Formal Aspects of Linguistic Modelling

Sylvain Schmitz
LSV, ENS Paris-Saclay & CNRS & Inria, Université Paris-Saclay
March 4, 2019 (r9182M)

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 modelling. The course is subdivided into two parts: the first, which is the topic of these notes, covers grammars, automata, and logics for syntax modelling, while the second part focuses on logical approaches to semantics. Among the prerequisites to the course are

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

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

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

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

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

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

Contents

1 Introduction 5
  1.1 Levels of Description ........................................... 6
    1.1.1 From Text to Meaning ..................................... 6
    1.1.2 Ambiguity at Every Turn .................................. 8
    1.1.3 Romantics and Revolutionaries .......................... 8
  1.2 Models of Syntax ............................................... 8
    1.2.1 Constituent Syntax ....................................... 9
    1.2.2 Dependency Syntax ..................................... 10
  1.3 Further Reading .............................................. 12
# Formal Aspects of Linguistic Modelling

## 2 Morphology

2.0.1 A Bit of English Morphology ........................................ 14
2.0.2 Part-of-Speech Tags ..................................................... 15

2.1 Finite-State Morphology .................................................. 16
2.1.1 Background: Rational Transductions ............................... 16
2.1.2 Morphological Analysis ................................................. 19
2.1.3 Phonological Rules ...................................................... 21

2.2 Part-of-Speech Tagging .................................................... 24
2.2.1 Rule-Based Tagging ...................................................... 24
2.2.2 HMM Tagging ............................................................ 29

## 3 Context-Free Syntax

3.1 Grammars ........................................................................... 37
3.1.1 The Parsing Problem ...................................................... 38
3.1.2 Background: Tree Automata .......................................... 39

3.2 Tabular Parsing ................................................................... 41
3.2.1 Parsing as Intersection ................................................. 41
3.2.2 Parsing as Deduction .................................................... 42

## 4 Model-Theoretic Syntax

4.1 Introduction ......................................................................... 45
4.1.1 Model-Theoretic vs. Generative ....................................... 45
4.1.2 Tree Structures ........................................................... 46

4.2 Monadic Second-Order Logic .............................................. 47
4.2.1 Linguistic Analyses in wMSO ........................................ 49
4.2.2 MSO on Finite Binary Trees .......................................... 52

4.3 Propositional Dynamic Logic .............................................. 53
4.3.1 Model-Checking ........................................................... 54
4.3.2 Satisfiability ............................................................... 54
4.3.3 Expressiveness ............................................................ 58

4.4 Parsing as Intersection ..................................................... 60

## 5 Mildly Context-Sensitive Syntax

5.1 Tree Adjoining Grammars ................................................. 62
5.1.1 Linguistic Analyses Using TAGs ....................................... 65
5.1.2 Background: Context-Free Tree Grammars .................... 67

5.2 Well-Nested MCSLS .......................................................... 72
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td><strong>Probabilistic Syntax</strong></td>
<td>77</td>
</tr>
<tr>
<td>6.1</td>
<td>Weighted and Probabilistic CFGs</td>
<td>78</td>
</tr>
<tr>
<td>6.1.1</td>
<td><strong>Background</strong>: Semirings and Formal Power Series</td>
<td>78</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Weighted Grammars</td>
<td>79</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Probabilistic Grammars</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Partition Functions</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Normalisation</td>
<td>81</td>
</tr>
<tr>
<td>6.2</td>
<td>Learning PCFGs</td>
<td>82</td>
</tr>
<tr>
<td>6.3</td>
<td>Probabilistic Parsing as Intersection</td>
<td>84</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Weighted Product</td>
<td>85</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Most Probable Parse</td>
<td>85</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Most Probable String</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>Hardness</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>Upper Bounds</td>
<td>89</td>
</tr>
<tr>
<td>7</td>
<td><strong>Categorial Grammars</strong></td>
<td>93</td>
</tr>
<tr>
<td>7.1</td>
<td>AB Categorial Grammars</td>
<td>94</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Alternative Views</td>
<td>94</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Equivalence with Context-Free Grammars</td>
<td>95</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Structural Limitations</td>
<td>96</td>
</tr>
<tr>
<td>7.2</td>
<td>Lambek Grammars</td>
<td>96</td>
</tr>
<tr>
<td>7.2.1</td>
<td><strong>Background</strong>: Substructural Proof Systems</td>
<td>96</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Lambek Calculus</td>
<td>97</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Equivalence with Context-Free Grammars</td>
<td>99</td>
</tr>
<tr>
<td>8</td>
<td><strong>First-Order Semantics</strong></td>
<td>101</td>
</tr>
<tr>
<td>8.1</td>
<td>Formal Semantics</td>
<td>101</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Event Semantics</td>
<td>102</td>
</tr>
<tr>
<td>8.1.2</td>
<td>Thematic Roles</td>
<td>103</td>
</tr>
<tr>
<td>8.2</td>
<td>A Dip into Description Logics</td>
<td>104</td>
</tr>
<tr>
<td>8.2.1</td>
<td>A Basic Description Logic</td>
<td>104</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Translation into First-Order Logic</td>
<td>105</td>
</tr>
<tr>
<td>8.3</td>
<td>Modal Semantics</td>
<td>106</td>
</tr>
<tr>
<td>8.3.1</td>
<td><strong>Background</strong>: Modal Logic</td>
<td>106</td>
</tr>
<tr>
<td>8.3.2</td>
<td>First-Order Modal Logic</td>
<td>109</td>
</tr>
<tr>
<td>8.4</td>
<td>Decidability</td>
<td>110</td>
</tr>
<tr>
<td>8.4.1</td>
<td>The Guarded Fragment</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>Guarded Bisimulations</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>Models of Bounded Treewidth</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>Limitations &amp; Extensions</td>
<td>113</td>
</tr>
<tr>
<td>9</td>
<td><strong>Tree Patterns</strong></td>
<td>115</td>
</tr>
<tr>
<td>9.1</td>
<td><strong>Background</strong>: Existential First-Order Logic</td>
<td>115</td>
</tr>
<tr>
<td>9.1.1</td>
<td>Characterisations over Finite Models</td>
<td>116</td>
</tr>
<tr>
<td>9.1.2</td>
<td>Tree Models</td>
<td>118</td>
</tr>
<tr>
<td>9.2</td>
<td>Meta-Grammars</td>
<td>118</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Diathesis Alternation</td>
<td>118</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Complexity</td>
<td>119</td>
</tr>
<tr>
<td>9.3</td>
<td>Underspecified Semantics</td>
<td>120</td>
</tr>
</tbody>
</table>
9.3.1 Scope Ambiguities ............................................. 120
9.3.2 Hole Semantics ............................................. 121
Constructive Satisfiability ........................................ 122

10 Higher-Order Semantics ........................................ 127

10.1 Compositional Semantics .................................... 127
10.1.1 Background: Simply Typed Lambda Calculus ........... 127
10.1.2 Ground Terms over Second-Order Signatures .......... 129
10.1.3 Higher-Order Homomorphisms ......................... 131
10.1.4 Tree Transductions ..................................... 132

10.2 Intensionality .................................................. 134

10.3 Higher-Order Logic ........................................... 136
10.3.1 Background: Church’s Simple Theory of Types ........ 136
10.3.2 Type-Logical Semantics ................................. 137
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 summarisation, to automated translation, and these technologies have a considerable impact in our daily lives.

The case for computational linguistics is however not limited to its technological applications. 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.

![Figure 1.1: The levels of linguistic description.](image-url)
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 as depicted in Figure 1.1 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 recognise 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? Modelling 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 1.2. 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 1.2: Constituent (on the left) and dependency (on the right) analyses.](image-url)

An alternative analysis, illustrated on the right of Figure 1.2, 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.
respective. In turn, \textit{bird} governs its determiner \textit{a}. Again, additional labels can decorate the nodes and relations in dependency structures, as shown in Figure 1.2.

\textbf{Ambiguity}. The following sentence is a classical example of a syntactic ambiguity, illustrated by the two derivation trees of Figure 1.3:

She watches a man with a telescope.

This is called a \textit{PP attachment} ambiguity: who exactly is using a telescope?

![Figure 1.3: An ambiguous sentence.](image)

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

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

or the description logic statement

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

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”

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

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

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

\textbf{Pragmatics} considers the ways in which meaning is affected by the \textit{context} of a sentence: it includes the study of \textit{discourse} and of \textit{referential expressions}.

As usual, the models have to account for massive ambiguity, as in this \textit{anaphora} resolution:

Mary asks Eve about her father

where \textit{her} might refer to \textit{Mary} or \textit{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 enterprise. To better emphasise 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 generalised to arbitrary lengths $n$, exhibits a number of parses related to the Catalan numbers $C_n \sim \frac{4^n}{n\sqrt{\pi}}$.

- In more recent experiments with treebank-induced grammars, Moore (2004) reports an average number of $7.2 \times 10^{27}$ 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

There is a historical chasm in linguistic modelling between symbolic models and statistical models. This is an important topic, given the success the latter have enjoyed since the eighties (have a look at Pereira, 2000; Steedman, 2011).

Taking the case of syntactic modelling for instance, symbolic models since Chomsky’s seminal work attempt to model the competence of native language speakers to form syntactically correct sentences, by opposition with their actual performance, i.e. the set of sentences uttered in real life. Here, what a linguist ought to model are the rules of grammar, allowing to analyse any sentence, and not only those seen before—mirroring the speaker’s ability to create new sentences.

Nevertheless, statistical models are trained over annotated real life corpora, hence might be taken to be models of performance, inadequate as models of grammar. There is however a simplification in this reasoning: although statistical models are indeed based on real life utterances, they are specifically designed (thanks in particular to smoothing techniques) to allow for previously unseen inputs—this is the very source of their robustness. We will discuss this topic further in Chapter 6.

1.2 Models of Syntax

To conclude this 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.
Flavours of syntactic modelling. For each of the two kinds of analyses, using constituents or dependencies, three different flavours of models can be distinguished:

generative models, which construct syntactic structures through rewrite systems;
model-theoretic approaches rather describe syntactic 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 (see Manning and Schütze [1999]), allowing for more robust modelling. 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 using rewrite systems can be traced back to 350BC and the Sanskrit grammar of Pāṇini, the Aṣṭādhyāyī. This large grammar employs contextual rewrite rules like

\[ A \rightarrow B / C_D \]

for “rewrite A to B in the context C_D”, i.e. the rewrite rule

\[ CAD \rightarrow CBD. \]

The grammar already features auxiliary symbols (like the labels on the inner nodes of Figure 1.2), 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 modelling.

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

\[ A \rightarrow BC | DE. \]

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

\[
\forall x. A(x) \supset \exists y. \exists z. x \downarrow y \land x \downarrow z \land y \rightarrow z \land ((B(y) \land C(z)) \lor (D(y) \land E(z))) \\
\land \forall c. x \downarrow c \supset c = y \lor c = z .
\]
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 4.

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

\[ A(xy) \vdash B(x), C(y). \]  
\[ A(xy) \vdash D(x), E(y). \]

(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

\[ B_1(u_1) \ldots B_m(u_m) \]
\[ A(u_1 \ldots u_m) \]
\[ \{ A(x_1 \ldots x_m) \vdash B_1(x_1), \ldots, B_m(x_m) \} \]

where \( u_1, \ldots, u_m \) are finite strings.

The interest of this proof-theoretic view is that it is readily generalisable 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 1.2, we can observe that it can be encoded through rewrite rules of the form

\[ h \rightarrow L \ast R \]

where \( L \) is the list of left dependents and \( R \) that of right dependents of the head word \( h \), and \( \ast \) marks the position of this word: more concretely, the rules

\[ VBZ \rightarrow \text{PRP} \ast \text{NN} \]  
\[ \text{PRP} \rightarrow \ast \]  
\[ \text{NN} \rightarrow \text{DT} \ast \]

would allow to generate the dependency tree on the right of Figure 1.2. 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 1.2, a dependency tree can be recovered by identifying the head of each phrase as in Figure 1.4. Applying this transformation to a context-free grammar results in a head lexicalised grammar, which is a fairly common idea in statistical parsing (e.g. Charniak, 1997; Collins, 2003).
Figure 1.4: A head lexicalised 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 $A$:

$$\gamma ::= A | \gamma\backslash\gamma | \gamma/\gamma$$  \hspace{1cm} \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:

$$\frac{w_1 \vdash \gamma_1 \quad w_2 \vdash \gamma_2}{w_1 \cdot w_2 \vdash \gamma_1 \backslash \gamma_2} /E \quad \frac{w_1 \vdash \gamma_2/\gamma_1 \quad w_2 \vdash \gamma_1}{w_1 \cdot w_2 \vdash \gamma_2} /E$$

For instance, the dependencies from the right of Figure 1.2 can be described in a lexicon over $A \overset{\text{def}}{=} \{s, n, d\}$ by

She $\vdash$ $n$  watches $\vdash$ $(n\backslash s)/n$  a $\vdash$ $d$  bird $\vdash$ $d\backslash n$

with a proof

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

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 favour 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/](http://www.aclweb.org/anthology/).

The more mathematics-oriented linguistics community is scattered around several sub-communities, each with its own meeting. Let me mention two special interest groups of the ACL: SIGMoL on “Mathematics of Language” and SIGParse on “natural language parsing”.
Chapter 2

Morphology

We consider in this chapter how to represent sets of words of a natural language in linguistically meaningful and computationally efficient ways.

The purpose of morphology is to describe the mechanisms that underlie the formation of words. Intuitively, one can recognise the existence of a relation between the words sings and singing, and further find that the same relation holds between dances and dancing. A morphological analysis of these words

- splits them into basic components, called morphemes: here the stems sing and dance, and the affixes -s and -ing, standing for singular third person and present participle, and thus

- recognises them as inflected forms of the lemmas to sing and to dance.

Already observe some difficulties in the word formation rules: the realisation of the present participle of dance is not *danceing; and some words may be outright irregular, e.g. sang and sung for the preterite and past participle forms of to sing. We will consider formal systems describing the derivation of words.

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? etc. Table 2.1 illustrates the kind of information one can expect to find in a morphological lexicon. If our rules are well-designed, we should be able to extract this information from the various derivations that account for the word.

Table 2.1: Example of entries in English along with their POS tags (from the Penn Treebank tagset) and some morphological features.

<table>
<thead>
<tr>
<th>Input</th>
<th>Lemma</th>
<th>POS</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>race</td>
<td>race</td>
<td>NN</td>
<td>[cat=n; num=sg; case=nom</td>
</tr>
<tr>
<td>races</td>
<td>race</td>
<td>NNS</td>
<td>[cat=n; num=pl; case=nom</td>
</tr>
<tr>
<td>race</td>
<td>to race</td>
<td>VB</td>
<td>[cat=v; mode=inf]</td>
</tr>
<tr>
<td>race</td>
<td>to race</td>
<td>VBP</td>
<td>[cat=v; pers=1</td>
</tr>
<tr>
<td>race</td>
<td>to race</td>
<td>VBZ</td>
<td>[cat=v; pers=3; num=sg; tense=pres; mode=ind]</td>
</tr>
<tr>
<td>race</td>
<td>to race</td>
<td>VBG</td>
<td>[cat=v; pers=2; num=sg; tense=pres; mode=imp]</td>
</tr>
<tr>
<td>raced</td>
<td>to race</td>
<td>VBD</td>
<td>[cat=v; tense=past; mode=ind]</td>
</tr>
<tr>
<td>raced</td>
<td>to race</td>
<td>VBN</td>
<td>[cat=v; mode=ppart]</td>
</tr>
<tr>
<td>racing</td>
<td>to race</td>
<td>VBG</td>
<td>[cat=v; mode=ger]</td>
</tr>
</tbody>
</table>
2.0.1 A Bit of English Morphology

Morphology is the study of the rules of word composition from their basic meaning-bearing elements, known as morphemes.

One usually distinguishes between the main morphemes, called stems (like sing, dance, etc.), that carry the main meaning, from affixes (like -s, -ing, etc.). Four main types of composition rules are commonly considered, and we will briefly review them in the following.

Inflection. Inflectional morphology composes a word stem along with a grammatical morpheme. Inflection can mark various syntactic features like case, tense, mood, genre, or number.

The various inflected forms of race and to race in Table 2.1 show the regular inflections of nouns and verbs in English: the -s plural marking for nouns, and the -s present third person, -ed preterite or past participle, and -ing present participle for verbs. Short gradable adjectives can take a comparative suffix -er (as in cuter) and a superlative suffix -est (as in cutest).

The regular rules are productive in the sense that new-formed words fall prey to them, for instance to twit/twits/twitted/twitting. In contrast, there are few irregular nouns and verbs, but they tend to be very frequent words. For nouns, ox/oxen, mouse/mice, sheep/sheep are a few examples, and for verbs to sing/sang/sung, to eat/ate/eaten, or to cut/cut/cut.

Derivation. A combination of a stem with an affix that results in a different lemma is called a derivation. The obtained word is often of a different category—e.g. from noun to verb with hospital and hospitalise, and back to noun with hospitalisation—, but this is not mandatory—e.g. pseudohospitalisation. Beyond prefixes and suffixes, English can employ expletives such as bloody, motherfucking, sodding etc. as infixes: absobloominglutely, Massafrippingchussets.

Compounding. Like derivation, compounding results in a different lemma, but links several stems: doghouse, bed-time, rock ’n’ roll, bull’s eye, etc. We will not treat compounding, which in practical processing is mostly a tokenisation issue.

Note that there are also finite-state approaches to tokenisation (see e.g. Karttunen et al., 1996, Section 4.1).

Cliticisation. A clitic is a morpheme that acts syntactically like a word, but is bound to another word like an affix. English has auxiliary verbs that may become simple clitics: has/’s, have/’ve, had/’d, am/’m, is/’s, are/’re, will/’ll, and would/’d. Such simple clitics are usually replaced by their expanded forms prior any further processing by the tokeniser.

The possessive marker ’s can also be seen as a special clitic, only applicable to nouns.

Orthographic Rules. In addition to morpheme composition rules, a morphological description has to take some orthographic rules into account. These are often caused by phonological issues.

An -s suffix is turned into -es in ibises, waltzes, thrushes, finches, boxes for nouns, and similarly tosses, waltzes, washes, catches, boxes for verbs. An ending y is turned into ies in butterflies and tries. Regarding -ed and -ing suffixes, ending consonant letters are doubled as in begging, while silent ending ‘e’s are deleted as in dancing.

Other examples of orthographic rules include in- prefixes before some conso-
nants \((b, p, m)\) turning into \textit{im-}, e.g. \textit{impractical}, but this does not apply to \textit{un-}prefixes (unperturbable).

**A Formal Approach.** Let us define the **morphological analysis** problem as the problem, given a single word in isolation, of recovering all its possible structures and morphological features. The exact formulation of course depends on how word structures and morphological features are formalised; we will consider a particular case where a word is decomposed into a sequence of stems, affixes, and feature structures. For instance, from \textit{hospitalised}, we want to recover the sequence \texttt{hospital+ise+ed[cat=v; mode=ppart]}. Note that we also want to recover the sequence \texttt{hospital+ise+ed[cat=v; tense=past; mode=ind]}, i.e. we need to account for ambiguity.

To solve the morphological analysis problem, if the set of words is finite, we can store all the forms in a plain dictionary and simply lookup the various entries. A much more efficient structure is a trie or a directed acyclic graph with the word structure and morphological information attached to the nodes.

Of course, a finite set approach is linguistically unsatisfying: limits to affix stacking are more of a performance issue, and are easily violated by extreme or playful uses, like \textit{antidisestablishmentarianism} or \texttt{*mystery-y-ish-y}. In order to represent infinite sets of words, we switch naturally to automata with outputs, i.e. transducers. We first review some basic results on transducers in \textsection 2.1.1 before returning to morphological analysis in \textsection 2.1.2.

### 2.0.2 Part-of-Speech Tags

**Part-of-speech tags** are refinements of the usual, basic categories like \textit{noun} or \textit{verb}. Different sets of tags (or **tagsets**) will provide different amounts of morphological or syntactic information about a word; for instance, one can see in Table 2.1 that, in the Penn Treebank tagset \cite{Marcus93}, VBP tags verbs in present tense, indicative mode, except for the third person singular case, which uses the VBZ tag. Table 2.2 presents the 36 POS tags of the Penn Treebank tagset, 12 further tags for punctuation and currency symbols being omitted.

By **POS tagging** we refer to the process of associating a POS tag to each word of a sentence. There are two important differences with the morphological analysis problem:

1. we only care about the POS tag, not about other morphological information,
2. the tagging of a word depends on its surrounding context: we should take it into account in order to accurately disambiguate between different possible tags.

For instance, we have several possible tags for \textit{hospitalised}, but the tagging is unambiguous in the context of

\begin{verbatim}
He/PRP hospitalised/VBD his/PP$ mother/NN ./.
His/PP$ mother/NN was/VBD hospitalised/VBN on/IN Saturday/NNP ./.
He/PRP ’s/VBZ visiting/VBG his/PP$ hospitalised/JJ mother/NN ./.
\end{verbatim}

Note that syntactic context is not always enough:

\begin{verbatim}
*Gator/NN attacks/VBZ puzzle/NN experts/NNS
Gator/NN attacks/NNS puzzle/VBP experts/NNS
\end{verbatim}
Table 2.2: The Penn Treebank POS tagset, punctuation excepted (Marcus et al., 1993).

<table>
<thead>
<tr>
<th>Tag</th>
<th>POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
<tr>
<td>CD</td>
<td>Cardinal number</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
</tr>
<tr>
<td>EX</td>
<td>Existential there</td>
</tr>
<tr>
<td>FW</td>
<td>Foreign word</td>
</tr>
<tr>
<td>IN</td>
<td>Preposition or subordinating conjunction</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>JJR</td>
<td>Adjective, comparative</td>
</tr>
<tr>
<td>JJS</td>
<td>Adjective, superlative</td>
</tr>
<tr>
<td>LS</td>
<td>List item marker</td>
</tr>
<tr>
<td>MD</td>
<td>Modal</td>
</tr>
<tr>
<td>NN</td>
<td>Noun, singular or mass</td>
</tr>
<tr>
<td>NNP</td>
<td>Proper noun, singular</td>
</tr>
<tr>
<td>NNPS</td>
<td>Proper noun, plural</td>
</tr>
<tr>
<td>NNS</td>
<td>Noun, plural</td>
</tr>
<tr>
<td>PDT</td>
<td>Predeterminer</td>
</tr>
<tr>
<td>POS</td>
<td>Possessive ending</td>
</tr>
<tr>
<td>PRP</td>
<td>Possessive ending</td>
</tr>
</tbody>
</table>

In fact, newspaper headlines—like the previous example, from AOL News, spotted in a New York Times “On Language” column—can be quite puzzling. Another example, from The Guardian, spotted on Language Log:

May/NNP axes/VBZ Labour/NNP police/NN beat/NN pledge/NN

where May—Theresa May, British Home Secretary—could also be a MD, axes a NNS, and each of Labour, police, beat, and pledge VBPs. A last example, from the BBC news, also spotted on Language Log:

Council/NN hires/VBZ ban/NN bid/NN taxi/NN firm/NN

where hires could also be a NNS, each of ban, bid, and taxi could VBPs, and firm a JJ. This illustrates the amount of ambiguity that POS taggers have to cope with.

The POS tagging task can in fact be decomposed into two subtasks:

1. training a POS tagger from already-tagged data, for instance the annotated texts from the Wall Street Journal present in the Penn Treebank corpus, which represent approximately 50,000 sentences and 1.2 million tokens, and

2. using the tagger on tokenised text.

We describe two finite-state approaches for tackling the POS tagging tasks in Section 2.2.

### 2.1 Finite-State Morphology

#### 2.1.1 Background: Rational Transductions

Whereas the theory of regular languages is typically presented on rational and recognisable subsets of $\Sigma^*$ for $\Sigma$ a finite alphabet, with Kleene’s Theorem stating their equality, the landscape of results changes on products of monoids, for instance on $\Sigma^* \times \Delta^*$ for $\Sigma$ and $\Delta$ two finite alphabets: rational and recognisable
sets do not coincide. Nevertheless, rational subsets of $\Sigma^* \times \Delta^*$, aka rational relations, have an operational presentation through finite-state devices, namely finite transducers.

We only review basic results on rational relations, and briefly mention two subclasses with applications in computational morphology, namely the length preserving relations and sequential functions. We will also draw parallels in Section 2.2.2 between hidden Markov models, which are probabilistic models commonly employed in statistical language processing, and recognisable series.

**Rational Relations.** Observe that $(\Sigma^* \times \Delta^*, \cdot, (\varepsilon, \varepsilon))$ with $(u, v)\cdot (u', v') = (uu', vv')$ is a monoid, generated by $\{\varepsilon \} \times \Delta^* \cup (\Sigma^* \times \{\varepsilon\})$, but not freely generated (e.g. $(a, b) = (a, \varepsilon) \cdot (\varepsilon, b) = (\varepsilon, b) \cdot (a, \varepsilon)$). We use $u:v$ as a shorthand for $(u, v) \in \Sigma^* \times \Delta^*$.

The rational subsets of $\Sigma^* \times \Delta^*$ are defined as per usual: the finite subsets are rational, and we take their closure by union, concatenation and Kleene star. Note that subsets of $\Sigma^* \times \Delta^*$ are relations, hence the name rational relations between $\Sigma^*$ and $\Delta^*$. We can also define rational expressions over $\Sigma^* \times \Delta^*$ using the abstract syntax

$$E ::= \emptyset \mid \varepsilon \cdot \varepsilon \mid a \cdot \varepsilon \mid \varepsilon \cdot b \mid E^* \mid E_1 + E_2 \mid E_1 \cdot E_2$$

with $a \in \Sigma$ and $b \in \Delta$.

