use **intensional models** as in (Muskens, 2007). Recall that Church's simple type theory verifies the extensionality axioms!*

In order to obtain the appropriate type, a possibility is to set $t(walks) = \iota \to s \to o$ and $t(John) = \iota$. Looking at more complex examples (for instance Example 3.6), we arrive at the types of Table 3.2. The semantics of a sentence can then be

syntactic category	examples	type
intransitive verbs	walk, talk, eat_1, ...	$\iota \to s \to o$
transitive verbs	eat_2, love, ...	$\iota \to \iota \to s \to o$
common nouns	apple, man, woman, ...	$\iota \to s \to o$
adjectives	red, ...	$\iota \to s \to o$
determiners	every, a, the, no, ...	$(\iota \to s \to o) \to (\iota \to s \to o) \to s \to o$
proper nouns	John, Mary, ...	ι
modal adverbs	necessarily, possibly, ...	$(s \to o) \to s \to o$
modal verbs	know, believe, ...	$(s \to o) \to \iota \to s \to o$
negation	not	$(s \to o) \to s \to o$

Table 3.2: Some constants and their possible types.

$$
\begin{aligned}
[\![\text{walk}]\!] &= walk_{\iota\to s\to o} \\
[\![\text{eat}_2]\!] &= eat2_{\iota\to\iota\to s\to o} \\
[\![\text{apple}]\!] &= apple_{\iota\to s\to o} \\
[\![\text{red}]\!] &= \lambda P_{\iota\to s\to o} x_\iota w_s.red_{\iota\to s\to o}\, x\, w \wedge P\, x\, w \\
[\![\text{every}]\!] &= \lambda P_{\iota\to s\to o} P'_{\iota\to s\to o} w_s.\forall_\iota x.(P\, x\, w \supset P'\, x\, w) \\
[\![\text{a}]\!] &= \lambda P_{\iota\to s\to o} P'_{\iota\to s\to o} w_s.\exists_\iota x.(P\, x\, w \wedge P'\, x\, w) \\
[\![\text{no}]\!] &= \lambda P_{\iota\to s\to o} P'_{\iota\to s\to o} w_s.\forall_\iota x.(P\, x\, w \supset \neg P'\, x\, w) \\
[\![\text{the}]\!] &= \lambda P_{\iota\to s\to o} P'_{\iota\to s\to o} w_s.\exists_\iota x.(P'\, x\, w \wedge \forall_\iota y.(P\, x\, w \equiv x = y)) \\
[\![\text{John}]\!] &= John_\iota \\
[\![\text{necessarily}]\!] &= \lambda p_{s\to o} w_s.\forall_s w'.(R_{s\to s\to o}\, w\, w') \supset p\, w' \\
[\![\text{possibly}]\!] &= \lambda p_{s\to o} w_s.\exists_s w'.(R_{s\to s\to o}\, w\, w') \wedge p\, w' \\
[\![\text{know}]\!] &= \lambda p_{s\to o} x_\iota w_s.\forall_s w'.(K_{\iota\to s\to s\to o}\, x\, w\, w') \supset p\, w' \\
[\![\text{believe}]\!] &= \lambda p_{s\to o} x_\iota w_s.\forall_s w'.(B_{\iota\to s\to s\to o}\, x\, w\, w') \supset p\, w' \\
[\![\text{not}]\!] &= \lambda p_{s\to o} w_s.\neg\, p\, w
\end{aligned}
$$

Table 3.3: Examples of semantics associated with lexical elements.

computed by a higher-order homomorphism as in Section 3.2, but there will be no need to translate back from λ-terms to first-order terms in order to reason about the semantics: the λ-term is a meaning representation with full-fledged model theory. See Table 3.3 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 $s \to s \to o$, $\iota \to s \to s \to o$, and $\iota \to s \to s \to o$ respectively. The desired properties of these relations can also be enforced; for instance $\forall_s ww'.\, R\, ww'$ forces R to be total.

