Notes on Computational Aspects of Syntax

Sylvain Schmitz
LSV, ENS Cachan & CNRS & INRIA
October 8, 2013 (r3589M)

These notes cover the first part of an introductory course on computational linguistics, also known as **MPRI 2-27-1**: *Logical and computational structures for linguistic modeling*. The course is subdivided into two parts: the first, which is the topic of these notes, is focused almost exclusively on syntax, while the second part, taught this year by Philippe de Groote, covers semantics and discourse representation. Among the prerequisites to the course are

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

These notes are based on numerous articles—and I have tried my best to provide stable hyperlinks to online versions in the references—, and on the excellent material of Benoît Crabbé and Philippe de Groote who teach this course with me in round-robin fashion.

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

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

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

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

Contents

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 $R_1 \mathbin{;} R_2 = \{(a,c) \mid \exists b \in B, (a,b) \in R_1 \wedge (b,c) \in R_2\}$. 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 $R^{i+1} = R \mathbin{;} R^i$.

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 $L^* = \bigcup_i L^i$.

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.

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.

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

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 $t : \{0, \dots, k\}^* \to \Sigma$ 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 $\{0, \dots, k\}$; 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 $\xRightarrow{r} = \{(C[t\{x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n\}], C[t'\{x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n\}]) \mid C \in \mathcal{C}(\Sigma), t_1, \dots, t_n \in T(\Sigma)\}$. We write $\xRightarrow{r_1 r_2}$ for $\xRightarrow{r_1} ⨾ \xRightarrow{r_2}$, and $\xRightarrow{R}$ for $\bigcup_{r \in R} \xRightarrow{r}$.

Chapter 1

Introduction

If linguistics is about the description and understanding of human language, a computational linguist thrives in developing *computational* models of language. By computational, we mean models that are not only mathematically elegant, but also amenable to an algorithmic treatment. Such models are certainly useful for practical applications in *natural language processing*, which range from text mining, question answering, and text summarization, to automated translation.

In spite of the large impact such technologies have on our lives, the case of computational linguistics is even stronger. Consider that human brains have limited capacity for holding language information (think for instance of dictionaries and common turns of phrase), and that being able to learn, understand, and produce a potentially unbounded number of utterances, we need to rely on some form or other of computation—quite an efficient one at that if you think about it.

A computational model, rather than a "mere" mathematical one, also allows for *experimentation*, and thus validation or refinement of the model. For example, a theoretical linguist might test her predictions about which sentences are grammatical by parsing large corpora of presumably correct text—does the model undergenerate?—, or about the syntax rules of a particular phenomenon by generating random sentences and checking against over-generation. As another example, a psycholinguist might try to match some measured degree of linguistic difficulty of sentences with various aspects of the model: frequency of the lexemes and of the syntactic rules, type and size of the involved rules, degree of ambiguity, etc.

1.1 Levels of Description

Language models are classically divided into several layers, first some specific to speech processing: **phonetics** and **phonology**, then more generally applicable: **morphology**, **syntax**, **semantics**, and **pragmatics**. This forms a *pipeline*, that inputs utterances in oral or written form and outputs meaning representations in context.

1.1.1 From Text to Meaning

Let us give a quick overview of the phases from text to meaning.

Morphology The purpose of morphology is to describe the mechanisms that underlie the formation of words. Intuitively, one can recognize the existence of a relation between the words *sings* and *singing*, and further find that the same relation holds between *dances* and *dancing*. Beyond the simple enumeration of words, we usually want to retrieve some linguistic information that will be helpful for further processing: are we dealing with a noun or a verb (its **category**)? Is it plural or singular (its **number**)? What is its **part-of-speech** (**POS**) tag? Modeling morphology often involves (probabilistic) word automata and transducers.

This process is quite prone to ambiguity: in the sentence

> Gator attacks puzzle experts

is *attacks* a verb in third person singular (VBZ) or a plural noun (NNS)? Is *puzzle* a verb (VB) or a noun (NN)? Should crossword experts avoid Florida?

Syntax deals with the structure of sentences: how do we combine words into phrases and sentences?

Constituents and Dependencies. Two main types of analysis are used by syntacticians: one as **constituents**, where the sentence is split into phrases, themselves further split until we reach the word level, as in

> [[She] [watches [a bird]]]

Such a constituent analysis can also be represented as a tree, as on the left of Figure 1.1. Here we introduced part-of-speech tags and syntactic categories to label the internal nodes: for instance, *VBZ* stands for a verb conjugated in present third person, *NP* stands for a noun phrase, and *VP* for a verb phrase.

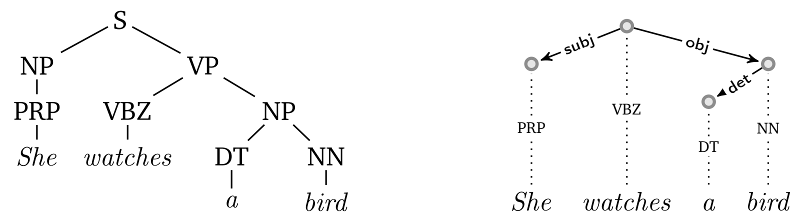

Figure 1.1: Constituent (on the left) and dependency (on the right) analyses.

An alternative analysis, illustrated on the right of Figure 1.1, rather exhibits the **dependencies** between words in the sentence: its **head** is the verb *watches*, with two dependents *She* and *bird*, which play the roles of subject and object respectively. In turn, *bird* governs its determiner *a*. Again, additional labels can decorate the nodes and relations in dependency structures, as shown in Figure 1.1.

Ambiguity. The following sentence is a classical example of a syntactic ambiguity, illustrated by the two derivation trees of Figure 1.2:

> She watches a man with a telescope.

This is called a *PP attachment* ambiguity: who exactly is using a telescope?

Semantics studies meaning. We often use logical languages to describe meaning, like the following (guarded) first-order sentence for "Every man loves a woman":

$$\forall x.\text{man}(x) \supset \exists y.\text{love}(x, y) \land \text{woman}(y)$$

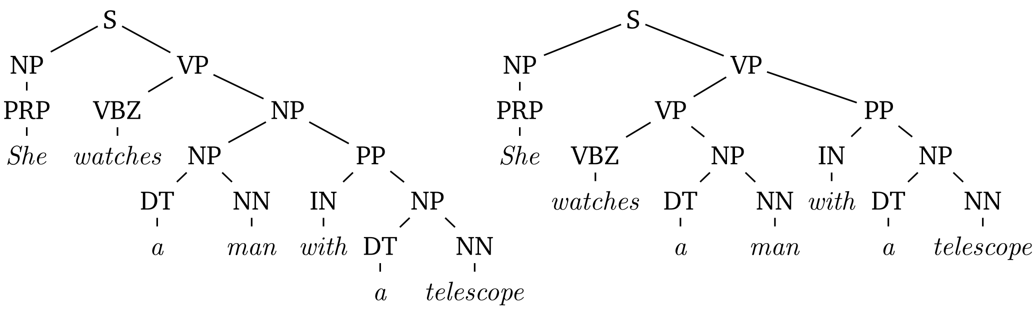

Figure 1.2: An ambiguous sentence.

or the description logic statement

$$\mathsf{Man} \sqsubseteq \exists \mathsf{love}.\mathsf{Woman}\ .$$

Ambiguity is of course present as in every aspect of language: for instance, scope ambiguities, as in this alternate reading of "Every man loves a woman"

$$\exists y.\text{woman}(y) \wedge \forall x.\text{man}(x) \supset \text{love}(x, y)$$

where there exists one particular woman loved by every man in the world.

More difficulties arise when we attempt to build meaning representations *compositionally*, based on syntactic structures, and when intensional phenomena must be modeled. The solutions often mix higher-order logics with possible-worlds semantics and modalities.

Pragmatics considers the ways in which meaning is affected by the *context* of a sentence: it includes the study of **discourse** and of **referential expressions**.

As usual, the models have to account for massive ambiguity, as in this **anaphora** resolution:

> Mary asks Eve about her father

where *her* might refer to *Mary* or *Eve*; only the context of the sentence will allow to disambiguate.

1.1.2 Ambiguity at Every Turn

The above succinct presentation should convince the reader that ambiguity permeates every layer of the linguistic entreprise. To better emphasize the importance of ambiguity, let us look at experimental results in real-world syntax grammars: Martin et al. (1987) presents a typical sentence found in a corpus, which when generalized to arbitrary lengths n, exhibits a number of parses related to the Catalan numbers $C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$. In more recent experiments with treebank-induced grammars, Moore (2004) reports an average number of 7.2×10^{27} different derivations for sentences of 5.7 words on average. The rationale behind these staggering levels of ambiguity is that any formal grammar that accounts for a realistic part of natural language, must allow for so many constructions, that it also yields an enormous number of different analyses: robustness of the model comes at a steep price in ambiguity.

The practical answer to this issue is to refine the models with **weights**, allowing to attach a grammaticality estimation to each structure. Those weights are typically probabilities inferred from frequencies found in large corpora. Stochastic methods are now ubiquitous in natural language processing (Manning and Schütze, 1999), and no purely symbolic model is able to compete with statistical models on practical benchmarks.

1.1.3 Romantics and Revolutionaries

Read (Pereira, 2000; Steedman, 2011).

1.2 Models of Syntax

To conclude this already long introduction, here is a short presentation of the kinds of models employed for describing syntax. Not every one will be covered in class, but there are pointers to the relevant literature.

For each of the two kinds of analyses, using constituents or dependencies, three different flavors of models can be distinguished: *generative* models, which construct structures through rewriting systems; *model-theoretic* approaches rather describe structures in a logical language and allow any model of a formula as an answer; *proof-theoretic* techniques establish the grammaticality of sentences through a proof in some formal deduction system. Finally, *stochastic* methods might be mixed with any of the previous frameworks (Manning and Schütze, 1999). This gives rise to twelve combinations—which should however not be distinguished too strictly, as their borders are often quite blurry.

1.2.1 Constituent Syntax

Generative Syntax The formal description of morpho-syntactic phenomena through rewriting systems can be traced back to 350BC and the Sanskrit grammar of Pāṇini, the *Aṣṭādhyāyī*. This large grammar employs *contextual* rewriting rules like

$$A \to B \mathbin{/} C\_\_D \tag{1.1}$$

for "rewrite A to B in the context $C\_\_D$", i.e. the rewrite rule

$$CAD \to CBD\ . \tag{1.2}$$

The grammar already features auxiliary symbols (like the labels on the inner nodes of Figure 1.1), and this type of formal systems is therefore already Turing-complete.

The adoption of **phrase-structure grammars** to derive constituent structures stems mostly from Chomsky's *Three Models for the Description of Language* (1956), which considers the suitability of finite automata, context-free grammars, and transformational grammars for syntactic modeling.

Readers with a computer science background are likely to be rather familiar with context-free grammars from a compilers or formal languages course; it is quite interesting to see that the equivalent BNF notation (Backus, 1959) was developed at roughly the same time to specify the syntax of ALGOL 60 (Ginsburg and Rice, 1962). The focus in linguistics applications is however on *trees*, for which tree languages provide a more appropriate framework (Comon et al., 2007).

Model-Theoretic Syntax Because the focus of linguistic models of syntax is on trees, there is an alternative way of understanding a disjunction of context-free production rules

$$A \to BC \mid DE\ . \tag{1.3}$$

It posits that in a valid tree, a node labeled by A should feature two children, labeled either by B and C or by D and E. In first-order logic, assuming $A, B, \dots$ to be predicates and using $\downarrow$ and $\to$ to denote the *child* and *right sibling* relations, this could be expressed as

$$\begin{aligned} \forall x.A(x) \supset \exists y.\exists z.x \downarrow y \wedge x \downarrow z \wedge y \to z \wedge \big((B(y) \wedge C(z)) \vee (D(y) \wedge E(z))\big) \\ \wedge \forall c.x \downarrow c \supset c = y \vee c = z\ . \end{aligned} \tag{1.4}$$

A constituent tree is valid if it satisfies the constraints stated by the grammar, and a language is the set of models, in a logical sense, of the grammar. See the survey by Pullum (2007) on the early developments of the model-theoretic approach.

Of course, the logical language of context-free rules is rather limited, and more expressive logics can be employed: we will consider **monadic second-order logic** and **propositional dynamic logic** in Chapter 3.

Proof-Theoretic Syntax Yet another way of viewing a context-free rule like (1.3) is as a deduction rules

$$A(xy) :- B(x), C(y). \tag{1.5}$$

$$A(xy) :- D(x), E(y). \tag{1.6}$$

(in Prolog-like syntax). Here the variables x and y range over finite strings, and a sentence w is accepted by the grammar if the judgement $S(w)$ (for "$w \in L(S)$") can be derived using the rules

$$\frac{B_1(u_1)\ \dots\ B_m(u_m)}{A(u_1 \cdots u_m)}\{A(x_1 \cdots x_m) :- B_1(x_1), \dots, B_m(x_m) \tag{1.7}$$

where $u_1, \dots, u_m$ are finite strings.

The interest of this proof-theoretic view is that it is readily generalizable beyond context-free grammars, for instance by removing the restriction to monadic predicates, as in **multiple context-free grammars** (Seki et al., 1991). It also encourages annotations of proofs with terms (as with the Curry-Howard isomorphism) to construct a semantic representation of the sentence, and thus provides an elegant syntax/semantics interface.

1.2.2 Dependency Syntax

Dependency analyses take their roots in the work of Tesnière, and are especially well-suited to language with "relaxed" word order, where **discontinuities** come handy (Mel'čuk, 1988, e.g. Meaning-Text Theory for Czech). It also turns out that several of the best statistical parsing systems today rely on dependencies rather than constituents.

Generative Syntax If we look at the dependency structure of Figure 1.1, we can observe that it can be encoded through rewrite rules of the form

$$h \to L * R \tag{1.8}$$

where L is the list of left dependents and R that of right dependents of the head word h, and $*$ marks the position of this word: more concretely, the rules

$$\text{VBZ} \to \text{PRP} * \text{NN} \tag{1.9}$$
$$\text{PRP} \to * \tag{1.10}$$
$$\text{NN} \to \text{DT} * \tag{1.11}$$

would allow to generate the dependency tree on the right of Figure 1.1. This general idea has been put forward by Gaifman (1965) and Hays (1964).

Conversely, given a constituent tree like the one on the left of Figure 1.1, a dependency tree can be recovered by identifying the head of each phrase as in Figure 1.3. Applying this transformation to a context-free grammar results in a **head lexicalized** grammar, which is a fairly common idea in statistical parsing (e.g. Charniak, 1997; Collins, 2003).

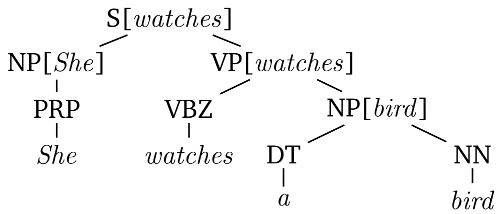

Figure 1.3: A head lexicalized constituent tree.

Model-Theoretic Syntax As with constituency analysis, dependency structures can be described in a model-theoretic framework. Here I do not know much work on the subject, besides a constraint-solving approach for a (positive existential) logic: the **topological dependency grammars** of Duchier and Debusmann (2001), along with related formalisms.

Proof-Theoretic Syntax Regarding the proof-theoretic take on dependency syntax, there is a very rich literature on **categorial grammar**. In the basic system of Bar-Hillel (1953), **categories** are built using left and right quotients over a finite set of symbols A:

$$\gamma ::= A \mid \gamma\backslash\gamma \mid \gamma/\gamma \qquad \text{(categories)}$$

The proof system then features three deduction rules: one that looks up the possible categories associated to a word in a finite lexicon

$$\frac{}{w \vdash \gamma}\,\text{Lexicon}$$

and two rules to eliminate the $\backslash$ and $/$ connectors:

$$\frac{w_1 \vdash \gamma_1 \qquad w_2 \vdash \gamma_1\backslash\gamma_2}{w_1 \cdot w_2 \vdash \gamma_2}\,\backslash E$$

$$\frac{w_1 \vdash \gamma_2/\gamma_1 \qquad w_2 \vdash \gamma_1}{w_1 \cdot w_2 \vdash \gamma_2}\,/E$$

For instance, the dependencies from the right of Figure 1.1 can be described in a lexicon over $A \stackrel{\text{def}}{=} \{s, n, d\}$ by

$$\text{She} \vdash n \qquad \text{watches} \vdash (n\backslash s)/n \qquad \text{a} \vdash d \qquad \text{bird} \vdash d\backslash n$$

with a proof

$$\frac{\text{She} \vdash n \qquad \dfrac{\text{watches} \vdash (s\backslash n)/n \qquad \dfrac{\text{a} \vdash d \qquad \text{bird} \vdash d\backslash n}{\text{a bird} \vdash n}\,\backslash E}{\text{watches a bird} \vdash n\backslash s}\,/E}{\text{She watches a bird} \vdash s}\,\backslash E$$

By adding *introduction* rules to this proof system, Lambek (1958) has defined the **Lambek calculus**, which can be viewed as a non-commutative variant of linear logic (e.g. Troelstra, 1992). As with constituency analyses, one of the interests of proof-theoretic methods is that it provides an elegant way of building compositional semantics interpretations.

1.3 Further Reading

Interested readers will find a good general textbook on natural language processing by Jurafsky and Martin (2009). The present notes have a strong bias towards logical formalisms, but this is hardly representative of the general field of natural language processing. 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/`.

The more mathematics-oriented linguistics community is scattered around several sub-communities, each with its meeting. Let me mention two special interest groups of the ACL: *SIGMoL* on "Mathematics of Language" and *SIGParse* on "natural language parsing".