A **finite-state transducer** is a finite automaton over $\Sigma^* \times \Delta^*$: $T = (Q, \Sigma^* \times \Delta^*, \delta, I, F)$ with $Q$ a finite set of states, $\delta \subseteq Q \times \Sigma^* \times \Delta^* \times Q$ a finite transition relation, and $I$ and $F$ the initial and final subsets of $Q$. The behaviour of $T$ is a relation in $\Sigma^* \times \Delta^*$ defined by

$$\parallel T \parallel = \{ u:v \mid \delta(I, u:v) \cap F \neq \emptyset \}$$

and is called a rational transduction. Without loss of generality, we can always assume our finite transducers to be normalised, i.e. to be over $(\{\varepsilon\} \times \Delta^*) \cup (\Sigma^* \times \{\varepsilon\})$.

**Exercise 2.1.** Show that the range $R(\Sigma^*)$ of a rational transduction $R$ is a rational language. (*)&

**Exercise 2.2.** Show that if $L$ is a rational language over $\Sigma$, then $\text{Id}_L$ is a rational transduction over $\Sigma^* \times \Sigma^*$. (*)&

**Exercise 2.3.** Show that rational transductions are closed under inverse and composition. (**) &

**Exercise 2.4.** Let $R$ be a relation over $\Sigma^* \times \Delta^*$. Show that $R$ is a rational relation iff it is a rational transduction.

**Remark 2.1** (Non-closure). Rational relations are not closed under intersection:

$$\{e^n:a^mb^m \mid m, n \geq 0\} \cap \{e^n:a^mb^n \mid m, n \geq 0\} = \{e^n:a^nb^n \mid n \geq 0\}$$

has a non-rational language $\{a^nb^n \mid n \geq 0\}$ for range. Thus rational relations are not closed under complement either.

Similarly, let $R = \text{Id}_{\{a\}^*} \circ ba:ab \circ \text{Id}_{\{a\}^*}$ is a rational relation, but its reflexive transitive closure $R^*$ is not: let $L = \{(ab)^n \mid n \geq 0\}$, then

$$\text{Id}_L \circ R^* \circ \{a, b\}^* \cap \{a^nb^m \mid m, n \geq 0\} = \{a^nb^n \mid n \geq 0\}$$

is not a rational language.
Sequential Functions. The equivalence of deterministic and nondeterministic finite state automata breaks when entering the realm of rational relations. The closest substitute for a deterministic transducer is called a sequential transducer. Formally, a sequential transducer from $\Sigma$ to $\Delta$ is a tuple $T = \langle Q, \Sigma, \Delta, q_0, \delta, \eta, \iota, \rho \rangle$ where $\delta : Q \times \Sigma \to Q$ is a partial transition function, $\eta : Q \times \Sigma \to \Delta^*$ a partial transition output function with the same domain as $\delta$, $\iota \in \Delta^*$ is an initial output, and $\rho : Q \to \Delta^*$ is a partial final output function. $T$ defines a partial sequential function $\|T\| : \Sigma^* \to \Delta^*$ with

$$\|T\|(w) = \iota \cdot \eta(q_0, w) \cdot \rho(\delta(q_0, w))$$

for all $w$ in $\Sigma^*$ for which $\delta(q_0, w)$ and $\rho(\delta(q_0, w))$ are defined, where $\eta(q, \varepsilon) = \varepsilon$ and $\eta(q, wa) = \eta(q, w) \cdot \eta(\delta(q, w), a)$ for all $w$ in $\Sigma^*$ and $a$ in $\Sigma$.

(**) Exercise 2.5. Show that sequential transducers are closed under composition.

Normalisation. Let us note $T_q$ for the sequential transducer with $q$ for initial state. The longest common prefix of all the outputs from state $q$ can be written formally as $\bigwedge_{v \in \Sigma^*} \|T_q\|(v)$. A sequential transducer is normalised if this value is $\varepsilon$ for all $q \in Q$ such that $\text{dom}(\|T_q\|) \neq \emptyset$.

(**) Exercise 2.6. Show that any sequential transducer can be normalised.

Minimisation. The translation of a sequential function $f$ by a word $w$ in $\Sigma^*$ is defined by

$$\text{dom}(w^{-1}f) = w^{-1}\text{dom}(f) \quad w^{-1}f(u) = \left( \bigwedge_{v \in \Sigma^*} f(wv) \right)^{-1} f(wu)$$

for all $u$ in $\text{dom}(w^{-1}f)$.

Theorem 2.2 (Raney, 1958). A function $f : \Sigma^* \to \Delta^*$ is sequential iff the set of translations $\{w^{-1}f \mid w \in \Sigma^*\}$ is finite.

As in the finite automata case where minimal automata are isomorphic with residual automata, the minimal sequential transducer for a sequential function $f$ is defined as the translation transducer $\langle Q, \Sigma, \Delta, q_0, \delta, \eta, \iota, \rho \rangle$ where

- $Q = \{w^{-1}f \mid w \in \Sigma^*\}$ (which is finite according to Theorem 2.2),
- $q_0 = \varepsilon^{-1}f$,
- $\iota = \bigwedge_{v \in \Sigma^*} f(v)$ if $\text{dom}(f) \neq \emptyset$ and $\iota = \varepsilon$ otherwise,
- $\delta(w^{-1}, a) = (wa)^{-1}f$,
- $\eta(w^{-1}f, a) = \bigwedge_{v \in \Sigma^*} (w^{-1}f)(av)$ if $\text{dom}((wa)^{-1}f) \neq \emptyset$ and $\eta(w^{-1}f, a) = \varepsilon$ otherwise, and
- $\rho(w^{-1}f) = (w^{-1}f)(\varepsilon)$ if $\varepsilon \in \text{dom}(w^{-1}f)$, and is otherwise undefined.
Recognisable Series. The idea of relations in $\Sigma^* \times \Delta^*$ can be extended to map words of $\Sigma^*$ with values in a semiring $\mathbb{K}$.

A finite weighted automaton (or automaton with multiplicity, or $\mathbb{K}$-automaton) in a semiring $\mathbb{K}$ is a generalisation of a finite automaton: $A = (Q, \Sigma, \mathbb{K}, \delta, I, F)$ 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 $\langle \rho \rangle = kw$ where $w = a_1 \cdots a_n$ is the word label of $\rho$ and $k = I(q_0)k_1 \cdots k_n F(q_n)$ its multiplicity. The behaviour $[A]$ of $A$ is the sum of the monomials for all runs in $A$: it is a formal power series on $\Sigma^*$ with coefficients in $\mathbb{K}$, i.e. a map $\Sigma^* \rightarrow \mathbb{K}$. The coefficient of a word $w$ in $[A]$ is denoted $\langle [A], w \rangle$ and is the sum of the multiplicities of all the runs with $w$ for word label:

$$\langle [A], a_1 \cdots a_n \rangle = \sum_{q_0 \xrightarrow{a_1,k_1} q_1 \xrightarrow{a_2,k_2} q_2 \cdots q_{n-1} \xrightarrow{a_n,k_n} q_n} I(q_0)k_1 \cdots k_n F(q_n).$$

A matrix $\mathbb{K}$-representation for $A$ is $\langle I, \mu, F \rangle$, where $I$ is seen as a row matrix in $\mathbb{K}^{1 \times Q}$, the morphism $\mu : \Sigma^* \rightarrow \mathbb{K}^{Q \times Q}$ 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

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

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

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

Exercise 2.7 (Hadamard Product). Let $\mathbb{K}$ be a commutative semiring. Show that $\mathbb{K}$-recognisable series are closed under product: given two $\mathbb{K}$-recognisable 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 $\Sigma^*$ is $\mathbb{K}$-recognisable. What can you tell about the support of $s \odot s'$?

Exercise 2.8 (Rational relations as series). Given a relation $R$ in $\Sigma^* \times \Delta^*$, define the series $[R]$ by $\langle [R], w \rangle = R(w)$ for all $w$ in $\Sigma^*$. Show that $R$ is rational iff $[R]$ is a $\text{Rat}(\Delta^*)$-recognisable series over $\Sigma$.

2.1.2 Morphological Analysis

Let us return to the problem of morphological analysis. We assume that we have at our disposal a lexicon of all the possible stems, affixes, and feature structures, and want to model how these morphemes can be combined.

Actually, we use in our examples POS tags as shorthand notations for both morphological features and inflectional affixes. For instance, the morphological analysis of hospitalised will rather be hospital-ise[-vbd], where the POS tag VBG, noted [-vbd], is a shorthand notation for both the inflectional affix -ed and the features [cat=v; tense=past; mode=ind]. Accordingly, our lexicon gathers stems, derivational affixes, and POS tags.

As we will see, using finite-state methods solely, we can

1. model the various possible morpheme associations, and their various possible orderings; this is called morphotactics (e.g. [-vbd] can follow any verb stem, but the suffix -ation should be applied to verbs ending with -ise as in hospitalisation),
2. implement the morphological rules (e.g. \([-vbd]\) translates into \(-ed\) for regular verbs, but behaves differently for irregular ones) and orthographic rules (e.g. hospitalised and not *hospitalised).

**Affix Selection and Morphotactics.** The issue of finite-state morphotactics is rather straightforward from a formal language viewpoint: the various orderings can be stored in a simple finite state automaton over the lexicon as alphabet. This automaton can then be turned into one over the Latin alphabet plus POS tags and a morpheme boundary marker “−”—which we will denote by \(\Sigma\) from now on—, and further minimised.

The automaton of Figure 2.1 presents morphotactics that derive for instance hospitalisations (hospital−ize−ation[−nns]) or hospitalised (hospital−ize[−vbd]).

What is the linguistic value of such a finite automaton representation?

- For one thing, it merely stores information about the possible combinations without providing any rationale. Consider for instance the rules of adjective suffixing with the comparative -er: the general rule is that adjectives of two syllables or less can use it, like sadder (produced by sad[−jjr]) or nicer (nice[−jjr]), but not the adjectives with more than two syllables, like *curiouser or *eleganter (see Pesetsky, 1985, for a related discussion). Contrast these affixation constraints with those of un- on adjectives: we can have unwell, unhappy, or uncheerful, but not *unill, *unsad or *unsorrowful. The explanation is that un- can only apply to an adjective with a positive connotation (usually attributed to Zimmer, 1964). Such phonological and semantic explanations are absent from the automaton model.

- Another issue comes from duplication of some parts of the automaton (Karttunen, 1983; Sproat, 1992). For example, contrast enjoyable (en−joy−able) and enrichable (en−rich−able) with the incorrect *joyable and *richable: in these examples, the -able suffix is only acceptable if the en- prefix is present, resulting in duplication in the automaton in order to record whether en- was added or not. This seems to indicate that the finite state model is not perfectly adequate; for instance a more compact representation would be obtained through pushdown automata.
Morphological and Orthographic Rules. A natural way to represent morphological and orthographic rules is to use string rewrite systems. The formation of begging out of the stem beg and the affix -ed can be explained as an application of the morphological rule

\[ [-\text{vbn}] \rightarrow -\text{ed} \quad (2.1) \]

followed by the orthographic rule

\[ g-e \rightarrow gge \quad (2.2) \]

to beg[-vbn]. The one-step derivation relation \( R \) defined by a finite string rewrite system \( R \) is a rational relation, which can also be expressed as \( \text{Id}_{\Sigma^*} \cdot (\sum_{u \rightarrow v \in R} u \cdot v) \cdot \text{Id}_{\Sigma^*} \). For instance this corresponds to

\[ \text{Id}_{\Sigma^*} \cdot g-e:gge \cdot \text{Id}_{\Sigma^*} \quad (2.3) \]

for the one-rule system consisting of rule (2.2), which can be implemented by a finite transducer, as the one of Figure 2.2 for (2.2).

The remainder of this section is dedicated to the translation from string rewriting formalisms into finite state transducers: Section 2.1.3 presents a formalism of cascaded phonological rules to this end.

2.1.3 Phonological Rules

Chomsky and Halle (1968) introduced a particular notation for the rewrite rules used in phonology: the general form of a phonological rule is

\[ \alpha \rightarrow \beta / \lambda \rho \quad (2.4) \]

where \( \alpha, \beta, \lambda, \) and \( \rho \) are rational languages over \( \Sigma \) (for instance represented by rational expressions). Such a rule stands roughly for “rewrite \( \alpha \) into \( \beta \) in the context of \( \lambda, \rho \)”, i.e. for the (generally infinite) string rewrite system

\[ \left\{ x \ u \ y \rightarrow x \ v \ y \mid (x, u, v, y) \in \lambda \times \alpha \times \beta \times \rho \right\} \quad (2.5) \]

— but not quite, as we will see later when considering the implicit restrictions on derivations assumed by phonologists. The same formalism can also be employed for the treatment of morphological and orthographic rules.

Example 2.3. Let us focus for instance on past participle inflection: (2.2) can be restated with this notation as

\[ - \rightarrow g / g \ e \quad (2.6) \]
Keeping with past participle composition, we would also need
\[ - \rightarrow \varepsilon / \_ \_ \_ \] (2.7)
in order to obtain \textit{faxed} from \textit{fax}−\textit{ed} (contexts are left blank for the rational expression \( \varepsilon \)), and
\[ e- \rightarrow \varepsilon / \_ \_e \] (2.8)
in order to obtain \textit{danced} from \textit{dance}−\textit{ed}. On the other hand, the formation of \textit{sung} for \textit{to sing} requires the addition of
\[ \text{sing}_vbn \rightarrow \text{sung} / \_ \_ \_ \] (2.9)
Observe that we should be able to \textit{order} our rules if we want to avoid spurious rewrites, like \textit{*beged} or \textit{*singged}. In our case, we should apply (2.9), then (2.1), then (2.6) and (2.8), and (2.7) last. Furthermore, we should make the rewrites \textit{obligatory}: if (2.9) or (2.6) can be applied, then they should be. But not all rules should be obligatory: for instance, with
\[ \text{prove}_vbn \rightarrow \text{proven} / \_ \_ \_ \] (2.10)
both \textit{proven} and \textit{proved} are accepted as past participles of \textit{to prove}, thus (2.10) should be \textit{optional}. Adding some derivational morphology to the mix allows to witness another issue with rule application:
\[ [-vbg] \rightarrow -ing / \_ \_ \_ \] (2.11)
\[ e- \rightarrow \varepsilon / \_ \_i \] (2.12)
could model gerund inflection in \textit{dancing}. In order to obtain \textit{passivizing}, we need to apply (2.11) once to passive\textit{−ize}[-vbg], and (2.12) on all the applicable factors of passive\textit{−ize}−\textit{ing}: we are actually applying the \textit{transitive closure} of the one-step derivation relation defined by rule (2.12).

To sum up, a \textbf{phonological rule system} consists of a finite sequence \( P = r_1 \cdots r_n \) of phonological rules \( r_i \), each rule \( r_i \) defining a rewrite relation \( [r_i] \) over \( \Sigma^* \times \Sigma^* \), such that the behaviour of \( P \) is the composition \( [P] = [r_1] \# \cdots \# [r_n] \).

\textbf{Restrictions on Rewrites.} In the optional case, the rewrite relation defined by a phonological rule of form (2.4) seems to be exactly the derivation relation of the system (2.5). General string rewrite systems are already Turing-complete in the finite case (in fact, three rules are enough to yield undecidable accessibility (Matiyasevich and Sénizergues, 2005)), thus there is no hope to be able to compute the effect of a phonological rule without further restricting derivations.

Fortunately, linguists give a particular semantics to rewrites: after the application of a rule like (2.4), the newly written word from \( \beta \) cannot be later rewritten. Moreover, rewrites are constrained to occur left-to-right (or right-to-left or simultaneously, but we will only consider the first case).

Formally, given a phonological rule \( r = \alpha \rightarrow \beta / \lambda \cdots \rho \), a \textbf{derivation} \( w_0 \xrightarrow{r} w_1 \Rightarrow \cdots \Rightarrow w_n \) is such that for each \( 0 \leq i < n \), \( w_i = x_i u_i y_i \) and \( w_{i+1} = x_i v_i y_i \) for some \( (x_i, u_i, v_i, y_i) \in (\Sigma^* \times \lambda) \times \alpha \times \beta \times (\rho \cdot \Sigma^*) \). A derivation is \textit{left-to-right} if for each \( 0 \leq i < n - 1 \), \( |x_i v_i| \leq |x_{i+1}| \); we only consider left-to-right derivations in the following. It is furthermore
leftmost if for each $0 \leq i < n$ and for all $(z, z')$ in $\Sigma^* \times \Sigma^*$ such that $zz' = x_iu_i$ and for each $i = 0$, or if $i > 0$ and $|z| > |x_{i-1}v_{i-1}|$, $(z, z'y_i) \notin (\Sigma^*\lambda\alpha) \times (\rho\Sigma^*)$

irreducible if either $n = 0$ and $w_0 \notin \Sigma^*\lambda\alpha\rho\Sigma^*$, or $n > 0$ and for all $z, z'$ in $\Sigma^*$ with $y_{n-1} = z'$, $(x_{n-1}v_{n-1}z, z') \notin (\Sigma^*\lambda\alpha) \times (\rho\Sigma^*)$

Given a phonological rule $r$, its behaviour $\llbracket r \rrbracket$ is a relation over $\Sigma^*$ defined by $w \llbracket r \rrbracket w'$ iff

- there exists a left-to-right derivation $w = w_0 \xrightarrow{r} w_1 \xrightarrow{r} \cdots \xrightarrow{r} w_n = w'$ if $r$ is optional, or iff
- there exists a left-to-right, leftmost, and irreducible derivation $w = w_0 \xrightarrow{r} w_1 \xrightarrow{r} \cdots \xrightarrow{r} w_n = w'$ if $r$ is obligatory.

Note that the definition of left-to-right derivations justifies the context notation in phonological rules: using directly rules of form $\lambda\alpha\rho \rightarrow \lambda\beta\rho$ in left-to-right mode would not allow to later rewrite the factor matched by $\rho$.

Phonological Rules as Rational Relations. We show in this section that the behaviour of a phonological rule $r$ is a rational relation. We only consider the simpler case of optional rules; see Kaplan and Kay (1994) and Mohri and Sproat (1996) for the obligatory case.

First observe that, in a left-to-right derivation $w_0 \xrightarrow{r} w_1 \xrightarrow{r} \cdots \xrightarrow{r} w_n$, since each of the $n$ rewrites has to occur to the right of the previous one, we can decompose each $w_i$ as

$$w_0 = z_0u_0z_1u_1z_2 \cdots z_{n-2}u_{n-1}z_{n-1}$$

$$w_1 = z_0v_0z_1u_1z_2 \cdots z_{n-2}u_{n-1}z_{n-1}$$

$$\vdots$$

$$w_n = z_0v_0z_1v_1z_2 \cdots z_{n-2}u_{n-1}z_{n-1}$$

verifying

$$(z_0v_0z_1 \cdots z_{i-1}, u_i, v_i, z_i \cdots z_{n-2}u_{n-1}z_{n-1}) \in (\Sigma^*\lambda) \times \alpha \times \beta \times (\rho\Sigma^*)$$

(2.13)

for each $0 \leq i < n$.

The second observation is that right contexts can be checked against $w_0$, whereas left contexts should be checked against $w_n$. Hence a decomposition of $\llbracket r \rrbracket$ as the composition of three relations

$$\llbracket r \rrbracket = \text{right}_\rho \circ \text{replace}_{\alpha,\beta} \circ \text{left}_\lambda$$

(2.14)

that respectively check the right contexts, perform the rewrites, and check the left contexts.

It remains to implements these relations as rational transductions. Let us introduce a fresh delimiter symbol $\#$ and the projection $\pi_\# : (\Sigma \cup \{\#\})^* \rightarrow \Sigma^*$. The relation $\text{right}_\rho$ nondeterministically introduces $\#$s before factors in $\rho$. The relation $\text{replace}_{\alpha,\beta}$ replaces a factor $u\#$ in $\alpha\#$ by a factor $\#v$ in $\#\beta$. The relation $\text{left}_\lambda$ erases $\#$s after factors in $\pi^{-1}_\#(\lambda)$.

Exercise 2.9. Propose transducer constructions for each of $\text{right}_\rho$, $\text{replace}_{\alpha,\beta}$, and $\text{left}_\lambda$. (***
Exercise 2.10. Given a rational relation $R$ in $\Sigma^* \times \Delta^*$, build a phonological rule system $P$ of optional rules such that, for all $(w, w')$ in $\Sigma^* \times \Delta^*$, $\$w\$P\$w'\$ iff $wRw'$, where $\$ is an end-of-string marker neither in $\Sigma$ nor in $\Delta$.

Deduce that the morphological analysis problem for phonological rule systems, i.e. given a phonological rule system $P$ of optional rules and a word $w'$, to decide whether there exists $w$ such that $w \parallel P \parallel w'$, is PSPACE-hard.

2.2 Part-of-Speech Tagging

Recall that the POS tagging task consists in assigning the appropriate part-of-speech tag to a word in the context of its sentence. The program that performs this task, the POS tagger, can be learned from an annotated corpus like the Penn Treebank—called supervised learning.

Formally, we are given a finite tagset $\Theta$ and an annotated corpus. For benchmarking purposes, the corpus is typically partitioned into

- a training corpus, on which the tagger is trained,
- optionally a development corpus, used to tune the tagger training algorithm, and
- a test corpus, on which the performance of the tagger is measured.

Thus the training corpus is made of sequences of (word, POS tag) pairs in $\Sigma \times \Theta$, where $\Sigma$ is the set of words in the training corpus. A consequence of this subdivision is that $\Sigma$ is likely to be a strict subset of the set of words in the entire corpus; in particular, there are bound to be unknown words in the test corpus. For instance, Brants (2000) reports that 2.9% of the words in his 10%-sized test set from the Penn Treebank corpus were unknown; unsurprisingly, tagging accuracies tend to be significantly lower for unknown words.

The accuracy of taggers trained on a corpus similar enough to the test set, for instance using a partitioned corpus, is quite high: Brill (1992) reports tagging accuracy scores around 95% using his rule-based tagger on the Brown corpus, while Brants (2000) reports an overall 96.7% accuracy on the WSJ parts of the Penn Treebank with his trigram-HMM tagger (these values are not directly comparable due to differing tagsets and corpora). One has to contrast such numbers with the mean inter-annotator agreement rate: Marcus et al. (1993) report that on average two linguists agree over 96.6% of the tags. Hence the accuracy scores of taggers trained on a corpus similar to the test set is pretty much optimal!

2.2.1 Rule-Based Tagging

The most famous rule-based POS tagging technique is due to Brill (1992). He introduced a three-parts technique comprising:

1. a lexical tagger, which associates a unique POS tag to each word from an annotated training corpus. This lexical tagger simply associates to each known word its most probable tag according to the training corpus annotation, i.e. a unigram maximum likelihood estimation;

2. an unknown word tagger, which attempts to tag unknown words based on suffix or capitalisation features. It works like the contextual tagger, using
the presence of a capital letter and bounded sized suffixes in its rules: for instance, a \textit{-able} suffix usually denotes an adjective;

3. a contextual tagger, on which we focus here. It consists of a cascade of \textbf{contextual rules} of form $uv \rightarrow ubv$ for $a,b$ in $\Theta$ and $uv$ in $\Theta^{\leq k}$ for some predefined $k$, which correct tag assignments based on the $u,v$ contexts. We present in this section how such rules are learned from the training or the development corpus, and how they can be compiled into sequential transducers.

\section*{Learning Contextual Rules}

We start with an example by \textit{Roche and Schabes (1995)}: Let us suppose the following sentences were tagged by the lexical tagger

*Chapman/NNP killed/VBN John/NNP Lennon/NNP
*John/NNP Lennon/NNP was/VBD shot/VBD by/IN Chapman/NNP
He/PRP witnessed/VBD Lennon/NNP killed/VBN by/IN Chapman/NNP

with mistakes in the first two sentences: \textit{killed} should be tagged as a past tense form, and \textit{shot} as a past participle form.

The contextual tagger learns contextual rules of form $uv \rightarrow ubv$ for $a,b$ in $\Theta$ and $uv$ in $\Theta^{\leq k}$ for some predefined $k$; in practice, $k = 2$ or $k = 3$. The learning algorithm simply consists in comparing the effect of all possible contextual rules on the tagging accuracy, and keeping the one with the best results. The learning phase always terminate since a rule is kept only if it actually improves tagging accuracy, and there is only a finite number of possible pairs in $\Sigma \times \Theta$ for each token of the training corpus. In fact, \textit{Brill (1992)} reports that 71 rules are enough when learning on 5\% of the Brown corpus; \textit{Roche and Schabes (1995)} obtain 280 rules on 90\% of the Brown corpus.

For instance, a first contextual rule could be

\begin{equation}
\text{nnp vbn} \rightarrow \text{nnp vbd}
\end{equation}

resulting in a new tagging

Chapman/NNP killed/VBD John/NNP Lennon/NNP
*John/NNP Lennon/NNP was/VBD shot/VBD by/IN Chapman/NNP
*He/PRP witnessed/VBD Lennon/NNP killed/VBN by/IN Chapman/NNP

A second contextual rule could be

\begin{equation}
\text{vbd in} \rightarrow \text{vbn in}
\end{equation}

resulting in the correct tagging

Chapman/NNP killed/VBD John/NNP Lennon/NNP
John/NNP Lennon/NNP was/VBD shot/VBN by/IN Chapman/NNP
He/PRP witnessed/VBD Lennon/NNP killed/VBN by/IN Chapman/NNP

\section*{Contextual Rules as Sequential Functions}

As stated before, our goal is to compile the entire sequence of contextual rules learned from a corpus into a single sequential function.
Let us first formalise the semantics of Brill’s contextual rules. Let \( C = r_1 r_2 \ldots r_n \) be a finite sequence of rewrite rules in \( \Theta^* \times \Theta^* \). In practice the rules constructed in Brill’s contextual tagger are length-preserving and modify a single letter, but this is not a useful consideration from a theoretical viewpoint. Each rule \( r_i = u_i \rightarrow v_i \) defines a leftmost rewrite relation \( \overset{r_i}{\sim} \) defined by

\[
w \overset{r_i}{\sim} w' \text{ iff } \exists x, y \in \Sigma^*, w = xu_iy \land w' = xv_iy \land (\forall z, z' \in \Sigma^*, w \neq zu_i z' \lor x \leq \text{pref } z)
\]

(2.17)

Note that the domain of \( \overset{r_i}{\sim} \) is \( \Theta^* \cdot u_i \cdot \Theta^* \). The behaviour of a single rule is then

\[
\llbracket r_i \rrbracket = \overset{r_i}{\sim} \cup \text{Id}_{\Theta^* \cdot u_i \cdot \Theta^*}
\]

(2.18)

i.e. it applies \( \overset{r_i}{\sim} \) on \( \Theta^* \cdot u_i \cdot \Theta^* \) and the identity on its complement \( \Theta^* \cdot u_i \cdot \Theta^* \). The behaviour of \( C \) is then the composition

\[
\llbracket C \rrbracket = \llbracket r_1 \rrbracket \cdot \llbracket r_2 \rrbracket \cdot \ldots \cdot \llbracket r_n \rrbracket.
\]

(2.19)

A naive implementation of \( C \) would try to match each \( u_i \) at every position of the input string \( w \) in \( \Sigma^* \), resulting in an overall complexity of \( O(|w| \cdot \sum_i |u_i|) \). However, one often faces the problem of tagging a set of sentences \( \{w_1, \ldots, w_m\} \), which yields \( O((\sum_i |u_i|) \cdot (\sum_j |w_j|)) \). As shown in Roche and Schabes’ experiments, compiling \( C \) into a single sequential transducer \( r \) results in practice in huge savings, with overall complexities in \( O(|w| + |T|) \) and \( O(|T| + \sum_j |w_j|) \) respectively.