Chapter 4

References

Afanasiev, L., Blackburn, P., Dimitriou, I., Gaiffe, B., Goris, E., Marx, M., and de Rijke, M., 2005. PDL for ordered trees. *Journal of Applied Non-Classical Logic*, 15(2):115–135. doi:10.3166/jancl.15.115-135. Cited on page 28.

Aho, A.V., 1968. Indexed grammars—An extension of context-free grammars. *Journal of the ACM*, 15(4):647–671. doi:10.1145/321479.321488. Cited on page 10.

Althaus, E., Duchier, D., Koller, A., Mehlhorn, K., Niehren, J., and Thiel, S., 2003. An efficient graph algorithm for dominance constraints. *Journal of Algorithms*, 48(1):194–219. doi:10.1016/S0196-6774(03)00050-6. Cited on pages 43, 45, 47.

Berstel, J., 1979. *Transductions and Context-Free Languages*. Teubner Studienbücher: Informatik. Teubner. ISBN 3-519-02340-7. http://www-igm.univ-mlv.fr/~berstel/LivreTransductions/LivreTransductions.html. Cited on page 2.

Björklund, H., Martens, W., and Schwentick, T., 2007. Conjunctive query containment over trees. In Arenas, M. and Schwartzbach, M.I., editors, *DBPL 2007*, volume 4797 of *Lecture Notes in Computer Science*, pages 66–80. Springer. doi:10.1007/978-3-540-75987-4_5. Cited on page 44.

Blackburn, P., Gardent, C., and Meyer-Viol, W., 1993. Talking about trees. In *EACL '93, Sixth Meeting of the European Chapter of the Association for Computational Linguistics*, pages 21–29. ACL Press. doi:10.3115/976744.976748. Cited on page 28.

Blackburn, P., Meyer-Viol, W., and Rijke, M.d., 1996. A proof system for finite trees. In Kleine Büning, H., editor, *CSL '95, 9th International Workshop on Computer Science Logic*, volume 1092 of *Lecture Notes in Computer Science*, pages 86–105. Springer. doi:10.1007/3-540-61377-3_33. Cited on page 28.

Blackburn, P., de Rijke, M., and Venema, Y., 2001. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press. Cited on pages 29, 48, 50, 51.

Blackburn, P. and Bos, J., 2005. *Representation and Inference for Natural Language: A First Course in Computational Semantics*. CSLI Studies in Computational Linguistics. CSLI Publications. ISBN 1-57586-496-7. Cited on pages 40, 44.

Book, R. and Otto, F., 1993. *String Rewriting Systems*. Texts and monographs in Computer Science. Springer. ISBN 3-540-97965-4. Cited on page 3.

Bos, J., 1996. Predicate logic unplugged. In Dekker, P. and Stokhof, M., editors, *AC '96, Tenth Amsterdam Colloquium*, pages 133–143. ILLC/Department of Philosophy, University of Amsterdam. Cited on pages 43, 44.

Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M., 2009. An automata-theoretic approach to Regular XPath. In Gardner, P. and Geerts, F., editors, *DBPL 2009, 12th International Symposium on Database Programming Languages*, volume 5708 of *Lecture Notes in Computer Science*, pages 18–35. Springer. doi:10.1007/978-3-642-03793-1_2. Cited on page 32.

Cate, B.T. and Segoufin, L., 2010. Transitive closure logic, nested tree walking automata, and XPath. *Journal of the ACM*, 57(3):18:1–18:41. doi:10.1145/1706591.1706598. Cited on page 34.

Chomsky, N., 1957. *Syntactic Structures*. Mouton de Gruyter. Cited on page 18.

Church, A., 1940. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68. doi:10.2307/2266170. Cited on pages 38, 55, 56.

Collins, M., 1999. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania. http://www.cs.columbia.edu/~mcollins/papers/thesis.ps. Cited on page 26.

Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M., 2007. *Tree Automata Techniques and Applications*. http://tata.gforge.inria.fr/. Cited on pages 2, 4, 10, 23, 27.

Copestake, A., Flickinger, D., Pollard, C., and Sag, I., 2005. Minimal recursion semantics: An introduction. *Research on Language & Computation*, 3(2):281–332. doi:10.1007/s11168-006-6327-9. Cited on page 43.

Crabbé, B., 2005. Grammatical development with XMG. In Blache, P., Stabler, E., Busquets, J., and Moot, R., editors, *LACL 2005, 5th International Conference on Logical Aspects of Computational Linguistics*, volume 3492 of *Lecture Notes in Computer Science*, pages 84–100. Springer. doi:10.1007/11422532_6. Cited on page 9.

Davidson, D., 1967. The logical form of action sentences. In Rescher, N., editor, *The Logic of Decision and Action*. University of Pittsburgh Press. doi:10.1093/0199246270.001.0001. Cited on page 37.

de Groote, P., 2001. Towards abstract categorial grammars. In *ACL 2001, 39th Annual Meeting of the Association for Computational Linguistics*, pages 252–259. ACL Press. doi:10.3115/1073012.1073045. Cited on pages 9, 18, 40, 42.

Doner, J., 1970. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4(5):406–451. doi:10.1016/S0022-0000(70)80041-1. Cited on page 27.

Duchier, D., Prost, J.P., and Dao, T.B.H., 2009. A model-theoretic framework for grammaticality judgements. In *FG 2009, 14th International Conference on Formal Grammar*. http://hal.archives-ouvertes.fr/hal-00458937/. Cited on page 21.

Egg, M., Koller, A., and Niehren, J., 2001. The constraint language for lambda structures. *Journal of Logic, Language and Information*, 10(4):457–485. doi:10.1023/A:1017964622902. Cited on page 43.

Engelfriet, J. and Vogler, H., 1985. Macro tree transducers. *Journal of Computer and System Sciences*, 31:71–146. doi:10.1016/0022-0000(85)90066-2. Cited on page 15.

Engelfriet, J. and Heyker, L., 1992. Context-free hypergraph grammars have the same term-generating power as attribute grammars. *Acta Informatica*, 29(2):161–210. doi:10.1007/BF01178504. Cited on page 42.

Engelfriet, J. and Maneth, S., 2000. Tree languages generated by context-free graph grammars. In Ehrig, H., Engels, G., Kreowski, H.J., and Rozenberg, G., editors, *TAGT '98, 6th International Workshop on Theory and Application of Graph Transformations*, volume 1764 of *Lecture Notes in Computer Science*, pages 15–29. Springer. doi:10.1007/978-3-540-46464-8_2. Cited on page 42.

Filmus, Y., 2011. Lower bounds for context-free grammars. *Information Processing Letters*, 111(18):895–898. doi:10.1016/j.ipl.2011.06.006. Cited on page 48.

Fischer, M.J., 1968. Grammars with macro-like productions. In *SWAT '68, 9th Annual Symposium on Switching and Automata Theory*, pages 131–142. IEEE Computer Society. doi:10.1109/SWAT.1968.12. Cited on pages 10, 11, 12, 15.

Fischer, M.J. and Ladner, R.E., 1979. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211. doi:10.1016/0022-0000(79)90046-1. Cited on pages 28, 29.

Fitting, M., 2004. First-order intensional logic. *Annals of Pure and Applied Logic*, 127 (1–3):173–193. doi:10.1016/j.apal.2003.11.014. Cited on pages 52, 53.

Fujiyoshi, A. and Kasai, T., 2000. Spinal-formed context-free tree grammars. *Theory of Computing Systems*, 33(1):59–83. doi:10.1007/s002249910004. Cited on page 12.

Gallin, D., 1975. *Intensional and Higher-Order Modal Logic*, volume 19 of *Mathematic Studies*. Elsevier. ISBN 0-444-11002-X. Cited on page 56.