Chapter 2

Context-Free Syntax

Syntax deals with how words are arranged into sentences. An important body of linguistics proposes **constituent** analyses for sentences, where for instance

> Those torn books are completely worthless.

can be decomposed into a **noun phrase** *those torn books* and a **verb phrase** *are completely worthless*. These two constituants can be recursively decomposed until we reach the individual words, effectively describing a tree:

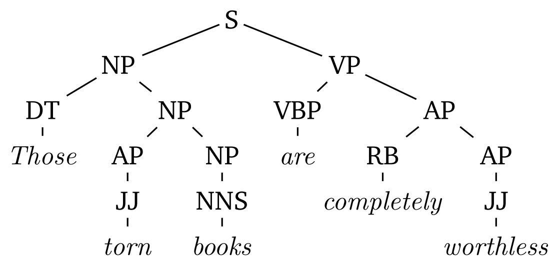

Figure 2.1: A context-free derivation tree.

You have probably recognized in this example a derivation tree for a **context-free grammar** (CFG). Context-free grammars, proposed by Chomsky (1956), constitute the primary example of a *generative* formalism for syntax, which we take to include all string- or term-rewriting systems.

2.1 Grammars

Definition 2.1 (Phrase-Structured Grammars)**.** A **phrase-structured grammar** is a tuple $\mathcal{G} = \langle N, \Sigma, P, S \rangle$ where N is a finite *nonterminal alphabet*, Σ a finite *terminal alphabet* disjoint from N, $V = N \uplus \Sigma$ the *vocabulary*, $P \subseteq V^* \times V^*$ a finite set of rewrite rules or *productions*, and S a *start symbol* or *axiom* in N.

A phrase-structure grammar defines a string rewrite system over V. Strings α in V^* s.t. $S \Rightarrow^\star \alpha$ are called **sentential forms**, whereas strings w in Σ^* s.t. $S \Rightarrow^\star w$ are called **sentences**. The *language* of $\mathcal{G}$ is its set of sentences, i.e.

$$L(\mathcal{G}) = L_{\mathcal{G}}(S) \qquad\qquad L_{\mathcal{G}}(A) = \{w \in \Sigma^* \mid A \Rightarrow^\star w\} .$$

Different restrictions on the shape of productions lead to different classes of grammars; we will not recall the entire **Chomsky hierarchy** (Chomsky, 1959) here, but only define **context-free grammars** (aka **type 2 grammars**) as phrase-structured grammars with $P \subseteq N \times V^*$.

Example 2.2. The derivation tree of Figure 2.1 corresponds to the context-free grammar with

$$N = \{\text{S}, \text{NP}, \text{AP}, \text{VP}, \text{DT}, \text{JJ}, \text{NNS}, \text{VBP}, \text{RB}\}\ ,$$
$$\Sigma = \{\textit{those}, \textit{torn}, \textit{books}, \textit{are}, \textit{completely}, \textit{worthless}\}\ ,$$

$$\begin{aligned} P = \{ \quad & \text{S} \to \text{NP VP}, & \text{NP} \to \text{DT NP} \mid \text{AP NP} \mid \text{NNS}, \\ & \text{VP} \to \text{VBP AP}, & \text{AP} \to \text{RB AP} \mid \text{JJ}, \\ & \text{DT} \to \textit{Those}, & \text{JJ} \to \textit{torn} \mid \textit{worthless}, \\ & \text{NNS} \to \textit{books}, & \text{VBP} \to \textit{are}, \\ & \text{RB} \to \textit{completely}\}\ , \\ S = \text{S}\ . \end{aligned}$$

Note that it also generates sentences such as *Those books are torn.* or *Those completely worthless books are completely completely torn.* Also note that this grammar describes part-of-speech tagging information, based on the Penn TreeBank tagset (Santorini, 1990). A different formalization could set $\Sigma = \{\text{DT}, \text{JJ}, \text{NNS}, \text{VBP}, \text{RB}\}$ and delegate the POS tagging issues to an external device.

2.1.1 The Parsing Problem

Context-free grammars enjoy a number of nice computational properties:

- both their **uniform membership** problem—i.e. given $\langle \mathcal{G}, w\rangle$ does $w \in L(\mathcal{G})$—and their **emptiness** problem—i.e. given $\langle \mathcal{G}\rangle$ does $L(\mathcal{G}) = \emptyset$—are PTIME-complete (Jones and Laaser, 1976),
- their **fixed grammar membership** problem—i.e. for a fixed $\mathcal{G}$, given $\langle w\rangle$ does $w \in L(\mathcal{G})$—is by very definition LOGCFL-complete (Sudborough, 1978),
- they have a natural notion of **derivation trees**, which constitute a local **regular tree language** (Thatcher, 1967).

The monograph of Grune and Jacobs (2007) is a rather exhaustive resource on context-free parsing.

Recall that our motivation in context-free grammars lies in their ability to model constituency through their *derivation trees*. Thus much of the linguistic interest in context-free grammars revolves around a variant of the membership problem: given $\langle \mathcal{G}, w\rangle$, compute the set of derivation trees of $\mathcal{G}$ that yield w—the **parsing problem**.

The asymptotically best parsing algorithm is that of Valiant (1975), with complexity $\Theta(B(|w|))$ where $B(n)$ is the complexity of n-dimensional boolean matrix multiplication, currently known to be in $O(n^{2.3727})$ (Williams, 2012). A converse reduction from boolean matrix multiplication to context-free parsing by Lee (2002) shows that any improvement for one problem would also yield one for the other.

Parsing Techniques Outside the realm of deterministic parsing algorithms for restricted classes of CFGs, for instance for LL(k) or LR(k) grammars (Knuth, 1965; Kurki-Suonio, 1969; Rosenkrantz and Stearns, 1970)—which are often studied in computer science curricula—, there exists quite a variety of methods for *general* context-free parsing. Possibly the best known of these is the CKY algorithm (Cocke and Schwartz, 1970; Kasami, 1965; Younger, 1967), which in its most basic form works with complexity $O(|\mathcal{G}|\,|w|^3)$ on grammars in Chomsky normal form. Both the CKY algorithm(s) and the advanced methods (Earley, 1970; Lang, 1974; Graham et al., 1980; Tomita, 1986; Billot and Lang, 1989) can be seen as refinement of the construction first described by Bar-Hillel et al. (1961) to prove the closure of context-free languages under intersection with recognizable sets, which will be central in these notes on syntax.

Ambiguity and Parse Forests The key issue in general parsing and parsing for natural language applications is grammatical **ambiguity**: the existence of several derivation trees sharing the same string yield.

The following sentence is a classical example of a PP attachment ambiguity, illustrated by the two derivation trees of Figure 2.2:

She watches a man with a telescope.

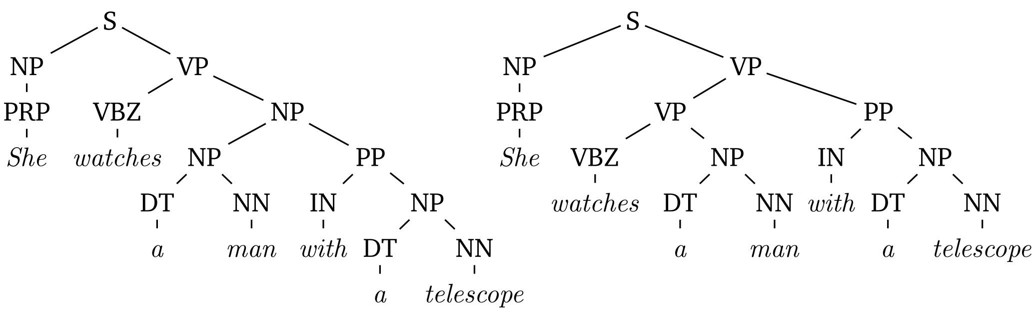

Figure 2.2: An ambiguous sentence.

In the case of a **cyclic** CFG, with a nonterminal A verifying $A \Rightarrow^+ A$, the number of different derivation trees for a single sentence can be infinite. For **acyclic** CFGs, it is finite but might be exponential in the length of the grammar and sentence:

Example 2.3 (Wich, 2005)**.** The grammar with rules

$$S \to a\,S \mid a\,A \mid \varepsilon, \qquad\qquad A \to a\,S \mid a\,A \mid \varepsilon$$

has exactly 2^n different derivation trees for the sentence a^n.

Such an explosive behavior is not unrealistic for CFGs in natural languages: Moore (2004) reports an average number of 7.2×10^{27} different derivations for sentences of 5.7 words on average, using a CFG extracted from the Penn Treebank.

The solution in order to retain polynomial complexities is to represent all these derivation trees as the language of a finite tree automaton (or using a CFG). This is sometimes called a **shared forest** representation.

2.1.2 *Background:* Tree Automata

Because our focus in linguistics analyses is on *trees*, context-free grammars are mostly useful as a means to define tree languages. Let us first recall basic definitions on regular tree languages.

Definition 2.4 (Finite Tree Automata)**.** A **finite tree automaton** (NTA) is a tuple $\mathcal{A} = \langle Q, \mathcal{F}, \delta, F \rangle$ where Q is a finite set of *states*, $\mathcal{F}$ a *ranked alphabet*, δ a finite *transition relation* in $\bigcup_n Q \times \mathcal{F}_n \times Q^n$, and $I \subseteq Q$ a set of *initial states*.

See Comon et al. *(2007).*

The semantics of a NTA can be defined by term rewrite systems over $\mathcal{F} = Q \uplus \mathcal{F}$ where the states in Q have arity 0: either **bottom-up**:

$$R_B = \{a^{(n)}(q_1^{(0)}, \ldots, q_n^{(0)}) \to q^{(0)} \mid (q, a^{(n)}, q_1, \ldots, q_n) \in \delta\}$$
$$L(\mathcal{A}) = \{t \in T(\mathcal{F}) \mid \exists q \in I, t \xrightarrow{R_B}{}^{\star}\, q\}\ ,$$

or **top-down**:

$$R_T = \{q^{(0)} \to a^{(n)}(q_1^{(0)}, \ldots, q_n^{(0)}) \mid (q, a^{(n)}, q_1, \ldots, q_n) \in \delta\}$$
$$L(\mathcal{A}) = \{t \in T(\mathcal{F}) \mid \exists q \in I, q \overset{R_T}{\Longrightarrow}{}^\star t\} .$$

A tree language $L \subseteq T(\mathcal{F})$ is **regular** if there exists an NTA $\mathcal{A}$ such that $L = L(\mathcal{A})$.

Example 2.5. The 2^n derivation trees for a^n in the grammar of Example 2.3 are generated by the $O(n)$-sized automaton $\langle\{q_S, q_a, q_\varepsilon, q_1, \ldots, q_n\}, \{S, A, a, \varepsilon\}, \delta, \{q_S\}\rangle$ with rules

$$\delta = \{(q_S, S^{(2)}, q_a, q_1), (q_a, a^{(0)}), (q_\varepsilon, \varepsilon^{(0)})\}$$
$$\cup\ \{(q_i, X, q_a, q_{i+1}) \mid 1 \leq i < n, X \in \{S^{(2)}, A^{(2)}\}\}$$
$$\cup\ \{(q_n, X, q_\varepsilon) \mid X \in \{S^{(1)}, A^{(1)}\}\} .$$

It is rather easy to define the set of derivation trees of a CFG through an NTA. The only slightly annoying point is that nonterminals in a CFG do not have a fixed arity; for instance if $A \to BC \mid a$ are two productions, then an A-labeled node in a derivation tree might have two children B and C or a single child a. This motivates the notation $A^{(r)}$ for an A-labeled node with rank r.

Definition 2.6 (Derived Tree Language)**.** Let $\mathcal{G} = \langle N, \Sigma, P, S\rangle$ be a context-free grammar and let m be its maximal right-hand side length. Its **derived tree language** $T(\mathcal{G})$ is defined as the language of the NTA $\mathcal{A} = \langle V \uplus \{\varepsilon\}, \mathcal{F}, \delta, \{S\}\rangle$, where

$$\mathcal{F} \overset{\text{def}}{=} \{a^{(0)} \mid a \in \Sigma\} \cup \{\varepsilon^{(0)}\} \cup \{A^{(1)} \mid A \to \varepsilon \in P\}$$
$$\cup \{A^{(m)} \mid m > 0 \text{ and } \exists A \to X_1 \cdots X_m \in P \text{ with } \forall i.X_i \in V\}$$
$$\delta \overset{\text{def}}{=} \{(A, A^{(m)}, X_1, \ldots, X_m) \mid m > 0 \wedge A \to X_1 \cdots X_m \in P\}$$
$$\cup\ \{(A, A^{(1)}, \varepsilon) \mid A \to \varepsilon \in P\}$$
$$\cup\ \{(a, a^{(0)}) \mid a \in \Sigma\}$$
$$\cup\ \{(\varepsilon, \varepsilon^{(0)})\} .$$

The class of derived tree languages of context-free grammars is a strict subclass of the class of regular tree languages.

($\ast\ast$) **Exercise 2.1** (Local Tree Languages)**.** Let $\mathcal{F}$ be a ranked alphabet and t a term of $T(\mathcal{F})$. We denote by $\mathsf{r}(t)$ the root symbol of t and by $\mathsf{b}(t)$ the set of *local branches* of t, defined inductively by

$$\mathsf{r}(a^{(0)}) \overset{\text{def}}{=} a^{(0)} \qquad \mathsf{b}(a^{(0)}) \overset{\text{def}}{=} \emptyset$$
$$\mathsf{r}(f^{(n)}(t_1, \ldots, t_n)) \overset{\text{def}}{=} f^{(n)} \quad \mathsf{b}(f^{(n)}(t_1, \ldots, t_n)) \overset{\text{def}}{=} \{f^{(n)}(\mathsf{r}(t_1), \ldots, \mathsf{r}(t_n))\} \cup \bigcup_{i=1}^{n} \mathsf{b}(t_i) .$$

For instance $\mathsf{b}(\ f(g(a), f(a,b))\ ) = \{f(g,f), g(a), f(a,b)\}$.

A tree language $L \subseteq T(\mathcal{F})$ is **local** if and only if there exist two sets $R \subseteq \mathcal{F}$ of root symbols and $B \subseteq \mathsf{b}(T(\mathcal{F}))$ of local branches, such that $t \in L$ iff $\mathsf{r}(t) \in R$ and $\mathsf{b}(t) \subseteq B$. Let

$$L(R, B) = \{t \in T(\mathcal{F}) \mid \mathsf{r}(t) \in R \text{ and } \mathsf{b}(t) \subseteq B\} ;$$

then a tree language L is local if and only if $L = L(\mathsf{r}(L), \mathsf{b}(L))$.

1. Show that $\{ f(g(a), g(b)) \}$ is not a local tree language.

2. Show that any local tree language is the language of some NTA.

3. Show that a tree language included in $T(\mathcal{F})$ is local with $R \subseteq \mathcal{F}_{>0}$ if and only if it is the derived tree language of some CFG.

4. Show that any regular tree language is the homomorphic image of a local tree language by an *alphabetic tree morphism*, i.e. the application of a relabeling to the tree nodes.

5. Given a tree language L, let $\mathrm{Yield}(L) \stackrel{\text{def}}{=} \bigcup_{t\in L} \mathrm{Yield}(t)$ and define inductively $\mathrm{Yield}(a^{(0)}) \stackrel{\text{def}}{=} a$ and $\mathrm{Yield}(f^{(r)}(t_1, \ldots, t_r) \stackrel{\text{def}}{=} \mathrm{Yield}(t_1) \cdots \mathrm{Yield}(t_r)$. Show that, if L is a regular tree language, then $\mathrm{Yield}(L)$ is a context-free language.

2.2 Tabular Parsing

We briefly survey the principles of general context-free parsing using **dynamic** or **tabular** algorithms. For more details, see the survey by Nederhof and Satta (2004).

2.2.1 Parsing as Intersection

The basic construction underlying all the tabular parsing algorithms is the *intersection* grammar of Bar-Hillel et al. (1961). It consists in an intersection between an $(|w|+1)$-sized automaton with language $\{w\}$ and the CFG under consideration. The intersection approach is moreover quite convenient if several input strings are possible, for instance if the input of the parser is provided by a speech recognition system.

A landmark paper on the importance of the construction of Bar-Hillel et al. *(1961) for parsing is due to Lang (1994).*

Theorem 2.7 (Bar-Hillel et al., 1961). *Let $\mathcal{G} = \langle N, \Sigma, P, S\rangle$ be a CFG and $\mathcal{A} = \langle Q, \Sigma, \delta, I, F\rangle$ be a NFA. The set of derivation trees of $\mathcal{G}$ with a word of $L(\mathcal{A})$ as yield is generated by the NTA $\mathcal{T} = \langle (V \uplus \{\varepsilon\}) \times Q \times Q, \Sigma \uplus N \uplus \{\varepsilon\}, \delta', \{S\} \times I \times F\rangle$ with*

$$\begin{aligned}\delta' = \{((A, q_0, q_m), A^{(m)}, (X_1, q_0, q_1), \ldots, (X_m, q_{m-1}, q_m))&\\ \mid m \geq 1, A \to X_1 \cdots X_m \in P, q_0, q_1, \ldots, q_m \in Q\}&\\ \cup\ \{((A, q, q), A^{(1)}, (\varepsilon, q, q)) \mid A \to \varepsilon \in P, q \in Q\}&\\ \cup\ \{((\varepsilon, q, q), \varepsilon^{(0)}) \mid q \in Q\}&\\ \cup\ \{((a, q, q'), a^{(0)}) \mid (q, a, q') \in \delta\}\ .&\end{aligned}$$

The size of the resulting NTA is in $O(|\mathcal{G}| \cdot |Q|^{m+1})$ where m is the maximal arity of a nonterminal in N. We can further reduce this NTA to only keep useful states, in linear time on a RAM machine. It is also possible to determinize and minimize the resulting tree automaton.