By (2.19) and the closure of sequential functions under composition, it suffices to prove that \( \llbracket r_i \rrbracket \) is a sequential function for each \( i \) in order to prove that \( \llbracket C \rrbracket \) is a sequential function. Since each \( \llbracket r_i \rrbracket \) is a rational function, being the union of two rational functions over disjoint domains, our efforts are not doomed from the start.

**Sequential Transducer of a Rule.** Intuitively, the sequential transducer for \( \llbracket r_i \rrbracket \) is related to the string matching automaton for \( u_i \), i.e. the automaton for the language \( \Theta^* u_i \). This insight yields a direct construction of the minimal sequential transducer of a contextual rule, with \( |u_i| + 1 \) states in most cases. Let us recall a few definitions:

**Definition 2.4** (Overlap, Border). The **overlap** \( ov(u, v) \) of two words \( u \) and \( v \) is the longest suffix of \( u \) which is simultaneously a prefix of \( v \):

\[
\text{overlap } ov(u, v) = \{ u \} \cap \{ v \}
\]

A word \( u \) is a **border** of a word \( v \) if it is both a prefix and a suffix of \( v \), i.e. if there exist \( v_1, v_2 \in \Theta^* \) such that \( v = vu_1 = v_2u \). For \( v \neq \varepsilon \), the longest border of \( v \) different from \( v \) itself is denoted \( \overline{\text{bord}}(v) \).
Exercise 2.11. Show that for all $u, v$ in $\Theta^*$ and $a$ in $\Theta$, \eqref{ov_eq}:

$$ov(ua, v) = \begin{cases} ov(u, v) \cdot a & \text{if } ov(u, v) \cdot a \leq \text{pref } v \\ \text{bord}(ov(u, v) \cdot a) & \text{otherwise.} \end{cases} \quad (2.20)$$

Definition 2.5 (Transducer of a Contextual Rule). The sequential transducer $T_r$ associated with a contextual rule $r = u \rightarrow v$ with $u \neq \varepsilon$ is defined as

$$T_r = \langle \text{Pref}(u), \Theta, \Theta, \varepsilon, \delta, \eta, \varepsilon, \rho \rangle$$

with the set of prefixes of $u$ as state set, $\varepsilon$ as initial state and initial output, and for all $a$ in $\Theta$ and $w$ in $\text{Pref}(u)$,

$$\delta(w, a) = \begin{cases} wa & \text{if } wa \leq \text{pref } u \\ w & \text{if } w = u \\ \text{bord}(wa) & \text{otherwise} \end{cases}$$

$$\rho(w) = \begin{cases} \varepsilon & \text{if } w \leq \text{pref } (u \land v) \\ (u \land v)^{-1} \cdot w & \text{if } (u \land v) < \text{pref } w < \text{pref } u \\ \varepsilon & \text{otherwise, i.e. if } w = u \end{cases}$$

$$\eta(w, a) = \begin{cases} a & \text{if } wa \leq \text{pref } (u \land v) \\ \varepsilon & \text{if } (u \land v) < \text{pref } wa < \text{pref } u \\ a & \text{if } w = u \\ \rho(w)a \cdot \rho(\text{bord}(wa))^{-1} & \text{otherwise.} \end{cases}$$

For instance, the sequential transducer for the rule $ababb \rightarrow abbbb$ is shown in Figure 2.3 (one can check that $ababb \land abbbb = ab$, bord($b$) = $\varepsilon$, bord($aa$) = $a$, bord($abb$) = $\varepsilon$, bord($abaa$) = $a$, and bord($ababa$) = $aba$).

Proposition 2.6. Let $r = u \rightarrow v$ with $u \neq \varepsilon$. Then $[T_r] = [r]$.

Proof. Let us first consider the case of input words in $\hat{\Theta}\cdot u \cdot \bar{\Theta}$:

Claim 2.6.1. For all $w$ in $\hat{\Theta}\cdot u \cdot \bar{\Theta}$,

$$\delta(\varepsilon, w) = ov(w, u) \quad \eta(\varepsilon, w) = w \cdot \rho(ov(w, u))^{-1}.$$  

Proof of Claim 2.6.1 By induction on $w$: since $u \neq \varepsilon$, the base case is $w = \varepsilon$ with

$$\delta(\varepsilon, \varepsilon) = \varepsilon = ov(\varepsilon, u) \quad \eta(\varepsilon, \varepsilon) = \varepsilon = \varepsilon \cdot \varepsilon^{-1} = \varepsilon \cdot \rho(\varepsilon)^{-1}.$$  

Figure 2.3: The sequential transducer constructed for $ababb \rightarrow abbbb$. 
Case $w'a \not<_{\text{pref}} u$ Then $\eta(w', a) = \rho(w') \cdot a \cdot \rho(\text{border}(w'a))^{-1}$, which yields

\[
\eta(\varepsilon, wa) = w \cdot \rho(w')^{-1} \cdot \rho(w') \cdot a \cdot \rho(\delta(\varepsilon, wa))^{-1} = wa \cdot \rho(\delta(\varepsilon, wa))^{-1}.
\]

\section*{Claim 2.6.1} yields that $[T_r]$ coincides with $[r]$ on words in with $\Theta^* \cdot u \cdot \Theta^*$, i.e. is the identity over $\Theta^* \cdot u \cdot \Theta^*$. Then, since $u \neq \varepsilon$, a word in $\Theta^* \cdot u \cdot \Theta^*$ can be written as $waw'$ with $w$ in $\Theta^* \cdot u \cdot \Theta^*$, $a$ in $\Theta$ with $wa$ in $\Theta^* \cdot u$, and $w'$ in $\Theta^*$. Let $u = u'a$; Claim 2.6.1 implies that

\[
\delta(\varepsilon, w) = u' \quad \quad \eta(\varepsilon, w) = w \cdot \rho(u')^{-1}.
\]

Thus, by definition of $T_r$, $\delta(\varepsilon, wa) = u'a = u$ and

\[
\eta(\varepsilon, wa) = \eta(\varepsilon, w) \cdot \eta(u', a) = w \cdot \rho(u')^{-1} \cdot (u \wedge v)^{-1} \cdot v;
\]

if $(u \wedge v) \not<_{\text{pref}} u'$

\[
\eta(\varepsilon, wa) = w \cdot ((u \wedge v)^{-1} \cdot u')^{-1} \cdot (u \wedge v)^{-1} \cdot v = w \cdot u'^{-1} \cdot v = wa \cdot u^{-1} \cdot v;
\]

otherwise i.e. if $u' = (u \wedge v)$:

\[
\eta(\varepsilon, wa) = w \cdot u'^{-1} \cdot v = wa \cdot u^{-1} \cdot v.
\]
Thus in all cases \([T_r](wa) = [r](wa)\), and since \(T_r\) starting in state \(u\) (i.e. \(T_r(u)\)) implements the identity over \(\Theta^*\), we have more generally \([T_r] = [r]\). □

**Lemma 2.7.** Let \(r = u \rightarrow v\). Then \(T_r\) is normalised.

**Proof.** Let \(w \in \text{Prefix}(u)\) be a state of \(T_r\); we need to show that \(\bigwedge [T_r(w)](\Theta^*) = \varepsilon\).

If \((u \land v) \prec_{\text{pref}} w \prec_{\text{pref}} u\) let \(u' = w^{-1}u \in \Theta^+\), and consider the two outputs

\[
[T_r(w)](u') = \eta(w, u')\rho(u) = (u \land v)^{-1}v \quad [T_r(w)](\varepsilon) = \rho(w) = (u \land v)^{-1}w.
\]

Since \((u \land v) \prec_{\text{pref}} u\) we can write \(u = (u \land v)au''u'\), and either \(v = (u \land v)bv'\) or \(v = u \land v\), for some \(a \neq b\) in \(\Theta\) and \(u'', v'\) in \(\Theta^*\); this yields \(w = (u \land v)au''\) and thus \([T_r(w)](u') \land [T_r(w)](\varepsilon) = \varepsilon\).

**Proposition 2.8.** Let \(r = u \rightarrow v\) with \(u \neq \varepsilon\) and \(u \neq v\). Then \(T_r\) is the minimal sequential transducer for \([r]\).

**Proof.** Let \(w \prec_{\text{pref}} w'\) be two different states in \(\text{Prefix}(u)\); we proceed to prove that \([w^{-1}T_r] \neq [w'^{-1}T_r]\), hence that no two states of \(T_r\) can be merged. By Lemma 2.7 it suffices to prove that \([T_r(w)] \neq [T_r(w')]\), thus to exhibit some \(x \in \Theta^*\) such that \([T_r(w)](x) \neq [T_r(w')]\). We perform a case analysis:

- **if** \(w' \prec_{\text{pref}} (u \land v)\) then \(w \prec_{\text{pref}} (u \land v)\) thus \([T_r(w)](x) = x\) for all \(x \notin w^{-1} \cdot \Theta^*\).
- Otherwise \(u \cdot \Theta^*\) consider
  \[
  [T_r(w)](w'^{-1}u) = w'^{-1}u \neq w'^{-1}v = [T_r(w')]\).

- **if** \(w \leq_{\text{pref}} (u \land v)\) and \(w' = u\) then \([T_r(w')]\)(x) = x for all x and we consider
  \[
  [T_r(w)](w^{-1}u) = w^{-1}u \neq w^{-1}v = [T_r(w')]\).

- **otherwise** that is if \(w \leq_{\text{pref}} (u \land v)\) and \((u \land v) \prec_{\text{pref}} w' \prec_{\text{pref}} u_*,\) or \((u \land v) \prec_{\text{pref}} w \prec_{\text{pref}} w' \leq_{\text{pref}} u_*,\) we have \(\rho(w) \neq \rho(w')\) thus
  \[
  [T_r(w)](\varepsilon) \neq [T_r(w')]\).

**Exercise 2.12.** Define the minimal sequential transducers for \(r = u \rightarrow v\) in the cases \(u = \varepsilon\) and \(u = v\).

### 2.2.2 HMM Tagging

Other approaches to the POS tagging problem rely on probabilistic models to find an appropriate tag sequence given a word sequence. A simple formalism to this end is that of **hidden Markov models (HMM)**, where the observed sequences of symbols (here the words) depend on hidden sequences of states (here the tags) that spanned them.

We need to define a notion of probabilities for sequences. Consider \(n\) variables \(Y_1, \ldots, Y_n\) with values in \(\Sigma\), and a sequence \(w\) of \(n\) words in \(\Sigma\). Variable \(Y_i\) is the
act of observing the $i$th word in the sequence of $n$ words. The probability of a particular sequence $w = a_1 \cdots a_n$ is then

\[
p(a_1 \cdots a_n) = \Pr(Y_1 = a_1, Y_2 = a_2, \ldots, Y_n = a_n) = \Pr(Y_1 = a_1) \cdot \Pr(Y_2 = a_2|Y_1 = a_1) \cdots \Pr(Y_n = a_n|Y_1 = a_1, \ldots, Y_{n-1} = a_{n-1}) = \prod_{i=1}^{n} \Pr(Y_i = a_i|Y_1 = a_1, \ldots, Y_{i-1} = a_{i-1}) .
\]

Add an extra variable $Y_0$ and a “beginning-of-sequence” marker $\$$ with $\Pr(Y_0 = \$$) = 1; we obtain a simpler expression

\[
p(a_1 \cdots a_n) = \prod_{i=1}^{n} \rho(a_i|\$$a_1 \cdots a_{i-1}) .
\]

(2.21)

Hidden Markov model provide a means to define the probability of an observed sequence as the result of another, hidden, sequence of states.

Given a set $S$, $\text{Disc}(S)$ denotes the set of discrete probability distributions over $S$, i.e. $\{p : S \to [0, 1] | \sum_{e \in S} p(e) = 1\}$.

**Definition 2.9 (HMM).** A hidden Markov model is a tuple $H = (Q, \Sigma, S, T, E)$ where $Q$ is a finite set of states, $\Sigma$ a finite output alphabet, $S \in \text{Disc}(Q)$ the starting state probabilities, $T : Q \to \text{Disc}(Q)$ the transition probabilities, and $E : Q \to \text{Disc}(\Sigma)$ the emission probabilities.

The entries of $S$ represent the conditional probability $S(q) = p(q|\$$)$ of starting a sequence of states in state $q$, $T$ the conditional probability $T(q)(q') = p(q'|q)$ of moving to $q'$ when in $q$, and $E$ the conditional probability $E(q)(a) = p(a|q)$ of emitting $a$ when in $q$. The probability for a run $\rho = q_1 \cdots q_n$ to occur is defined to be

\[
p(\rho) = \prod_{i=1}^{n} p(q_i|\$$q_1 \cdots q_{i-1}) = \prod_{i=1}^{n} p(q_i|q_{i-1}) = S(q_1) \cdot \prod_{i=2}^{n} T(q_{i-1})(q_i)
\]

(with $q_0 = \$$), i.e. the conditional probability distribution of the next state depends only upon the current state—the Markov property—, while the probability for this run to emit $w = a_1 \cdots a_n$ is defined to depend solely on the currently visited states,

\[
p(w|\rho) = \prod_{i=1}^{n} p(a_i|q_i) = \prod_{i=1}^{n} E(q_i)(a_i) ;
\]

and the probability of $w$ is thus

\[
p(w) = \sum_{\rho \in Q^n} p(w|\rho) \cdot p(\rho) .
\]

Observe that a HMM defines a discrete probability distribution over $\Sigma^n$ for all $n$:

\[
\forall n, \sum_{w \in \Sigma^n} p(w) = 1 .
\]

(2.22)
Example 2.10. Consider the HMM defined by \( Q = \{q_1, q_2, q_3\} \), \( \Sigma = \{a, b\} \), and

\[
S = \begin{pmatrix} 0.5 & 0.5 & 0 \end{pmatrix} \quad T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \quad E = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0.5 & 0.5 \end{pmatrix}.
\]

It starts with probability 0.5 in either \( q_1 \) or \( q_2 \). Supposing it starts in \( q_2 \), it remains there with probability 0.5 and emits \( a \), or moves to \( q_3 \) and emits \( a \) or \( b \). The run \( q_2q_2 \) has probability \( p(q_2q_2) = 0.25 \) and emits \( aa \) with probability \( p(aa | q_2q_2) = 1 \). There are other runs that emit \( aa \), for instance \( q_3q_3 \) is such that \( p(aa | q_3q_3) = 0.25 \), but \( p(q_3q_3) = 0 \), and in fact there are only two other runs with non-null probability that emit \( aa \): \( p(q_1q_1) = 0.5 \) with \( p(aa | q_1q_1) = 1 \) and \( p(q_2q_3) = 0.25 \) with \( p(aa | q_2q_3) = 0.5 \), thus we have \( p(aa) = 0.875 \).

Constructing HMMs from N-Grams

As we have seen in (2.21), the probability of a given sequence is a complex expression that involves the full history of the sequence at each step. The idea of N-grams is to approximate this full history by considering only the last \( N - 1 \) events as conditioning the current one, i.e. by replacing (2.21) with

\[
p(a_1 \cdots a_n) \approx \prod_{i=1}^{n} p(a_i | a_{i-N+1} \cdots a_{i-1}), \tag{2.23}
\]

(with the convention that \( a_j = \$ \) is a dummy observation for each \( j \leq 0 \)). In the particular cases of \( N = 1 \), \( N = 2 \), and \( N = 3 \), N-grams are called unigrams, bigrams, and trigrams respectively.

Maximum Likelihood Estimation. Suppose now that we have an annotated corpus made of sequences of (word, POS tag) pairs in \( \Sigma \times \Theta \). Then we can estimate the probability of a given tag \( t \) appearing after \( N - 1 \) other tags \( t_1 \cdots t_{N-1} \) by counting the number of occurrences \( C(t_1 \cdots t_{N-1}) \) of the sequence \( t_1 \cdots t_{N-1}t \) and dividing by the number of occurrences of \( C(t_1 \cdots t_{N-1}) \) of \( t_1 \cdots t_{N-1} \):

\[
p(t | t_1 \cdots t_{N-1}) = \frac{C(t_1 \cdots t_{N-1}t)}{C(t_1 \cdots t_{N-1})} \tag{2.24}
\]

(assuming we pad our corpus sequences with dummy \$s both on the left and on the right); this is called a maximum likelihood estimation.

We can build a HMM from such estimations by setting \( Q = (\Theta \cup \{\$\})^N \), i.e. using states of form \( q = t_1 \cdots t_N \), and computing the next state probabilities as

\[
p(t_1' \cdots t_{N'} | t_1 \cdots t_N) = \begin{cases} p(t_N' | t_2 \cdots t_N) & \text{if } \forall 1 \leq i \leq N - 1, t_i' = t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.25}
\]

the initial state probabilities being the particular case \( p(t_1' \cdots t_N' | \$^N) \), and the emission probabilities as

\[
p(a | t_1 \cdots t_N) = \frac{1}{|\Sigma|^{N-1}} \sum_{a_1 \cdots a_{N-1} \in \Sigma^{N-1}} \frac{C((a_1, t_1) \cdots (a, t_N))}{\sum_{a_N \in \Sigma} C((a_1, t_1) \cdots (a_N, t_N))} \tag{2.26}
\]

estimated from occurrences of sequences of pairs. One can then reconstruct a sequence of tags from a sequence of states by projection on the \( N \)th component.
Smoothness. 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. 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 numerators of (2.24) and (2.25) and normalise, called Laplace smoothing), or more involved ones that take into account the probability of observations with a single occurrence (Good-Turing discounting) or the probabilities of \((N-1)\)-grams (interpolation and backoff). A common side-effect of all these techniques is that there are no zero-probability values left in the constructed HMMs.

HMM Decoding

Recall the POS tagging problem: find the best possible sequence of tags \(t_1 \cdots t_n\), given a sequence of words \(w = a_1 \cdots a_n\). Let us assume we are given a HMM model where we can reconstruct the sequence \(t_1 \cdots t_n\) from the most probable execution \(\rho = q_1 \cdots q_n\), that emits \(w\), i.e. we want to compute

\[
\rho = \arg\max_{\rho' \in Q^n} p(\rho' | w) ,
\]

which is also known as HMM decoding. By Bayes’ inversion rule, this is the same as

\[
\rho = \arg\max_{\rho' \in Q^n} \frac{p(w | \rho') p(\rho')}{p(w)} = \arg\max_{\rho' \in Q^n} p(w | \rho') p(\rho') .
\]

The usual procedure to compute the result of (2.28) is to use the Viterbi algorithm, a dynamic programming algorithm. We also present another approach based on weighted automata products and shortest path algorithms, like Dijkstra’s algorithm.

The Viterbi Algorithm. Let \(w = a_1 \cdots a_n\), \(0 \leq i < n\), and consider the maximal joint probability \(V(i + 1, q)\) among all sequences of \(i + 1\) states ending in a given state \(q\) and of a sequence of emissions \(a_1 \cdots a_{i+1}\):

\[
V(i + 1, q) = \max_{\rho' \in Q^i} p(a_1 \cdots a_{i+1} | \rho' q) p(\rho' q) .
\]

For \(i = 0\), this probability is clearly

\[
V(1, q) = p(a_1 | q) p(q | S) = E(q) (a_1) S(q) .
\]

Then, for \(1 \leq i < n\),

\[
V(i + 1, q) = \max_{\rho' \in Q^i, q' \in Q} p(a_1 \cdots a_{i+1} | \rho' q' q) p(\rho' q')
\]

\[
= \max_{\rho' \in Q^i, q' \in Q} p(a_1 \cdots a_i | \rho' q') p(a_{i+1} | q) p(\rho' q') p(q | q')
\]

\[
= \max_{q' \in Q} V(i, q') p(a_{i+1} | q) p(q | q')
\]

\[
= E(q) (a_{i+1}) \max_{q' \in Q} V(i, q') T(q')(q) .
\]
Let us introduce backpointers for the best choice at each step $1 \leq i < n$ and for each state $q$:

$$B(i, q) = \arg\max_{q' \in Q} V(i, q') T(q')(q) .$$

(2.32)

Let $\rho = q_1 \cdots q_n$, then the last state $q_n$ of the most likely explanation is

$$q_n = \arg\max_{q \in \mathcal{Q}} V(n, q) ,$$

and we can work our way back from there using

$$q_i = B(i, q_{i+1}) ,$$

for each $1 \leq i < n$ to reconstruct $\rho$.

**Example 2.11.** For the HMM of Example 2.10 and the input $aa$, we obtain

$$V = \begin{pmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0.25 & 0.125 \end{pmatrix}$$

$$B = (q_1, q_2, q_2)$$

from which we reconstruct the most likely state sequence $q_1q_1$.

**Complexity of the Viterbi Algorithm.** The algorithm proceeds by computing $V(i+1, q)$ for each $0 \leq i < n$ and $q$ in $\mathcal{Q}$; the computation given by (2.31) of one of these probabilities is in $O(|\mathcal{Q}|)$. The complexity of the computation of $V$ dominates the other operations, and the overall complexity is thus in $O(|w||\mathcal{Q}|^2)$.

**Shortest Path Approach.** A HMM defines a rational series $[\mathcal{H}]$ on the probabilistic semiring. Indeed, set $Q' = \mathcal{Q} \cup \{q_0\}$, $I(q_0) = 1$ and $I(q \neq q_0) = 0$ and define the representation $\langle I, \mu, \bar{1} \rangle$ where, for all $a$ in $\Sigma$ and $q, q'$ in $\mathcal{Q}$,

$$\mu(a)(q_0, q_0) = 0 \quad \mu(a)(q_0, q') = S(q')(a) \quad \mu(a)(q, q') = T(q)(q') \cdot E(q')(a) \quad \mu(a)(q_0, q_0) = 0$$

(2.35)

combines the transition and emission probabilities. Observe that the support of this series is prefix-closed: if $\langle [\mathcal{H}], uv \rangle \neq 0$, then $\langle [\mathcal{H}], u \rangle \neq 0$—this is reflected by the $\bar{1}$ final matrix in the representation. Figure 2.4 shows the probabilistic automaton that corresponds to the HMM of Example 2.10.

Our HMM decoding problem then reduces to choosing the path of maximal weight labelled by $w = a_1 \cdots a_n$ in the probabilistic automaton associated to $\mathcal{H}$ by
Indeed, there are in total (2.35):

\[ \rho = \arg\max_{\rho' \in Q^m} p(w|\rho') p(\rho') \]

(2.28)

\[ = \arg\max_{q_1 \cdots q_n} \prod_{i=1}^{n} p(a_i|q_i) p(q_i|q_{i-1}) \]

\[ = \arg\max_{q_1 \cdots q_n} S(q_1) E(q_1)(a_1) \prod_{i=2}^{n} E(q_i)(a_i) T(q_{i-1})(q_i). \]  

(2.36)

Figure 2.5: The tropical automaton \(- \log \mathcal{H}\) for the HMM \(\mathcal{H}\) of Example 2.10

For performance reasons, we rather look for the path of minimal weight in the tropical automaton \(- \log \mathcal{H}\) of representation \((- \log I, - \log \mu, 0)\). (See Figure 2.5) From a practical standpoint, this allows to use addition instead of multiplication, and avoids issues with the floating-point representation of real numbers close to zero. From a theoretical standpoint, a solution to (2.36) becomes

\[ \rho = \arg\min_{q_1 \cdots q_n} \left( -\log S(q_1) - E(q_1)(a_1) - \sum_{i=2}^{n} \log E(q_i)(a_i) + \log T(q_{i-1})(q_i) \right), \]

i.e. a path with weight \(\langle - \log \mathcal{H}, w \rangle\) assigned by the tropical automaton to \(w\).

We can effectively build the product of our weighted automaton \(- \log \mathcal{H}\) with an automaton \(\mathcal{W}\) for the singleton language \(\{ w \}\) (Figure 2.6a). The transition labels in the resulting weighted automaton \(- \log \mathcal{H} + \mathcal{W}\) (Figure 2.6b) are then useless, and we can see it more simply as a weighted graph with weights in \(\mathbb{R}_+\). Adding a single sink node \(s\) with edges \(( (p, q), 0, s)\) for each final state \((p, q)\) of the product automaton (Figure 2.6c) then allows to use a single-pair shortest path algorithm between \((p_0, q_0)\) and \(s\) to find a solution for (2.37) \((q_1, q_1)\) in the example of Figure 2.6.

Complexity of the Shortest Path Approach. Recall that Dijkstra’s algorithm with Fibonacci heaps runs in \(O(m + n \log n)\) in a graph with \(m\) edges and \(n\) vertices. The product automaton \(- \log \mathcal{H} + \mathcal{W}\) has at most \(n = |w| \cdot |Q| + 1\) states (there is a single initial state \((p_0, q_0)\) by construction).

In practice, due to smoothing, the transition relation of \(- \log \mathcal{H}\) is complete except that \(q_0\) does not have any incoming transition: the number of transitions with weight different from \(+\infty\) is \(|Q|^2 + |Q|\). Luckily, the situation is not as bad with \(- \log \mathcal{H} + \mathcal{W}\): its number of transitions is not \(n^2\) but \(m = (|w| - 1) \cdot |Q|^2 + |Q|\).