Gardent, C. and Kallmeyer, L., 2003. Semantic construction in feature-based TAG. In *EACL 2003, Tenth Meeting of the European Chapter of the Association for Computational Linguistics*, pages 123–130. ACL Press. doi:10.3115/1067807.1067825. Cited on page 9.

Gécseg, F. and Steinby, M., 1997. Tree languages. In Rozenberg, G. and Salomaa, A., editors, *Hanbook of Formal Languages*, volume 3: Beyond Words, chapter 1. Springer. ISBN 3-540-60649-1. Cited on page 10.

Guessarian, I., 1983. Pushdown tree automata. *Theory of Computing Systems*, 16(1): 237–263. doi:10.1007/BF01744582. Cited on page 10.

Harel, D., Kozen, D., and Tiuryn, J., 2000. *Dynamic Logic*. Foundations of Computing. MIT Press. Cited on page 29.

Harrison, M.A., 1978. *Introduction to Formal Language Theory*. Series in Computer Science. Addison-Wesley. ISBN 0-201-02955-3. Cited on page 2.

Henkin, L., 1950. Completeness in the theory of types. *Journal of Symbolic Logic*, 15(2): 81–91. doi:http://dx.doi.org/10.2307/2266967. Cited on page 56.

Hidders, J., 2004. Satisfiability of XPath expressions. In Lausen, G. and Suciu, D., editors, *DBPL 2003*, volume 2921 of *Lecture Notes in Computer Science*, pages 21–36. Springer. doi:10.1007/978-3-540-24607-7_3. Cited on page 44.

Hindley, J.R., 1997. *Basic Simple Type Theory*, volume 42 of *Cambride Tracts in Theoretical Computer Science*. Cambridge University Press. ISBN 0-521-46518-4. doi:10.1017/CBO9780511608865. Cited on pages 38, 40.

Hobbs, J.R. and Shieber, S.M., 1987. An algorithm for generating quantifier scopings. *Computational Linguistics*, 13(1–2):47–63. http://aclweb.org/anthology/J87-1005.pdf. Cited on page 43.

Janssen, T.M., 1997. Compositionality. In Benthem, J.F. and ter Meulen, A., editors, *Handbook of Logic and Language*, chapter 7, pages 417–473. Elsevier. ISBN 0-444-81714-3. doi:10.1016/B978-044481714-3/50011-4. Cited on page 38.

Joshi, A.K., Levy, L.S., and Takahashi, M., 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163. doi:10.1016/S0022-0000(75)80019-5. Cited on page 6.

Joshi, A.K., 1985. Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions? In Dowty, D.R., Karttunen, L., and Zwicky, A.M., editors, *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*, chapter 6, pages 206–250. Cambridge University Press. Cited on page 5.

Joshi, A.K., Vijay-Shanker, K., and Weir, D., 1991. The convergence of mildly context-sensitive grammatical formalisms. In Sells, P., Shieber, S., and Wasow, T., editors, *Foundational Issues in Natural Language Processing*. MIT Press. http://repository.upenn.edu/cis_reports/539. Cited on page 5.

Joshi, A.K. and Schabes, Y., 1997. Tree-adjoining grammars. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages*, volume 3: Beyond Words, chapter 2, pages 69–124. Springer. ISBN 3-540-60649-1. http://www.seas.upenn.edu/∼joshi/joshi-schabes-tag-97.pdf. Cited on page 6.

Jurafsky, D. and Martin, J.H., 2009. *Speech and Language Processing*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, second edition. ISBN 978-0-13-187321-6. Cited on pages 2, 35.

Kallmeyer, L. and Romero, M., 2004. LTAG semantics with semantic unification. In Rambow, O. and Stone, M., editors, *TAG+7, Seventh International Workshop on Tree-Adjoining Grammars and Related Formalisms*, pages 155–162. http://www.cs.rutgers.edu/TAG+7/papers/kallmeyer-c.pdf. Cited on page 9.