In order to reduce the complexity of this construction to $O(|\mathcal{G}| \cdot |Q|^3)$, one can put the CFG in **quadratic form**, so that $P \subseteq N \times V^{\leq 2}$. This changes the shape of trees, and thus the linguistic analyses, but the transformation is reversible:

Lemma 2.8. *Given a CFG $\mathcal{G} = \langle \Sigma, N, P, S\rangle$, one can construct in time $O(|\mathcal{G}|)$ an equivalent CFG $\mathcal{G}' = \langle \Sigma, N', P', S\rangle$ in quadratic form s.t. $V \subseteq V'$, $L_{\mathcal{G}}(X) = L_{\mathcal{G}'}(X)$ for all X in V, and $|\mathcal{G}'| \leq 5 \cdot |\mathcal{G}|$.*

Proof. For every production $A \to X_1 \cdots X_m$ of P with $m \geq 2$, add productions

$$\begin{aligned} A &\to [X_1][X_2 \cdots X_m] \\ [X_2 \cdots X_m] &\to [X_2][X_3 \cdots X_m] \\ &\vdots \\ [X_{m-1}X_m] &\to [X_{m-1}][X_m] \end{aligned}$$

and for all $1 \leq i \leq m$

$$[X_i] \to X_i \ .$$

Thus an $(m+1)$-sized production is replaced by $m-1$ productions of size 3 and m productions of size 2, for a total less than $5m$. Formally,

$$\begin{aligned} N' &= N \cup \{[\beta] \mid \beta \in V^+ \text{ and } \exists A \in N, \alpha \in V^+, A \to \alpha\beta \in P\} \\ &\quad \cup \{[X] \mid X \in V \text{ and } \exists A \in N, \alpha, \beta \in V^*, A \to \alpha X \beta \in P\} \\ P' &= \{A \to \alpha \in P \mid |\alpha| \leq 1\} \\ &\quad \cup \{A \to [X][\beta] \mid A \to X\beta \in P, X \in V \text{ and } \beta \in V^+\} \\ &\quad \cup \{[X\beta] \to [X][\beta] \mid [X\beta] \in N', X \in V \text{ and } \beta \in V^+\} \\ &\quad \cup \{[X] \to X \mid [X] \in N' \text{ and } X \in V\} \ . \end{aligned}$$

Grammar $\mathcal{G}'$ est clearly in quadratic form with $N \subseteq N'$ and $|\mathcal{G}'| \leq 5 \cdot |\mathcal{G}|$. It remains to show equivalence, which stems from $L_{\mathcal{G}}(X) = L_{\mathcal{G}'}(X)$ for all X in V. Obviously, $L_{\mathcal{G}}(X) \subseteq L_{\mathcal{G}'}(X)$. Conversely, by induction on the length n of derivations in $\mathcal{G}'$, we prove that

$$X \Rightarrow^n_{\mathcal{G}'} w \text{ implies } X \Rightarrow^\star_{\mathcal{G}} w \quad (2.1)$$

$$[\alpha] \Rightarrow^n_{\mathcal{G}'} w \text{ implies } \alpha \Rightarrow^\star_{\mathcal{G}} w \quad (2.2)$$

for all X in V, w in Σ^*, and $[\alpha]$ in $N' \backslash N$. The base case $n = 0$ implies X in Σ and the lemma holds. Suppose it holds for all $i < n$.

From the shape of the productions in $\mathcal{G}'$, three cases can be distinguished for a derivation

$$X \Rightarrow_{\mathcal{G}'} \beta \Rightarrow^{n-1}{}_{\mathcal{G}'} w \ :$$

1. $\beta = \varepsilon$ implies immediately $X \Rightarrow^\star_{\mathcal{G}} w = \varepsilon$, or
2. $\beta = Y$ in V implies $X \Rightarrow^\star_{\mathcal{G}} w$ by induction hypothesis (2.1), or
3. $\beta = [Y][\gamma]$ with $[Y]$ and $[\gamma]$ in N' implies again $X \Rightarrow^\star_{\mathcal{G}} w$ by induction hypothesis (2.2) and context-freeness, since in that case $X \to Y\gamma$ is in P.

Similarly, a derivation

$$[\alpha] \Rightarrow_{\mathcal{G}'} \beta \Rightarrow^{n-1}_{\mathcal{G}'} w$$

implies $\alpha \Rightarrow^\star w$ by induction hypothesis (2.1) if $|\alpha| = 1$ and thus $\beta = \alpha$, or by induction hypothesis (2.2) and context-freeness if $\alpha = Y\gamma$ with Y in V and γ in V^+, and thus $\beta = [Y][\gamma]$. □

2.2.2 Parsing as Deduction

In practice, we want to perform at least some of the reduction of the tree automaton constructed by Theorem 2.7 *on the fly*, in order to avoid constructing states and transitions that will be later discarded as useless.

Bottom-Up Tabular Parsing One way is to restrict ourselves to **co-accessible** states, by which we mean states q of the NTA such that there exists at least one tree t with $t \xRightarrow{R_B}{}^\star q$. This is the principle underlying the classical CKY parsing algorithm (but here we do not require the grammar to be in Chomsky normal form).

We describe the algorithm using deduction rules (Pereira and Warren, 1983; Sikkel, 1997), which conveniently represent how new tabulated **items** can be constructed from previously computed ones: in this case, items are states (A, q, q') in $V \times Q \times Q$ of the constructed NTA. Side conditions constrain how a deduction rule can be applied.

$$\frac{(X_1, q_0, q_1), \ldots, (X_m, q_{m-1}, q_m)}{(A, q_0, q_m)} \begin{cases} m > 0,\ A \to X_1 \cdots X_m \in P \\ q_0, q_1, \ldots, q_m \in Q \end{cases} \qquad \text{(Internal)}$$

$$\frac{}{(A, q, q)} \begin{cases} A \to \varepsilon \in P \\ q \in Q \end{cases} \qquad \text{(Empty)}$$

$$\frac{}{(a, q, q')} \left\{\ (q, a, q') \in \delta \right. \qquad \text{(Leaf)}$$

The construction of the NTA proceeds by creating new states following the rules, and transitions of δ' as output to the deduction rules, i.e. an application of (Internal) outputs if $m \geq 1$ $((A, q_0, q_m), A^{(m)}, (X_1, q_0, q_1), \ldots, (X_m, q_{m-1}, q_m))$, or if $m = 0$ $((A, q_0, q_0), A^{(1)}, (\varepsilon, q_0, q_0))$, and one of (Leaf) outputs $((a, q, q'), a^{(0)})$. We only need to add states (ε, q, q) and transitions $((\varepsilon, q, q), \varepsilon^{(0)})$ for each q in Q in order to obtain the co-accessible part of the NTA of Theorem 2.7.

The algorithm performs the deduction closure of the system; the intersection itself is non-empty if an item in $\{S\} \times I \times F$ appears at some point. The complexity depends on the "free variables" in the premices of the rules and on the side constraints; here it is dominated by the (Internal) rule, with at most $|\mathcal{G}| \cdot |Q|^{m+1}$ applications.

We could similarly construct a system of **top-down** deduction rules that only construct **accessible** states of the NTA, starting from (S, q_i, q_f) with q_i in I and q_f in F, and working its way towards the leaves.

Exercise 2.2. Give the deduction rules for top-down tabular parsing. **(∗)**

Earley Parsing The algorithm of Earley (1970) uses a mix of accessibility and co-accessibility. An *Earley item* is a triple $(A \to \alpha \cdot \beta, q, q')$, q, q' in Q and $A \to \alpha\beta$ in P, constructed iff

This invariant proves the correctness of the algorithm. For a more original proof using abtract interpretation, see Cousot and Cousot (2003).

1. there exists both (i) a run of $\mathcal{A}$ starting in q and ending in q' with label v and (ii) a derivation $\alpha \Rightarrow^\star v$, and furthermore

2. there exists (i) a run in $\mathcal{A}$ from some q_i in I to q with label u and (ii) a derivation $S \underset{\text{lm}}{\Rightarrow}{}^\star uA\gamma$ for some γ in V^*.

$$\frac{}{(S \to \cdot\alpha, q_i, q_i)} \left\{ \begin{array}{l} S \to \alpha \in P \\ q_i \in I \end{array} \right. \qquad \text{(Init)}$$

$$\frac{(A \to \alpha \cdot B\alpha', q, q')}{(B \to \cdot\beta, q', q')} \left\{ \; B \to \beta \in P \right. \qquad \text{(Predict)}$$

$$\frac{(A \to \alpha \cdot a\alpha', q, q')}{(A \to \alpha a \cdot \alpha', q, q'')} \left\{ \; (q', a, q'') \in \delta \right. \qquad \text{(Scan)}$$

$$\frac{\begin{array}{c}(A \to \alpha \cdot B\alpha', q, q') \\ (B \to \beta\cdot, q', q'')\end{array}}{(A \to \alpha B \cdot \alpha', q, q'')} \qquad \text{(Complete)}$$

The intersection is non empty if an item $(S \to \alpha\cdot, q_i, q_f)$ is obtained for some q_i in I and q_f in F.

The algorithm run as a recognizer works in $O(|\mathcal{G}|^2 \cdot |Q|^3)$ regardless of the arity of symbols in $\mathcal{G}$ ((Complete) dominates this complexity), and can be further optimized to run in $O(|\mathcal{G}| \cdot |Q|^3)$, which is the object of Exercise 2.3. This cubic complexity in the size of the automaton can be understood as the effect of an on-the-fly quadratic form transformation into $\mathcal{G}' = \langle N', \Sigma, P', S' \rangle$ with

$$\begin{aligned} N' &= \{S'\} \uplus \{[A \to \alpha \cdot \beta] \mid A \to \alpha\beta \in P\} \\ P' &= \{S' \to [S \to \alpha\cdot] \mid S \to \alpha \in P\} \\ &\quad \cup \; \{[A \to \alpha B \cdot \alpha'] \to [A \to \alpha \cdot B\alpha'] \; [B \to \beta\cdot] \mid B \to \beta \in P\} \\ &\quad \cup \; \{[A \to \alpha a \cdot \alpha'] \to [A \to \alpha \cdot a\alpha'] \; a \mid a \in \Sigma\} \\ &\quad \cup \; \{[A \to \cdot\alpha'] \to \varepsilon\} \; . \end{aligned}$$

Note that the transformation yields a grammar of quadratic size, but can be modified to yield one of linear size—this is the same simple trick as that of Exercise 2.3. It is easier to output a NTA for this transformed grammar $\mathcal{G}'$:

- create state $([S \to \cdot\alpha], q_i, q_i)$ and transition $(([S \to \cdot\alpha], q_i, q_i), \varepsilon^{(0)})$ when applying (Init),
- create state $([B \to \cdot\beta], q', q')$ and transition $(([B \to \cdot\beta], q', q'), \varepsilon^{(0)})$ when applying (Predict),
- create states $([A \to \alpha a \cdot \alpha'], q, q'')$ and (a, q', q''), and transitions $(([A \to \alpha a \cdot \alpha'], q, q''), [A \to \alpha a \cdot \alpha']^{(2)}, ([A \to \alpha \cdot a\alpha'], q, q'), (a, q', q''))$ and $((a, q', q''), a^{(0)})$ when applying (Scan),
- create state $([A \to \alpha B \cdot \alpha'], q, q'')$ and transition $(([A \to \alpha B \cdot \alpha'], q, q''), [A \to \alpha B \cdot \alpha']^{(2)}, ([A \to \alpha \cdot B\alpha'], q, q'), ([B \to \beta\cdot], q', q'')$ when applying (Complete).

We finally need to add states (S', q_i, q_f) for q_i in I and q_f in F, and transitions $((S', q_i, q_f), S'^{(1)}, ([S \to \alpha\cdot], q_i, q_f)$ for each $S \to \alpha$ in P.

(∗) **Exercise 2.3.** How should the algorithm be modified in order to run in time $O(|\mathcal{G}| \cdot |Q|^3)$ instead of $O(|\mathcal{G}|^2 \cdot |Q|^3)$?

(∗) **Exercise 2.4.** Show that the Earley recognizer works in time $O(|\mathcal{G}| \cdot |Q|^2)$ if the grammar is unambiguous and the automaton deterministic.

A related open problem is whether fixed grammar membership can be solved in time $O(|w|)$ if $\mathcal{G}$ is unambiguous. See Leo (1991) for a partial answer in the case where $\mathcal{G}$ is LR-Regular.

Chapter 3

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.

3.0.1 Model-Theoretic vs. Generative

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

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

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 Figure 1.3 to lexicalize CFGs. Remember that the solution there was to explicitly annotate each nonterminal with the head information of its subtree—which is the only way to percolate the head information up the trees in a context-free grammar. On the other hand, one can write a logic formula postulating the existence of a unique head word for each node of a tree (see (3.19) and (3.20)).

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

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

> *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 probabilistic methods, because estimated probabilities are not grammaticality judgments per se, but occurrence-based (although smoothing techniques attempt to account for missing events).

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

3.0.2 Tree Structures

Before we turn to the two logical languages that we consider for model-theoretic syntax, let us introduce the structures we will consider as possible models. Because we work with constituent analyses, these will be **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

$$\downarrow \stackrel{\text{def}}{=} \{(w, wi) \mid i \in \mathbb{N} \wedge wi \in W\} \tag{3.1}$$

$$\rightarrow \stackrel{\text{def}}{=} \{(wi, w(i+1)) \mid i \in \mathbb{N} \wedge w(i+1) \in W\} \ . \tag{3.2}$$

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).*

$$\mathrm{fcns}(\varepsilon) \stackrel{\text{def}}{=} \varepsilon \qquad \mathrm{fcns}(w0) \stackrel{\text{def}}{=} \mathrm{fcns}(w)0 \qquad \mathrm{fcns}(w(i+1)) \stackrel{\text{def}}{=} \mathrm{fcns}(wi)1 \tag{3.3}$$

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

$$\mathrm{fcns}^{-1}(\varepsilon) \stackrel{\text{def}}{=} \varepsilon \quad \mathrm{fcns}^{-1}(w0) \stackrel{\text{def}}{=} \mathrm{fcns}^{-1}(w)0 \qquad \mathrm{fcns}^{-1}(w1) \stackrel{\text{def}}{=} \mathrm{fcns}^{-1}(w)+1 \tag{3.4}$$

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

$$\downarrow_0 \stackrel{\text{def}}{=} \{(w, w0) \mid w0 \in \mathrm{dom}(t)\} \tag{3.5}$$
$$\downarrow_1 \stackrel{\text{def}}{=} \{(w, w1) \mid w1 \in \mathrm{dom}(t)\} \tag{3.6}$$
$$P_a \stackrel{\text{def}}{=} \{w \in \mathrm{dom}(t) \mid a \in t(w)\}\ . \tag{3.7}$$

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

$$\mathrm{dom}(\bar{t}) \stackrel{\text{def}}{=} \{\varepsilon, 1\} \cup \{0w \mid w \in \mathrm{dom}(t)\} \cup \{0wi \mid w \in \mathrm{dom}(t) \wedge i \in \{0,1\}\} \tag{3.8}$$

and labels

$$\bar{t}(w) \stackrel{\text{def}}{=} \begin{cases} t(w') & \text{if } w = 0w' \wedge w' \in \mathrm{dom}(t) \\ \{\#\} & \text{otherwise.} \end{cases} \tag{3.9}$$

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

3.1 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

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

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

$$P_B(x) \stackrel{\text{def}}{=} \bigvee_{a \in B} P_a(x)$$

$$P_B(X) \stackrel{\text{def}}{=} \forall x.x \in X \supset P_B(x)\ .$$

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 $u, v \in \mathrm{FV}(\psi)$:

$$x\ [\mathrm{TC}_{u,v}\ \psi(u, v)]\ y \stackrel{\text{def}}{=} \forall X.(x \in X \wedge \forall uv.(u \in X \wedge \psi(u, v) \supset v \in X) \supset y \in X) \tag{3.10}$$

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

3.1.1 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

$$\text{labels}_{N,\Theta} \stackrel{\text{def}}{=} \forall x.\text{root}(x) \supset P_S(x) \;, \tag{3.11}$$

which forces the root label to be S;

$$\wedge\, \forall x.\text{internal}(x) \supset \bigvee_{a \in N \uplus \Theta} P_a(x) \wedge \bigwedge_{b \in N \uplus \Theta \setminus \{a\}} \neg P_b(x) \tag{3.12}$$

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

$$\wedge\, \forall x.\text{leaf}(x) \supset \neg P_{N \uplus \Theta}(x) \tag{3.13}$$

forbids grammatical labels on leaves;

$$\wedge\, \forall y.\text{leaf}(y) \supset \exists x.x \downarrow y \wedge P_\Theta(x) \tag{3.14}$$

expresses that leaves should have POS-labeled parents;

$$\wedge\, \forall x.\exists y_0 y_1 y_2.x \downarrow^\star y_0 \wedge y_0 \downarrow y_1 \wedge y_1 \downarrow y_2 \wedge \text{leaf}(y_2) \supset P_N(x) \tag{3.15}$$

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 (3.12), and thus cannot have a leaf as a child by (3.13);

$$\wedge\, \forall x.P_\Theta(x) \supset \neg \exists yz.y \neq z \wedge x \downarrow y \wedge x \downarrow z \tag{3.16}$$

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

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

$$\text{lexicon}_{L,\text{pos}} \stackrel{\text{def}}{=} \forall x. \bigvee_{\ell \in L} \left(P_\ell(x) \supset \text{leaf}(x) \wedge \bigwedge_{\ell' \in L \setminus \{\ell\}} \neg P_{\ell'}(x) \wedge \forall y.y \downarrow x \supset P_{\text{pos}(\ell)}(y) \right) \tag{3.17}$$

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

$$\text{grammar}_{\mathcal{G}} \stackrel{\text{def}}{=} \forall x.(P_\varepsilon(x) \supset \neg P_{N \uplus \Theta \uplus L}(x)) \wedge \bigvee_{B \in N} P_B(x) \supset \bigvee_{B \to \beta \in P} \exists y.x \downarrow_0 y \wedge \text{rule}_\beta(y) \tag{3.18}$$

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

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

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 and modifiers.