Indeed, there are in total \(|Q|\) outgoing transitions from \((p_0, q_0)\) to each \((p_1, q)\) with \(q\) in \(Q\), and after that for each \(1 \leq i < |w|\) there are in total \(|Q|^2\) transitions between some state \((p_i, q)\) and some state \((p_{i+1}, q')\) with \(q, q'\) in \(Q\).

Overall, we obtain a complexity of

\[ O((|w| - 1) |Q|^2 + |Q| + (|w| |Q| + 1) \log(|w| |Q| + 1)) \]

\[ = O(|w| |Q|^2 + |w| |Q| \log(|w| |Q|)), \]
which is close enough to that of the Viterbi algorithm.

**Sequential Series Approach.** One might rightly think that, even if we end up with similar complexities, the weighted automata approach induces quite a bit of extra machinery, and is of limited practical interest compared to the Viterbi algorithm.

There is however one case where the weighted automata approach yields practical advantages: when the automaton $-\log \mathcal{H}$ can be **determinised**. Recall that a solution $\rho$ of (2.37) is a path with weight $\langle J - \log \mathcal{H}, w \rangle$. One could thus issue the state information along with this weight and hope to perform HMM tagging deterministically, with a $O(|w| + |A|)$ complexity where $A$ is the determinised and minimised automaton for $-\log \mathcal{H}$.

However, unlike the rule-based tagging technique of Section 2.2.1, there is no general determinisation procedure for (weighted automata translations of) HMMs. Consider for instance the automaton of Figure 2.5 for the HMM of Example 2.10: it implements the series over the tropical semiring defined by

$$\langle s, w \rangle = \begin{cases} 1 & \text{if } w = a^n, \\ n + 2 + |w'| & \text{if } w = a^nbw', \ w' \in \{a, b\}^*. \end{cases}$$

(2.38)

The set of translations $w^{-1}s$ for all $w \in \Sigma^*$ is not finite: for each $m$ and $n$, one gets a different $\langle (a^n)^{-1}s, a^mb \rangle = n + m + 1$; thus by the pendant of Theorem 2.2 for sequential series, $s$ is not sequential (see Lombardy and Sakarovitch (2006), Theorem 8, where $w^{-1}s$ is noted $|w^{-1}s|^\sharp$).

Still, an incomplete determinisation algorithm that might work in practice is described by Mohri (1997), and can be followed by a minimisation step.

A related open problem is the decidability of sequentiaity for rational series over the tropical semiring: see Lombardy and Sakarovitch (2006).
Chapter 3

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 constituents can be recursively decomposed until we reach the individual words, effectively describing a tree:

![Derivation tree](image)

Figure 3.1: A context-free derivation tree.

You have probably recognised 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-rewrite systems.

3.1 Grammars

Definition 3.1 (Phrase-Structured Grammars). A phrase-structured grammar is a tuple $G = (N, \Sigma, P, S)$ where $N$ is a finite nonterminal alphabet, $\Sigma$ 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 $\alpha$ in $V^*$ s.t. $S \Rightarrow^* \alpha$ are called sentential forms, whereas strings $w$ in $\Sigma^*$ s.t. $S \Rightarrow^* w$ are called sentences. The language of $G$ is its set of sentences, i.e.

$$L(G) = L_G(S) \quad L_G(A) = \{ w \in \Sigma^* \mid A \Rightarrow^* w \} .$$

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 3.2. The derivation tree of Figure 3.1 corresponds to the context-free grammar with

\[ N = \{ S, NP, AP, VP, DT, JJ, NNS, VBP, RB \} , \]
\[ \Sigma = \{ those, torn, books, are, completely, worthless \} , \]
\[ P = \{ \]
  \[ S \rightarrow NP \ VP, \]
  \[ NP \rightarrow DT \ NP \mid AP \ NP \mid NNS, \]
  \[ VP \rightarrow VBP \ AP, \]
  \[ AP \rightarrow RB \ AP \mid JJ, \]
  \[ DT \rightarrow Those, \]
  \[ JJ \rightarrow torn \mid worthless, \]
  \[ NNS \rightarrow books, \]
  \[ VBP \rightarrow are, \]
  \[ RB \rightarrow completely \} , \]
\[ S = S . \]

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 formalisation could set \( \Sigma = \{ DT, JJ, NNS, VBP, RB \} \) and delegate the POS tagging issues to an external device.

3.1.1 The Parsing Problem

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

- both their uniform membership problem—i.e. given \( \langle G, w \rangle \) does \( w \in L(G) \)—
  and their emptiness problem—i.e. given \( \langle G \rangle \) does \( L(G) = \emptyset \)—are P-complete (Jones and Laaser, 1976),

- their fixed grammar membership problem—i.e. for a fixed \( G \), given \( \langle w \rangle \)
  does \( w \in L(G) \)—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).

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 \( \langle G, w \rangle \), compute the set of derivation trees of \( G \) that yield \( w \)—the parsing problem.

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 \( O(|G| \ |w|^3) \) 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 refinements of the construction first described by Bar-Hillel et al. (1961) to prove the closure of context-free languages under intersection with recognisable 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 3.2:

She watches a man with a telescope.

She watches a man with a telescope.

Figure 3.2: 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 3.3 (Wich, 2005). The grammar with rules

$$S \rightarrow a S \mid a A \mid \varepsilon, \quad A \rightarrow a S \mid a A \mid \varepsilon$$

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

Such an explosive behaviour is not unrealistic for CFGs in natural languages: Moore (2004) reports an average number of $7.2 \times 10^{27}$ 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.

3.1.2 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 3.4 (Finite Tree Automata). A finite tree automaton (NTA) is a tuple $A = \langle Q, F, \delta, I \rangle$ where $Q$ is a finite set of states, $F$ a ranked alphabet, $\delta$ a finite transition relation in $\bigcup_n Q \times F^n \times Q^n$, and $I \subseteq Q$ a set of initial states.

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

$$R_B = \{ a^{(n)}(q_{1}^{(0)}, \ldots , q_{n}^{(0)}) \rightarrow q^{(0)} \mid (q, a^{(m)}, q_{1}, \ldots , q_{n}) \in \delta \}$$

$$L(A) = \{ t \in T(F) \mid \exists q \in I, t \xrightarrow{R_B}^* q \} ,$$
or top-down:

\[ R_T = \{ q(0) \rightarrow a^{(n)}(q_1^{(0)}, \ldots, q_n^{(0)}) \mid (q, a^{(m)}, q_1, \ldots, q_n) \in \delta \} \]

\[ L(A) = \{ t \in T(F) \mid \exists q \in I, q \xrightarrow{R_T^*} t \} . \]

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

**Example 3.5.** The \( 2^n \) derivation trees for \( a^n \) in the grammar of **Example 3.3** are generated by the \( O(n) \)-sized automaton \( \{qs, qa, q\epsilon, q_1, \ldots, q_n\}, \{S^{(2)}, A^{(2)}, S^{(1)}, A^{(1)}, a^{(0)}, \epsilon\} \)

with rules

\[ \delta = \{ (qs, S^{(2)}, qa, q_1), (qa, a^{(0)}), (q\epsilon, \epsilon^{(0)}) \} \]

\[ \cup \{ (q_i, X, qa, q_{i+1}) \mid 1 \leq i < n, X \in \{S^{(2)}, A^{(2)}\} \} \]

\[ \cup \{ (q_n, X\epsilon) \mid X \in \{S^{(1)}, A^{(1)}\} \} . \]

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 \( A \rightarrow BC \mid a \) are two productions, then an \( A \)-labelled node in a derivation tree might have two children \( B \) or \( C \) or a single child \( a \). This motivates the notation \( A^{(r)} \) for an \( A \)-labelled node with rank \( r \).

**Definition 3.6** (Derived Tree Language). Let \( G = \langle N, \Sigma, P, S \rangle \) be a context-free grammar and let \( m \) be its maximal right-hand side length. Its **derived tree language** \( T(G) \) is defined as the language of the NTA \( A = \langle V \cup \{\epsilon\}, F, \delta, \{S\} \rangle \), where

\[ F = \{ a^{(0)} \mid a \in \Sigma \} \cup \{ \epsilon^{(0)} \} \cup \{ A^{(1)} \mid A \rightarrow \epsilon \in P \} \]

\[ \cup \{ A^{(m)} \mid m > 0 \text{ and } \exists A \rightarrow X_1 \cdots X_m \in P \text{ with } \forall i.X_i \in V \} \]

\[ \delta = \{ (A, A^{(m)}, X_1, \ldots, X_m) \mid m > 0 \land A \rightarrow X_1 \cdots X_m \in P \} \]

\[ \cup \{ (A, A^{(1)}, \epsilon) \mid A \rightarrow \epsilon \in P \} \]

\[ \cup \{ (a, a^{(0)}) \mid a \in \Sigma \} \]

\[ \cup \{ (\epsilon, \epsilon^{(0)}) \} . \]

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

**Exercise 3.1** (Local Tree Languages). Let \( F \) be a ranked alphabet and \( t \) a term of \( T(F) \). We denote by \( r(t) \) the root symbol of \( t \) and by \( b(t) \) the set of **local branches** of \( t \), defined inductively by

\[ r(a^{(0)}) \overset{\text{def}}{=} a^{(0)} \]

\[ b(a^{(0)}) \overset{\text{def}}{=} \emptyset \]

\[ r(f^{(m)}(t_1, \ldots, t_n)) \overset{\text{def}}{=} f^{(m)} \]

\[ b(f^{(m)}(t_1, \ldots, t_n)) \overset{\text{def}}{=} \{ f^{(m)}(r(t_1), \ldots, r(t_n)) \} \cup \bigcup_{i=1}^{n} b(t_i) . \]

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

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

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

then a tree language \( L \) is local if and only if \( L = L(r(L), b(L)) \).
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 \( R \subseteq \mathcal{F}_{>0} \) and \(|R| = 1\) 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 relabelling to the tree nodes.

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

### 3.2 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).

#### 3.2.1 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.

**Theorem 3.7** (Bar-Hillel et al., 1961). Let \( G = \langle N, \Sigma, P, S \rangle \) be a CFG and \( A = \langle Q, \Sigma, \delta, I, F \rangle \) be a NFA. The set of derivation trees of \( G \) with a word of \( L(A) \) as yield is generated by the NTA \( T = ((V \cup \{\varepsilon\}) \times Q \times Q, \Sigma \cup N \cup \{\varepsilon\}, \delta', \{S\} \times I \times F) \) with

\[
\delta' = \{((A, q_0, q_m), A^{(m)}, (X_1, q_0, q_1), \ldots, (X_m, q_m-1, q_m)) \mid m \geq 1, A \rightarrow X_1 \cdots X_m \in P, q_0, \ldots, q_m \in Q \}
\]

\[
\cup \{((A, q, q), A^{(1)}, (\varepsilon, q, q)) \mid A \rightarrow \varepsilon \in P, q \in Q \}
\]

\[
\cup \{((\varepsilon, q, q), \varepsilon^{(0)}) \mid q \in Q \}
\]

\[
\cup \{((a, q, q'), a^{(0)}) \mid (q, a, q') \in \delta \}.
\]

The size of the resulting NTA is in \( O(|G| \cdot |Q|^{m+1}) \) 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 determinise and minimise the resulting tree automaton.

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

**Lemma 3.8.** Given a CFG \( G = \langle \Sigma, N, P, S \rangle \), one can construct in time \( O(|G|) \) an equivalent CFG \( G' = \langle \Sigma, N', P', S \rangle \) in quadratic form s.t. \( V \subseteq V' \), \( L_G(X) = L_{G'}(X) \) for all \( X \) in \( V \), and \( |G'| \leq 5 \cdot |G| \).
Proof. For every production $A \to X_1 \cdots X_m$ of $P$ with $m \geq 2$, add productions

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

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,

$$N' = N \cup \{[\beta] \mid \beta \in V^+ \text{ and } \exists A \in N, \alpha \in V^+, A \to \alpha\beta \in P\}$$
$$\cup \\{[X] \mid X \in V \text{ and } \exists A \in N, \alpha, \beta \in V^+, A \to \alpha X\beta \in P\}$$

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

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

$$X \Rightarrow_{G'}^n w \text{ implies } X \Rightarrow_G^* w$$  \hspace{1cm} (3.1)
$$[\alpha] \Rightarrow_{G'}^n w \text{ implies } \alpha \Rightarrow_G^* w$$  \hspace{1cm} (3.2)

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

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

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

1. $\beta = \varepsilon$ implies immediately $X \Rightarrow_{G'}^* w = \varepsilon$, or
2. $\beta = Y$ in $V$ implies $X \Rightarrow_{G'}^* w$ by induction hypothesis (3.1), or
3. $\beta = [Y][\gamma]$ with $[Y]$ and $[\gamma]$ in $N'$ implies again $X \Rightarrow_{G'}^* w$ by induction hypothesis (3.2) and context-freeness, since in that case $X \Rightarrow Y \gamma$ is in $P$.

Similarly, a derivation

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

implies $\alpha \Rightarrow_G^* w$ by induction hypothesis (3.1) if $|\alpha| = 1$ and thus $\beta = \alpha$, or by induction hypothesis (3.2) and context-freeness if $\alpha = Y \gamma$ with $Y$ in $V$ and $\gamma$ in $V^+$, and thus $\beta = [Y][\gamma]$.

\[ \square \]

### 3.2.2 Parsing as Deduction

In practice, we want to perform at least some of the reduction of the tree automaton constructed by Theorem 3.7 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 $t \Rightarrow^* q$. 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.

$$\frac{(X_1, q_0, q_1), \ldots, (X_m, q_{m-1}, q_m)}{(A, q_0, q_m)} \quad \left\{ \begin{array}{l} A \Rightarrow X_1 \cdots X_m \in P \\ q_0, q_1, \ldots, q_m \in Q \end{array} \right. \quad \text{(Internal)}$$

$$\frac{(a, q, q')}{(q, a, q') \in \delta} \quad \text{(Leaf)}$$

The construction of the NTA proceeds by creating new states following the rules, and transitions of $\delta'$ 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 $m = 0$ ($(A, q_0, q_0), A^{(i)}(\varepsilon, q_0, q_0)$, and one of (Leaf) outputs ($(a, q, q'), a^{(0)}$). We only need to add states $(\varepsilon, q, q)$ and transitions $(\varepsilon, q, \varepsilon^{(0)})$ for each $q$ in $Q$ in order to obtain the co-accessible part of the NTA of Theorem 3.7.

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 premises of the rules and on the side constraints; here it is dominated by the (Internal) rule, with at most $|G| \cdot |Q|^{m+1}$ 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 3.2.** 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 $(A \rightarrow \alpha \cdot \beta, q, q')$, $q, q'$ in $Q$ and $A \rightarrow \alpha \beta$ in $P$, constructed iff

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

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

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

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

This invariant proves the correctness of the algorithm. For a more original proof using abstract interpretation, see Cousot and Cousot (2003).
Rule (Init) initialises the accessibility information, which is propagated by (Predict).
Rule (Scan) initialises the co-accessibility information, which is propagated by (Complete). The intersection is non empty if an item \((S \rightarrow \alpha \cdot q_i, q_f)\) is obtained for some \(q_i\) in \(I\) and \(q_f\) in \(F\).

The algorithm run as a recogniser works in \(O(|G|^2 \cdot |Q|^3)\) regardless of the arity of symbols in \(G\) (Complete dominates this complexity), and can be further optimised to run in \(O(|G| \cdot |Q|^3)\), which is the object of Exercise 3.3. This cubic complexity in the size of the automaton can be understood as the effect of an on-the-fly quadratic form transformation into \(G' = \langle N', \Sigma, P', S' \rangle\) with

\[
\begin{align*}
N' & = \{ S' \} \cup \{ [A \rightarrow \alpha \cdot \beta] \mid A \rightarrow \alpha \beta \in P \} \\
P' & = \{ S' \rightarrow [S \rightarrow \alpha] \mid S \rightarrow \alpha \in P \} \\
& \quad \cup \{ [A \rightarrow \alpha B \cdot \alpha'] \rightarrow [A \rightarrow \alpha \cdot B \alpha'] [B \rightarrow \beta] \mid B \rightarrow \beta \in P \} \\
& \quad \cup \{ [A \rightarrow \alpha a \cdot \alpha'] \rightarrow [A \rightarrow \alpha \cdot a \alpha'] a \mid a \in \Sigma \} \\
& \quad \cup \{ [A \rightarrow \cdot \alpha'] \rightarrow \varepsilon \}.
\end{align*}
\]

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 3.3.

It is easier to output a NTA for this transformed grammar \(G'\):

- create state \(([S \rightarrow \cdot \alpha], q_i, q_f)\) and transition \(([S \rightarrow \cdot \alpha], q_i, q_f, \varepsilon^{(0)})\) when applying (Init),
- create state \(([B \rightarrow \cdot \beta], q', q')\) and transition \(([B \rightarrow \cdot \beta], q', q', \varepsilon^{(0)})\) when applying (Predict),
- create states \(([A \rightarrow \alpha a \cdot \alpha'], q, q'')\) and \((a, q', q'')\), and transitions \(([A \rightarrow \alpha a \cdot \alpha'], q, q''), ([A \rightarrow \alpha a \cdot \alpha']^{(2)}, ([A \rightarrow \alpha a \cdot \alpha'], q', q''), (a, q', q''))\) and \((a, q', q'')\) when applying (Scan),
- create state \(([A \rightarrow \alpha B \cdot \alpha'], q, q'')\) and transition \(([A \rightarrow \alpha B \cdot \alpha'], q, q''), [A \rightarrow \alpha B \cdot \alpha']^{(2)}, ([A \rightarrow \alpha B \cdot \alpha'], q', q''), ([B \rightarrow \beta], q', q'')\) 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 \(((S', q_i, q_f), S')\) for each \(S \rightarrow \alpha \in P\).

(*) Exercise 3.3. How should the algorithm be modified in order to run in time \(O(|G| \cdot |Q|^3)\) instead of \(O(|G|^2 \cdot |Q|^3)\)?

(*) Exercise 3.4. Show that the Earley recogniser works in time \(O(|G| \cdot |Q|^2)\) if the grammar is unambiguous and the automaton deterministic.
Chapter 4

Model-Theoretic Syntax

In contrast with the generative approaches of chapters 3 and 5, we take here a different stance on how to formalise constituent-based syntax. Instead of a more or less operational description using some string or term rewrite system, the trees of our linguistic analyses are now models of logical formulae.

4.1 Introduction

4.1.1 Model-Theoretic vs. Generative

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

Local vs. Global View. The model-theoretic approach simplifies the specification of global properties of syntactic analyses. Let us consider for instance the problem of finding the head of a constituent, which can be used to lexicalise CFGs. Remember that the solution there was to explicitly annotate each nonterminal with the head information of its subtree—which is the only way to percolate the head information up the trees in a context-free grammar. On the other hand, one can write a logic formula postulating the existence of a unique head word for each node of a tree (see (4.19) and (4.20)).

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

* In a hole in 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 obtain through probabilistic methods (cf. Chapter 6), because estimated probabilities are not grammaticality judgements per se, but occurrence-based (although smoothing techniques attempt to account for missing events).
A natural way to obtain a gradience of grammaticality using model theoretic methods is to structure formulæ as large conjunctions $\bigwedge_i \varphi_i$, where each conjunct $\varphi_i$ implements a specific linguistic notion. A degree of grammaticality can be derived from (possibly weighted) counts of satisfied conjuncts.

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

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

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

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

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

Jones believes that iron rusts, and Smith believes that iron rusts, and Jones believes that Smith believes that iron rusts, and Smith believes that Jones believes that iron rusts, and Jones believes that Smith believes that Jones believes that iron rusts, and…

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

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

4.1.2 Tree Structures

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

\begin{align*}
\downarrow & \overset{\text{def}}{=} \{(w, wi) \mid i \in \mathbb{N} \land wi \in W\} \quad (4.1) \\
\rightarrow & \overset{\text{def}}{=} \{(wi, w(i+1)) \mid i \in \mathbb{N} \land w(i+1) \in W\}. \quad (4.2)
\end{align*}

Note that (a) we do not limit ourselves to a single label per node, i.e. we actually prefix-closed isomorphic to some \(\Sigma\overset{\text{def}}{=}2^A\), (b) we do not bound the rank of our trees, and (c) we do not assume the set of labels to be finite.

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

\[
\text{fcns}(\varepsilon) = \varepsilon \quad \text{fcns}(w0) = \text{fcns}(w)0 \quad \text{fcns}(w(i+1)) = \text{fcns}(w)i1 \quad (4.3)
\]

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

\[
\text{fcns}^{-1}(\varepsilon) = \varepsilon \quad \text{fcns}^{-1}(w0) = \text{fcns}^{-1}(w)0 \quad \text{fcns}^{-1}(w1) = \text{fcns}^{-1}(w) + 1 \quad (4.4)
\]

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

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

\begin{align*}
\downarrow_0 & \overset{\text{def}}{=} \{(w, w0) \mid w0 \in \text{dom}(t)\} \quad (4.5) \\
\downarrow_1 & \overset{\text{def}}{=} \{(w, w1) \mid w1 \in \text{dom}(t)\} \quad (4.6) \\
P_a & \overset{\text{def}}{=} \{w \in \text{dom}(t) \mid a \in t(w)\}. \quad (4.7)
\end{align*}

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

\[
\text{dom}(\tilde{t}) = \{\varepsilon, 1\} \cup \{0w \mid w \in \text{dom}(t)\} \cup \{0wi \mid w \in \text{dom}(t) \land i \in \{0, 1\}\} \quad (4.8)
\]

and labels

\[
\tilde{t}(w) = \begin{cases} t(w') & \text{if } w = 0w' \land w' \in \text{dom}(t) \\ \# & \text{otherwise.} \end{cases} \quad (4.9)
\]

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

\section*{4.2 Monadic Second-Order Logic}

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

\[
\psi ::= x = y \mid x \in X \mid x \downarrow y \mid x \rightarrow y \mid P_a(x) \mid \neg \psi \mid \psi \lor \psi \mid \exists x. \psi \mid \exists X. \psi
\]
where $x, y$ range over $X_1$, $X$ over $X_2$, and $a$ over $A$. We write $\text{FV}(\psi)$ for the set of variables free in a formula $\psi$; a formula without free variables is called a \textit{sentence}.

First-order variables are interpreted as nodes in $W$, while second-order variables are interpreted as \textit{finite} subsets of $W$ (it would otherwise be the full second-order logic). Let $\nu : X_1 \to W$ and $\mu : X_2 \to \mathcal{P}_f(W)$ be two corresponding assignments; then the satisfaction relation is defined by

$$
\models_{\nu, \mu} x = y \quad \text{if } \nu(x) = \nu(y)
$$

$$
\models_{\nu, \mu} x \in X \quad \text{if } \nu(x) \in \mu(X)
$$

$$
\models_{\nu, \mu} x \downarrow y \quad \text{if } \nu(x) \downarrow \nu(y)
$$

$$
\models_{\nu, \mu} x \rightarrow y \quad \text{if } \nu(x) \rightarrow \nu(y)
$$

$$
\models_{\nu, \mu} P_a(x) \quad \text{if } P_a(\nu(x))
$$

$$
\models_{\nu, \mu} \neg \psi \quad \text{if } M \not\models_{\nu, \mu} \psi
$$

$$
\models_{\nu, \mu} \psi \lor \psi' \quad \text{if } M \models_{\nu, \mu} \psi \text{ or } M \models_{\nu, \mu} \psi'
$$

$$
\models_{\nu, \mu} \exists x. \psi \quad \text{if } \exists w \in W, \models_{\nu_{\{x=w\}}, \mu} \psi
$$

$$
\models_{\nu, \mu} \exists X. \psi \quad \text{if } \exists U \subseteq W, U \text{ finite } \land M \models_{\nu, \mu}(X \cup \{U\}) \psi.
$$

As usual, we define conjunctions as $\psi \land \psi' \overset{\text{def}}{=} \neg \neg \psi \lor \neg \psi'$, implications as $\psi \supset \psi' \overset{\text{def}}{=} \neg \psi \lor \psi'$, and equivalences as $\psi \equiv \psi' \overset{\text{def}}{=} \psi \lor \psi' \land \psi' \lor \psi$.

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

$$
\psi' \overset{\text{def}}{=} \exists \text{FV}(\psi). \psi \land \left( \bigwedge_{x \in X_1 \cap \text{FV}(\psi)} P_a(x) \land \forall y. x \neq y \supset \neg P_a(y) \right)
$$

$$
\land \left( \bigwedge_{X \in X_2 \cap \text{FV}(\psi)} \forall y. y \in X \equiv P_X(y) \right).
$$

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

\textbf{Example 4.1.} Here are a few useful wMSO formulæ: To allow any label in a finite set $B \subseteq A$:

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

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

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

$$
\text{root}(x) \overset{\text{def}}{=} \neg \exists y. y \downarrow x
$$

$$
\text{leaf}(x) \overset{\text{def}}{=} \neg \exists y. x \downarrow y
$$

$$
\text{internal}(x) \overset{\text{def}}{=} \neg \text{leaf}(x)
$$

$$
\text{children}(x, X) \overset{\text{def}}{=} \forall y. y \in X \equiv x \downarrow y
$$

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

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

(4.10)

For example,

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

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

4.2.1 Linguistic Analyses in wMSO

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).

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

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

Define the formula

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

which forces the root label to be $S$;

$$\land \forall x. \text{internal}(x) \supset \bigvee_{a \in N \cup \Theta} P_a(x) \land \bigwedge_{b \in N \cup \Theta \setminus \{a\}} \neg P_b(x)$$

(4.12)

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

$$\land \forall x. \text{leaf}(x) \supset \neg P_{N \cup \Theta}(x)$$

(4.13)

forbids grammatical labels on leaves;

$$\land \forall y. \text{leaf}(y) \supset \exists x. x \downarrow^* y \land P_\Theta(x)$$

(4.14)

expresses that leaves should have POS-labelled parents;

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

(4.15)

verifies that internal nodes at distance at least two from some leaf should have labels drawn from $N$, and are thus not POS-labelled by (4.12), and thus cannot have a leaf as a child by (4.13);

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

(4.16)

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

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

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

(4.17)
makes sure that only leaves can be labelled by words, and that when a word is
known (i.e. if it appears in \(L\)), it should have one of its allowed POS tag as imme-
diate parent. If the current POS tagging information of our lexicon is incomplete,
then this particular constraint will not be satisfied. For an unknown word however,
any POS tag can be used.