Kanazawa, M., 2007. Parsing and generation as Datalog queries. In *ACL 2007, 45th Annual Meeting of the Association for Computational Linguistics*, pages 176–183. Annual Meeting of the Association for Computational Linguistics. http://www.aclweb.org/anthology/P07-1023. Cited on page 40.

Kanazawa, M., 2009. The pumping lemma for well-nested multiple context-free languages. In Diekert, V. and Nowotka, D., editors, *DLT 2009, 13th International Conference on Developments in Language Theory*, volume 5583 of *Lecture Notes in Computer Science*, pages 312–325. Springer. doi:10.1007/978-3-642-02737-6_25. Cited on page 15.

Kanazawa, M., 2010. Second-order abstract categorial grammars as hyperedge replacement grammars. *Journal of Logic, Language and Information*, 19(2):137–161. doi:10.1007/s10849-009-9109-6. Cited on page 42.

Kepser, S. and Mönnich, U., 2006. Closure properties of linear context-free tree languages with an application to optimality theory. *Theoretical Computer Science*, 354(1):82–97. doi:10.1016/j.tcs.2005.11.024. Cited on page 16.

Kepser, S., 2004. Querying linguistic treebanks with monadic second-order logic in linear time. *Journal of Logic, Language and Information*, 13(4):457–470. doi:10.1007/s10849-004-2116-8. Cited on page 25.

Kepser, S. and Rogers, J., 2011. The equivalence of tree adjoining grammars and monadic linear context-free tree grammars. *Journal of Logic, Language and Information*, 20(3):361–384. doi:10.1007/s10849-011-9134-0. Cited on pages 12, 15.

Koller, A., Niehren, J., and Treinen, R., 2001. Dominance constraints: Algorithms and complexity. In Moortgat, M., editor, *LACL 1998, Third International Conference on Logical Aspects of Computational Linguistics*, volume 2014 of *Lecture Notes in Computer Science*, pages 106–125. doi:10.1007/3-540-45738-0_7. Cited on page 44.

Koller, A., Niehren, J., and Thater, S., 2003. Bridging the gap between underspecification formalisms: Hole semantics as dominance constraints. In *EACL 2003, 10th Meeting of the European Chapter of the Association for Computational Linguistics*, pages 195–202. ACL Press. doi:10.3115/1067807.1067834. Cited on page 46.

Koller, A., Regneri, M., and Thater, S., 2008. Regular tree grammars as a formalism for scope underspecification. In *ACL 2008:HLT, 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 218–226. ACL Press. http://www.aclweb.org/anthology/P08-1026. Cited on page 47.

Kracht, M., 1995. Syntactic codes and grammar refinement. *Journal of Logic, Language and Information*, 4(1):41–60. doi:10.1007/BF01048404. Cited on page 28.

Kroch, A.S. and Joshi, A.K., 1985. The linguistic relevance of tree adjoining grammars. Technical Report MS-CIS-85-16, University of Pennsylvania, Department of Computer and Information Science. http://repository.upenn.edu/cis_reports/671/. Cited on page 8.

Kroch, A.S. and Santorini, B., 1991. The derived constituent structure of the West Germanic verb-raising construction. In Freidin, R., editor, *Principles and Parameters in Comparative Grammar*, chapter 10, pages 269–338. MIT Press. Cited on page 5.

Kupfermana, O., Pnueli, A., and Vardi, M.Y. Once and for all. *Journal of Computer and System Sciences*. doi:10.1016/j.jcss.2011.08.006. In Press, Corrected Proof. Cited on page 49.

Lai, C. and Bird, S., 2010. Querying linguistic trees. *Journal of Logic, Language and Information*, 19(1):53–73. doi:10.1007/s10849-009-9086-9. Cited on page 29.