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

Head Percolation. The first step is to 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 $X_1, \dots, X_n$ 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, this unless X_1 is not among the children, in which case we should try X_2 and so on, and if X_n also fails simply choose the leftmost (resp. rightmost) child (see e.g. Collins, 1999, Appendix A). For instance, the function

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

would result in the correct head annotations in Figure 5.1.

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

$$\text{head}(x) \stackrel{\text{def}}{=} \text{leaf}(x) \vee \bigvee_{B \in N} \exists y Y. y \downarrow x \wedge \text{children}(y, Y) \wedge P_B(y) \wedge \text{head}_{h(B)}(x, Y) \tag{3.19}$$

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

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

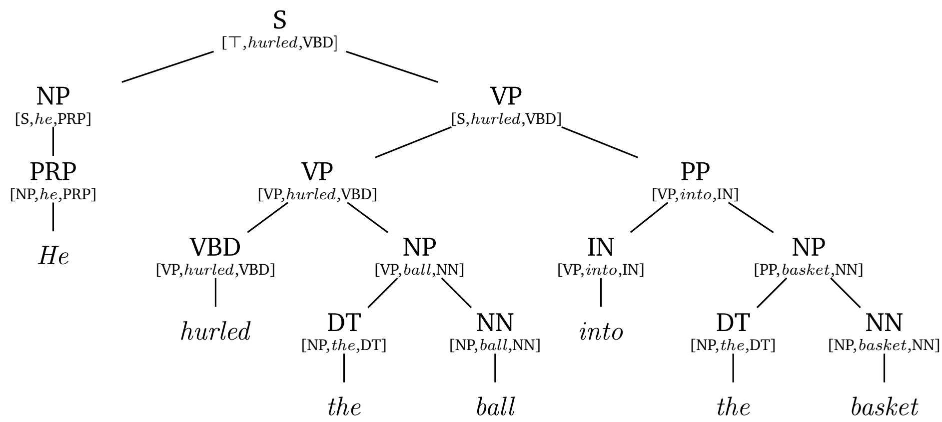

Figure 3.1: A derivation tree refined with lexical and parent information.

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

$$\text{lexicalize}(x, y) \stackrel{\text{def}}{=} \text{leaf}(y) \wedge x\, [\text{TC}_{u,v}\, u \downarrow v \wedge \text{head}(v)]\, y \; . \tag{3.20}$$

This formula recovers the lexical information in Figure 5.1.

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

$$\begin{aligned}
\text{modify}(x, y) \stackrel{\text{def}}{=} &\exists x' y' z. z \downarrow x \wedge P_{\text{RB}}(z) \wedge \text{lexicalize}(x', x) \wedge y' \downarrow x' \\
&\wedge \neg\text{lexicalize}(y', x) \wedge \text{lexicalize}(y', y)
\end{aligned} \tag{3.21}$$

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 4.7.

Exercise 3.1. Modify (3.21) to make sure that any leaf with a parent tagged by the POS RB modifies either a verb or an adjective. **(∗)**

3.1.2 wS2S

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

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

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).

$$L(\mathcal{A}_\psi) = \{\bar{t} \in T(\Sigma \uplus \{\{\#\}\}) \mid t \text{ finite} \wedge t \models \psi\}\ . \tag{3.22}$$

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

$$\psi' \stackrel{\text{def}}{=} \psi \wedge \exists x.\neg(\exists z.z \downarrow_0 x \vee z \downarrow_1 x) \wedge \neg(\exists y.x \downarrow_1 y) \tag{3.23}$$

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

$$x \downarrow y \stackrel{\text{def}}{=} \exists x_0.x \downarrow_0 x_0 \wedge (x_0 = y \vee x_0\ [\mathrm{TC}_{u,v}\ u \downarrow_1 v]\ y) \tag{3.24}$$

$$x \rightarrow y \stackrel{\text{def}}{=} x \downarrow_1 y\ . \tag{3.25}$$

The conclusion of this construction is

Theorem 3.2. *Satisfiability and model-checking for wMSO are decidable.*

(∗) **Exercise 3.2** (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.

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

$$\pi ::= \downarrow \mid \rightarrow \mid \pi^{-1} \mid \pi;\pi \mid \pi + \pi \mid \pi^* \mid \varphi? \qquad \text{(path formulæ)}$$

$$\varphi ::= a \mid \top \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\varphi \qquad \text{(node formulæ)}$$

where a ranges over A.

The **semantics** of a node formula on a tree structure $\mathfrak{M} = \langle W, \downarrow, \rightarrow, (P_a)_{a\in A}\rangle$ 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:

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

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

$$\bot \stackrel{\text{def}}{=} \neg\top \qquad \varphi_1 \wedge \varphi_2 \stackrel{\text{def}}{=} \neg(\neg\varphi_1 \vee \neg\varphi_2) \qquad [\pi]\varphi \stackrel{\text{def}}{=} \neg\langle\pi\rangle\neg\varphi \;. \tag{3.26}$$

We also define

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

Exercise 3.3 (Converses)**.** Prove the following equivalences: **(∗)**

$$(\pi_1;\pi_2)^{-1} \equiv \pi_2^{-1};\pi_1^{-1} \tag{3.27}$$
$$(\pi_1 + \pi_2)^{-1} \equiv \pi_1^{-1} + \pi_2^{-1} \tag{3.28}$$
$$(\pi^*)^{-1} \equiv (\pi^{-1})^* \tag{3.29}$$
$$(\varphi?)^{-1} \equiv \varphi? \;. \tag{3.30}$$

Exercise 3.4 (Reductions)**.** Prove the following equivalences: **(∗)**

$$\langle\pi_1;\pi_2\rangle\varphi \equiv \langle\pi_1\rangle\langle\pi_2\rangle\varphi \tag{3.31}$$
$$\langle\pi_1 + \pi_2\rangle\varphi \equiv (\langle\pi_1\rangle\varphi) \vee (\langle\pi_2\rangle\varphi) \tag{3.32}$$
$$\langle\pi^*\rangle\varphi \equiv \varphi \vee \langle\pi;\pi^*\rangle\varphi \tag{3.33}$$
$$\langle\varphi_1?\rangle\varphi_2 \equiv \varphi_1 \wedge \varphi_2 \;. \tag{3.34}$$

3.2.1 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).

3.2.2 Satisfiability

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

Theorem 3.3 (Fischer and Ladner, 1979)**.** *Satisfiability for PDL is* EXPTIME*-hard.*

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.*

As with wMSO, it is more convenient to work on binary trees t of the form $\langle \mathrm{dom}(t), \downarrow_0, \downarrow_1, (P_a)_{a\in A\uplus\{0,1\}}\rangle$ that encode our tree structures. Compared with the wMSO case, we add two atomic predicates 0 and 1 that hold on left and right children respectively. The syntax of PDL over such models simply replaces $\downarrow$ and $\rightarrow$ by $\downarrow_0$ and $\downarrow_1$; as with wMSO in Section 3.1.2 we can *interpret* these relations in PDL by

$$\downarrow \stackrel{\text{def}}{=} \downarrow_0;\downarrow_1^* \qquad\qquad \rightarrow \stackrel{\text{def}}{=} \downarrow_1 \tag{3.35}$$

and translate any PDL formula φ into a formula

$$\varphi' \stackrel{\text{def}}{=} \varphi \wedge ([\uparrow^*;\downarrow^*;\downarrow_0]0 \wedge \neg 1) \wedge ([\uparrow^*;\downarrow^*;\downarrow_1]1 \wedge \neg 0) \wedge [\uparrow^*;\mathrm{root}?;\downarrow_1]\bot \tag{3.36}$$

that checks that φ holds, that the 0 and 1 labels are correct, and verifies $\mathfrak{M}, w \models \varphi$ iff $\mathrm{fcns}(\mathfrak{M}), \mathrm{fcns}(w) \models \varphi'$. The conditions in (3.36) 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 3.3 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 (3.26), 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 3.4. *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 3.4 and their duals allow to simplify PDL formulæ into a reduced normal form we will soon see, which is a form of disjunctive normal form with atomic propositions and atomic modalities for literals. In order to obtain algorithmic complexity results, it will be important to be able to bound the number of possible such literals, which we do now.

The **Fisher-Ladner closure** of a PDL formula in normal form 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 3.5. *Let φ be a PDL formula in normal form. Its Fisher-Ladner closure is of size $|\mathrm{FL}(\varphi)| \leq |\varphi|$.*

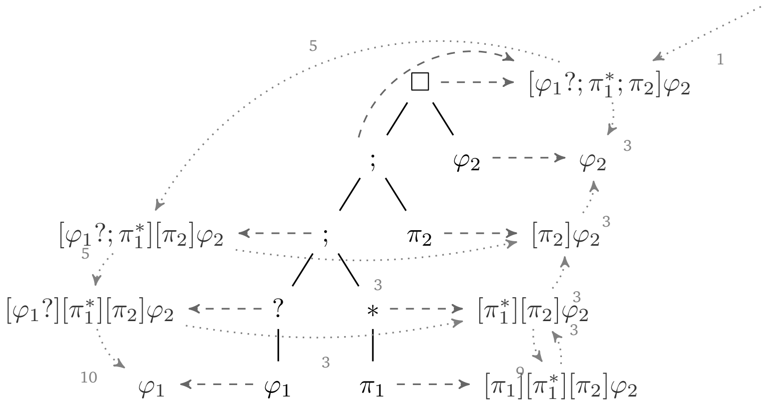

Figure 3.2: The surjection σ from positions in $\varphi \overset{\text{def}}{=} [\varphi_1?;\pi_1^*;\pi_2]\varphi_2$ to $\mathrm{FL}(\varphi)$ (dashed), and the rules used to construct $\mathrm{FL}(\varphi)$ (dotted).

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

- 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 3.2 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 3.4 and their duals, and by putting the formula into disjunctive normal form, i.e.

$$\varphi \equiv \bigvee_i \bigwedge_j \chi_{i,j} \tag{3.37}$$

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 3.6. *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 3.7. 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 $J = \{(d_0, q_0), \ldots, (d_n, q_n)\}$ s.t. $J \models f$ and for all $0 \leq i \leq n$ we have

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

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

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

Theorem 3.8 (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 3.8. We assume wlog. that $A \subseteq \mathrm{Sub}(\varphi)$. We define

$$Q \stackrel{\text{def}}{=} \mathrm{FL}(\varphi) \uplus \{q_i, q_\varphi, q_\#\}$$
$$\Sigma \stackrel{\text{def}}{=} \{\#^{(0)}, \#^{(2)}\} \cup \{a^{(2)} \mid a \subseteq A \uplus \{0, 1\}\}\ .$$

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 $a \subseteq A \uplus \{0, 1\}$

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

$$\delta(\langle\downarrow_0\rangle\varphi', a) \stackrel{\text{def}}{=} (1, \varphi') \qquad \delta([\downarrow_0]\varphi', a) \stackrel{\text{def}}{=} (1, \varphi') \vee (1, q_\#)$$
$$\delta(\langle\downarrow_1\rangle\varphi', a) \stackrel{\text{def}}{=} (2, \varphi') \qquad \delta([\downarrow_1]\varphi', a) \stackrel{\text{def}}{=} (2, \varphi') \vee (2, q_\#)$$
$$\delta(\langle\uparrow_0\rangle\varphi', a) \stackrel{\text{def}}{=} (-1, \varphi') \wedge (0, 0) \qquad \delta([\uparrow_0]\varphi', a) \stackrel{\text{def}}{=} ((-1, \varphi') \wedge (0, 0)) \vee (-1, q_\#) \vee (0, 1)$$
$$\delta(\langle\uparrow_1\rangle\varphi', a) \stackrel{\text{def}}{=} (-1, \varphi') \wedge (0, 1) \qquad \delta([\uparrow_1]\varphi', a) \stackrel{\text{def}}{=} ((-1, \varphi') \wedge (0, 1)) \vee (-1, q_\#) \vee (0, 0)$$

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

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

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 3.9. *Satisfiability of PDL can be decided in* ExpTime.

Proof sketch. Given a PDL formula φ, by Lemma 3.4 construct an equivalent formula in normal form φ' with $|\varphi'| = O(|\varphi|)$. We then construct $\mathcal{A}_{\varphi'}$ with $O(|\varphi|)$ states by Lemma 3.5 and an alphabet of size at most $2^{O(|\varphi|)}$, s.t. $\bar{t}$ is accepted by $\mathcal{A}_{\varphi'}$ iff $t, \mathrm{root} \models \varphi$. By Theorem 3.8 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. ☐

3.2.3 Expressiveness

See Cate and Segoufin (2010).

PDL can be expressed in FO[TC[1]] the **first-order logic with monadic transitive closure**. The translation can be expressed by induction, yielding formulæ $\mathrm{ST}_x(\varphi)$ with one free variable x for node formulæ and $\mathrm{ST}_{x,y}(\pi)$ with two free variables for path formulæ, such that $\mathfrak{M} \models_{x\mapsto w} \mathrm{ST}_x(\varphi)$ iff $w \in [\![\varphi]\!]_{\mathfrak{M}}$ and $\mathfrak{M} \models_{x\mapsto u, y\mapsto v} \mathrm{ST}_{x,y}(\pi)$ iff $u\ [\![\pi]\!]_{\mathfrak{M}}\ v$:

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

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

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

Chapter 4

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:

See Pullum (1986).

- 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.

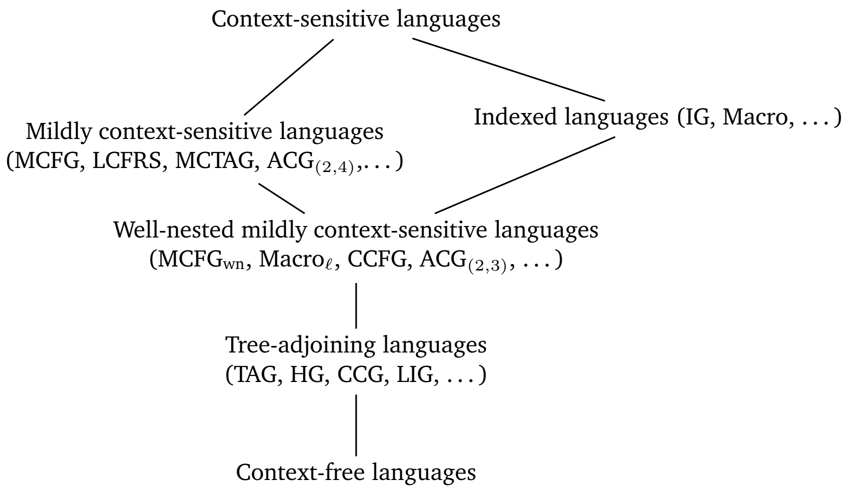

Figure 4.1: Hierarchies between context-free and full context-sensitive languages.

We will however concentrate on two strict subclasses: tree adjoining languages (TALs, Section 4.1) and well-nested MCSLs (wnMCSLs, Section 4.2); Figure 4.1 illustrates the relationship between these classes. As in Section 2.1.1 our main focus will be on the corresponding tree languages, representing linguistic constituency analyses and sentence composition.

4.1 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 4.1 (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* alphabet 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,
- $N^{\mathrm{na}} \stackrel{\text{def}}{=} \{A^{\mathrm{na}} \mid A \in N\}$ the unranked alphabet of **null adjunction** labels,
- $N_\star \stackrel{\text{def}}{=} \{A_\star \mid A \in N \cup N^{\mathrm{na}}\}$ 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 $N^{\mathrm{na}}_{>0}$ 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 $N \cup N^{\mathrm{na}}$, and we define accordingly $\mathrm{rl}(t)$ of a tree t as its *unranked* root label modulo $^{\mathrm{na}}$: $\mathrm{rl}(t) \stackrel{\text{def}}{=} A$ if there exists m in $\mathbb{N}_{>0}$, $t(\varepsilon) = A^{(m)}$ or $t(\varepsilon) = A^{\mathrm{na}(m)}$. Then

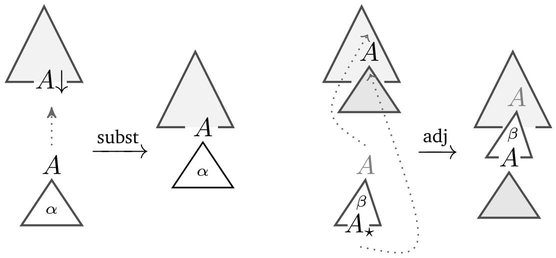

Figure 4.2: Schematics for the substitution and adjunction operations.

- $T_\alpha \subseteq T(N \cup N{\downarrow} \cup N^{\mathrm{na}} \cup \Sigma \cup \{\varepsilon^{(0)}\})$ 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 4.2)

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

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

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

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 4.2. Figure 4.3 presents a tree adjoining grammar with

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

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

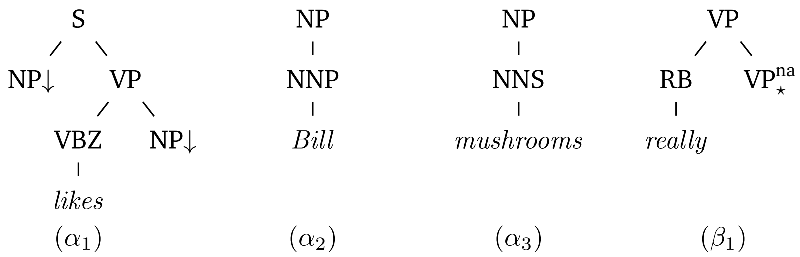

Figure 4.3: A tree adjoining grammar.