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

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

(4.18)

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

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

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

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

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

**Head Percolation.** The first step is to find which child is the head among its
siblings; several heuristics have been developed to this end, and a simple way to
describe such heuristics is to use a head percolation function \(h : N \rightarrow \{l, r\} \times (N \cup \Theta)^*\) that describes for a given parent label \(A\) a list of potential labels \(X_1, \ldots, X_n\)
in \(N \cup \Theta\) in order of priority and a direction \(d \in \{l, r\}\) standing for “leftmost” or
“rightmost”: such a value means that the leftmost (resp. rightmost) occurrence of
\(X_1\) is the head, this unless \(X_1\) is not among the children, in which case we should
try \(X_2\) and so on, and if \(X_n\) also fails simply choose the leftmost (resp. rightmost)
child (see e.g. [Collins, 1999](#collins1999) Appendix A). For instance, the function

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

would result in the correct head annotations in Figure 6.1.
Given such a head percolation function $h$, we can express the fact that a given node is a head:

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

(4.19)

$$\text{head}_{d,X\beta}(x, Y) \equiv \neg \text{priority}_{d,X}(x, Y) \lor (\text{head}_{d,X\beta}(x, Y) \land \neg P_X(Y))$$

$$\text{head}_{l,X}(x, Y) \equiv \forall y. y \in Y \supset x \rightarrow y$$

$$\text{head}_{r,X}(x, Y) \equiv \forall y. y \in Y \supset y \rightarrow x$$

$$\text{priority}_{l,X}(x, Y) \equiv P_X(x) \land \forall y. y \in Y \land y \rightarrow x \supset \neg P_X(y)$$

$$\text{priority}_{r,X}(x, Y) \equiv P_X(x) \land \forall y. y \in Y \land x \rightarrow y \supset \neg P_X(y)$$

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

$$\text{lexicalise}(x, y) \equiv \text{leaf}(y) \land x \downarrow [\text{TC}_{u,v}, u \downarrow v \land \text{head}(v)] y$$

(4.20)

This formula recovers the lexical information in Figure 6.1.

**Exercise 4.1.** Propose wMSO formulae to recover the parent and lexical POS information in constituent trees, as illustrated in Figure 6.1. (*)

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

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

$$\land \neg \text{lexicalise}(y', x) \land \text{lexicalise}(y', y)$$

(4.21)

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 4.2.

**Exercise 4.2.** Modify (4.21) to make sure that any leaf with a parent tagged by the POS RB modifies either a verb or an adjective. (*)

**Exercise 4.3.** Consider the $\varepsilon$ node in Figure 4.2: modify (4.20) to recover that who lexicalises the bottommost NP node. (**)
4.2.2 MSO on Finite Binary Trees

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

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

$$L(A_\psi) = \{ \bar{t} \in T(\Sigma \cup \{ \# \}) \mid t \text{ finite } \land t \models \psi \} \quad (4.22)$$

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

$$\psi' \overset{\text{def}}{=} \psi \land \exists x. \neg (\exists z. \downarrow_0 x \lor z \downarrow_1 x) \land \neg (\exists y. x \downarrow_1 y) \land \forall x. \forall y. (P_\#(x) \land (x \downarrow_0 y \lor x \downarrow_1 y)) \lor P_\#(y) \lor \text{root}(x) \quad (4.23)$$

where the conditions ensure the root does not have any right child, and where $\psi$ uses the macros

$$x \downarrow y \overset{\text{def}}{=} \exists x_0. x \downarrow_0 x_0 \land (x_0 [\text{TC}_u,v u \downarrow_1 v] y) \quad (4.24)$$

$$x \rightarrow y \overset{\text{def}}{=} x \downarrow_1 y \quad (4.25)$$

$$\exists x. \psi \overset{\text{def}}{=} \exists x. \neg P_\#(x) \land \psi \quad (4.26)$$

$$\exists X. \psi \overset{\text{def}}{=} \exists X. (\forall x. x \in X \supset \neg P_\#(x)) \land \psi \quad (4.27)$$

The conclusion of this construction is

**Theorem 4.2.** Satisfiability and model-checking for wMSO are decidable.

(∗) **Exercise 4.4** (ω Successors). Show that the weak second-order logic of ω successors (wSωS), i.e. with $\downarrow_i \overset{\text{def}}{=} \{(w, wi) \mid wi \in W\}$ defined for every $i \in \mathbb{N}$, has decidable satisfiability and model-checking problems.
4.3 Propositional Dynamic Logic

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

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

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

(path formulae)

where $a$ ranges over $A$.

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

$$
\begin{align*}
\llbracket a \rrbracket &\defeq \{ w \in W | P_a(w) \} & \llbracket \downarrow \rrbracket &\defeq \downarrow \\
\llbracket \top \rrbracket &\defeq W & \llbracket \rightarrow \rrbracket &\defeq \rightarrow \\
\llbracket \neg \varphi \rrbracket &\defeq W \setminus \llbracket \varphi \rrbracket & \llbracket \pi^{-1} \rrbracket &\defeq \llbracket \pi \rrbracket^{-1} \\
\llbracket \varphi \lor \varphi' \rrbracket &\defeq \llbracket \varphi \rrbracket \cup \llbracket \varphi' \rrbracket & \llbracket \pi; \pi' \rrbracket &\defeq \llbracket \pi \rrbracket \uplus \llbracket \pi' \rrbracket \\
\llbracket \langle \pi \rangle \varphi \rrbracket &\defeq \llbracket \pi \rrbracket^{-1}(\llbracket \varphi \rrbracket) & \llbracket \pi^* \rrbracket &\defeq \llbracket \pi \rrbracket^* \\
\llbracket \varphi^? \rrbracket &\defeq \text{Id}_{\llbracket \varphi \rrbracket}.
\end{align*}
$$

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

We define the classical dual operators

$$
\begin{align*}
\bot &\defeq \neg \top & \varphi_1 \land \varphi_2 &\defeq \neg (\neg \varphi_1 \lor \neg \varphi_2) & [\pi] \varphi &\defeq \neg (\langle \pi \rangle \neg \varphi).
\end{align*}
$$

We also define

$$
\begin{align*}
\uparrow &\defeq \downarrow^{-1} & \leftarrow &\defeq \rightarrow^{-1} \\
\text{root} &\defeq \llbracket \uparrow \rrbracket \bot & \text{leaf} &\defeq \llbracket \downarrow \rrbracket \bot \\
\text{first} &\defeq \llbracket \leftarrow \rrbracket \bot & \text{last} &\defeq \llbracket \rightarrow \rrbracket \bot.
\end{align*}
$$

Exercise 4.5 (Converses). Prove the following equivalences:

$$
\begin{align*}
(\pi_1; \pi_2)^{-1} &\equiv \pi_2^{-1}; \pi_1^{-1} \tag{4.29} \\
(\pi_1 + \pi_2)^{-1} &\equiv \pi_1^{-1} + \pi_2^{-1} \tag{4.30} \\
(\pi^*)^{-1} &\equiv (\pi^{-1})^* \tag{4.31} \\
(\varphi^?)^{-1} &\equiv \varphi^? \tag{4.32}
\end{align*}
$$

Exercise 4.6 (Reductions). Prove the following equivalences:

$$
\begin{align*}
\langle \pi_1; \pi_2 \rangle \varphi &\equiv \langle \pi_1 \rangle \langle \pi_2 \rangle \varphi \tag{4.33} \\
\langle \pi_1 + \pi_2 \rangle \varphi &\equiv \langle \pi_1 \rangle \varphi \lor \langle \pi_2 \rangle \varphi \tag{4.34} \\
\langle \pi^* \rangle \varphi &\equiv \varphi \lor \langle \pi; \pi^* \rangle \varphi \tag{4.35} \\
\langle \varphi^? \rangle \varphi_2 &\equiv \varphi_1 \land \varphi_2 \tag{4.36}
\end{align*}
$$
4.3.1 Model-Checking

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

4.3.2 Satisfiability

Unlike the model-checking problem, the satisfiability problem for PDL is rather demanding: it is \( \text{ExpTime}\)-complete.

Theorem 4.3 (Fischer and Ladner, 1979). Satisfiability for PDL is \( \text{ExpTime}\)-hard.

As with wMSO, it is more convenient to work on binary trees \( t \) of the form \( \langle \text{dom}(t), \downarrow_0, \downarrow_1, (P_a)_{a \in A} \{0, 1\} \rangle \) 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 4.2.2 we can interpret these relations in PDL by

\[
\downarrow \overset{\text{def}}{=} \downarrow_0; \downarrow_1 \\
\rightarrow \overset{\text{def}}{=} \downarrow_1
\]

and translate any PDL formula \( \varphi \) into a formula

\[
\varphi' \overset{\text{def}}{=} \varphi \land ([\uparrow^*; \downarrow^*; \downarrow_0]0 \land \neg 1) \land ([\uparrow^*; \downarrow^*; \downarrow_1]1 \land \neg 0) \land [\uparrow^*; \text{root}?; \downarrow_1] \bot
\]

that checks that \( \varphi \) holds, that the 0 and 1 labels are correct, and verifies \( M, w \models \varphi \) iff \( \text{fcns}(M), \text{fcns}(w) \models \varphi' \). The conditions in (4.38) ensure that the tree we are considering is the image of some tree structure by \( \text{fcns} \): we first go back to the root by the path \( \uparrow^*; \text{root}? \), and then verify that the root does not have a right child.

Normal Form. Let us write

\[
\uparrow_0 \overset{\text{def}}{=} \downarrow^{-1}_0 \\
\uparrow_1 \overset{\text{def}}{=} \downarrow^{-1}_1
\]

then using the equivalences of Exercise 4.5 we can reason on PDL with a restricted path syntax

\[
\alpha := \downarrow_0 \mid \uparrow_0 \mid \downarrow_1 \mid \uparrow_1 \\
\pi := \alpha \mid \pi \mid \pi + \pi \mid \pi^* \mid \varphi^?
\]

and using the dualities of (4.28), we can restrict node formulæ to be of form

\[
\varphi := a \mid \neg a \mid T \mid \bot \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \langle \pi \rangle \varphi \mid [\pi] \varphi
\]

(node formulæ)

Lemma 4.4. For any PDL formula \( \varphi \), we can construct an equivalent formula \( \varphi' \) in normal form with \( |\varphi'| = O(|\varphi|) \).

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

The equivalences found in Exercise 4.6 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 $\phi$ is the smallest set $S$ of formulæ in normal form s.t.

1. $\phi \in S$,
2. if $\phi_1 \lor \phi_2 \in S$ or $\phi_1 \land \phi_2 \in S$ then $\phi_1 \in S$ and $\phi_2 \in S$,
3. if $\langle \pi \rangle \phi' \in S$ or $[\pi] \phi' \in S$ then $\phi' \in S$,
4. if $\langle \pi_1; \pi_2 \rangle \phi' \in S$ then $\langle \pi_1 \rangle \langle \pi_2 \rangle \phi' \in S$,
5. if $[\pi_1; \pi_2] \phi' \in S$ then $[\pi_1][\pi_2] \phi' \in S$,
6. if $\langle \pi_1 + \pi_2 \rangle \phi' \in S$ then $\langle \pi_1 \rangle \phi' \in S$ and $\langle \pi_2 \rangle \phi' \in S$,
7. if $[\pi_1 + \pi_2] \phi' \in S$ then $[\pi_1] \phi' \in S$ and $[\pi_2] \phi' \in S$,
8. if $\langle \pi^* \rangle \phi' \in S$ then $\langle \pi \rangle \langle \pi^* \rangle \phi' \in S$,
9. if $[\pi^*] \phi' \in S$ then $[\pi][\pi^*] \phi' \in S$,
10. if $\langle \phi_1 ? \rangle \phi_2 \in S$ or $[\phi_1 ?] \phi_2 \in S$ then $\phi_1 \in S$.

We write $\text{FL}(\phi)$ for the Fisher-Ladner closure of $\phi$.

**Lemma 4.5.** Let $\phi$ be a PDL formula in normal form. Its Fisher-Ladner closure is of size $|\text{FL}(\phi)| \leq |\phi|$.

We construct a surjection $\sigma$ between positions $p$ in the term $\phi$ and the formulae in $S$:

- for positions $p$ spanning a node subformula $\text{span}(p) = \phi_1$, we can map to $\phi_1$ (this corresponds to cases 1–3 and 10 on subformulae of $\phi'$);

![Figure 4.3: The surjection $\sigma$ from positions in $\phi \overset{\text{def}}{=} [\phi_1?; \pi_1^*; \pi_2] \phi_2$ to $\text{FL}(\phi)$ (dashed), and the rules used to construct $\text{FL}(\phi)$ (dotted).]
• for positions \( p \) spanning a path subformula \( \text{span}(p) = \pi \), we find the closest ancestor spanning a node subformula (thus of form \( \langle \pi' \rangle \varphi_1 \) or \( [\pi'] \varphi_1 \)). If \( \pi = \pi' \) we map \( p \) to the same \( \langle \pi' \rangle \varphi_1 \) or \( [\pi'] \varphi_1 \). Otherwise we consider the parent position \( p' \) of \( p \), which is mapped to some formula \( \sigma(p') \), and distinguish several cases:

- for \( \sigma(p') = \langle \pi_1; \pi_2 \rangle \varphi_2 \) we map \( p \) to \( \langle \pi_1 \rangle \varphi_2 \) if \( \text{span}(p) = \pi_1 \) and to \( \langle \pi_2 \rangle \varphi_2 \) if \( \text{span}(p) = \pi_2 \) (this matches case 4 and the further application of 3);
- for \( \sigma(p') = [\pi_1; \pi_2] \varphi_2 \) we map \( p \) to \( [\pi_1] \varphi_2 \) if \( \text{span}(p) = \pi_1 \) and to \( [\pi_2] \varphi_2 \) if \( \text{span}(p) = \pi_2 \) (this matches case 5 and the further application of 3);
- for \( \sigma(p') = (\pi_1 + \pi_2) \varphi_2 \) and \( \text{span}(p) = \pi_1 \) with \( i \in \{1, 2\} \), we map \( p \) to \( \langle \pi_i \rangle \varphi_2 \) (this matches case 6);
- for \( \sigma(p') = (\pi_1 + \pi_2) \varphi_2 \) and \( \text{span}(p) = \pi_2 \) with \( i \in \{1, 2\} \), we map \( p \) to \( [\pi_i] \varphi_2 \) (this matches case 7);
- for \( \sigma(p') = (\pi^*) \varphi_2 \), \( \text{span}(p) = \pi \) and we map \( p \) to \( \langle \pi \rangle (\pi^*) \varphi_2 \) (this matches case 8);
- for \( \sigma(p') = [\pi^*] \varphi_2 \), \( \text{span}(p) = \pi \) and we map \( p \) to \( [\pi[\pi^*]] \varphi_2 \) (this matches case 9).

The function \( \sigma \) we just defined is indeed surjective: we have covered every formula produced by every rule. Figure 4.3 presents an example term and its mapping.

### Reduced Formulae

**Reduced Normal Form.** We try now to reduce formulæ into a form where any modal subformula is under the scope of some atomic modality \( \langle \alpha \rangle \) or \( [\alpha] \). Given a formula \( \varphi \) in normal form, this is obtained by using the equivalences of Exercise 4.6 and their duals, and by putting the formula into disjunctive normal form, i.e.

\[
\varphi \equiv \bigvee_i \bigwedge_j \chi_{i,j} \quad (4.39)
\]

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

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

(reduced formulæ)

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

**Lemma 4.6.** Given a PDL formula \( \varphi \) in normal form, we can construct an equivalent formula \( \bigvee_i \bigwedge_j \chi_{i,j} \) where each \( \chi_{i,j} \) is a reduced formula in FL(\( \varphi \)).

### Two-Way Alternating Tree Automata

We finally turn to the construction of a tree automaton that recognises the models of a normal form formula \( \varphi \). To simplify matters, we use a powerful model for this automaton: a **two-way alternating tree automaton** (2ATA) over finite ranked trees.
Definition 4.7. A two-way alternating tree automaton (2ATA) is a tuple $A = \langle Q, \Sigma, q_i, F, \delta \rangle$ where $Q$ is a finite set of states, $\Sigma$ is a ranked alphabet with maximal rank $k$, $q_i \in Q$ is the initial state, and $\delta$ is a transition function from pairs of states and symbols $(q, a)$ in $Q \times \Sigma$ to positive Boolean formulae $f$ in $B_+((-1, \ldots, k) \times Q)$, defined by the abstract syntax

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

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

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

1. $\varepsilon$ is in $\operatorname{dom}(\rho)$ with $\rho(\varepsilon) = (\varepsilon, q_i)$,

2. if $w$ is in $\operatorname{dom}(\rho)$, $\rho(w) = (u, q)$ 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

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

with each $u_i' \in \operatorname{dom}(t)$.

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

Theorem 4.8 (Vardi, 1998). Given a 2ATA $A = \langle Q, \Sigma, q_i, F, \delta \rangle$, deciding the emptiness of $L(A)$ can be done in deterministic time $|\Sigma| \cdot 2^{O(k |Q|^3)}$.

Automaton of a Formula. Let $\varphi$ be a formula in normal form. We want to construct a 2ATA $A_\varphi = \langle Q, \Sigma, q_i, \delta \rangle$ that recognises exactly the closed models of $\varphi$, so that we can test the satisfiability of $\varphi$ by Theorem 4.8. We assume wlog. that $A \subseteq \operatorname{Sub}(\varphi)$. We define

$$Q \overset{\text{def}}{=} \operatorname{FL}(\varphi) \uplus \{q_i, q_{\varphi}, q_\#\},$$

$$\Sigma \overset{\text{def}}{=} \{\#(0), \#(2)\} \cup \{a^{(2)} \mid a \subseteq A \uplus \{0, 1\}\}.$$ 

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

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

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

\[
\delta(\langle \downarrow 0 \rangle \varphi', a) \overset{\text{def}}{=} (1, \varphi') \quad \delta([\downarrow 0] \varphi', a) \overset{\text{def}}{=} (1, \varphi') \lor (1, q_\#)
\]

\[
\delta(\langle \downarrow 1 \rangle \varphi', a) \overset{\text{def}}{=} (2, \varphi') \quad \delta([\downarrow 1] \varphi', a) \overset{\text{def}}{=} (2, \varphi') \lor (2, q_\#)
\]

\[
\delta(\langle \uparrow 0 \rangle \varphi', a) \overset{\text{def}}{=} (-1, \varphi') \land (0, 0) \quad \delta([\uparrow 0] \varphi', a) \overset{\text{def}}{=} (-1, \varphi') \land (0, 0) \lor (-1, q_\#) \lor (0, 1)
\]

\[
\delta(\langle \uparrow 1 \rangle \varphi', a) \overset{\text{def}}{=} (-1, \varphi') \land (0, 1) \quad \delta([\uparrow 1] \varphi', a) \overset{\text{def}}{=} (-1, \varphi') \land (0, 1) \lor (-1, q_\#) \lor (0, 0)
\]

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 \( # \):

\[
\delta(q_\#, #) \overset{\text{def}}{=} \top \quad \delta(q_\#, a) \overset{\text{def}}{=} \bot.
\]

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

\[
\delta(q_i, #) \overset{\text{def}}{=} (1, q_\varphi) \land (2, q_\#) \quad \delta(q_i, a) \overset{\text{def}}{=} \bot
\]

\[
\delta(q_\varphi, a) \overset{\text{def}}{=} \delta(\varphi, a) \land (2, q_\#).
\]

For any state \( q \) beside \( q_i \) and \( q_\# \)

\[
\delta(q, #) \overset{\text{def}}{=} \bot.
\]

**Corollary 4.9.** Satisfiability of PDL can be decided in \( \text{EXPTIME} \).

**Proof sketch.** Given a PDL formula \( \varphi \), by **Lemma 4.4** construct an equivalent formula in normal form \( \varphi' \) with \( |\varphi'| = O(|\varphi|) \). We then construct \( \mathcal{A}_{\varphi'} \) with \( O(|\varphi|) \) states by **Lemma 4.5** and an alphabet of size at most \( 2^{O(|\varphi|)} \), s.t. \( \bar{t} \) is accepted by \( \mathcal{A}_{\varphi'} \) iff \( t, \text{root} \models \varphi \). By Theorem 4.8 we can decide the existence of such a tree \( \bar{t} \) in time \( 2^{O(|\varphi|)} \). The proof carries to satisfiability on tree structures rather than binary trees. \( \square \)

### 4.3.3 Expressiveness

**Monadic Transitive Closure.** PDL can be expressed in \( \text{FO}[\text{TC}^1] \) the first-order logic with monadic transitive closure. The translation can be expressed by induction, yielding formulæ \( \text{ST}_x(\varphi) \) with one free variable \( x \) for node formulæ and \( \text{ST}_{x,0}(\pi) \) with two free variables for path formulæ, such that \( \mathcal{M} \models_{x \rightarrow \omega} \text{ST}_x(\varphi) \) iff
$w \in [\varphi]_\mathfrak{M}$ and $\mathfrak{M} \models x \rightarrow u, y \rightarrow v$ if $w \mathfrak{M} \models \pi$:

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

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

**Exercise 4.7 (Within modality).** Let $\mathfrak{M} = (W, \downarrow, \rightarrow, (P_a)_{a \in A})$ be a tree structure and $p$ be a point in $\mathfrak{M}$. We define the substructure at $p$, noted $\mathfrak{M} \models p \overset{\text{def}}{=} \{ w \in W \mid p \downarrow^* w \}$. The semantics of a PDLW formulæ is defined by $\mathfrak{M}, w \models \varphi$ iff $\mathfrak{M} \models w \models \varphi$.

Propose a translation of PDLW formulæ into FO[TC].

**Conditional PDL.** A particular fragment of PDL called conditional PDL (cPDL) is equivalent to FO[$\downarrow^*, \rightarrow^*$]: \cite{Marx2005} See Marx (2005).

\[
\pi ::= \alpha | \alpha^* | \tau | \pi | \pi + \pi | (\alpha; \varphi^?)^* | \varphi^? \quad \text{(conditional paths)}
\]

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

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

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

**Exercise 4.8.** Express the formulæ (4.12)–(4.21) in cPDL.
4.4 Parsing as Intersection

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

(*) Exercise 4.9. Fix a finite word $w$ and a finite alphabet $\Gamma$ of internal nodes. Define a non-deterministic tree automaton that recognises the set of closed binary trees with $w$ as yield; more formally, it should recognise the tree language $\{t \in T(\Sigma \cup \{\#\}) \mid \text{yield}(t) = w\}$.
Chapter 5

Mildly Context-Sensitive Syntax

Recall that context-sensitive languages (aka type-1 languages) are defined by phrase structure grammars with rules of form $\lambda A \rho \rightarrow \lambda \alpha \rho$ with $A$ in $N$, $\lambda, \rho$ in $V^*$, and $\alpha$ 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.

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

- some natural language constructs are not context-free, the Swiss-German account by Shieber (1985) being the best known example. Such fragments typically involve so-called limited cross-serial dependencies, as in the languages $\{a^n b^m c^n d^m \mid n,m \geq 0\}$ or $\{ww \mid w \in \{a,b\}^*\}$—see the next paragraph;

- 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 modelling point of view.

Cross-Serial Dependencies in Swiss-German. One of the first widely accepted instance of a syntactic phenomenon in a natural language that exceeds the expressive power of context-free grammar is due to Shieber (1985). The typical sentence is as follows:

Jan säit das mer d’chind em Hans es huus haend wele laa hälfe aastriiche
Jan says that we the children-ACC Hans-DAT the house-ACC have wanted let help paint

‘Jan says that we have wanted to let the children help Hans paint the house’

In this sentence, one should distinguish between

- $a$: noun phrases with accusative marking, like d’chind,
- $b$: noun phrases with dative marking, like em Hans,
- $c$: verbs with an accusative argument, like laa, and
d: verbs with a dative argument, like hälfe.

Observe that the sentence above is of the form \( xabycdz \) where \( x \) stands for Jan säit das mer, \( y \) for es huus haend wele, and \( z \) for aastriche. The core of the argument is that this example is productive: for all \( k \), there exist grammatical sentences of the form \( xa^mb^ncyd^pz \) exist with \( m, n, o, p \geq k \), but those are constrained by the dependencies between \( a \)'s and \( c \)'s on the one hand and \( b \)'s and \( d \)'s on the other hand: one must have \( m = o \) and \( n = p \) for the sentence to be grammatically correct. But then that means that any grammar for Swiss-German has a language whose intersection with \( xa^*b^*yc^*d^*z \) is \( \{ xa^mb^ncyd^p \mid m, n \geq 0 \} \), which is not context-free: Swiss German as a whole is therefore not context-free either.

**Mildly-Context Sensitive Syntax.** 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 requires the Parikh image of the language to be a semilinear subset of \( \mathbb{N}^{\Sigma} \) (Parikh, 1966), and
3. polynomial time recognition for a fixed grammar.

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.

We will however concentrate on two strict subclasses: tree adjoining languages (TALs, Section 5.1) and well-nested MCSLs (wnMCSLs, Section 5.2); Figure 5.1 illustrates the relationship between these classes. As in Section 3.1.1 our main focus will be on the corresponding tree languages, representing linguistic constituency analyses and sentence composition.

### 5.1 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 subsequently extended in various ways; see Joshi and Schabes (1997) for the ‘standard’ definitions.

**Definition 5.1** (Tree Adjoining Grammars). A tree adjoining grammar (TAG) is a tuple \( G = \langle N, \Sigma, T_\alpha, T_\beta, S \rangle \) where \( N \) is a finite nonterminal alphabet, \( \Sigma \) a finite terminal alphabet with \( N \cap \Sigma = \emptyset \), \( T_\alpha \) and \( T_\beta \) 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
Logic and Linguistic Modelling

Figure 5.1: Hierarchies between context-free and full context-sensitive languages.