Maneth, S., Perst, T., and Seidl, H., 2007. Exact XML type checking in polynomial time. In Schwentick, T. and Suciu, D., editors, *ICDT 2007, 11th International Conference on Database Theory*, volume 4353 of *Lecture Notes in Computer Science*, pages 254–268. Springer. doi:10.1007/11965893_18. Cited on page 17.

Manning, C.D. and Schütze, H., 1999. *Foundations of Statistical Natural Language Processing*. MIT Press. ISBN 978-0-262-13360-9. Cited on page 2.

Marx, M., 2005. Conditional XPath. *ACM Transactions on Database Systems*, 30(4):929–959. doi:10.1145/1114244.1114247. Cited on pages 28, 34.

Marx, M. and de Rijke, M., 2005. Semantic characterizations of navigational XPath. *SIGMOD Record*, 34(2):41–46. doi:10.1145/1083784.1083792. Cited on page 28.

Maryns, H. and Kepser, S., 2009. MonaSearch - a tool for querying linguistic treebanks. In Van Eynde, F., Frank, A., De Smedt, K., and van Noord, G., editors, *TLT 7, 7th International Workshop on Treebanks and Linguistic Theories*, pages 29–40. http://lotos.library.uu.nl/publish/articles/000260/bookpart.pdf. Cited on page 25.

Meyer, A., 1975. Weak monadic second order theory of successor is not elementary-recursive. In Parikh, R., editor, *Logic Colloquium '75*, volume 453 of *Lecture Notes in Mathematics*, pages 132–154. Springer. doi:10.1007/BFb0064872. Cited on page 27.

Mönnich, U., 1997. Adjunction as substitution: An algebraic formulation of regular, context-free and tree adjoining languages. In *FG '97, Second Conference on Formal Grammar*. arXiv:cmp-lg/9707012. Cited on page 12.

Montague, R., 1970. Universal grammar. *Theoria*, 36(3):373–398. doi:10.1111/j.1755-2567.1970.tb00434.x. Cited on page 38.

Montague, R., 1973. The proper treatment of quantification in ordinary English. In Hintikka, J., Moravcsik, J., and Suppes, P., editors, *Approaches to Natural Language*, pages 221–242. Reidel. https://www.blackwellpublishing.com/content/BPL_Images/Content_store/Sample_chapter/0631215417/Portner.pdf. Cited on pages 38, 40.

Muskens, R., 2007. Intensional models for the theory of types. *Journal of Symbolic Logic*, 72(1):98–118. doi:10.2178/jsl/1174668386. Cited on page 56.

Muskens, R., 2011. Type-logical semantics. In Craig, E., editor, *Routledge Encyclopedia of Philosophy Online*. Routledge. http://let.uvt.nl/general/people/rmuskens/pubs/rep.pdf. (to appear). Cited on page 56.

Palm, A., 1999. Propositional tense logic of finite trees. http://www.phil.uni-passau.de/linguistik/palm/papers/mol99.pdf. Cited on page 28.

Parikh, R.J., 1966. On context-free languages. *Journal of the ACM*, 13(4):570–581. doi:10.1145/321356.321364. Cited on page 5.

Parsons, T., 1990. *Events in the Semantics of English: A Study in Subatomic Semantics*, volume 19 of *Current Studies in Linguistics*. MIT Press. ISBN 0-262016120-6. http://www.humnet.ucla.edu/humnet/phil/faculty/tparsons/EventSemantics/download.htm. Cited on page 37.

Partee, B.H., ter Meulen, A.G., and Wall, R.E. *Mathematical Methods in Linguistics*, volume 30 of *Studies in Linguistics and Philosophy*. Springer. Cited on page 38.

Poesio, M., 1994. Ambiguity, underspecification and discourse interpretation. In *IWCS-1, First International Workshop on Computational Semantics*. Cited on page 43.