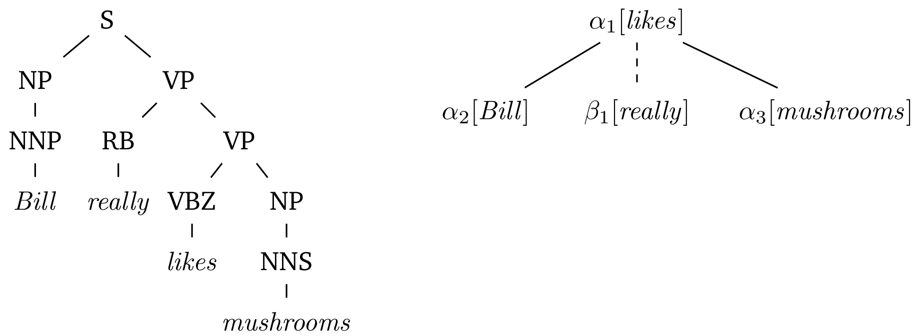

Figure 4.4: A derived tree and the corresponding derivation tree for the TAG of Example 4.2.

VP node of α_1 also yields *Bill really likes mushrooms* (see Figure 4.4) or *mushrooms really really really likes Bill*. In the TAG literature, a tree in $T(N \cup N^{\text{na}} \cup \Sigma \cup \{\varepsilon^{(0)}\})$ obtained through the substitution and adjunction operations is called a **derived tree**, while a **derivation tree** records how the rewrites took place (see Figure 4.4 for an example; children of an elementary tree are shown in addressing order, with plain lines for substitutions and dashed lines for adjunctions).

Example 4.3 (Copy Language)**.** The **copy language** $L_{\text{copy}} \stackrel{\text{def}}{=} \{ww \mid w \in \{a,b\}^*\}$ is generated by the TAG of Figure 4.5 with $N = \{S\}$, $\Sigma = \{a,b\}$, $T_\alpha = \{\alpha_\varepsilon\}$, and $T_\beta = \{\beta_a, \beta_b\}$.

(∗) **Exercise 4.1.** Give a TAG for the language $\{a^n b^m c^n d^m \mid n,m \geq 0\}$.

4.1.1 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

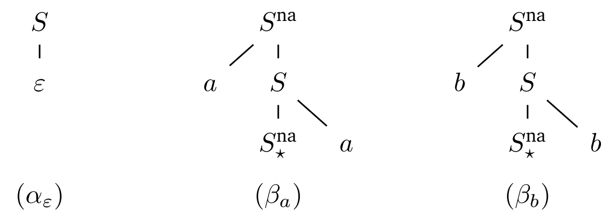

Figure 4.5: A TAG for L_{copy}.

coanchors. An anchor serves as head word for at least a part of the elementary tree, as *likes* for α_1 in Figure 4.3. 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 4.6.

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 4.3; similarly, *to think* takes a clausal S complement, as instantiated by β_2 in Figure 4.6. 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 4.4. Extend the TAG of Figure 4.3 with the trees of Figure 4.6. 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 4.3 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 4.3 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, ...).

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). See also (Kallmeyer and Kuhlmann, 2012) on the extraction of dependency analyses from TAG derivations.

Long-Distance Dependencies

Let us focus on α_4 in Figure 4.6. 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 4.7. We leave the question of syntax/semantics interfaces using derivation trees to later chapters.

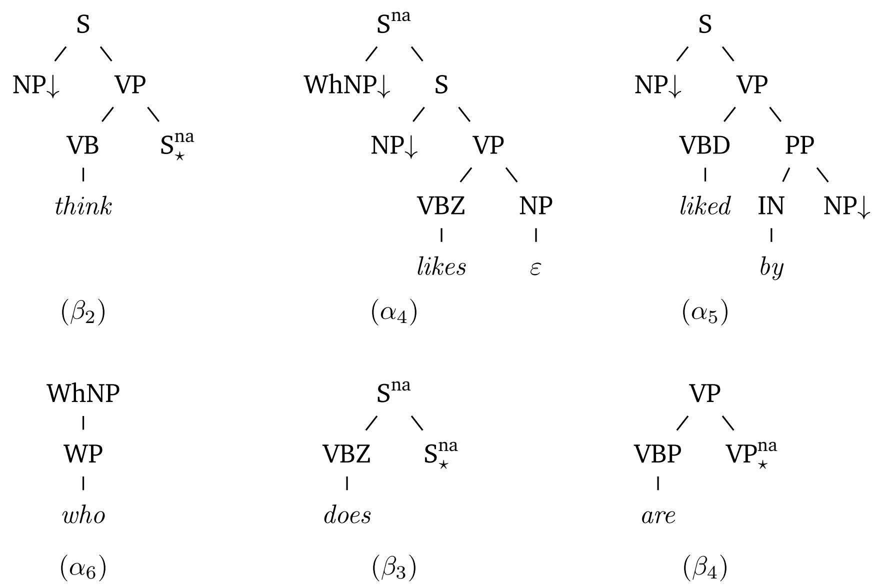

Figure 4.6: More elementary trees for the tree adjoining grammar of Example 4.2.

4.1.2 *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 4.5 (Context-Free Tree Grammars)**.** A **context-free tree grammar** (CFTG) is a tuple $\mathcal{G} = \langle N, \mathcal{F}, S, R\rangle$ consisting of a ranked *nonterminal* alphabet N, a ranked *terminal* alphabet $\mathcal{F}$, 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 $e \in T(N \cup \mathcal{F}, \mathcal{Y}_n)$ where $\mathcal{Y}$ is an infinite countable set of *parameters*. The *language* of $\mathcal{G}$ is defined as

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

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

Example 4.6 (Squares)**.** The CFTG with rules

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

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

$$\sum_{i=0}^{n-1} 2i + 1 = n + 2\sum_{i=0}^{n-1} i = n^2 \qquad (4.1)$$

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

Example 4.7 (Non-primes)**.** The CFTG with rules

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

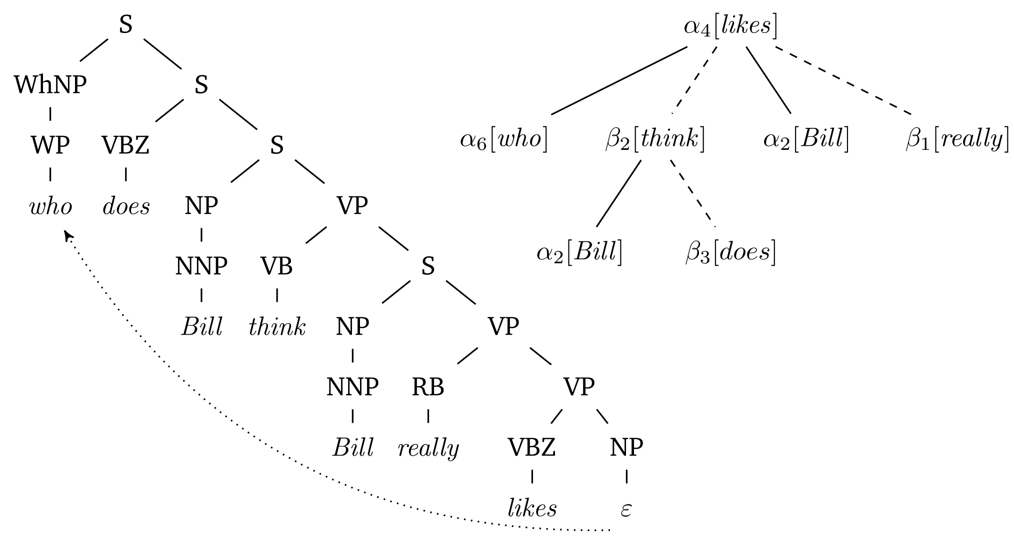

Figure 4.7: Derived and derivation trees for *Who does Bill think Bill really likes?* using the TAG of Figures 4.3 and 4.6.

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 $T(\mathcal{F})$, 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 4.2 (Powers of 2)**.** Give a CFTG with $\text{yield}(L(\mathcal{G})) = \{a^n b a^{2^n} \mid n \geq 1\}$. **(∗)**

Exercise 4.3 (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 $\mathcal{F}_n$ 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 $T(\mathcal{F})$ 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).

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

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

Example 4.8 (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 4.9 (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 4.8, the case $L_{\text{IO}}(\mathcal{G}) \subsetneq L_{\text{OI}}(\mathcal{G})$ can occur. Theorem 4.9 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 4.10. *Let $\mathcal{G} = \langle N, \mathcal{F}, S, R\rangle$. If $t_1, \ldots, t_n$ are trees in $T(N \cup \mathcal{F})$, C is a context in $\mathcal{C}^n(\mathcal{F})$, and $t = C[t_1, \ldots, t_n] \overset{R}{\Longrightarrow}{}^m t'$ for some m, then there exist $m_1, \ldots, m_n$ in $\mathbb{N}$ and $t'_1, \ldots, t'_n$ in $T(N \cup \mathcal{F})$ s.t. $t_i \overset{R}{\Longrightarrow}{}^{m_i} t'_i$, $m = m_1 + \cdots + m_n$, and $t' = C[t'_1, \ldots, t'_n]$.*

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 $t = C[t_1, \ldots, t_n] \overset{R}{\Longrightarrow}{}^m t' \overset{R}{\Longrightarrow} t''$. By induction hypothesis, we find $m_1, \ldots, m_n$ and $t'_1, \ldots, t'_n$ with $t_i \overset{R}{\Longrightarrow}{}^{m_i} t'_i$, $m = \sum_{i=1}^{n} m_i$, and $t' = C[t'_1, \ldots, t'_n] \overset{R}{\Longrightarrow} t''$. Since $C \in \mathcal{C}^n(\mathcal{F})$ 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. $t'' = C[t'_1, \ldots, t''_i, \ldots, t'_n]$. □

In contrast with Theorem 4.9, 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).

4.1.3 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 $\mathcal{G}' = \langle N', \mathcal{F}, S{\downarrow}, R \cup R'\rangle$ with

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

where $\tau : T(\Delta\cup\{\Box\}) \to T(\Delta'\cup\{\Box\})$ for $\Delta \stackrel{\text{def}}{=} N{\downarrow}\cup N^{\text{na}}_{>0}\cup N\cup\Sigma_0$ and $\Delta' \stackrel{\text{def}}{=} N'\cup\mathcal{F}$ is a tree homomorphism generated by

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

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

Example 4.11. Consider again the TAG of Figure 4.5 for the copy language: we obtain $\mathcal{G}' = \langle N', \mathcal{F}, S{\downarrow}, R\cup R'\rangle$ with $N' = \{S{\downarrow}, \bar{S}\}$, $\mathcal{F} = \{S, a, b, \varepsilon\}$, and rules

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

Proposition 4.12. $L_T(\mathcal{G}) = L(\mathcal{G}')$.

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

Claim 4.12.1. For all trees t in $T(\Delta)$, $t \xRightarrow{R_\mathcal{G}}{}^\star t'$ implies t' is in $T(\Delta)$ and $\tau(t) \xRightarrow{R}{}^\star \tau(t')$.

Proof of Claim 4.12.1. 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 $C[A{\downarrow}] \xRightarrow{R_\mathcal{G}} C[\alpha]$ occurs iff α is in T_α with $\text{rl}(\alpha) = A$, which implies $\tau(C[A{\downarrow}]) = \tau(C)[\tau(A{\downarrow})] = \tau(C)[A{\downarrow}] \xRightarrow{R} \tau(C)[\tau(\alpha)] = \tau(C[\alpha])$.

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

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

The case of a tree $\beta[A^{\text{na}}_\star]$ is similar. [4.12.1]

Claim 4.12.2. If t is a tree in $T(N^{\text{na}}\cup\mathcal{F})$, then there exists a derivation $\tau(t) \xRightarrow{R'}{}^\star h(t)$ in $\mathcal{G}'$.

Proof of Claim 4.12.2. We proceed by induction on t:

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

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

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

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

The case of a tree rooted by a in $\Sigma \cup \{\varepsilon\}$ is trivial. [4.12.2]

For the main proof: Let t be a tree in $L_T(\mathcal{G})$; there exist t' in $T(N^{\text{na}} \cup \mathcal{F})$ and α in T_α with $\text{rl}(\alpha) = S$ 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 4.12.1, and then $\tau(t') \overset{R'}{\Longrightarrow}{}^{\star}\ t$ removes all its nonterminals according to Claim 4.12.2. □

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

Proof of Claim 4.12.3. 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

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

We see immediately that $[\![\mathcal{T}]\!](t) = \tau^{-1}(t)$ for all t in L. [4.12.3]

Thanks to Claim 4.12.3, we can use τ^{-1} in our proofs. We obtain claims mirroring Claim 4.12.1 and Claim 4.12.2 using the same types of arguments:

Claim 4.12.4. 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 4.12.5. If t is a tree in $L \cap T(\bar{N} \cup \mathcal{F})$, t' a tree in $T(\mathcal{F})$, 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 $t \in T(\mathcal{F})$ 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 $\text{rl}(\alpha) = S$ and t' in $L \cap T(\bar{N} \cup \mathcal{F})$ (i.e. t' does not contain any symbol from $N{\downarrow}$). By Claim 4.12.4, $\alpha \overset{R_{\mathcal{G}}}{\Longrightarrow}{}^{\star}\ t'$ and by Claim 4.12.5 $h(t') = \tau^{-1}(t)$. Since t belongs to $T(\mathcal{F})$, $\tau^{-1}(t) = t$, 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 4.4 (Non-Strict TAGs)**.** Definition 4.1 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
$$S((A(a,\Box))^* \cdot b, (A(\Box,a))^* \cdot b)\ . \tag{4.2}$$
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:
$$\{A(a), B(a)\} \tag{4.3}$$
$$\{a\} \tag{4.4}$$
Note that allowing distinct foot and root labels in auxiliary trees is useless for these examples.

4.2 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 4.1. This provides characterizations both in terms of

See (Kuhlmann, 2013) for related definitions in terms of dependency syntax.

- **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.

4.2.1 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).

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:

See Kepser and Mönnich (2006).

Proposition 4.13. *Let $\mathcal{G} = \langle N, \mathcal{F}, S, R\rangle$ 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 4.9, we can assume this derivation to be OI. Let us pick the last non-IO step within this OI derivation:

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

using some rule $r_A : A^{(n)}(y_1, \dots, y_n) \to e_A$, where an e_i contains a nonterminal. By Lemma 4.10, 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:

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

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:

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

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 4.13 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 $\mathcal{G} = \langle N, \mathcal{F}, S, R\rangle$: the semantics of a nonterminal $A^{(n)}$ can be recast as a subset of the relation $[\![A^{(n)}]\!] \subseteq (T(\mathcal{F}))^{n+1}$:

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

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

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

The consequence of this definition is

$$L_{\text{IO}}(\mathcal{G}) = \llbracket S^{(0)} \rrbracket\ .$$

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

4.2.2 Parsing as Intersection

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

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 4.14), the second is closure under intersection with regular sets (Proposition 4.15). We actually prove these results for IO derivations in CFTGs rather than for linear CFTGs solely.

Proposition 4.14 (Emptiness)**.** *Given a CFTG $\mathcal{G}$, one can decide whether $L_{\text{IO}}(\mathcal{G}) = \emptyset$ in $O(|\mathcal{G}|)$.*

Proof sketch. Given $\mathcal{G} = \langle N, \mathcal{F}, S, R\rangle$, 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

$$\begin{aligned} P' \stackrel{\text{def}}{=} & \{A \to e \mid A^{(m)}(y_1,\ldots,y_m) \to e \in R\} && \text{(rules)}\\ & \cup \{a^{(m)}(e_1,\ldots,e_m) \to e_1 \cdots e_m \mid a \in \mathcal{F} \cup \mathcal{Y}\} && (\mathcal{F}\text{- or } \mathcal{Y}\text{-labeled positions})\\ & \cup \{A^{(m)}(e_1,\ldots,e_m) \to Ae_1 \cdots e_m\}\ . && (N\text{-labeled positions}) \end{aligned}$$

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 $\llbracket X \rrbracket \neq \emptyset$ to the reader. □

Proposition 4.15 (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 $\mathcal{G} = \langle N, \mathcal{F}, S, R\rangle$ and $\mathcal{A} = \langle Q, \mathcal{F}, \delta, F\rangle$. We define $\mathcal{G}' = \langle N', \mathcal{F}, S', R'\rangle$ 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:

$$\begin{aligned} \theta_{q_0 q_1 \cdots q_m}(a^{(m)}(e_1, \ldots, e_m)) &\stackrel{\text{def}}{=} \{a^{(m)}(e'_1, \ldots, e'_m) \mid \exists (q_0, a, q'_1, \ldots, q'_m) \in \delta, \\ &\qquad\qquad \forall 1 \le i \le m, e'_i \in \theta_{q'_i q_1 \cdots q_m}(e_i)\} \\ \theta_{q_0 q_1 \cdots q_m}(B^{(m)}(e_1, \ldots, e_m)) &\stackrel{\text{def}}{=} \{\langle B, q_0, q'_1, \ldots, q'_m\rangle(e'_1, \ldots, e'_m) \mid \forall 1 \le i \le m, \\ &\qquad\qquad q'_i \in Q \wedge e'_i \in \theta_{q'_i q_1 \cdots q_m}(e_i)\} \\ \theta_{q_i q_1 \cdots q_m}(y_i) &\stackrel{\text{def}}{=} \{y_i\} . \end{aligned}$$

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 $[\![q_1 \cdots q_m]\!] \subseteq (T(\mathcal{F}))^m$ as the cartesian product of the sets $[\![q_i]\!] \stackrel{\text{def}}{=} \{t \in T(\mathcal{F}) \mid q_i \stackrel{R_T}{\Longrightarrow}{}^\star\, t\}$. We can check that, for all $m \le M$, 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. □

In order to use these results for *string* parsing, we merely need to construct, given a string w and a ranked alphabet $\mathcal{F}$, 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*.