Figure 5.2: Schematics for the substitution and adjunction operations.

- \( N \downarrow \) the ranked alphabet of substitution labels, all with arity 0,
- \( N^{na} \) the unranked alphabet of null adjunction labels,
- \( N^* \) the ranked alphabet of foot variables, all with arity 0.

In order to work on ranked trees, we add positive arities to \( N \) and \( N^{na} \) and null arities to \( \Sigma_0 \). 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 labelled by a symbol in \( N \cup N^{na} \), and we define accordingly \( rl(t) \) of a tree \( t \) as its unranked root label modulo \( na \): \( rl(t) \) if there exists \( m \) in \( N_{>0} \), \( t(\varepsilon) = A^{(m)} \) or \( t(\varepsilon) = A^{na(m)} \). Then

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

Figure 5.3: A tree adjoining grammar.

\[ R^\text{def}_G \equiv \{ \alpha \in T_\alpha \land \text{rl}(\alpha) = A \} \]

(substitution)

\[ \cup \{ A^{(m)}(x_1, \ldots, x_m) \rightarrow \beta[A^{(m)}(x_1, \ldots, x_m)] | m \in \mathbb{N}_{>0}, A^{(m)} \in N_m, \beta[A_*] \in T_\beta \} \]

(adjunction)

A derivation starts with an initial tree in \( T_\alpha \) and applies rules from \( R_G \) until no substitution node is left:

\[ L_T(G) \equiv \{ h(t) | \exists t \in T(N \cup \Sigma \cup \{ \varepsilon(0) \}), \exists \alpha \in T_\alpha, \text{rl}(\alpha) = S \land \alpha \xrightarrow{R}^* t \} \]

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

\[ L(G) \equiv \text{yield}(L_T(G)) \]

the set of yields of all its trees.

Example 5.2. Figure 5.3 presents a tree adjoining grammar with

\[ N = \{ S, \text{NP}, \text{VP}, \text{VBZ}, \text{NNP}, \text{NNS}, \text{RB} \} , \]

\[ \Sigma = \{ \text{likes}, \text{Bill}, \text{mushrooms}, \text{really} \} , \]

\[ T_\alpha = \{ \alpha_1, \alpha_2, \alpha_3 \} , \]

\[ T_\beta = \{ \beta_1 \} , \]

\[ S = S . \]

Its sole S-rooted initial tree is \( \alpha_1 \), on which one can substitute \( \alpha_2 \) or \( \alpha_3 \) in order to get \( \text{Bill likes mushrooms} \) or \( \text{mushrooms likes mushrooms} \); the adjunction of \( \beta_1 \) on the VP node of \( \alpha_1 \) also yields \( \text{Bill really likes mushrooms} \) (see Figure 5.4) or \( \text{mushrooms really really really likes Bill} \). In the TAG literature, a tree in \( T(N \cup N^{na} \cup \Sigma \cup \{ \varepsilon(0) \}) \) obtained through the substitution and adjunction operations is called a derived tree, while a derivation tree records how the rewrites took place (see Figure 5.4 for an example; children of an elementary tree are shown in addressing order, with plain lines for substitutions and dashed lines for adjunctions).

Example 5.3 (Copy Language). The copy language \( L_{\text{copy}} \equiv \{ ww | w \in \{ a, b \}^* \} \) is generated by the TAG of Figure 5.5 with \( N = \{ S \} , \Sigma = \{ a, b \} , T_\alpha = \{ \alpha_e \} , \) and \( T_\beta = \{ \beta_\alpha, \beta_\beta \} \).

(*) Exercise 5.1. Give a TAG for the language \( \{ a^n b^m c^n d^m | n, m \geq 0 \} \).
5.1.1 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 lexicalised, 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 how it treats some long-distance dependencies in English.

Lexicalised Grammar

A TAG is lexicalised if all its elementary trees have at least one terminal symbol as a leaf. In linguistic modelling, it will actually have one distinguished terminal symbol, called the anchor, plus possibly some other terminal symbols, called coanchors. An anchor serves as head word for at least a part of the elementary tree, as \( \alpha_1 \) for \( \alpha_1 \) in Figure 5.3. Coanchors serve for particles, prepositions, etc., whose use is mandatory in the syntactic phenomenon modelled by the elementary tree, as by for \( \alpha_5 \) in Figure 5.6.

Subcategorisation Frames. Each elementary tree then instantiates a subcategorisation 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 \( \alpha_1 \) in Figure 5.3; similarly, to think takes a clausal S complement, as instantiated by \( \beta_2 \) in Figure 5.6. These first two examples are canonical instantiations of the subcategorisation frames of to like and to think, but there are other possible instantiations, for instance interrogative with \( \alpha_4 \) or passive with \( \alpha_5 \) for to like.
Example 5.4. Extend the TAG of Figure 5.3 with the trees of Figure 5.6. 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 $\beta_3$ on the $S$ node of $\alpha_4$, and that of a single $\beta_4$ on the VP node of $\alpha_5$ are controlled through the feature structures, and there is no over-generation from this simple grammar.

Syntactic Lexicon. In practice, elementary trees as the ones of Figure 5.3 are not present as such in the XTAG grammar. It rather contains unanchored versions of these trees, with a specific marker ⋄ for the anchor position. For instance, $\alpha_2$ in Figure 5.3 would be stored as a context NP(NNP(⋄)) 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 subcategorisation frame and type of construction (e.g. canonical, passive, ...).

Long-Distance Dependencies

Let us focus on $\alpha_4$ in Figure 5.6. 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 5.7. We leave the question of syntax/semantics interfaces using derivation trees to later chapters.
5.1.2 Background: Context-Free Tree Grammars

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

Definition 5.5 (Context-Free Tree Grammars). A context-free tree grammar (CFTG) is a tuple $G = \langle N, F, S, R \rangle$ consisting of a finite ranked nonterminal alphabet $N$, a finite ranked terminal alphabet $F$, an axiom $S^{(0)}$ in $N_0$, and a finite set of rules $R$ of form $A^{(n)}(y_1, \ldots, y_n) \rightarrow e$ with $e \in T(N \cup F, \mathcal{Y})$ where $\mathcal{Y}$ is an infinite countable set of parameters. The language of $G$ is defined as

$$L(G) \overset{\text{def}}{=} \{ t \in T(F) \mid S^{(0)} R^* t \}.$$  

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

Example 5.6 (Squares). The CFTG with rules

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

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

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

(5.1)

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

Example 5.7 (Non-primes). The CFTG with rules

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

See Gécseg and Steinby (1997) Section 15 and Comon et al. (2007) Section 2.5. Regarding string languages, the set yield($L(G)$) of CFTGs characterises the class of indexed languages (Aho 1968, Fischer 1968). Context-free tree languages are also defined through top-down pushdown tree automata (Guessarian 1983).
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 \(\text{yield}(t') = a^{(2+m)(n+1)}\).

\[\star\] Exercise 5.2 (Powers of 2). Give a CFTG with \(\text{yield}(L(\mathcal{G})) = \{a^n ba^{2^n} \mid n \geq 1\}\).

\[\star\] Exercise 5.3 (Normal Form). Show that any CFTG can be put in a normal form where every rule in \(R\) is either of form \(A^{(n)}(y_1, \ldots, y_n) \rightarrow a^{(m)}(y_1, \ldots, y_n)\) with \(a\) in \(\mathcal{F}_n\) or of form \(A^{(n)}(y_1, \ldots, y_n) \rightarrow e\) with \(e\) in \(T(N, \mathcal{F}_n)\).

**IO and OI Derivations**

If we see derivations in a CFTG as evaluation in a recursive program with non-terminals as 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 \(R\),

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

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

**Example 5.8 (IO vs. OI).** Consider the CFTG with rules

\[
S \rightarrow A(B) \quad A(y) \rightarrow f(y, y) \\
B \rightarrow g(B) \quad B \rightarrow a.
\]

Then OI derivations are all of form

\[
S \overset{\text{OI}}{\Rightarrow} A(B) \overset{f}{\Rightarrow} (B, B) \overset{\text{OI}}{\Rightarrow} n+m f(g^m(a), g^n(a))
\]

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

\[
S \overset{\text{IO}}{\Rightarrow} A(B) \overset{\text{IO}}{\Rightarrow} n A(g^n(a)) \overset{\text{IO}}{\Rightarrow} f(g^n(a), g^n(a)).
\]

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

**Theorem 5.9** [Fischer 1968]. For any CFTG \(\mathcal{G}\), \(L_{\text{IO}}(\mathcal{G}) \subseteq L_{\text{IO}}(\mathcal{G}) = L(\mathcal{G})\).

As seen with Example 5.8, the case \(L_{\text{IO}}(\mathcal{G}) \subseteq L_{\text{IO}}(\mathcal{G})\) can occur. Theorem 5.9 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 5.10.** 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 \(C^n(\mathcal{F})\), and \(l = C[t_1, \ldots, t_n] \Rightarrow^m t'\) 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. \(l \overset{R}{\Rightarrow} m t'_1, \ldots, t'_n, m = m_1 + \ldots + m_n\), and \(t' = C[t'_1, \ldots, t'_n]\).
Proposition 5.12. Consider again the TAG of Figure 5.5 for the copy language: we obtain

\[ \text{Translation from TAGs to CFTGs.} \]

Given a TAG \( G = \langle N, \Sigma, T, S, T, S \rangle \), we construct a CFTG \( G' = \langle N', \mathcal{F}, S', R \rangle \) with

\[
N' \overset{\text{def}}{=} N \cup \{ A^{(1)} \mid A \in N \} \\
\mathcal{F} \overset{\text{def}}{=} \Sigma_0 \cup \{ \varepsilon(0) \} \cup N_{>0} \\
R \overset{\text{def}}{=} \{ A \downarrow \to \tau(\alpha) \mid \alpha \in T_\alpha \land \rl(\alpha) = A \} \\
\quad \cup \{ A^{(1)}(y) \to \tau(\beta)[A^{(1)}(y)] \mid \beta[A\varepsilon] \in T_\beta \} \\
\quad \cup \{ A^{(1)}(y) \to \tau(\beta)[y] \mid \beta[A\varepsilon] \in T_\beta \} \\
R' \overset{\text{def}}{=} \{ A^{(1)}(y) \to y \mid A^{(1)} \in \bar{N} \}
\]

where \( \tau : T(\Delta \cup \{ \square \}) \to T(\Delta' \cup \{ \square \}) \) for \( \Delta \overset{\text{def}}{=} N \downarrow \cup N_{>0} \uparrow N \cup \Sigma_0 \) and \( \Delta' \overset{\text{def}}{=} N' \cup \mathcal{F} \) is a tree homomorphism generated by

\[
\tau(A^{(m)}(x_1, \ldots, x_m)) \overset{\text{def}}{=} A^{(1)}(A^{(m)}(x_1, \ldots, x_m)) \\
\tau(A^{na(m)}) \overset{\text{def}}{=} A^{(m)}(x_1, \ldots, x_m)
\]

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

Example 5.11. Consider again the TAG of Figure 5.5 for the copy language: we obtain \( G' = \langle N', \mathcal{F}, S', R \rangle \) with \( N' = \{ S, S \} \), \( \mathcal{F} = \{ S, a, b, \varepsilon \} \), and rules

\[
R = \{ S \downarrow \to S(S(\varepsilon)), \quad (\text{corresponding to } \alpha_\varepsilon) \\
S(y) \to S(a, S(S(y, a))), \quad (\text{corresponding to } \beta_a) \\
\bar{S}(y) \to S(b, S(S(y, b)))) \quad (\text{corresponding to } \beta_b) \\
R' = \{ S(y) \to y \}.
\]

Proposition 5.12. \( L_T(G) = L(G') \).
Proof of $L_T(\mathcal{G}) \subseteq L(\mathcal{G}')$. We first prove by induction on the length of derivations:

Claim 5.12.1. For all trees $t$ in $T(\Delta)$, $\frac{R_G}{\tau} t \implies * t'$ implies $t'$ is in $T(\Delta)$ and $\tau(t) \frac{R}{\tau} * \tau(t')$.

Proof of Claim 5.12.1. That $T(\Delta)$ is closed under $R_G$ is immediate. For the second part of the claim, we only need to consider the case of a single derivation step:

For a substitution $C[A] \frac{R_G}{\tau} C[\alpha]$ occurs iff $\alpha$ is in $T_\alpha$ with $\text{rl}(\alpha) = A$, which implies $\tau(C[A]) = \tau(C)[\alpha]$.

For an adjunction $C[A^{(m)}(t_1, \ldots, t_m)] \frac{R_G}{\tau} C[\beta[A^{(m)}(t_1, \ldots, t_m)]]$ occurs iff $\beta[A]$ is in $T_\beta$, implying

\[
\tau(C[A^{(m)}(t_1, \ldots, t_m)]) = \tau(C)[\bar{A}(1)(A^{(m)}(\tau(t_1), \ldots, \tau(t_m)))]
\]

\[
\frac{R}{\tau} \tau(C)[\bar{A}(1)(A^{(m)}(\tau(t_1), \ldots, \tau(t_m)))]
\]

\[
= \tau(C)[\beta[A^{(m)}(t_1, \ldots, t_m)]).
\]

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

Claim 5.12.2. If $t$ is a tree in $T(\text{na} \cup F)$, then there exists a derivation $\tau(t) \frac{R}{\tau} * h(t)$ in $\mathcal{G}'$.

Proof of Claim 5.12.2. We proceed by induction on $t$:

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

\[
\tau(A^{(m)}(t_1, \ldots, t_m)) = \bar{A}(1)(A^{(m)}(\tau(t_1), \ldots, \tau(t_m)))
\]

\[
\frac{R}{\tau} A^{(m)}(\tau(t_1), \ldots, \tau(t_m))
\]

\[
\frac{R}{\tau} * A^{(m)}(h(t_1), \ldots, h(t_m)) \quad \text{(by ind. hyp.)}
\]

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

\[
\tau(A^{na(m)}(t_1, \ldots, t_m)) = A^{(m)}(\tau(t_1), \ldots, \tau(t_m))
\]

\[
\frac{R}{\tau} A^{(m)}(h(t_1), \ldots, h(t_m)) \quad \text{(by ind. hyp.)}
\]

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

For the main proof: Let $t$ be a tree in $L_T(\mathcal{G})$; there exist $t'$ in $T(\text{na} \cup F)$ and $\alpha$ in $T_\alpha$ with $\text{rl}(\alpha) = S$ s.t. $\alpha \frac{R_G}{\tau} t'$ and $t = h(t')$. Then $S \frac{R}{\tau} \tau(\alpha) \frac{R}{\tau} * \tau(t')$ according to Claim 5.12.1 and then $\tau(t') \frac{R}{\tau} * t$ removes all its nonterminals according to Claim 5.12.2.

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 $L \subseteq T(\Delta' \cup \{\square\})$ as the language of all trees and contexts where every node labelled $\bar{A}(1)$ in $\mathcal{N}$ has $A^{(m)}$ in $\mathcal{N}$ as the label of its daughter—$L$ is defined formally in the proof of the following claim:
Claim 5.12.3. The homomorphism \( \tau \) is a bijection from \( T(\Delta \cup \{\square\}) \) to \( L \).

Proof of Claim 5.12.3. It should be clear that \( \tau \) is injective and has a range included in \( L \). We can define \( \tau^{-1} \) as a deterministic top-down tree transduction from \( T(\Delta' \cup \{\square\}) \) into \( T(\Delta \cup \{\square\}) \) with \( L \) for domain, thus proving surjectivity: Let \( T = \{\{q\} \cup \{q_A \mid A \in N\}, \Delta' \cup \{\square\}, \Delta \cup \{\square\}, \rho, \{q\}\} \) with rules

\[
\rho = \{q(A^{(1)}(x)) \rightarrow q_A(x) \mid A^{(1)} \in N\} \\
\cup \{q(A^{(m)}(x_1, \ldots, x_m)) \rightarrow A^{(m)}(q(x_1), \ldots, q(x_m)) \mid A^{(m)} \in N\} \\
\cup \{q(A^{(m)}(x_1, \ldots, x_m)) \rightarrow A^{\rho A(m)}(q(x_1), \ldots, q(x_m)) \mid A^{(m)} \in N\} \\
\cup \{q(a^{(m)}(x_1, \ldots, x_m)) \rightarrow a^{(m)}(q(x_1), \ldots, q(x_m)) \mid a^{(m)} \in N \downarrow \cup \Sigma \uplus \{e^{(0)}, \square^{(0)}\}\} .
\]

We see immediately that \( \|T\|(t) = \tau^{-1}(t) \) for all \( t \) in \( L \).

Thanks to Claim 5.12.3, we can use \( \tau^{-1} \) in our proofs. We obtain claims mirroring Claim 5.12.1 and Claim 5.12.2 using the same types of arguments:

Claim 5.12.4. For all trees \( t \) in \( L \), \( t \overset{R}{\Rightarrow}^* \tau'(t) \) implies \( t' \) in \( L \) and \( \tau^{-1}(t) \overset{\tau}{\Rightarrow}^* \tau^{-1}(t') \).

Claim 5.12.5. If \( t \) is a tree in \( L \cap T(\bar{N} \cup F) \), \( t' \) a tree in \( T(F) \), and \( t \overset{\tau}{\Rightarrow}^* t' \), then \( h(\tau^{-1}(t')) = \tau^{-1}(t) \).

For the main proof, consider a derivation \( S \downarrow \overset{R}{\Rightarrow}^* t \) with \( t \in T(F) \) of \( G \). We can reorder this derivation so that \( S \downarrow \overset{R}{\Rightarrow}^* \tau(\alpha) \overset{\tau}{\Rightarrow}^* \tau(t') \overset{\tau'}{\Rightarrow}^* t \) for some \( \alpha \) in \( T_\alpha \) with \( rl(\alpha) = S \) and \( t' \) in \( L \cap T(\bar{N} \cup F) \) (i.e. \( t' \) does not contain any symbol from \( N_\downarrow \)). By Claim 5.12.4 \( \alpha \overset{\tau}{\Rightarrow}^* t' \) and by Claim 5.12.5 \( h(t') = \tau^{-1}(t) \). Since \( t \) belongs to \( T(F) \), \( \tau^{-1}(t) = t \), which shows that \( t \) belongs to \( L_T(G) \).

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

Exercise 5.4 (Non-Strict TAGs). Definition 5.1 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

\[ S((A(a, □))^* \cdot b, (A(□, a))^* \cdot b) \]. (5.2)

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:

\[ \{ A(a), B(a) \} \] (5.3)
\[ \{ a \} \] (5.4)

Note that allowing distinct foot and root labels in auxiliary trees is useless for these examples.

5.2 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 5.1. This provides characterisations both in terms of

- well-nested multiple context-free grammars (or equivalently well-nested linear context-free rewrite systems) (Kanazawa, 2009), and in terms of

- linear macro grammars (Seki and Kato, 2008), a subclass of the macro grammars of Fischer (1968), also characterised via linear context-free tree grammars (Rounds, 1970) or linear macro tree transducers (Engelfriet and Vogler, 1985).

We concentrate on this second view.

5.2.1 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 modelling is that arguments in a subcategorisation 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:

Proposition 5.13. Let \( G = \langle N, F, S, R \rangle \) be a linear CFTG. Then \( L_{IO}(G) = L_{OI}(G) \).

Proof. Consider a derivation \( S \xrightarrow{R}^* t \) in a linear CFTG. Thanks to Theorem 5.9, we can assume this derivation to be OI. Let us pick the last non-IO step within this OI derivation:

\[ S \xrightarrow{OI}^* C[A^{(n)}(e_1, \ldots, e_n)] \]
\[ \xrightarrow{r_A} C[e_A\{y_1 \leftarrow e_1, \ldots, y_n \leftarrow e_n\}] \]
\[ \xrightarrow{IO}^* t \]

using some rule \( r_A : A^{(n)}(y_1, \ldots, y_n) \rightarrow e_A \): one of the \( e_i \) must contain a nonterminal, or that step would have been IO. By Lemma 5.10, we can 'pull' all the independent rewrites occurring after this \( \xrightarrow{r_A} \) so that they occur before the \( \xrightarrow{r_A} \) rewrite,
so that the next rewrite occurs outside the context $C$ in $e_A\{y_1 \leftarrow e_1, \ldots, y_n \leftarrow e_n\}$. Since everything after this $\overset{r}{\Rightarrow}$ is IO, this rewrite has to involve an innermost non-terminal, thus a non-terminal that was not introduced in $e_A$, but one that already appeared in some $e_i$: in the context $C$:

$$e_A\{y_1 \leftarrow e_1, \ldots, y_i \leftarrow C'[B'(m)(e'_1, \ldots, e'_m)], \ldots, y_n \leftarrow e_n\}$$

$$\overset{r_B}{\Rightarrow} e_A\{y_1 \leftarrow e_1, \ldots, y_i \leftarrow C'[e_B\{x_1 \leftarrow e'_1, \ldots, x_m \leftarrow e'_m\}], \ldots, y_n \leftarrow e_n\}$$

for some rule $r_B : B'(m)(x_1, \ldots, x_m) \rightarrow e_B$; this is only correct 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 $C$:

$$A(n)(e_1, \ldots, C'[B'(m)(e'_1, \ldots, e'_m)], \ldots, e_n)$$

$$\overset{r_B}{\Rightarrow} A(n)(e_1, \ldots, C'[e_B\{x_1 \leftarrow e'_1, \ldots, x_m \leftarrow e'_m\}], \ldots, e_n)$$

$$\overset{r_B}{\Rightarrow} e_A\{y_1 \leftarrow e_1, \ldots, y_i \leftarrow C'[e_B\{x_1 \leftarrow e'_1, \ldots, x_m \leftarrow e'_m\}], \ldots, y_n \leftarrow e_n\}.$$  

Repeating this operation for every non-terminal that occurred in the $e_i$’s yields a derivation of the same length for $S \overset{R}{\Rightarrow}^* t$ with a shorter OI prefix and a longer IO suffix. Repeating the argument at this level yields a full IO derivation.

Proposition 5.13 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 $G = \langle N, F, S, R \rangle$: the semantics of a non-terminal $A(n)$ can be recast as the relation $\llbracket A(n) \rrbracket \subseteq (T(F))^{n+1}$:

$$\llbracket A(n) \rrbracket(t_1, \ldots, t_n) \overset{A(n)(y_1, \ldots, y_n) \rightarrow e \in R}{=} \bigcup_{(A(n)(y_1, \ldots, y_n) \rightarrow e \in R)} \llbracket e \rrbracket(t_1, \ldots, t_n)$$

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

$$\llbracket a(m)\{e_1, \ldots, e_m\}\rrbracket(t_1, \ldots, t_n) \overset{a(m)(t'_1, \ldots, t'_m)}{=} \{a(m)(t'_1, \ldots, t'_m) \mid \forall 1 \leq i \leq m. t'_i \in \llbracket e_i \rrbracket(t_1, \ldots, t_n)\}$$

$$\llbracket B(m)\{e_1, \ldots, e_m\}\rrbracket(t_1, \ldots, t_n) \overset{B(m)(t'_1, \ldots, t'_m)}{=} \{B(m)(t'_1, \ldots, t'_m) \mid \forall 1 \leq i \leq m. t'_i \in \llbracket e_i \rrbracket(t_1, \ldots, t_n)\}$$

$$\llbracket y_i \rrbracket(t_1, \ldots, t_n) \overset{y_i}{=} \{t_i\}.$$  

The consequence of this definition is

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

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

### 5.2.2 Parsing as Intersection

Let us look into more algorithmic issues and consider the parsing problem for linear CFTGs. In order to apply the parsing as intersection paradigm, we need two main ingredients: the first is emptiness testing [Proposition 5.14], the second is closure under intersection with regular sets [Proposition 5.15]. We actually prove these results for IO derivations in CFTGs rather than for linear CFTGs solely.

**Proposition 5.14 (Emptiness).** Given a CFTG $G$, one can decide whether $L_{IO}(G) = \emptyset$ in $O(|G|)$.  

*This section relies heavily on Maneth et al. (2007).*
Proof sketch. Given $G = \langle N, F, S, R \rangle$, we construct a context-free grammar $G' = \langle N', \emptyset, P, S, R \rangle$ s.t. $L(\mathcal{G'}) = \emptyset$ iff $L(G) = \emptyset$ and $|G'| = O(|G|)$. Since emptiness of CFGs can be tested in linear time, this will yield the result. We define for this

$$N' \defeq N \cup \bigcup_{A(m)(y_1, \ldots, y_m) \rightarrow e \in R} \text{Sub}(e),$$

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

$$P' \defeq \{ A \rightarrow e \mid A(m)(y_1, \ldots, y_m) \rightarrow e \in R \} \quad \text{(rules)}$$
$$\cup \{ a(m)(e_1, \ldots, e_m) \rightarrow e_1 \cdots e_m \mid a \in F \cup \mathcal{Y} \} \quad \text{(F- or \mathcal{Y}-labelled positions)}$$
$$\cup \{ A(m)(e_1, \ldots, e_m) \rightarrow A e_1 \cdots e_m \} \quad \text{(N-labelled positions)}$$

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

The constructed grammar is clearly of linear size; we leave the fixpoint induction proof of $X \xrightarrow{\mathcal{G}'} * \in \mathcal{D}$ iff $[X] \neq \emptyset$ to the reader. \hfill $\square$

**Proposition 5.15** (Closure under Intersection with Regular Tree Languages). Let $G$ be a (linear) CFTG with maximal nonterminal rank $M$ and maximal number of nonterminals in a right-hand side $D$, and $A$ a NTA with $|Q|$ states. Then we can construct a (linear) CFTG $G'$ with $L(\mathcal{G'}) = L \cap L(G)$ and $|G'| = O(|G| \cdot |Q|^{M+D+1})$.