Pullum, G.K. and Scholz, B.C., 2001. On the distinction between model-theoretic and generative-enumerative syntactic frameworks. In de Groote, P., Morrill, G., and Retoré, C., editors, *LACL 2001, 4th International Conference on Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Computer Science*, pages 17–43. Springer. doi:10.1007/3-540-48199-0_2. Cited on page 21.

Rabin, M.O., 1969. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35. doi:10.2307/1995086. Cited on page 27.

Rogers, J., 1996. A model-theoretic framework for theories of syntax. In *ACL '96, 34th Annual Meeting of the Association for Computational Linguistics*, pages 10–16. ACL Press. doi:10.3115/981863.981865. Cited on page 27.

Rogers, J., 1998. *A Descriptive Approach to Language-Based Complexity*. Studies in Logic, Language, and Information. CSLI Publications. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.912&rep=rep1&type=pdf. Cited on page 25.

Rogers, J., 2003. wMSO theories as grammar formalisms. *Theoretical Computer Science*, 293(2):291–320. doi:10.1016/S0304-3975(01)00349-8. Cited on page 27.

Rounds, W.C., 1970. Mappings and grammars on trees. *Theory of Computing Systems*, 4 (3):257–287. doi:10.1007/BF01695769. Cited on pages 10, 15.

Sakarovitch, J., 2009. *Elements of Automata Theory*. Cambridge University Press. ISBN 978-0-521-84425-3. Translated from *Éléments de théorie des automates*, Vuibert, 2003. Cited on page 2.

Schabes, Y. and Shieber, S.M., 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124. http://www.aclweb.org/anthology/J94-1004. Cited on page 9.

Schwichtenberg, H., 1991. An upper bound for reduction sequences in the typed λ-calculus. *Archive for Mathematical Logic*, 30(5–6):405–408. doi:10.1007/BF01621476. Cited on page 40.

Seki, H., Matsumura, T., Fujii, M., and Kasami, T., 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229. doi:10.1016/0304-3975(91)90374-B. Cited on page 5.

Seki, H. and Kato, Y., 2008. On the generative power of multiple context-free grammars and macro grammars. *IEICE Transactions on Information and Systems*, E91-D(2):209–221. doi:10.1093/ietisy/e91-d.2.209. Cited on page 15.

Shieber, S.M., 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343. doi:10.1007/BF00630917. Cited on page 5.

Shieber, S.M., 2006. Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In *EACL 2006, 11th Meeting of the European Chapter of the Association for Computational Linguistics*. ACL Press. http://www.aclweb.org/anthology/E06-1048. Cited on page 18.

Statman, R., 1979. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9(1):73–81. doi:10.1016/0304-3975(79)90007-0. Cited on page 40.

Thatcher, J.W. and Wright, J.B., 1968. Generalized finite automata theory with an application to a decision problem of second-order logic. *Theory of Computing Systems*, 2(1): 57–81. doi:10.1007/BF01691346. Cited on page 27.

Vardi, M., 1998. Reasoning about the past with two-way automata. In Larsen, K.G., Skyum, S., and Winskel, G., editors, *ICALP '98, 25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer. doi:10.1007/BFb0055090. Cited on page 32.

Weir, D.J., 1992. Linear context-free rewriting systems and deterministic tree-walking transducers. In *ACL '92, 30th Annual Meeting of the Association for Computational Linguistics*, pages 136–143. ACL Press. doi:10.3115/981967.981985. Cited on page 5.

Weyer, M., 2002. Decidability of S1S and S2S. In Grädel, E., Thomas, W., and Wilke, T., editors, *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*, chapter 12, pages 207–230. Springer. doi:10.1007/3-540-36387-4_12. Cited on page 27.

XTAG Research Group, 2001. A lexicalized tree adjoining grammar for English. Technical Report IRCS-01-03, University of Pennsylvania, Institute for Research in Cognitive Science. http://www.cis.upenn.edu/~xtag/. Cited on page 8.