Chapter 5

Probabilistic Syntax

Probabilistic approaches to syntax and parsing are helpful on (at least) two different grounds:

1. the first is *ambiguity* issues; in order to choose between the various possible parses of a sentence, like the PP attachment ambiguity of Figure 2.2, we can resort to several techniques: heuristics, semantic processing, and what interests us in this section, probabilities learned from a corpus.
2. the second is *robustness* of the parser: rather than discarding a sentence as agrammatical or returning a partial parse, a probabilistic parser with smoothed probabilities will still propose several parses, with low probabilities.

Smoothing and Hidden Variables The relevance of statistical models of syntax has been a subject of heated discussion: Chomsky (1957) famously wrote

See Pereira (2000).

> (1) Colorless green ideas sleep furiously.
> (2) Furiously sleep ideas green colorless.
>
> ... It is fair to assume that neither sentence (1) nor (2) (nor indeed any parts of these sentences) has ever occurred in an English discourse. Hence, in any statistical model for grammaticalness, these sentences will be rules out on identical grounds as equally 'remote' from English. Yet (1), though nonsensical, is grammatical, while (2) is not.

The main issue with this statement is the 'in any statistical model' bit, which actually assumes a rather impoverished statistical model, unable to assign a non-null probability to unseen events. The current statistical models are quite capable of handling them, mainly through two techniques:

smoothing which consists in assigning some weight to unseen events (and renormalizing probabilities). A very basic smoothing technique is called **Laplace smoothing**, and simply adds 1 to the counts of occurrence of any unseen event. Using such a technique over the *Google books corpus* from 1800 to 1954, Norvig trains a model where (1) is about 10^4 times more probable than (2).

hidden variables where the model assumes the existence of **hidden variables** responsible for the observations. Pereira trains a model using the **expectation maximization** method on newspaper text, where (1) is about 2.10^5 times more probable than (2).

We will *not* go much into the details of learning algorithms (which is the subject of another course at MPRI), but rather look at the algorithmics of weighted models.

5.1 Weighted and Probabilistic CFGs

The models we consider are actually *weighted* models defined over semirings, for which probabilities are only one particular case.

5.1.1 *Background:* Semirings and Formal Power Series

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 $\mathbb{K}$.

A semiring is **commutative** if $\langle \mathbb{K}, \odot, 1_\mathbb{K}\rangle$ is a commutative monoid.
Among the main semirings of interest are the

boolean semiring $\langle \mathbb{B}, \vee, \wedge, 0, 1\rangle$ where $\mathbb{B} = \{0, 1\}$,

probabilistic semiring $\langle \mathbb{R}_{\geq 0}, +, \cdot, 0, 1\rangle$ where $\mathbb{R}_{\geq 0} = [0, +\infty)$ is the set of non-negative reals (sometimes restricted to $[0, 1]$ when in presence of a probability distribution),

tropical semiring $\langle \mathbb{R}_{\geq 0} \uplus \{+\infty\}, \min, +, +\infty, 0\rangle$,

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. This is the only non-commutative example here.

Weighted Automata

A finite **weighted automaton** (or **automaton with multiplicity**, or $\mathbb{K}$**-automaton**) in a semiring $\mathbb{K}$ is a generalization of a finite automaton: $\mathcal{A} = \langle Q, \Sigma, \mathbb{K}, \delta, I, F\rangle$ where $\delta \subseteq Q \times \Sigma \times \mathbb{K} \times Q$ is a weighted transition relation, and I and F are maps from Q to $\mathbb{K}$ instead of subsets of Q. A run

$$\rho = q_0 \xrightarrow{a_1, k_1} q_1 \xrightarrow{a_2, k_2} q_2 \cdots q_{n-1} \xrightarrow{a_n, k_n} q_n$$

defines a **monomial** $[\![\rho]\!] = kw$ where $w = a_1 \cdots a_n$ is the **word label** of ρ and $k = I(q_0)k_1 \cdots k_n F(q_n)$ its **multiplicity**. The behavior $[\![\mathcal{A}]\!]$ of $\mathcal{A}$ is the sum of the monomials for all runs in $\mathcal{A}$: it is a formal power series on Σ^* with coefficients in $\mathbb{K}$, i.e. a map $\Sigma^* \to \mathbb{K}$. The **coefficient** of a word w in $[\![\mathcal{A}]\!]$ is denoted $\langle [\![\mathcal{A}]\!], w\rangle$ and is the sum of the multiplicities of all the runs with w for word label:

$$\langle [\![\mathcal{A}]\!], a_1 \cdots a_n\rangle = \sum_{q_0 \xrightarrow{a_1, k_1} q_1 \cdots q_{n-1} \xrightarrow{a_n, k_n} q_n} I(q_0)k_1 \cdots k_n F(q_n)\ .$$

A matrix $\mathbb{K}$**-representation** for $\mathcal{A}$ is $\langle I, \mu, F\rangle$, where I is seen as a row matrix in $\mathbb{K}^{1\times Q}$, the morphism $\mu : \Sigma^* \to \mathbb{K}^{Q\times Q}$ is defined by $\mu(a)(q, q') = k$ iff $(q, a, k, q') \in \delta$, and F is seen as a column matrix in $\mathbb{K}^{Q\times 1}$. Then

$$\langle [\![\mathcal{A}]\!], w\rangle = I\mu(w)F \ .$$

There is a notion of $\mathbb{K}$***-rational*** *series, which coincide with the* $\mathbb{K}$*-recognizable ones (Schützenberger, 1961).*

A series is $\mathbb{K}$**-recognizable** if there exists a $\mathbb{K}$-representation for it.

The **support** of a series $[\![\mathcal{A}]\!]$ is $\operatorname{supp}([\![\mathcal{A}]\!]) = \{w \in \Sigma^* \mid \langle [\![\mathcal{A}]\!], w\rangle \neq 0_{\mathbb{K}}\}$. This corresponds to the language of the underlying automaton of $\mathcal{A}$.

Exercise 5.1 (Hadamard Product)**.** Let $\mathbb{K}$ be a commutative semiring. Show that $\mathbb{K}$-recognizable series are closed under product: given two $\mathbb{K}$-recognizable series s and s', show that $s \odot s'$ with $\langle s \odot s', w\rangle = \langle s, w\rangle \odot \langle s', w\rangle$ for all w in Σ^* is $\mathbb{K}$-recognizable. What can you tell about the support of $s \odot s'$? **(∗∗)**

5.1.2 Weighted Grammars

Definition 5.1 (Weighted Context-Free Grammars)**.** A **weighted context-free grammar** $\mathcal{G} = \langle N, \Sigma, P, S, \rho\rangle$ over a semiring $\mathbb{K}$ ($\mathbb{K}$-CFG) is a context-free grammar $\langle N, \Sigma, P, S\rangle$ along with a mapping $\rho : P \to \mathbb{K}$, which is extended in a natural way into a morphism from $\langle P^*, \cdot, \varepsilon\rangle$ to $\langle \mathbb{K}, \odot, 1_{\mathbb{K}}\rangle$. The *weight* of a leftmost derivation $\alpha \overset{\pi}{\underset{\text{lm}}{\Longrightarrow}}{}^{\star}\ \beta$ is then defined as $\rho(\pi)$. It would be natural to define the weight of a sentential form γ as the sum of the weights $\rho(\pi)$ with $S \overset{\pi}{\underset{\text{lm}}{\Longrightarrow}}{}^{\star}\ \gamma$, i.e.

The presentation of this section follows closely Nederhof and Satta (2008).

Considering leftmost *derivations is only important if* $\langle \mathbb{K}, \odot, 1_{\mathbb{K}}\rangle$ *is non-commutative.*

$$\rho(\gamma) = \sum_{\pi \in P^*, S \overset{\pi}{\underset{\text{lm}}{\Longrightarrow}}{}^{\star} \gamma} \rho(\pi) \ .$$

However this sum might be infinite in general, and lead to weights outside $\mathbb{K}$. We therefore restrict ourselves to **acyclic** $\mathbb{K}$-CFGs, such that $A \Rightarrow^+ A$ is impossible for all A in N, ensuring that there exist only finitely many derivations for each sentential form. An acyclic $\mathbb{K}$-CFG $\mathcal{G}$ then defines a formal series $[\![\mathcal{G}]\!]$ with coefficients $\langle [\![\mathcal{G}]\!], w\rangle = \rho(w)$.

A $\mathbb{K}$-CFG $\mathcal{G}$ is **reduced** if each nonterminal A in $N\backslash\{S\}$ is **useful**, which means that there exist π_1, π_2 in P^*, u, v in Σ^*, and γ in V^* such that $S \overset{\pi_1}{\underset{\text{lm}}{\Longrightarrow}}{}^{\star}\ uA\gamma \overset{\pi_2}{\underset{\text{lm}}{\Longrightarrow}}{}^{\star}\ uv$ and $\rho(\pi_1\pi_2) \neq 0_{\mathbb{K}}$.

A $\mathbb{R}_{\geq 0}$-CFG $\mathcal{G} = \langle N, \Sigma, P, S, \rho\rangle$ is a **probabilistic context-free grammar** (PCFG) if ρ is a mapping $P \to [0, 1]$.

Exercise 5.2. A **right linear** $\mathbb{K}$-CFG $\mathcal{G}$ has its productions in $N \times (\Sigma^* \cup \Sigma^* \cdot N)$. Show that a series s over Σ is $\mathbb{K}$-recognizable iff there exists an acyclic right linear $\mathbb{K}$-CFG for it. **(∗∗)**

5.1.3 Probabilistic Grammars

Definition 5.1 makes no provision on the kind of probability distributions defined by a PCFG. We define here two such conditions, properness and consistency (Booth and Thompson, 1973).

A PCFG is **proper** if for all A in N,

$$\sum_{p = A \to \alpha \in P} \rho(p) = 1 \ , \tag{5.1}$$

i.e. ρ can be seen as a mapping from N to $\mathrm{Disc}(\{p \in P \mid p = A \to \alpha\})$, where $\mathrm{Disc}(S)$ denotes the set of discrete distributions over S, i.e. $\{p : S \to [0,1] \mid \sum_{e \in S} p(e) = 1\}$.

Partition Functions

The **partition function** Z maps each nonterminal A to

$$Z(A) = \sum_{w \in \Sigma^*, A \overset{\pi}{\underset{\mathrm{lm}}{\Longrightarrow}}{}^{\star} w} \rho(\pi)\ . \tag{5.2}$$

A PCFG is **convergent** if

$$Z(S) < \infty\ ; \tag{5.3}$$

in particular, it is **consistent** if

$$Z(S) = 1\ , \tag{5.4}$$

i.e. ρ defines a discrete probability distribution over the derivations of terminal strings. The intuition behind proper inconsistent grammars is that some of the probability mass is lost into infinite, non-terminating derivations.

Equation (5.2) can be decomposed using commutativity of multiplication into

$$Z(A) = \sum_{p = A \to \alpha \in P} \rho(p) \cdot Z(\alpha) \qquad \text{for all } A \text{ in } N \tag{5.5}$$

$$Z(a) = 1 \qquad \text{for all } a \text{ in } \Sigma \uplus \{\varepsilon\} \tag{5.6}$$

$$Z(X\beta) = Z(X) \cdot Z(\beta) \qquad \text{for all } (X, \beta) \text{ in } V \times V^*. \tag{5.7}$$

This describes a monotone system of equations with the $Z(A)$ for A in N as variables.

Example 5.2. Properness and consistency are two distinct notions. For instance, the PCFG

$$S \xrightarrow{p_1} S\,S \qquad \rho(p_1) = q$$
$$S \xrightarrow{p_2} a \qquad \rho(p_2) = (1-q)$$

is proper for all $0 \leq q \leq 1$, but $Z(S)$ is the least solution of $x = qx^2 + 1 - q$, thus if $q \leq \frac{1}{2}$ the grammar is consistent, but otherwise $Z(S) = \frac{1-q}{q} < 1$.

Conversely,

$$S \xrightarrow{p_1} A \qquad \rho(p_1) = \frac{q}{1-q}$$
$$A \xrightarrow{p_2} A\,A \qquad \rho(p_2) = q$$
$$A \xrightarrow{p_3} a \qquad \rho(p_3) = 1-q$$

is improper but consistent for $\frac{1}{2} < q < 1$.

See Booth and Thompson (1973); Gecse and Kovács (2010) for ways to check for consistency, and Etessami and Yannakakis (2009) for ways to compute $Z(A)$. In general, $Z(A)$ has to be approximated:

Remark 5.3 (Etessami and Yannakakis, 2009, Theorem 3.2)**.** The partition function of S can be irrational even when ρ maps productions to rationals in $[0,1]$:

$$S \xrightarrow{p_1} S\,S\,S\,S\,S \qquad \rho(p_1) = \frac{1}{6}$$
$$S \xrightarrow{p_2} a \qquad \rho(p_2) = \frac{1}{2}\;.$$

The associated equation is $x = \frac{1}{6}x^5 + \frac{1}{2}$, which has no rational root.

Normalization

Given $Z(A)$ for all A in N, one can furthermore **normalize** any reduced convergent PCFG $\mathcal{G} = \langle N, \Sigma, P, S, \rho\rangle$ with $Z(S) > 0$ into a proper and consistent PCFG $\mathcal{G}' = \langle N, \Sigma, P, S, \rho'\rangle$. Define for this

$$\rho'(p = A \to \alpha) = \frac{\rho(p)Z(\alpha)}{Z(A)}\;. \tag{5.8}$$

Exercise 5.3. Show that in a reduced convergent PCFG with $Z(S) > 0$, for each α in V^*, one has $0 < Z(\alpha) < \infty$. (This justifies that (5.8) is well-defined.) **(∗)**

Exercise 5.4. Show that $\mathcal{G}'$ is a proper PCFG. **(∗)**

Proposition 5.4. *The grammar $\mathcal{G}'$ defined by* (5.8) *is consistent if $\mathcal{G}$ is reduced and convergent.*

Proof. We rely for the proof on the following claim:

Claim 5.4.1. For all Y in V, π in P^*, and w in Σ^* with $Y \overset{\pi}{\underset{\text{lm}}{\Longrightarrow}}{}^{\star}\, w$,

$$\rho'(\pi) = \frac{\rho(\pi)}{Z(Y)}\;. \tag{5.9}$$

Proof of Claim 5.4.1. Note that, because $\mathcal{G}$ is reduced, $Z(Y) > 0$ for all Y in V, so all the divisions we perform are well-defined.

We prove the claim by induction over the derivation π. For the base case, in an empty derivation $\pi = \varepsilon$, $\rho'(\varepsilon) = \rho(\varepsilon) = 1$ and $Z(Y) = 1$ since Y is necessarily a terminal, hence the claim holds. For the induction step, consider a derivation $p\pi$ for some production $p = A \to X_1 \cdots X_m$: $A \overset{p}{\underset{\text{lm}}{\Longrightarrow}} X_1 \cdots X_m \overset{\pi}{\underset{\text{lm}}{\Longrightarrow}}{}^{\star}\, w$. This derivation can be decomposed using a derivation $X_i \overset{\pi_i}{\underset{\text{lm}}{\Longrightarrow}}{}^{\star}\, w_i$ for each i, such that $\pi = \pi_1 \cdots \pi_n$ and $w = w_1 \cdots w_n$. By induction hypothesis, $\rho'(\pi_i) = \rho(\pi_i)/Z(X_i)$. Hence

$$\begin{aligned}
\rho'(p\pi) &= \rho'(p) \cdot \prod_{i=1}^{m} \rho'(\pi_i) \\
&= \frac{\rho(p)Z(X_1 \cdots X_m)}{Z(A)} \cdot \prod_{i=1}^{m} \rho'(\pi_i) && \text{(by (5.8))} \\
&= \frac{\rho(p)}{Z(A)} \cdot \prod_{i=1}^{m} Z(X_i) \cdot \prod_{i=1}^{m} \frac{\rho(\pi_i)}{Z(X_i)} && \text{(by ind. hyp.)} \\
&= \frac{\rho(p)}{Z(A)} \cdot \prod_{i=1}^{m} \rho(\pi_i) \\
&= \frac{\rho(p\pi)}{Z(A)}\;.
\end{aligned}$$

[5.4.1]

Claim 5.4.1 shows that $\mathcal{G}'$ is consistent, since

$$Z'(S) = \sum_{w\in\Sigma^*, S\overset{\pi}{\underset{\text{lm}}{\Longrightarrow}}^{\star} w} \rho'(\pi) = \sum_{w\in\Sigma^*, S\overset{\pi}{\underset{\text{lm}}{\Longrightarrow}}^{\star} w} \frac{\rho(\pi)}{Z(S)} = \frac{Z(S)}{Z(S)} = 1\ . \qquad \square$$

Remark 5.5. Note that Claim 5.4.1 also yields for all w in Σ^*

$$\rho'(w) = \sum_{S\overset{\pi}{\underset{\text{lm}}{\Longrightarrow}}^{\star} w} \rho'(\pi) = \sum_{S\overset{\pi}{\underset{\text{lm}}{\Longrightarrow}}^{\star} w} \frac{\rho(\pi)}{Z(S)} = \frac{\rho(w)}{Z(S)}\ , \tag{5.10}$$

thus the ratios between derivation weights are preserved by the normalization procedure.

Example 5.6. Considering again the first grammar of Example 5.2, if $q > \frac{1}{2}$, then ρ' with $\rho'(p_1) = \frac{q\,Z(S)^2}{Z(S)} = 1 - q$ and $\rho'(p_2) = q$ fits.