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

$$N' \defeq \{ S' \} \cup \bigcup_{m \leq M} N_m \times Q^{m+1},$$

i.e. we add a new axiom and otherwise consider tuples of form $(A(m), q_0, q_1, \ldots, q_m)$ as nonterminals of rank $m$. Informally, such a nonterminal tells us that, if the trees plugged into parameters $y_1, \ldots, y_m$ of a rule from $A$ are recognised in states $q_1, \ldots, q_m$ of $A$, then the whole tree generated by $A$ will be recognised in state $q_0$ of $A$.

Before we define $R'$, let us first define a term rewriting system $\theta_{q_1, \ldots, q_m}$ for every tuple of states $q_1, \ldots, q_m \in Q$ with $m \leq M$. This rewriting system is actually a linear non-erasing **bottom-up tree transducer** from trees in $T(F \cup N, \mathcal{Y})$ to trees in $T(F \cup N' \cup Q, \mathcal{Y})$, where the symbols in $Q$ have all arity 1. The set of rules of $\theta_{q_1, \ldots, q_m}$ is

$$y_i \rightarrow q_i(y_i) \quad \text{for all } 1 \leq i \leq m$$
$$a(q_1(x_1), \ldots, q_n(x_n)) \rightarrow q(a(x_1, \ldots, x_n)) \quad \text{for all } (q, a, q_1', \ldots, q_n') \in \delta$$
$$B(y_1', \ldots, y_m') \rightarrow q(B, y_1', \ldots, y_m')(y_1, \ldots, y_m) \quad \text{for all } n, B \in N_n, q_1, \ldots, q_n \in Q.$$

The effect of $\theta_{q_1, \ldots, q_m}$ is to relabel a tree in $T(F \cup N, \mathcal{Y})$ with ‘guessed’ nonterminals in $N'$ consistent with the transitions of the automaton $A$. 

We can now define $R'$ by
\[
R' \overset{\text{def}}{=} \{ S' \to \langle S, q_f \rangle \mid q_f \in F \} \\
\cup \{ \langle A, q_0, \ldots, q_m \rangle^{(m)}(y_1, \ldots, y_m) \to e' \mid A^{(m)}(y_1, \ldots, y_m) \to e \in R \land e \xrightarrow{\theta_{q_1 \ldots q_m}}^* q_0(e') \}.
\]

The intuition behind this definition is that $G'$ guesses that the trees passed as $y_i$ parameters will be recognised by state $q_i$ of $A$, leading to a tree generated by $A^{(m)}$ and recognised 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 $G'$ is constructed through at most $|R| \cdot |Q|^{|M+1}$ calls to $\theta$ translations, each allowing at most $|Q|^D$ choices for the nonterminals in the argument right-hand side. In fine, each rule of $G$ 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 $\llbracket q_1 \cdots q_m \rrbracket \subseteq (T(F))^m$ as the Cartesian product of the sets $\llbracket q_i \rrbracket \overset{\text{def}}{=} \{ t \in T(F) \mid t \xrightarrow{R_{q_i}}^* \}$. We can check that, for all $m \leq M$, all states $q_0, q_1, \ldots, q_m$ of $Q$, and all nonterminals $A^{(m)}$ of $N$, $\llbracket \langle A, q_0, q_1, \ldots, q_m \rangle \rrbracket(\llbracket q_1 \cdots q_m \rrbracket) = \llbracket A^{(m)} \rrbracket \cap \llbracket q_0 \rrbracket$.

This last equality proves the correctness of the construction. \hfill \Box

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

See Gómez-Rodríguez et al. (2010) for better bounds on parsing wnMCSLs.
Chapter 6

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 3.2, we can resort to several techniques: heuristics, semantic processing, and what interests us in this section, probabilities learnt 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

(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 grammaticality, these sentences will be ruled 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’ part, 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 normalising 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 maximisation method on newspaper text, where (1) is about $2.10^5$ times more probable than (2).

See Pereira (2000).
It remains that statistical language models are not modelling grammaticality itself, but rely on an occurrence-based model as a proxy.

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.

### 6.1 Weighted and Probabilistic CFGs

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

#### 6.1.1 Background: Semirings and Formal Power Series

**Semirings**

A semiring \( \langle K, \oplus, \odot, 0_K, 1_K \rangle \) is endowed with two binary operations, an addition \( \oplus \) and a multiplication \( \odot \) such that

- \( \langle K, \oplus, 0_K \rangle \) is a commutative monoid for addition with \( 0_K \) for neutral element,
- \( \langle K, \odot, 1_K \rangle \) is a monoid for multiplication with \( 1_K \) 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 \( a, b, c \in K \),
- \( 0_K \) is a zero for multiplication, i.e. \( a \odot 0_K = 0_K \odot a = 0_K \) for all \( a \in K \).

A semiring is **commutative** if \( \langle K, \odot, 1_K \rangle \) is a commutative monoid.

Among the main semirings of interest are the

- **Boolean** semiring \( \langle \mathbb{B}, \lor, \land, 0, 1 \rangle \) 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 \( \langle \mathbb{R}_{\geq 0} \cup \{+\infty\}, \min, +, +\infty, 0 \rangle \),
- **rational** semiring \( \langle \text{Rat}(\Delta^*), \cup, \cap, \emptyset, \{\varepsilon\} \rangle \) where \( \text{Rat}(\Delta^*) \) is the set of rational sets over some alphabet \( \Delta \). This is the only non-commutative example here.

**Weighted Automata**

A finite **weighted automaton** (or automaton with multiplicity, or \( K \)-automaton) in a semiring \( K \) is a generalisation of a finite automaton: \( A = \langle Q, \Sigma, K, \delta, I, F \rangle \) where \( \delta \subseteq Q \times \Sigma \times K \times Q \) is a finite weighted transition relation, and \( I \) and \( F \) are maps from \( Q \) to \( K \) 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 \( \llbracket \rho \rrbracket = kw \) where \( w = a_1 \cdots a_n \) is the word label of \( \rho \) and \( k = I(q_0)k_1 \cdots k_n F(q_n) \) its **multiplicity**. The behaviour \( \llbracket A \rrbracket \) of \( A \) is the sum of the monomials for all runs in \( A \): it is a formal power series on \( \Sigma^* \) with coefficients in
\( \mathbb{K} \), i.e. a map \( \Sigma^* \to \mathbb{K} \). The coefficient of a word \( w \) in \( \langle [A], w \rangle \) is the sum of the multiplicities of all the runs with \( w \) for word label:

\[
\langle [A], a_1 \cdots a_n \rangle \overset{\text{def}}{=} \sum_{q_0 \overset{a_1,A_{k_1}}{\overrightarrow{\pi}} q_1 \overset{a_2,A_{k_2}}{\overrightarrow{\pi}} \cdots q_{n-1} \overset{a_n,A_{k_n}}{\overrightarrow{\pi}} q_n} I(q_0)k_1 \cdots k_n F(q_n).
\]

A matrix \( \mathbb{K} \)-representation for \( A \) 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 \( (a, q, k, q') \in \delta \), and \( F \) is seen as a column matrix in \( \mathbb{K}^{Q \times 1} \). Then

\[
\langle [A], w \rangle = I \mu(w) F.
\]

A series is \( \mathbb{K} \)-recognisable if there exists a \( \mathbb{K} \)-representation for it.

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

**Exercise 6.1** (Hadamard Product). Let \( \mathbb{K} \) be a commutative semiring. Show that \( \mathbb{K} \)-recognisable series are closed under product: given two \( \mathbb{K} \)-recognisable series \( s \) and \( s' \), show that \( s \circ s' \) with \( \langle s \circ s', w \rangle = \langle s, w \rangle \odot \langle s', w \rangle \) for all \( w \in \Sigma^* \) is \( \mathbb{K} \)-recognisable. What can you tell about the support of \( s \circ s' \)?

### 6.1.2 Weighted Grammars

**Definition 6.1** (Weighted Context-Free Grammars). A weighted context-free grammar \( G = \langle N, \Sigma, P, S, \rho \rangle \) over a semiring \( \mathbb{K} \) 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 \( \alpha \overset{\pi}{\Rightarrow}_{\text{lm}}^* \beta \) is then defined as \( \rho(\pi) \). It would be natural to define the weight of a sentential form \( \gamma \) as the sum of the weights \( \rho(\pi) \) with \( S \overset{\pi}{\Rightarrow}_{\text{lm}}^* \gamma \), i.e.

\[
\rho(\gamma) \overset{\text{def}}{=} \sum_{\pi \in P^*, S \overset{\pi}{\Rightarrow}_{\text{lm}}^* \gamma} \rho(\pi).
\]

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 \( G \) then defines a formal series \( [G] \) with coefficients \( \langle [G], w \rangle = \rho(w) \).

A \( \mathbb{K} \)-CFG \( G \) is reduced if each nonterminal \( A \) in \( N \setminus \{ S \} \) is useful, which means that there exist \( \pi_1, \pi_2 \) in \( P^* \), \( u, v \) in \( \Sigma^* \), and \( \gamma \) in \( V^* \) such that \( S \overset{\pi_1}{\Rightarrow}_{\text{lm}}^* u A \gamma \overset{\pi_2}{\Rightarrow}_{\text{lm}}^* u v \) and \( \rho(\pi_1 \pi_2) \neq 0_{\mathbb{K}} \).

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

**Exercise 6.2.** A right linear \( \mathbb{K} \)-CFG \( G \) has its productions in \( N \times (\Sigma^* \cup \Sigma^* : \Sigma) \). Show that a series \( s \) over \( \Sigma \) is \( \mathbb{K} \)-recognisable iff there exists an acyclic right linear \( \mathbb{K} \)-CFG for it.
6.1.3 Probabilistic Grammars

Definition 6.1 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 \( N \),

\[
\sum_{p = A \rightarrow \alpha \in P} \rho(p) = 1 , \tag{6.1}
\]

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

**Partition Functions**

The **partition function** \( Z \) maps each nonterminal \( A \) to

\[
Z(A) \overset{\text{def}}{=} \sum_{w \in \Sigma^*, A \xrightarrow{\pi} \beta} \rho(\pi) . \tag{6.2}
\]

A PCFG is **convergent** if

\[
Z(S) < \infty ; \tag{6.3}
\]

in particular, it is **consistent** if

\[
Z(S) = 1 , \tag{6.4}
\]

i.e. \( \rho \) 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 (6.2) can be decomposed using commutativity of multiplication into

\[
Z(A) = \sum_{p = A \rightarrow \alpha \in P} \rho(p) \cdot Z(\alpha) \quad \text{for all } A \text{ in } N \tag{6.5}
\]

\[
Z(a) = 1 \quad \text{for all } a \text{ in } \Sigma \cup \{ \epsilon \} \tag{6.6}
\]

\[
Z(X\beta) = Z(X) \cdot Z(\beta) \quad \text{for all } (X, \beta) \text{ in } V \times V^* . \tag{6.7}
\]

This describes a monotone, continuous system of equations with the \( Z(A) \) for \( A \) in \( N \) as variables. By Kleene’s Fixpoint Theorem, it has a smallest solution, which is the partition function. Its solutions can be approximated through several techniques, prominently fixpoint approximants and Newton’s method.

**Example 6.2.** Properness and consistency are two distinct notions. For instance, the PCFG

\[
S \overset{q}{\rightarrow} SS \\
S \overset{1-q}{\rightarrow} a
\]

is proper for all \( 0 \leq q \leq 1 \), but the equation \( x = qx^2 + 1 - q \) has two roots \( 1 \) and \( \frac{1-q}{q} \), and thus if \( q \leq \frac{1}{2} \) the grammar is consistent with \( Z(S) = 1 \), but otherwise
Logic and Linguistic Modelling

$$Z(S) = \frac{1-q}{q} < 1.$$ Conversely,

$$S \xrightarrow{q/(1-q)} A$$
$$A \xrightarrow{q} AA$$
$$A \xrightarrow{1-q} a$$

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

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

**Normalisation**

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

$$\rho'(p = A \to \alpha) \overset{\text{def}}{=} \frac{\rho(p) Z(\alpha)}{Z(A)}.$$ (6.8)

**Exercise 6.3.** Show that in a reduced convergent PCFG with $Z(S) > 0$, for each $\alpha$ in $V^*$, one has $0 < Z(\alpha) < \infty$. (This justifies that (6.8) is well-defined.)

**Exercise 6.4.** Show that $G'$ is a proper PCFG. (*

**Proposition 6.4.** The grammar $G'$ defined by (6.8) is consistent if $G$ is reduced and convergent.

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

**Claim 6.4.1.** For all $Y$ in $V$, $\pi$ in $P^*$, and $w$ in $\Sigma^*$ with $Y \overset{\pi}{\Rightarrow}_im w$,

$$\rho'(\pi) = \frac{\rho(\pi)}{Z(Y)}. \quad (6.9)$$

**Proof of Claim 6.4.1** Note that, because $G$ 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 $\pi$. 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 $p = A \to X_1 \cdots X_m$: $A \overset{p}{\Rightarrow}_im X_1 \cdots X_m \overset{\pi}{\Rightarrow}_im w$. This derivation
can be decomposed using a derivation \( X_i \xrightarrow{\pi_i}^{\text{lm}} w_i \) 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

\[
\rho'(p\pi) = \rho'(p) \cdot \prod_{i=1}^m \rho'(\pi_i) = \frac{\rho(p) Z(X_1 \cdots X_m)}{Z(A)} \cdot \prod_{i=1}^m \rho'(\pi_i) \tag{by (6.8)}
\]

\[
= \frac{\rho(p)}{Z(A)} \cdot \prod_{i=1}^m Z(X_i) \cdot \prod_{i=1}^m \rho(\pi_i) \tag{by ind. hyp.}
\]

\[
= \frac{\rho(p\pi)}{Z(A)} \cdot Z(S) = 1 \tag{by (6.10)}
\]

Claim 6.4.1 shows that \( G' \) is consistent, since

\[
Z'(S) = \sum_{w \in \Sigma^*, S \xrightarrow{\pi}^{\text{lm}} w} \rho'(\pi) = \sum_{w \in \Sigma^*, S \xrightarrow{\pi}^{\text{lm}} w} \frac{\rho(\pi)}{Z(S)} = Z(S) = 1 .
\]

Remark 6.5. Note that Claim 6.4.1 also yields for all \( w \) in \( \Sigma^* \)

\[
\rho'(w) = \sum_{S \xrightarrow{\pi}^{\text{lm}} w} \rho'(\pi) = \sum_{S \xrightarrow{\pi}^{\text{lm}} w} \frac{\rho(\pi)}{Z(S)} = \frac{\rho(w)}{Z(S)} ,
\]

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

Example 6.6. Considering again the first grammar of Example 6.2, if \( q > \frac{1}{2} \), then \( \rho' \) with \( \rho'(p_1) = q \frac{Z(S)^2}{Z(S)} = 1 - q \) and \( \rho'(p_2) = q \) fits.

### 6.2 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 \( S' \rightarrow S \) 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 \( \pi \), and \( C(A, \pi) = \sum_{p=A \rightarrow \alpha \in P} C(p, \pi) \). Summing over the entire treebank, we get
\[ C(p, D) = \sum_{\pi \in D} C(p, \pi) \text{ and } C(A, D) = \sum_{\pi \in D} C(A, \pi). \] The estimated probability of a production is then (see e.g. [Chi and Geman, 1998])

\[ \rho(p = A \rightarrow \alpha) = \frac{C(p, D)}{C(A, D)}. \] (6.11)

**Exercise 6.5.** Show that the obtained PCFG is proper and consistent.

**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 (6.11) and normalise, 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:

\[
\begin{align*}
&\text{[NP He]} \ [VP \text{hurled} [\text{NP the ball}]] \ [PP \text{into the basket}]]. \\
&\text{[NP She]} \ [VP \text{took} [\text{NP a sip}]] \ [PP \text{of water}]].
\end{align*}
\]

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 6.1 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

\[
\text{VP}[S, \text{hurled}, \text{VBD}] \rightarrow \text{VP}[VP, \text{hurled}, \text{VBD}] \ \text{PP}[VP, \text{into}, \text{IN}].
\]

allowing to give it a higher probability than that of

\[
\text{VP}[S, \text{took}, \text{VBD}] \rightarrow \text{VP}[VP, \text{took}, \text{VBD}] \ \text{PP}[VP, \text{of}, \text{IN}].
\]

**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

\[
\text{VP} \rightarrow \text{VBPP PP PP PP PP ADVP PP.}
\]
Figure 6.1: A derivation tree refined with lexical and parent information.

for sentence

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

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

\[
\text{VP} \rightarrow \text{VBP} \text{ VP}' \\
\text{VP}' \rightarrow \text{PP} | \text{PP VP}' | \text{ADVP VP}'.
\]

**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 labelled with the appropriate nonterminal symbol. The evaluation measures the

**labelled 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,

**labelled 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.

### 6.3 Probabilistic Parsing as Intersection

We generalise in this section the intersective approach of Theorem 3.7. 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 generalised version of Dijkstra’s algorithm due to Knuth (1977) to find the most probable parse in this grammar.
6.3.1 Weighted Product

We generalise here Theorem 3.7 to the weighted case. Observe that it also answers Exercise 6.1 since \( \mathbb{K} \)-automata are equivalent to right-linear \( \mathbb{K} \)-CFGs according to Exercise 6.2.

**Theorem 6.7.** Let \( \mathbb{K} \) be a commutative semiring, \( G = \langle N, \Sigma, P, S, \rho \rangle \) an acyclic \( \mathbb{K} \)-CFG, and \( A = \langle Q, \Sigma, \mathbb{K}, \delta, I, F \rangle \) a \( \mathbb{K} \)-automaton. Then the \( \mathbb{K} \)-CFG \( G' = (\{ S' \} \uplus (N \times Q \times Q), \Sigma, P', S', \rho') \) with

\[
P' \triangleq \{ S' \xrightarrow{I(q_i) \odot F(q_f)} (S_i, q_i, q_f) \mid q_i, q_f \in Q \} \cup \{(a, q_0, q_m) \xrightarrow{k} (X_1, q_0, q_1) \cdots (X_m, q_m-1, q_m) \mid m \geq 1, A \xrightarrow{k} X_1 \cdots X_m \in P, q_0, \ldots, q_m \in Q \} \cup \{(a, q, q') \xrightarrow{k} a \mid (q, a, k, q') \in \delta \}
\]

is acyclic and such that, for all \( w \in \Sigma^* \), \( \langle \langle G' \rangle, w \rangle = (\langle G \rangle, w) \odot (\langle A \rangle, w) \).

As with Theorem 3.7 the construction of Theorem 6.7 works in time \( O(|G| \cdot |Q|^{m+1}) \) with \( m \) the maximal length of a rule rightpart in \( G \). Again, this complexity can be reduced by first transforming \( G \) into quadratic form, thus yielding a \( O(|G| \cdot |Q|^3) \) construction.

**Exercise 6.6.** Modify the quadratic form construction of Lemma 3.8 for the weighted (\( * \)) case.

6.3.2 Most Probable Parse

The weighted CFG \( G' \) constructed by Theorem 6.7 can be reduced by a generalisation 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 \( G' \), 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 \( G' \), i.e. where there does not exist a derivation \( A \Rightarrow^+ \delta A \gamma \) for any \( A \in N \) and \( \delta, \gamma \) in \( V^* \) in the underlying grammar. This is necessarily the case with Theorem 6.7 if \( G \) is acyclic and \( A \) has a finite support language. Then a topological sort of the nonterminals of \( G' \) for the partial ordering \( B < A \) iff there exists a production \( A \Rightarrow \alpha B \beta \) in \( P' \) with \( \alpha, \beta \) 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 \( j = 1, \ldots, |N'| \)

\[
M(A_j) = \max_{A \xrightarrow{k} X_1 \cdots X_m} k \cdot M(X_1) \cdots M(X_m) \quad (6.12)
\]

in the probabilistic semiring, with \( M(a) = 1 \) for each \( a \) in \( \Sigma \). The topological sort ensures that the maximal values \( M(X_i) \) in the right-hand side have already been computed when we use (6.12) to compute \( M(A_j) \).
Data: $G = \langle N, \Sigma, P, S, \rho \rangle$

1. foreach $a \in \Sigma$ do
2. \hspace{1em} $M(a) = 1$
3. \hspace{1em} $D \leftarrow \Sigma$
4. while $D \neq V$ do
5. \hspace{1em} foreach $A \in V \setminus D$ do
6. \hspace{2em} $\nu(A) \leftarrow \max_{A \rightarrow X_1 \cdots X_m \text{ s.t. } X_1, \ldots, X_m \in D} k \cdot M(X_1) \cdots M(X_m)$
7. \hspace{1em} $A \leftarrow \arg\max_{V \setminus D} \nu(A)$
8. \hspace{1em} $M(A) \leftarrow \nu(A)$
9. \hspace{1em} $D \leftarrow D \cup \{A\}$
10. return $M(S)$

Algorithm 6.1: Most probable derivation.

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 6.1.

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 $V \setminus D$ in time $O(\log |N|)$ 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 $O(|P| \log |N| + |G|)$.

The correctness of the algorithm relies on the fact that $M(A) = \nu(A)$ at line 8. Assuming the opposite, and since $\nu(A) \leq M(A)$ is an invariant, there must exist a derivation $B \xrightarrow{\text{im}}^* w$ with $\rho(\pi) > \nu(A)$ for some $B \notin D$, and we consider the shortest such derivation. Since $B \in N$, we can split this derivation into $B \xrightarrow{\text{im}}^* X_1 \cdots X_m$ and $X_i \xrightarrow{\text{im}}^* w_i$ 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 $\pi_i$ is strictly shorter than $\pi$, again a contradiction.

6.3.3 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, namely when one attempts to extract the most probable string (aka the consensus string) generated by a PCFG.

Consider the following decision problems:

Most Probable String (MPS).

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

question is there a string $w$ in $\Sigma^*$ s.t. $\langle [G], w \rangle \geq p$?
Bounded Most Probable String (BMPS).

**input** a PCFG $G$ over $\Sigma$ with rational weights (coded in binary), a length $b$ (in unary), and a rational $p$ in $[0, 1]$ (in binary);

**question** is there a string $w$ in $\Sigma^* \leq b$ s.t. $\langle [G], w \rangle \geq p$?

**Example 6.8.** Consider the following right-linear proper consistent PCFG:

\[
\begin{align*}
S & \rightarrow 9/10 \ A S \\
A & \rightarrow 2/3 \ A A \\
B & \rightarrow 2/3 \ A B
\end{align*}
\]

The most probable derivations are for the strings $b$ and $aaa$, with probability $1/10$. The most probable strings are actually $aaaa$ and $aaaaa$, with probability $(4/3) \cdot (1/10)$.

**Hardness**

We show here that both problems are already hard for convergent right-linear PCFGs. Note that, in the non convergent right-linear case, MPS is known as the Threshold Problem for Rabin probabilistic automata and is undecidable (e.g. Blondel and Canterini, 2003).

**Theorem 6.9 (Casacuberta and de la Higuera, 2000).** MPS and BMPS for convergent right-linear PCFGs are NP-hard.

**Proof.** The proof reduces from SAT. Let $\varphi = \bigwedge_{i=1}^{k} C_i$ 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 $\langle G, p \rangle$ of MPS or $\langle G, b, p \rangle$ of BMPS such that $\varphi$ is satisfiable if and only if there exists $w$ in $\Sigma^*$ such that $\langle [G], w \rangle \geq p$.

We define for this $G \overset{\text{def}}{=} \langle N, \Sigma, P, S \rangle$ where

\[
\begin{align*}
N & \overset{\text{def}}{=} \{S\} \cup \{A_{i,j} \mid 1 \leq i \leq k \land 0 \leq j \leq n\} \cup \{B_j \mid 1 \leq j \leq n\} \\
\Sigma & \overset{\text{def}}{=} \{0, 1, \$$\}
\end{align*}
\]

\[
\begin{align*}
P & \overset{\text{def}}{=} \{S \overset{1/k}{\rightarrow} A_{i,0} \mid 1 \leq i \leq k\} \\
& \cup \{A_{i,j-1} \overset{1/2}{\rightarrow} vB_j, A_{i,j-1} \overset{1/2}{\rightarrow} (1-v)A_{i,j} \mid v \in \{0, 1\} \land x_j \mapsto v \models C_i \land 1 \leq i \leq k \land 1 \leq j \leq n\} \\
& \cup \{A_{i,j-1} \overset{1/2}{\rightarrow} A_{i,j}, A_{i,j-1} \overset{1/2}{\rightarrow} 0A_{i,j} \mid x_j \not\in C_i \land 1 \leq i \leq k \land 1 \leq j \leq n\} \\
& \cup \{A_{i,n} \overset{0}{\rightarrow} \$$ \mid 1 \leq i \leq k\} \\
& \cup \{B_{j-1} \overset{1/2}{\rightarrow} 0B_j, B_{j-1} \overset{1/2}{\rightarrow} 1B_j \mid 2 \leq j \leq n\} \\
& \cup \{B_n \overset{1}{\rightarrow} \$$\}
\end{align*}
\]
and fix
\[ p \overset{\text{def}}{=} \frac{1}{2^n} . \]

First note that the construction can indeed be carried in polynomial time—remember that \( p \) is encoded in binary. Second, \( G \) 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 \( \varphi \) is satisfiable if and only if there exists \( w \in \Sigma^* \) such that \( \langle [G], w \rangle \geq p \). Note that any string \( w \) with \( \langle [G], w \rangle > 0 \) is necessarily of form \( \$v_1 \cdots v_n\$ \) with each \( v_j \in \{0, 1\} \), i.e. describes a valuation for \( \varphi \).

Observe that, for each clause \( C_i \) and each string \( w = \langle [J] G K, w \rangle > 0 \) is of form \( \langle [J] G K, w \rangle > 0 \), is necessarily of form \( \$v_1 \cdots v_n\$ \) and describes an assignment \( V_w : x_j \mapsto v_j \) that

- either satisfies \( C_i \), and then the corresponding string \( w \) has a single derivation \( \pi_w \) (the one that uses \( A_{i,j-1} \)) for the lowest index \( j \) such that \( x_j \mapsto v_j = C_i \); 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 \( A_{i,n} \overset{0}{\rightarrow} \$ \), and is thus of probability 0.

Therefore, if \( \varphi \) is satisfiable, i.e. if there exists \( V \) that satisfies all the clauses, then the corresponding string \( w = \sum_{i=1}^{k} Z(A_i,0) \) converges in polynomial time, and the grammar can be normalised in polynomial time.