5.2 Learning PCFGs

We rely on an annotated corpus for **supervised** learning. We consider for this the Penn Treebank (Marcus et al., 1993) as an example of such an annotated corpus, made of n trees.

Maximum Likelihood Estimation Assuming the treebank to be well-formed, i.e. that the labels of internal nodes and those of leaves are disjoint, we can collect all the labels of internal tree nodes as nonterminals, all the labels of tree leaves as terminals, and all elementary subtrees (i.e. all the subtrees of height one) as productions. Introducing a new start symbol S' with productions $S' \to S$ for each label S of a root node ensures a unique start symbol. The treebank itself can then be seen as a multiset of leftmost derivations $D = \{\pi_1, \ldots, \pi_n\}$.

Let $C(p, \pi)$ be the count of occurrences of production p inside derivation π, and $C(A, \pi) = \sum_{p=A\to\alpha\in P} C(p, \pi)$. Summing over the entire treebank, we get $C(p, D) = \sum_{\pi\in D} C(p, \pi)$ and $C(A, D) = \sum_{\pi\in D} C(A, \pi)$. The estimated probability of a production is then (see e.g. Chi and Geman, 1998)

$$\rho(p = A \to \alpha) = \frac{C(p, D)}{C(A, D)}\ . \tag{5.11}$$

(∗∗) **Exercise 5.5.** Show that the obtained PCFG is proper and consistent.

*The statistical distribution of words in corpora can be approximated by **Zipf's law** (see Manning and Schütze, 1999, Section 1.4.3).*

See Jurafsky and Martin (2009, Section 4.5) and Manning and Schütze (1999, Chapter 6).

Smoothing Maximum likelihood estimations are accurate if there are enough occurrences in the training corpus. Nevertheless, some valid sequences of tags or of pairs of tags and words will invariably be missing, and be assigned a zero probability. Furthermore, the estimations are also unreliable for observations with low occurrence counts—they *overfit* the available data.

The idea of **smoothing** is to compensate data sparseness by moving some of the probability mass from the higher counts towards the lower and null ones. This can be performed in rather crude ways (for instance add 1 to the counts on the numerator of (5.11) and normalize, called **Laplace smoothing**), or more involved ones that take into account the probability of observations with a single occurrence (**Good-Turing discounting**).

Preprocessing the Treebank The PCFG estimated from a treebank is typically not very good: the linguistic annotations are too coarse-grained, and nonterminals do not capture enough context to allow for a precise parsing. Refining nonterminals allows to capture some *hidden state* information from the treebank.

Refining Nonterminals. For instance, PP attachment ambiguities are typically resolved as high attachments (i.e. to the VP) when the verb expects a PP complement, as with the following *hurled... into* construction, and a low attachment (i.e. to the NP) otherwise, as in the following *sip of ...* construction:

[$_{\text{NP}}$ He] [$_{\text{VP}}$[$_{\text{VP}}$ hurled [$_{\text{NP}}$ the ball]] [$_{\text{PP}}$ into the basket]].
[$_{\text{NP}}$ She] [$_{\text{VP}}$ took [$_{\text{NP}}$[$_{\text{NP}}$ a sip] [$_{\text{PP}}$ of water]]].

A PCFG cannot assign different probabilities to the attachment choices if the extracted rules are the same.

In practice, the tree annotations are refined in two directions: from the lexical leaves by tracking the **head** information, and from the root by remembering the **parent** or **grandparent** label. This greatly increases the sets of nonterminals and rules, thus some smoothing techniques are required to compensate for data sparseness. Figure 5.1 illustrates this idea by associating lexical head and parent information to each internal node. Observe that the PP attachment probability is now specific to a production

$$\text{VP}[\text{S}, \mathit{hurled}, \text{VBD}] \to \text{VP}[\text{VP}, \mathit{hurled}, \text{VBD}]\ \text{PP}[\text{VP}, \mathit{into}, \text{IN}]\ ,$$

allowing to give it a higher probability than that of

$$\text{VP}[\text{S}, \mathit{took}, \text{VBD}] \to \text{VP}[\text{VP}, \mathit{took}, \text{VBD}]\ \text{PP}[\text{VP}, \mathit{of}, \text{IN}]\ .$$

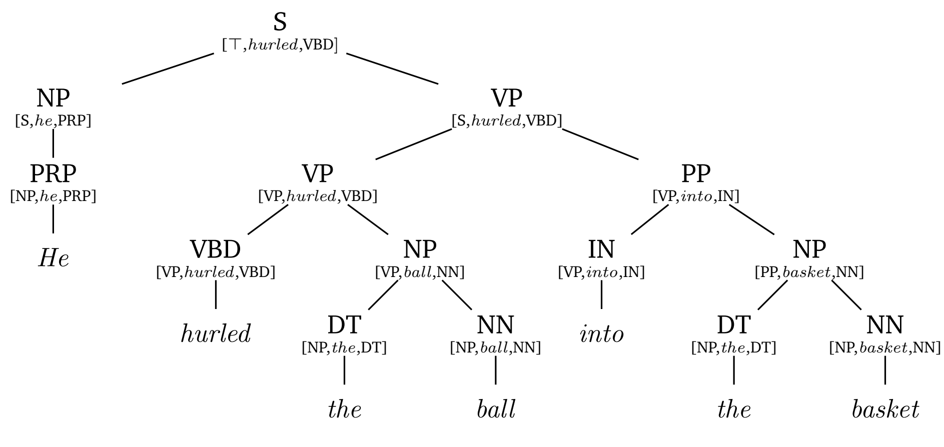

Figure 5.1: A derivation tree refined with lexical and parent information.

Binary Rules. Another issue, which is more specific to the kind of linguistic analyses found in the Penn Treebank, is that trees are mostly *flat*, resulting in a very large number of long, different rules, like

$$\text{VP} \to \text{VBP PP PP PP PP PP ADVP PP}$$

for sentence

This mostly happens because we [$_{\text{VP}}$ go [$_{\text{PP}}$ from football] [$_{\text{PP}}$ in the fall] [$_{\text{PP}}$ to lifting] [$_{\text{PP}}$ in the winter] [$_{\text{PP}}$ to football] [$_{\text{ADVP}}$ again] [$_{\text{PP}}$ in the spring]].

The *WSJ* part of the Penn Treebank yields about 17,500 distinct rules, causing important data sparseness issues in probability estimations. A solution is to transform the resulting grammar into **quadratic form** prior to probability estimation, for instance by having rules

$$\text{VP} \to \text{VBP VP'} \qquad\qquad \text{VP'} \to \text{PP} \mid \text{PP VP'} \mid \text{ADVP VP'}\ .$$

Parser Evaluation The usual measure of constituent parser performance is called PARSEVAL (Black et al., 1991). It supposes that some **gold standard** derivation trees are available for sentences, as in a test subcorpus of the *Wall Street Journal* part of the Penn Treebank, and compares the candidate parses with the gold ones. The comparison is constituent-based: correctly identified constituents start and end at the expected point and are labeled with the appropriate nonterminal symbol. The evaluation measures the

labeled recall which is the number of correct constituents in the candidate parse of a sentence, divided by the number of constituents in the gold standard analysis of the sentence,

labeled precision which is the number of correct constituents in the candidate parse of a sentence divided by the number of constituents in the same candidate parse.

Current probabilistic parsers on the *WSJ* treebank obtain a bit more than 90% precision and recall. Beware however that long sentences are often parsed incorrectly, i.e. have at least one misparsed constituent.

5.3 Probabilistic Parsing as Intersection

We generalize in this section the intersective approach of Theorem 2.7. More precisely, we show how to construct a product grammar from a weighted grammar and a weighted automaton over a commutative semiring, and then use a generalized version of Dijkstra's algorithm due to Knuth (1977) to find the most probable parse in this grammar.

5.3.1 Weighted Product

We generalize here Theorem 2.7 to the weighted case. Observe that it also answers Exercise 5.1 since $\mathbb{K}$-automata are equivalent to right-linear $\mathbb{K}$-CFGs according to Exercise 5.2.

We abuse notation and write $A \xrightarrow{k} \alpha$ for a production $p = A \to \alpha$ with $\rho(p) = k$.

Theorem 5.7. *Let $\mathbb{K}$ be a commutative semiring, $\mathcal{G} = \langle N, \Sigma, P, S, \rho\rangle$ an acyclic $\mathbb{K}$-CFG, and $\mathcal{A} = \langle Q, \Sigma, \mathbb{K}, \delta, I, F\rangle$ a $\mathbb{K}$-automaton. Then the $\mathbb{K}$-CFG $\mathcal{G}' = \langle \{S'\} \uplus (N \times Q \times Q), \Sigma, P', S', \rho'\rangle$ with*

$$\begin{aligned} P' \stackrel{\text{def}}{=} \{S' &\xrightarrow{I(q_i)\odot F(q_f)} (S, q_i, q_f) \mid q_i, q_f \in Q\} \\ \cup\ \{(A, q_0, q_m) &\xrightarrow{k} (X_1, q_0, q_1)\cdots(X_m, q_{m-1}, q_m) \\ &\qquad\qquad \mid m \geq 1, A \xrightarrow{k} X_1 \cdots X_m \in P, q_0, \ldots, q_m \in Q\} \\ \cup\ \{(a, q, q') &\xrightarrow{k} a \mid (q, a, k, q') \in \delta\} \end{aligned}$$

is acyclic and such that, for all w in Σ^*, $\langle [\![\mathcal{G}']\!], w\rangle = \langle [\![\mathcal{G}]\!], w\rangle \odot \langle [\![\mathcal{A}]\!], w\rangle$.

See Maletti and Satta (2009) for a version of Theorem 5.7 that works on weighted tree automata instead of CFGs.

As with Theorem 2.7, the construction of Theorem 5.7 works in time $O(|\mathcal{G}| \cdot |Q|^{m+1})$ with m the maximal length of a rule rightpart in $\mathcal{G}$. Again, this complexity can be reduced by first transforming $\mathcal{G}$ into **quadratic form**, thus yielding a $O(|\mathcal{G}| \cdot |Q|^3)$ construction.

Exercise 5.6. Modify the quadratic form construction of Lemma 2.8 for the weighted case. **(∗)**

5.3.2 Most Probable Parse

The weighted CFG $\mathcal{G}'$ constructed by Theorem 5.7 can be *reduced* by a generalization of the usual CFG reduction algorithm to the weighted case. Here we rather consider the issue of finding the best parse in this intersection grammar $\mathcal{G}'$, assuming we are working on the probabilistic semiring—we could also work on the tropical semiring.

Non Recursive Case The easiest case is that of a **non recursive** $\mathbb{K}$-CFG $\mathcal{G}'$, i.e. where there does not exist a derivation $A \Rightarrow^+ \delta A \gamma$ for any A in N and δ, γ in V^* in the underlying grammar. This is necessarily the case with Theorem 5.7 if $\mathcal{G}$ is acyclic and $\mathcal{A}$ has a finite support language. Then a **topological sort** of the nonterminals of $\mathcal{G}'$ for the partial ordering $B \prec A$ iff there exists a production $A \to \alpha B \beta$ in P' with α, β in V'^* can be performed in linear time, yielding a total order $(N', <)$: $A_1 < A_2 < \cdots < A_{|N'|}$. We can then compute the probability $M(S')$ of the most probable parse by computing for $j = 1, \ldots, |N'|$

$$M(A_j) = \max_{A \xrightarrow{k} X_1 \cdots X_m} k \cdot M(X_1) \cdots M(X_m) \tag{5.12}$$

in the probabilistic semiring, with $M(a) = 1$ for each a in Σ. The topological sort ensures that the maximal values $M(X_i)$ in the right-hand side have already been computed when we use (5.12) to compute $M(A_j)$.

Knuth's Algorithm In the case of a recursive PCFG, the topological sort approach fails. We can nevertheless use an extension of Dijkstra's algorithm to weighted CFGs proposed by Knuth (1977): see Algorithm 5.1.

Data: $\mathcal{G} = \langle N, \Sigma, P, S, \rho \rangle$
foreach $a \in \Sigma$ **do**
  $M(a) = 1$
$D \longleftarrow \Sigma$
while $D \neq V$ **do**
  foreach $A \in V \backslash D$ **do**
    $\nu(A) \longleftarrow \max_{A \xrightarrow{k} X_1 \cdots X_m \text{ s.t. } X_1, \ldots, X_m \in D} k \cdot M(X_1) \cdots M(X_m)$
  $A \longleftarrow \mathrm{argmax}_{V \backslash D}\, \nu(A)$
  $M(A) \longleftarrow \nu(A)$
  $D \longleftarrow D \uplus \{A\}$
return $M(S)$

Algorithm 5.1: Most probable derivation.

The set $D \subseteq V$ is the set of symbols X for which $M(X)$, the probability of the most probable tree rooted in X, has been computed. Using a **priority queue** for extracting elements of $V \backslash D$ in time $\log |N|$ at line 7, and tracking which productions to consider for the computation of $\nu(A)$ at line 6, the time complexity of the algorithm is in $O(|P| \log |N| + |\mathcal{G}|)$.

The correctness of the algorithm relies on the fact that $M(A) = \nu(A)$ at line 8; assuming the opposite, there must exist a shortest derivation $B \overset{\pi}{\underset{\text{lm}}{\Longrightarrow}}{}^{\star}\, w$ with $\rho(\pi) > \nu(A)$ for some $B \notin D$. We can split this derivation into $B \overset{p}{\underset{\text{lm}}{\Longrightarrow}}{}^{\star}\, X_1 \cdots X_m$ and $X_i \overset{\pi_i}{\underset{\text{lm}}{\Longrightarrow}}{}^{\star}\, w_i$ with $w = w_1 \cdots w_m$ and $\pi = p\pi_1 \cdots \pi_m$, thus with $\rho(\pi) = \rho(p) \cdot \rho(\pi_1) \cdots \rho(\pi_m)$. If each X_i is already in D, then $M(X_i) \geq \rho(\pi_i)$ for all i, thus $\rho(\pi) \leq \nu(B)$ computed at line 6, and finally $\rho(\pi) \leq \nu(B) \leq \nu(A)$ by line 8—a contradiction. Therefore there must be one X_i not in D for some i, but in that case $\rho(\pi_i) \geq \rho(\pi) > \nu(A)$ and π_i is strictly shorter than π, a contradiction.

5.3.3 Most Probable String

We have just seen that the algorithms for the Boolean case are rather easy to extend in order to handle general (commutative) semirings, including the probabilistic semiring. Let us finish with an example showing that *some* problems become hard.

Consider the following decision problem:

Most Probable String (MPS**)**

input a PCFG $\mathcal{G}$ over Σ with rational weights (coded in binary) and a rational p in $[0, 1]$ (also in binary);

question is there a string w in Σ^* s.t. $\langle [\![\mathcal{G}]\!], w \rangle \geq p$?

Theorem 5.8 (Casacuberta and de la Higuera, 2000)**.** *MPS for convergent right-linear PCFGs is NP-hard.*

See also the work of Sima'an (2002) for similar bounds.

Proof. The proof reduces from SAT. Let $\varphi = \bigwedge_{i=1}^{k} C_k$ be a propositional formula in conjunctive normal form, where each clause C_i is a non-empty disjunction of literals over the set of variables $\{x_1, \ldots, x_n\}$. Without loss of generality, we assume that each variable appears at most once in each clause, be it positively or negatively.

We construct in polynomial time an instance $\langle \mathcal{G}, p \rangle$ of MPS such that φ is satisfiable if and only if there exists w in Σ^* such that $\langle [\![\mathcal{G}]\!], w \rangle \geq p$. We define for this $\mathcal{G} \overset{\text{def}}{=} \langle N, \Sigma, P, S \rangle$ where

$$N \overset{\text{def}}{=} \{S\} \uplus \{A_{i,j} \mid 1 \leq i \leq k \wedge 0 \leq j \leq n\} \uplus \{B_j \mid 1 \leq j \leq n\}$$
$$\Sigma \overset{\text{def}}{=} \{0, 1, \$\}\ ,$$

$$
\begin{aligned}
P \stackrel{\text{def}}{=} & \{S \xrightarrow{1/k} \$A_{i,0} \mid 1 \leq i \leq k\} \\
& \cup \{A_{i,j-1} \xrightarrow{1/2} vB_j, A_{i,j-1} \xrightarrow{1/2} (1-v)A_{i,j} \mid v \in \{0,1\} \wedge x_j \mapsto v \models C_i \\
& \qquad \wedge 1 \leq i \leq k \wedge 1 \leq j \leq n\} \\
& \cup \{A_{i,j-1} \xrightarrow{1/2} 1A_{i,j}, A_{i,j-1} \xrightarrow{1/2} 0A_{i,j} \mid x_j \notin C_i \wedge 1 \leq i \leq k \wedge 1 \leq j \leq n\} \\
& \cup \{A_{i,n} \xrightarrow{0} \$ \mid 1 \leq i \leq k\} \\
& \cup \{B_{j-1} \xrightarrow{1/2} 0B_i, B_{j-1} \xrightarrow{1/2} 1B_i \mid 2 \leq j \leq n\} \\
& \cup \{B_n \xrightarrow{1} \$\}
\end{aligned}
$$

and fix

$$p \stackrel{\text{def}}{=} 1/2^n .$$

First note that the construction can indeed be carried in polynomial time—remember that p is encoded in binary. Second, $\mathcal{G}$ is visibly right-linear by construction, and also convergent because every derivation is of length $n+2$ and every string has finitely many derivations.

It remains to show that φ is satisfiable if and only if there exists w in Σ^* such that $\langle[\![\mathcal{G}]\!], w\rangle \geq p$. Note that any string w with $\langle[\![\mathcal{G}]\!], w\rangle > 0$ is necessarily of form $\$v_1 \cdots v_n\$$ with each v_j in $\{0,1\}$, i.e. describes a valuation for φ.

Observe that, for each clause C_i and each string $w = \$v_1 \cdots v_n\$$, w describes a valuation $V_w\colon x_j \mapsto v_j$ that

- either satisfies C_i, and then the corresponding string w has a single derivation π_w (the one that uses $A_{i,j-1} \xrightarrow{1/2} v_j B_j$ for the lowest index j such that $x_j \mapsto v_j \models C_i$); this derivation has probability $\rho(\pi_w) = 1/(k2^n)$,
- or does not satisfies C_i, and there is a single derivation, which must use the production $A_{i,n} \xrightarrow{0} \$$, and is thus of probability 0.

Therefore, if φ is satisfiable, i.e. if there exists V that satisfies all the clauses, then the corresponding string w_V has probability $\sum_{i=1}^{k} 1/(k2^n) = p$. Conversely, if φ is not satisfiable, then any w with $\langle[\![\mathcal{G}]\!], w\rangle > 0$ is of form $\$v_1 \cdots v_n\$$ and describes an assignment $V_w\colon x_j \mapsto v_j$ that does not satisfy at least one of the clauses, thus has a total probability $\rho(w) < p$. □

Corollary 5.9. *MPS for proper and consistent right-linear PCFGs is NP-hard.*

Proof. If suffices to reduce and normalize the PCFG constructed in Theorem 5.8. Because every derivation is of bounded length, the computation of the partition function for $\mathcal{G}$ converges in polynomial time, and the grammar can be normalized in polynomial time.

Let us nevertheless perform those computations by hand as an exercise. For instance, for all $1 \leq j \leq n$,

$$Z(B_j) = 1 , \tag{5.13}$$

$$Z(S) = \frac{1}{k} \sum_{i=1}^{k} Z(A_{i,0}) . \tag{5.14}$$

We need to introduce some notation in order to handle the computation of $Z(A_{i,j})$. For each clause C_i, and each $0 \leq j \leq n$, let $q_{i,j}$ be the number of variables

x_ℓ with $j < \ell \leq n$ that occur (positively or negatively) in C_i:

$$q_{i,j} \stackrel{\text{def}}{=} |\{x_\ell \in C_i \mid j < \ell \leq n\}| \; . \tag{5.15}$$

Claim 5.9.1. For all $1 \leq i \leq k$ and $0 \leq j \leq n$,

$$Z(A_{i,j}) = 1 - \frac{1}{2^{q_{i,j}}} \; .$$

Proof of Claim 5.9.1. Fix some $1 \leq i \leq k$; we proceed by induction over $n - j$. For the base case, $Z(A_{i,n}) = 0$ since the only production available is $A_{i,n} \xrightarrow{0} \$$. For the induction step, two cases arise:

1. $q_{i,j} = q_{i,j+1}$, i.e. when x_{j+1} does not appear in C_i. Then $Z(A_{i,j}) = 1/2 \cdot Z(A_{i,j+1}) + 1/2 \cdot Z(A_{i,j+1})$ by (5.5), and thus $Z(A_{i,j}) = Z(A_{i,j+1}) = 1 - 1/2^{q_{i,j+1}} = 1 - 1/2^{q_{i,j}}$ by induction hypothesis.

2. $q_{i,j} = 1 + q_{i,j+1}$, i.e. when x_{j+1} appears in C_i. Then $Z(A_{i,j}) = 1/2 \cdot Z(A_{i,j+1}) + 1/2 \cdot Z(B_{j+1})$, hence by (5.13) and the induction hypothesis, $Z(A_{i,j}) = 1/2 - 1/2^{q_{i,j+1}+1} + 1/2 = 1 - 1/2^{q_{i,j}}$. [5.9.1]

In particular, if we reduce from a 3SAT instance instead of any SAT instance, then $Z(A_{i,0}) = 7/8$ for all i, and thus $Z(S) = 7/8$.

Any nonterminal with probability mass 0 can be disposed of during the reduction phase, which can be performed in polynomial time. We use next (5.8) to normalize the grammar of Theorem 5.8, thereby obtaining a proper and consistent right-linear PCFG $\mathcal{G}'$ in polynomial time.

There remains the issue of computing an appropriate bound p' for this new grammar. By Remark 5.5, for any word w in Σ^*, $\langle [\![G]\!], w\rangle \geq p$ if and only if $\langle [\![G']\!], w\rangle \geq p/Z(S)$: we define therefore

$$p' \stackrel{\text{def}}{=} \frac{p}{Z(S)} \; . \tag{5.16}$$

□

Chapter 6

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 40.

Backus, J.W., 1959. The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM Conference. In *IFIP Congress*, pages 125–131. Cited on page 8.

Bar-Hillel, Y., 1953. A quasi-arithmetical notation for syntactic description. *Language*, 29 (1):47–58. doi:10.2307/410452. Cited on page 10.

Bar-Hillel, Y., Perles, M., and Shamir, E., 1961. On formal properties of simple phrase-structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft, und Kommunikations-forschung*, 14:143–172. Cited on pages 14, 17.

Billot, S. and Lang, B., 1989. The structure of shared forests in ambiguous parsing. In *ACL'89, 27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151. ACL Press. doi:10.3115/981623.981641. Cited on page 14.

Black, E., Abney, S., Flickenger, S., Gdaniec, C., Grishman, C., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., and Strzalkowski, T., 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *HLT '91, Fourth Workshop on Speech and Natural Language*, pages 306–311. ACL Press. doi:10.3115/112405.112467. Cited on page 56.

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 page 29.

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.

Booth, T.L. and Thompson, R.A., 1973. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22(5):442–450. doi:10.1109/T-C.1973.223746. Cited on pages 51, 52.

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.

Casacuberta, F. and de la Higuera, C., 2000. Computational complexity of problems on probabilistic grammars and transducers. In Oliveira, A.L., editor, *ICGI 2000, 5th International Conference on Grammatical Inference: Algorithms and Applications*, volume 1891 of *Lecture Notes in Artificial Intelligence*, pages 15–24. Springer. doi:10.1007/978-3-540-45257-7_2. Cited on page 58.

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.

Charniak, E., 1997. Statistical parsing with a context-free grammar and word statistics. In *AAAI '97/IAAI '97*, pages 598–603. AAAI Press. Cited on page 10.

Chi, Z. and Geman, S., 1998. Estimation of probabilistic context-free grammars. *Computational Linguistics*, 24(2):299–305. http://www.aclweb.org/anthology/J98-2005.pdf. Cited on page 54.

Chomsky, N., 1956. Three models for the description of language. *IEEE Transactions on Information Theory*, 2(3):113–124. doi:10.1109/TIT.1956.1056813. Cited on pages 8, 13.

Chomsky, N., 1957. *Syntactic Structures*. Mouton de Gruyter. Cited on page 49.

Chomsky, N., 1959. On certain formal properties of grammars. *Information and Control*, 2(2):137–167. doi:10.1016/S0019-9958(59)90362-6. Cited on page 13.

Cocke, J. and Schwartz, J.T., 1970. *Programming languages and their compilers*. Courant Institute of Mathematical Sciences, New York University. Cited on page 14.

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.

Collins, M., 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29:589–637. doi:10.1162/089120103322753356. Cited on page 10.

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 3, 8, 15, 23, 28, 40.

Cousot, P. and Cousot, R., 2003. Parsing as abstract interpretation of grammar semantics. *Theoretical Computer Science*, 290(1):531–544. doi:10.1016/S0304-3975(02)00034-8. Cited on page 19.

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 39.

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 page 39.

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 28.

Duchier, D. and Debusmann, R., 2001. Topological dependency trees: a constraint-based account of linear precedence. In *ACL 2001, 39th Annual Meeting of the Association for Computational Linguistics*, pages 180–187. Annual Meeting of the Association for Computational Linguistics. doi:10.3115/1073012.1073036. Cited on page 10.

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.

Earley, J., 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102. doi:10.1145/362007.362035. Cited on pages 14, 19.

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 45.

Etessami, K. and Yannakakis, M., 2009. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1–66. doi:10.1145/1462153.1462154. Cited on pages 52, 53.

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 40, 41, 42, 45.

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.

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 42.

Gaifman, H., 1965. Dependency systems and phrase-structure systems. *Information and Control*, 8(3):304–337. doi:10.1016/S0019-9958(65)90232-9. Cited on page 10.

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 39.

Gecse, R. and Kovács, A., 2010. Consistency of stochastic context-free grammars. *Mathematical and Computer Modelling*, 52(3–4):490–500. doi:10.1016/j.mcm.2010.03.046. Cited on page 52.

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 40.

Ginsburg, S. and Rice, H.G., 1962. Two families of languages related to ALGOL. *Journal of the ACM*, 9(3):350–371. doi:10.1145/321127.321132. Cited on page 8.

Graham, S.L., Harrison, M., and Ruzzo, W.L., 1980. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415–462. doi:10.1145/357103.357112. Cited on page 14.

Grune, D. and Jacobs, C.J.H., 2007. *Parsing Techniques*. Monographs in Computer Science. Springer, second edition. ISBN 0-387-20248-X. Cited on page 14.

Guessarian, I., 1983. Pushdown tree automata. *Theory of Computing Systems*, 16(1):237–263. doi:10.1007/BF01744582. Cited on page 40.

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

Hays, D.G., 1964. Dependency theory: A formalism and some observations. *Language*, 40(4):511–525. http://www.jstor.org/stable/411934. Cited on page 10.

Jones, N.D. and Laaser, W.T., 1976. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3(1):105–117. doi:10.1016/0304-3975(76)90068-2. Cited on page 14.

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 36.

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 35.

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 35.

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 36.

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 11, 54.

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 39.

Kallmeyer, L. and Kuhlmann, M., 2012. A formal model for plausible dependencies in lexicalized tree adjoining grammar. In *TAG+11, 11th International Workshop on Tree-Adjoining Grammars and Related Formalisms*, pages 108–116. http://user.phil-fak.uni-duesseldorf.de/~kallmeyer/papers/KallmeyerKuhlmann-TAG+11.pdf. Cited on page 39.

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 45.

Kasami, T., 1965. An efficient recognition and syntax analysis algorithm for context free languages. Scientific Report AF CRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachussetts. Cited on page 14.

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 46.

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 42, 45.

Knuth, D.E., 1965. On the translation of languages from left to right. *Information and Control*, 8(6):607–639. doi:10.1016/S0019-9958(65)90426-2. Cited on page 14.

Knuth, D.E., 1977. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5. doi:10.1016/0020-0190(77)90002-3. Cited on pages 56, 57.

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 38.

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 35.

Kuhlmann, M., 2013. Mildly non-projective dependency grammar. 39(2):355–387. doi: 10.1162/COLI_a_00125. Cited on page 45.

Kurki-Suonio, R., 1969. Notes on top-down languages. *BIT Numerical Mathematics*, 9 (3):225–238. doi:10.1007/BF01946814. Cited on page 14.

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.

Lambek, J., 1958. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3):154–170. doi:10.2307/2310058. Cited on page 11.

Lang, B., 1974. Deterministic techniques for efficient non-deterministic parsers. In Loeckx, J., editor, *ICALP'74, 2nd International Colloquium on Automata, Languages and Programming*, volume 14 of *Lecture Notes in Computer Science*, pages 255–269. Springer. doi:10.1007/3-540-06841-4_65. Cited on page 14.

Lang, B., 1994. Recognition can be harder than parsing. *Computational Intelligence*, 10 (4):486–494. doi:10.1111/j.1467-8640.1994.tb00011.x. Cited on page 17.

Lee, L., 2002. Fast context-free grammar parsing requires fast boolean matrix multiplication. *Journal of the ACM*, 49(1):1–15. doi:10.1145/505241.505242. Cited on page 14.

Leo, J.M.I.M., 1991. A general context-free parsing algorithm running in linear time on every LR(k) grammar without using lookahead. *Theoretical Computer Science*, 82(1): 165–176. doi:10.1016/0304-3975(91)90180-A. Cited on page 20.

Maletti, A. and Satta, G., 2009. Parsing algorithms based on tree automata. In *IWPT 2009, 11th International Workshop on Parsing Technologies*, pages 1–12. ACL Press. http://www.aclweb.org/anthology/W09-3801.pdf. Cited on page 56.

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 47.

Manning, C.D. and Schütze, H., 1999. *Foundations of Statistical Natural Language Processing*. MIT Press. ISBN 978-0-262-13360-9. Cited on pages 8, 11, 54.

Marcus, M.P., Marcinkiewicz, M.A., and Santorini, B., 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330. http://www.aclweb.org/anthology/J93-2004.pdf. Cited on page 54.

Martin, W.A., Church, K.W., and Patil, R.S., 1987. Preliminary analysis of a breadth-first parsing algorithm: Theoretical and experimental results. In Bolc, L., editor, *Natural Language Parsing Systems*, Symbolic Computation, pages 267–328. Springer. doi:10.1007/978-3-642-83030-3_8. Cited on page 7.

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.

Mel'čuk, I.A., 1988. *Dependency syntax: Theory and practice*. SUNY Press. Cited on page 9.

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 28.

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 42.

Moore, R.C., 2004. Improved left-corner chart parsing for large context-free grammars. In *New Developments in Parsing Technology*, pages 185–201. Springer. doi:10.1007/1-4020-2295-6_9. Cited on pages 7, 15.

Nederhof, M.J. and Satta, G., 2004. Tabular parsing. In Martín-Vide, C., Mitrana, V., and Paun, G., editors, *Formal Languages and Applications*, volume 148 of *Studies in Fuzziness and Soft Computing*, pages 529–549. Springer. arXiv:cs.CL/0404009. Cited on page 17.

Nederhof, M.J. and Satta, G., 2008. Probabilistic parsing. In Bel-Enguix, G., Jiménez-López, M., and Martín-Vide, C., editors, *New Developments in Formal Languages and Applications*, volume 113 of *Studies in Computational Intelligence*, pages 229–258. Springer. doi:10.1007/978-3-540-78291-9_7. Cited on page 51.

Palm, A., 1999. Propositional tense logic of finite trees. In *MOL 6, 6th Biennial Conference on Mathematics of Language*. 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 35.

Pereira, F.C.N. and Warren, D.H.D., 1983. Parsing as deduction. In *ACL '83, 21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144. ACL Press. doi:10.3115/981311.981338. Cited on page 19.

Pereira, F., 2000. Formal grammar and information theory: together again? *Philosophical Transactions of the Royal Society A*, 358(1769):1239–1253. doi:10.1098/rsta.2000.0583. Cited on pages 8, 49.

Pullum, G.K., 1986. Footloose and context-free. *Natural Language & Linguistic Theory*, 4(3):409–414. doi:10.1007/BF00133376. Cited on page 35.

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.

Pullum, G.K., 2007. The evolution of model-theoretic frameworks in linguistics. In *Model-Theoretic Syntax at 10*, pages 1–10. http://www.lel.ed.ac.uk/~gpullum/EvolutionOfMTS.pdf. Cited on page 9.

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 28.

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 28.

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 28.

Rosenkrantz, D.J. and Stearns, R.E., 1970. Properties of deterministic top-down grammars. *Information and Control*, 17(3):226–256. doi:10.1016/S0019-9958(70)90446-8. Cited on page 14.

Rounds, W.C., 1970. Mappings and grammars on trees. *Theory of Computing Systems*, 4 (3):257–287. doi:10.1007/BF01695769. Cited on pages 40, 45.

Santorini, B., 1990. Part-of-speech tagging guidelines for the Penn Treebank project (3rd revision). Technical Report MS-CIS-90-47, University of Pennsylvania, Department of Computer and Information Science. http://repository.upenn.edu/cis_reports/570/. Cited on page 14.

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 39.

Schützenberger, M.P., 1961. On the definition of a family of automata. *Information and Control*, 4(2–3):245–270. doi:10.1016/S0019-9958(61)80020-X. Cited on page 51.

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 pages 9, 35.

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 45.

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 35.

Sikkel, K., 1997. *Parsing Schemata - a framework for specification and analysis of parsing algorithms*. Texts in Theoretical Computer Science - An EATCS Series. Springer. ISBN 3-540-61650-0. Cited on page 19.

Sima'an, K., 2002. Computational complexity of probabilistic disambiguation. 5(2): 125–151. doi:10.1023/A:1016340700671. Cited on page 58.

Steedman, M., 2011. Romantics and revolutionaries. *Linguistic Issues in Language Technology*, 6. http://elanguage.net/journals/lilt/article/view/2587. Cited on page 8.

Sudborough, I.H., 1978. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414. doi:10.1145/322077.322083. Cited on page 14.

Thatcher, J.W., 1967. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences*, 1 (4):317–322. doi:10.1016/S0022-0000(67)80022-9. Cited on page 14.

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 28.

Tomita, M., 1986. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers. ISBN 0-89838-202-5. Cited on page 14.

Troelstra, A.S., 1992. *Lectures on Linear Logic*, volume 29 of *CSLI Lecture Notes*. CSLI Publications. http://standish.stanford.edu/bin/detail?fileID=1846861073. Cited on page 11.

Valiant, L.G., 1975. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–314. doi:10.1016/S0022-0000(75)80046-8. Cited on page 14.

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 33.

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 35.

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 28.

Wich, K., 2005. *Ambiguity Functions of Context-Free Grammars and Languages*. PhD thesis, Institut fur Formale Methoden der Informatik, Universität Stuttgart. ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/DIS-2005-01/DIS-2005-01.pdf. Cited on page 15.

Williams, V., 2012. Multiplying matrices faster than Coppersmith-Winograd. In *STOC 2012, 44th Symposium on Theory of Computing*, pages 887–898. ACM Press. doi:10.1145/2213977.2214056. Cited on page 14.

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 38.

Younger, D.H., 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208. doi:10.1016/S0019-9958(67)80007-X. Cited on page 14.