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

\[
Z(B_j) = 1 , \tag{6.13}
\]

\[
Z(S) = \frac{1}{k} \sum_{i=1}^{k} Z(A_i,0) . \tag{6.14}
\]

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

\[
q_{i,j} \overset{\text{def}}{=} |\{ x_\ell \in C_i \mid j < \ell \leq n \}| . \tag{6.15}
\]

\textbf{Claim 6.10.1.} 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 6.10.1. 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 $Z(A_{i,j}) = 1/2 \cdot Z(A_{i,j+1}) + 1/2 \cdot Z(A_{i+1,j+1})$ by (6.5), and thus $Z(A_{i,j}) = Z(A_{i,j+1}) = 1 - 1/2^{q_{i,j+1}}$ by induction hypothesis.

2. $q_{i,j} = 1 + q_{i,j+1}$, i.e. when $x_{j+1}$ appears in $C_i$. Then $Z(A_{i,j}) = 1/2 \cdot Z(A_{i,j+1}) + 1/2 \cdot Z(B_{i,j+1})$, hence by (6.13) and the induction hypothesis, $Z(A_{i,j}) = 1/2 - 1/2^{q_{i,j+1}}$.

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 (6.8) to normalise the grammar of Theorem 6.9 thereby obtaining a proper and consistent right-linear PCFG $G'$ in polynomial time.

There remains the issue of computing an appropriate bound $p'$ for this new grammar. By Remark 6.5 for any word $w$ in $\Sigma^*$, $\langle \mathcal{G}, w \rangle \geq p$ if and only if $\langle \mathcal{G'}, w \rangle \geq p/Z(S)$: we define therefore

$$p' \overset{\text{def}}{=} \frac{p}{Z(S)}.$$  \hspace{1cm} (6.16)

Upper Bounds

Bounded Case. Deciding BMPS is mostly straightforward: guess a string $w$ in $\Sigma^b$, compute the PCFG $G'$ for $w$ using Theorem 6.7 in polynomial time, and compute the partition function $Z$ for $G'$—which is non-recursive since $G$ is acyclic—, which can be performed in polynomial time: then $\langle \mathcal{G'}, w \rangle = Z(S')$. Hence:

**Proposition 6.11.** BMPS is NP-complete.

Right-Linear Case. The case of MPS is more involved: there is no reason for the most probable string to be short.

**Example 6.12 (Long Strings).** De la Higuera and Oncina (2011) exhibit a right-linear grammar, for which the most probable string is of exponential length. Let $m$ be a natural number and $q$ a rational in $(0, 1)$, then the right-linear grammar with axiom $A_{q,m,0}$ and productions

$$A_{q,m,0} \overset{q}{\rightarrow} \varepsilon \quad A_{q,m,0} \overset{1-q}{\rightarrow} a A_{q,m,1} \quad A_{q,m,i} \overset{1}{\rightarrow} a A_{q,m,i+1 \mod m}$$

for all $1 \leq i < m$ assigns a probability $\rho(a^{km}) = q(1 - q)^k$ to a string of $a$'s whose length is a multiple of $m$, and probability 0 to any other string. Consider now a set of primes $\{m_1, \ldots, m_n\}$ and add the production

$$S \overset{1/n}{\rightarrow} A_{q,m,j,0}$$

for each $m_j$. This grammar has a size in $O(\sum_{j=1}^n m_j)$. 


On the one hand, the probability of the string $a^M$ of length $M \equiv \prod_{i=j}^{n} m_j$ (which is exponential in $\sum_{j=1}^{n} m_j$) is

$$\rho(a^M) = \sum_{j=1}^{n} \frac{1}{n} q(1-q)^{M_j}$$

$$\geq \frac{q}{n} \sum_{j=1}^{n} (1-q)^{M_j} \quad \text{(since } \frac{M}{m_j} \leq M)$$

$$= q(1-q)^M.$$ 

On the other hand, a string $a^\ell$ of length $\ell < M$ is not accepted by at least one of the subgrammars, therefore its probability is at most

$$\rho(a^\ell) \leq q^{n-1}.$$ 

Hence, a choice of $q$ such that

$$q \leq 1 - \sqrt{\frac{n-1}{n}}$$

ensures that no shorter string can have probability higher than $\rho(a^M)$.

Fortunately, we are also provided with a probability $p$ in an instance of MPS. When taking this threshold into account, de la Higuera and Oncina (2013) can then provide a polynomial bound on the length of the most probable strings. The following proposition uses a normal form on right-linear PCFGs:

**Definition 6.13.** A right-linear PCFG $G = \langle N, \Sigma, P, S, \rho \rangle$ is in $\varepsilon$-free form if $P \subseteq N \times (\Sigma \cup \Sigma N)$.

**Exercise 6.7.** Show (****) that any (acyclic) right linear convergent PCFG can be put in $\varepsilon$-free form in polynomial time.

**Proposition 6.14 (Probable Strings are Short).** Let $G = \langle N, \Sigma, P, S, \rho \rangle$ be a right-linear reduced convergent PCFG in $\varepsilon$-free form and $w$ be a sequence in $\Sigma^*$ with $\rho(w) \geq p$. Then $|w| \leq \frac{Z(S) |N|^2}{p} + |N|$.

**Proof.** Let $w = a_1 \cdots a_\ell$ be a string of length $\ell$ with $\rho(w) \geq p$ (with $a_i$ in $\Sigma$ for every $i$). Any derivation for $w$ in the $\varepsilon$-free grammar $G$ is necessarily of the form

$$S = A_1 \xrightarrow{p_1} a_1A_2 \xrightarrow{p_2} a_1a_2A_3 \xrightarrow{p_3} \cdots \xrightarrow{p_\ell} a_1 \cdots a_\ell \quad (6.17)$$

using the productions $p_i = A_i \rightarrow a_iA_{i+1}$ for $1 \leq i < \ell - 1$ and $p_\ell = A_\ell \rightarrow a_\ell$. Define

$$D_w = \{ \pi \in P^* \mid S \xrightarrow{\pi} \ast \} \quad (6.18)$$

the set of derivations of $w$. Assuming some total ordering $\prec$ over the nonterminals in $N$, we write $D_w^A$ for the subset of $D_w$ where $A$ is the nonterminal that occurs as left-hand side the most often, using $\prec$ to choose between ties. Then

$$D_w = \bigcup_{A \in N} D_w^A; \quad \rho(w) = \sum_{\pi \in D_w} \rho(\pi) \geq p, \quad (6.19)$$
hence there exists \( A \) in \( N \) such that

\[
\sum_{\pi \in D^A_w} \rho(\pi) \geq \frac{p}{|N|} .
\] (6.20)

In any derivation \( \pi = p_1 \cdots p_\ell \) in \( D^A_w \), \( A \) appears as left-hand side at least \( \ell/|N| \) times. By removing a subderivation between two such occurrences, we obtain a derivation for a shorter sequence with at least the same probability. We call such shorter sequences alternatives for \( \pi \); there are at least \( \ell/|N| - 1 \) alternatives, that we gather in a set \( \text{Alt}(\pi, A) \). Hence

\[
\sum_{\pi' \in \text{Alt}(\pi, A)} \rho(\pi') \geq \left( \frac{\ell}{|N|} - 1 \right) \rho(\pi) .
\] (6.21)

We want to sum the probability mass of alternatives over all \( \pi \) in \( D^A_w \); however, there might be common alternatives for different derivations \( \pi_1 \) and \( \pi_2 \). This is not an issue, as shown by the following claim:

Claim 6.14.1. Let \( \pi_1 \) and \( \pi_2 \) be two different derivations in \( D^A_w \), and let \( \pi \) be a derivation in \( \text{Alt}(\pi_1, A) \cap \text{Alt}(\pi_2, A) \). Then \( \rho(\pi) \geq \rho(\pi_1) + \rho(\pi_2) \).

Hence

\[
\sum_{\pi \in D^A_w} \sum_{\pi' \in \text{Alt}(\pi, A)} \rho(\pi') \geq \sum_{\pi \in D^A_w} \left( \frac{\ell}{|N|} - 1 \right) \rho(\pi) \geq \left( \frac{\ell}{|N|} - 1 \right) \frac{p}{|N|} .
\] (6.22)

The probability mass on the left side of the previous inequality is contributed by strings different from \( w \); hence, summing with the probability of \( w \), we obtain

\[
\left( \frac{\ell}{|N|} - 1 \right) \frac{p}{|N|} + p \leq Z(S)
\]

thus

\[
\ell \leq \frac{(Z(S) - p)|N|^2}{p} + |N| ,
\]

from which we deduce the desired bound since \( Z(S) \geq \rho(w) \geq p \).

**General Case.** As mentioned at the beginning of this section, MPS is undecidable in general, already for right-linear PCFGs. However, if the PCFG is convergent, then MPS is decidable (de la Higuera et al., 2014, Proposition 8). The principle of the algorithm is to enumerate the strings \( w \in \Sigma^* \) and compute their probabilities \( \langle [G], w \rangle \). If we find a string with probability at least \( p \) we can stop. Otherwise, since \( Z(S) \) is finite and equal to \( \sum_{w \in \Sigma^*} \langle [G], w \rangle \), eventually the sum of probabilities of all the tested strings will exceed \( Z(S) - p \), and we will know for sure that no string has probability at least \( p \).
Chapter 7

Categorial Grammars

The last approach to formal syntax we will consider in these notes is also one of the oldest: categorial grammars were indeed introduced by [Bar-Hillel in 1953] based on earlier ideas of [Ajdukiewicz (1935)].

In their barest form, categorial grammars are defined using residuation types, which are usually called syntactic types or categories. Consider a finite set of primitive types $\Gamma$. We define syntactic types over $C$ as terms $\gamma$ defined by the abstract syntax

$$C ::= p \mid C \setminus C \mid C / C$$

where $p$ is in $\Gamma$; let $C(\Gamma)$ be the set of syntactic types over $\Gamma$. The second part of the course will better emphasize the interest of categorial grammars for semantics representation. Indeed, one can apply the Curry-Howard isomorphism and associate lambda terms (modeling semantics) to syntactic types (modeling syntax).

The interpretation of sequences of syntactic types over the free semigroup $\langle \Sigma^+, \cdot \rangle$ relies on a finite lexical relation $\ell$ between $\Sigma$ and $C(\Gamma)$, mapping words to set of syntactic types, so that the interpretation of $\llbracket C \rrbracket_\ell$ a syntactic type $C$ given $\ell$ is a subset of $\Sigma^+$:

$$\llbracket p \rrbracket_\ell = \ell^{-1}(p)$$
$$\llbracket C_1 \setminus C_2 \rrbracket_\ell = (\llbracket C_1 \rrbracket_\ell)^{-1} \cdot \llbracket C_2 \rrbracket_\ell$$
$$\llbracket C_1 / C_2 \rrbracket_\ell = \llbracket C_1 \rrbracket_\ell \cdot (\llbracket C_2 \rrbracket_\ell)^{-1}.$$ 

Having a distinguished axiom type $S$ then allows to define a language over $\Sigma$ as the interpretation $\llbracket S \rrbracket_\ell$. Categorial grammars are interested in quasi orderings $\vdash$ of derivability between sequences of types, such that if $\gamma \vdash \gamma'$ is derivable then $\llbracket \gamma \rrbracket_\ell \subseteq \llbracket \gamma' \rrbracket_\ell$.

Definition 7.1. A (product-free) categorial grammar $C = \langle \Sigma, \Gamma, S, \vdash, \ell \rangle$ comprises a finite alphabet $\Sigma$, a finite set of primitive types $\Gamma$, a distinguished syntactic type $S$ in $C(\Gamma)$, a derivability quasi ordering $\vdash$ over $(C(\Gamma))^+$, and a finite lexical relation $\ell$ in $\Sigma \times C(\Gamma)$.

The language of $C$ is defined as

$$L(C) = \{a_1 \cdots a_n \in \Sigma^+ \mid n > 0, \exists C_1 \in \ell(a_1), \ldots, \exists C_n \in \ell(a_n), C_1 \cdots C_n \vdash S\}.$$

We present two different systems to define derivability quasi orderings in sections 7.1 and 7.2.

See the lecture notes of Retoré (2005) for a more detailed treatment of categorial grammars.
7.1 AB Categorial Grammars

The derivability quasi ordering for AB categorial grammars (named after Ajdukiewicz and Bar-Hillel) can be defined by a string rewrite system $R$ over the free semigroup $(\mathbf{C}(\Gamma)^+, \cdot)$ with the two cancellation rule schemata

\[
B \cdot (B \setminus A) \rightarrow A \quad (\setminus / E)
\]
\[
(A / B) \cdot B \rightarrow A \quad (/ E)
\]

for all $A, B$ in $\mathbf{C}(\Gamma)$, so that $\vdash$ is the reflexive transitive closure of the single-step rewrite relation $\Rightarrow$—which is by definition a quasi ordering.

Example 7.2. Let $\Gamma = \{n, s\}$ and let us consider the following lexical relation:

<table>
<thead>
<tr>
<th>$\Sigma$</th>
<th>$\mathbf{C}(\Gamma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill, John, Mary, mushrooms</td>
<td>n</td>
</tr>
<tr>
<td>the, white</td>
<td>n / n</td>
</tr>
<tr>
<td>works</td>
<td>n \ n</td>
</tr>
<tr>
<td>likes</td>
<td>(n \ s) / n</td>
</tr>
<tr>
<td>thinks</td>
<td>(n \ s) / s</td>
</tr>
<tr>
<td>tells</td>
<td>((n \ s) / s) / n</td>
</tr>
<tr>
<td>really</td>
<td>(n \ s) / (n \ s)</td>
</tr>
<tr>
<td>who</td>
<td>(n \ n) / (n \ s)</td>
</tr>
</tbody>
</table>

We can derive sentences such as

- Bill really likes mushrooms.
- John thinks Bill likes mushrooms.
- Bill who likes mushrooms likes white mushrooms.
- John tells Mary Bill likes mushrooms.

Observe that the principles of lexicalization put forward in Section 5.1 for TAGs are also at work here: the syntactic types associated to works, likes, thinks, and tells reflect their subcategorization frames (only a subject, a subject and a nominal object, a subject and a clausal object, and a subject and both a nominal and a clausal object resp.).

7.1.1 Alternative Views

Axiomatic View  Other definitions are possible; for instance by algebraic laws over $(\mathbf{C}(\Gamma))^+$, where the laws left implicit in the string rewrite definition have to be expressed. More precisely, by definition of a semigroup, the rules are taken modulo associativity of $\cdot$, and by definition of a string rewrite system, $\vdash$ is monotone wrt. concatenation:

\[
A \vdash A \quad \text{(reflexivity)}
\]
\[
A \vdash B \text{ and } B \vdash C \implies A \vdash C \quad \text{(transitivity)}
\]
\[
A \cdot (B \cdot C) \vdash (A \cdot B) \cdot C \quad \text{(associativity)}
\]
\[
(A \cdot B) \cdot C \vdash A \cdot (B \cdot C) \quad \text{(associativity)}
\]
\[
A \vdash B \implies A \cdot C \vdash B \cdot C \quad \text{(left monotonicity)}
\]
\[
A \vdash B \implies C \cdot A \vdash C \cdot B \quad \text{(right monotonicity)}
\]

for all $A, B, C$ in $(\mathbf{C}(\Gamma))^+$. 
Proof-Theoretic View Yet another presentation would be as a natural deduction sequent calculus: a sequent $\gamma \vdash C$ pairs up a non empty sequence $\gamma$ in $(C(\Gamma))^+$ with a syntactic type $C$ in $C(\Gamma)$. The derivability quasi ordering is then defined by the following substructural proof system

$$
\begin{align*}
\frac{}{C \vdash C} \text{ (Id)} & \quad \frac{\beta \vdash B}{\beta \alpha \vdash A} \text{ (\land E)} & \quad \frac{\alpha \vdash A / B \quad \beta \vdash B}{\alpha \beta \vdash A} \text{ (/ E)}
\end{align*}
$$

where the two rules (\land E) and (/ E) are non-commutative versions of the traditional modus ponens rule (which we will recall later as rule (\rightarrow E)).

7.1.2 Equivalence with Context-Free Grammars

The equivalence of AB categorial grammars and context-free grammars is originally due to Bar-Hillel et al. (1960).

From AB Categorial Grammars to CFGs The encoding relies on a subformula property for the cancellation rules: the resulting types are always subtypes of the left-hand types. Thus, given an AB categorial grammar $C = \langle \Sigma, \Gamma, S, \vdash, \ell \rangle$, the set of types that can appear during a derivation is in $\text{sub}(\ell(\Sigma))^+$. A second property is a context-freeness one: a derivation of form $\beta R = \Rightarrow n A_1 \cdots A_m$ can be decomposed into $m$ subderivations $\beta_i R = \Rightarrow n_i A_i$ with $n = n_1 + \cdots + n_m$ and $\beta = \beta_1 \cdots \beta_m$.

Using these two properties, it is straightforward to check that the CFG $G = \langle \text{sub}(\ell(\Sigma)), \Sigma, P, S \rangle$ with

$$
P = \{ A \to B \ (B \ \land A) \mid (B \ \land A) \in \text{sub}(\ell(\Sigma)) \} \\
\cup \{ A \to (A / B) \ B \mid (A / B) \in \text{sub}(\ell(\Sigma)) \} \\
\cup \{ A \to a \mid (a, A) \in \ell \}
$$

encodes $C$.

From CFGs to AB Categorial Grammars Recall that any CFG can be transformed into an equivalent CFG in (quadratic) Greibach normal form (GNF, Greibach, 1965), i.e. such that all its productions are of form

$$
S \to \varepsilon \\
A \to a\alpha \\
sample { \in \Sigma, \alpha \in \{ N \setminus \{ S \} \} }^2
$$

This yields a straightforward encoding of a CFG $G$ with $\varepsilon \notin L(G)$ in GNF into an AB categorial grammar $C = \langle N, \Sigma, \vdash, S, \ell \rangle$ with

$$
\ell = \{(a, (A / C) / B) \mid A \to aBC \in P \} \\
\cup \{(a, A / B) \mid A \to aB \in P \} \\
\cup \{(a, A) \mid A \to a \in P \}.
$$

Exercise 7.1. Fill out the missing details of the proof of equivalence between AB categorial grammars and context-free grammars.

Exercise 7.2. The Dyck language $D_n$ over $n$ pairs of parentheses $a_i, \bar{a}_i$ is generated by the CFG with productions $\{ S \to a_iS \bar{a}_i \mid 1 \leq i \leq n \} \cup \{ S \to SS \} \cup \{ S \to \varepsilon \}$. Give an AB categorial grammar for the language $D_n$ where $\$ is an endmarker distinct from all the $a_i, \bar{a}_i$. 
7.1.3 Structural Limitations

There exist some structural limitations to AB categorial grammars. Consider for instance that introducing a subordination as in the mushrooms that Bill likes; in this frame the type for that would be \((n \setminus n) / (s / n)\) since Bill likes is intuitively of type \(s / n\). However, we cannot derive \(n \cdot ((n \setminus s) / n) \vdash s / n\) using only the cancellation rules, although it would be correct wrt. the free semigroup interpretation.

Several extensions were defined in order to circumvent the limitations of AB categorial grammars (often sparked by semantic rather than syntactic motivations):

- **type raising** \(B \rightarrow (A / B) \setminus A\) and \(B \rightarrow A / (B \setminus A)\),
- **composition** \((A / B)(B / C) \rightarrow A / C\) and \((C \setminus B)(B \setminus A) \rightarrow C \setminus A\),
- **Geach rules** \(A / B \rightarrow (A / C) / (B / C)\) and \(B \setminus A \rightarrow (C \setminus B) \setminus (C \setminus A)\).

All these extensions are captured by the Lambek calculus.

7.2 Lambek Grammars

The Lambek calculus ([Lambek](Lambek, 1958)) generalizes all the extensions of AB categorial rules by proposing instead to add introduction rules to the intuitionistic fragment of Section 7.1.1.

7.2.1 Background: Substructural Proof Systems

Let us first recall the implicative and conjunctive fragment of propositional calculus, with a presentation based on natural deduction for intuitionistic logic. A **proposition** \(C\) is defined in this fragment as

\[
C ::= p \mid C \rightarrow C \mid C \land C \quad \text{(propositions)}
\]

where \(p\) is taken from a set \(\Gamma\) of atomic propositions. An **assumption** a sequence of propositions, and a **judgement** has the form \(\gamma \vdash C\), meaning that from assumptions \(\gamma\) one can conclude proposition \(C\). A version of the propositional calculus is then defined by the rules

\[
\frac{}{C \vdash C} \quad \text{(Id)}
\]

\[
\frac{\alpha \beta \vdash A}{\beta \alpha \vdash A} \quad \text{(Ex)} \qquad \frac{\alpha A A \vdash B}{\alpha A \vdash B} \quad \text{(Con)} \qquad \frac{\alpha \vdash B}{\alpha A \vdash B} \quad \text{(W)}
\]

\[
\frac{\alpha \vdash A}{\alpha \vdash A \rightarrow B} \quad \text{(-I)} \qquad \frac{\beta \vdash B}{\beta \alpha \vdash A} \quad \text{(-E)}
\]

\[
\frac{\alpha \vdash A \beta \vdash B}{\alpha \beta \vdash A \land B} \quad \text{(-I)} \qquad \frac{\alpha \vdash A \land B}{\beta \alpha \vdash B \land C} \quad \text{(-E)}
\]

where **(Ex)**, **(Con)**, and **(W)** are the **structural rules** of exchange, contraction, and weakening, respectively.
There exists a rich literature on substructural logics, in particular linear logic (Girard, 1987) allows to restrict the use of the (Con) and (W) rules. If we completely forbid these two rules, then the fragment of propositional calculus we just saw corresponds to the multiplicative fragment of intuitionistic linear logic, which displays linear implication \(\multimap\) instead of implication, and tensor product \(\otimes\) instead of conjunction.

One can go a step further and also forbid the exchange rule (Ex). It has however the effect of refining the implication rules (\(\rightarrow I\)) and (\(\rightarrow E\)) into left and right implications, while conjunction becomes a form of concatenation, which we denote by \(\cdot\):

\[
\frac{C \vdash C}{(\text{Id})}
\]

\[
\frac{B\alpha \vdash A}{\alpha \vdash B \setminus A} (\setminus)\]

\[
\frac{\beta \vdash B \quad \alpha \vdash B \setminus A}{\alpha \beta \vdash A} (\setminus E)
\]

\[
\frac{\alpha B \vdash A}{\alpha \vdash A / B} (/)\]

\[
\frac{\beta \vdash B}{\alpha \beta \vdash A} (\text{ divides } A)
\]

\[
\frac{\alpha \vdash A \quad \beta \vdash B}{\alpha \beta \vdash A \cdot B} (\cdot)\]

\[
\frac{\beta \vdash A \cdot B \quad \alpha AB \gamma \vdash C}{\alpha \beta \gamma \vdash C} (\cdot E)
\]

Note that this system simply adds insertion counterparts to (\(\setminus E\)) and (\(\text{ divides } A\)) and product rules to the system of Section 7.1.1. What we have just defined is a natural deduction version of the Lambek calculus.

Example 7.3. Here is a derivation of a type raising rule from Section 7.1.3:

\[
\frac{A / B \vdash A / B}{(\text{Id})} \quad \frac{B \vdash B}{(\text{Id})} \quad \frac{(A / B)B \vdash A}{(\text{ divides } A)}\]

\[
\frac{B \vdash (A / B) \setminus A}{(\setminus)}
\]

Exercise 7.3. Show that the composition and Geach rules from Section 7.1.3 are also derivable in this natural deduction version of the Lambek calculus.

7.2.2 Lambek Calculus

Lambek (1958) actually presents his calculus in Gentzen sequent style, with rules

\[
\frac{C \vdash C}{(\text{Id})}
\]

\[
\frac{\beta \vdash B \quad \alpha \beta \gamma \vdash A}{\alpha \beta \gamma \vdash A} (\text{Cut})
\]

\[
\frac{B\alpha \vdash A}{\alpha \vdash B \setminus A} (\setminus R)\]

\[
\frac{\beta \vdash B \quad \alpha A\gamma \vdash C}{\alpha \beta (B \setminus A) \gamma \vdash C} (\setminus L)
\]

\[
\frac{\alpha B \vdash A}{\alpha \vdash A / B} (/ R)\]

\[
\frac{\beta \vdash B}{\alpha (A / B) \beta \gamma \vdash C} (/ L)
\]

\[
\frac{\alpha \vdash A \quad \beta \vdash B}{\alpha \beta \vdash A \cdot B} (\cdot R)
\]

\[
\frac{\alpha AB \beta \vdash C}{\alpha (A \cdot B) \beta \gamma \vdash C} (\cdot L)
\]

See e.g. Troelstra (1992) for a textbook on linear logic, and the course MPRI 2-1.

One can go one more step further and define a non-associative calculus, where sequents left parts are terms instead of sequences. This results in a variant called the non-associative Lambek calculus (Lambek, 1961).
Again, one can recognize a non-commutative variation of the multiplicative fragment of intuitionistic linear logic.

### Cut Elimination

The Lambek calculus enjoys cut elimination, i.e. for any proof in the sequent calculus, there exists a proof that does not employ the (Cut) rule. A byproduct of cut elimination is that cut-free proofs have the subformula property, in the following strong sense: each application of the rules besides (Cut) adds one symbol from \{\land, /, \cdot\} to the sequent. Thus working our way backward from a sequent \( \gamma \vdash C \) to be proven, there are only finitely many cut-free proofs possible: the calculus is decidable.

\[(*)\] **Exercise 7.4.** Show that the decision procedure sketched above is in NP.

Let us prove the cut elimination property. Suppose both \( \beta \vdash B \) and \( \alpha B \gamma \vdash A \) are provable in the cut-free calculus; we want to show that \( \alpha \beta \gamma \vdash A \) is also provable. The proof proceeds by induction on the sum of the sizes of the sequents—defined as their number of symbols from \{\land, /, \cdot\}—and consists mostly of a large case analysis depending on the last rule employed to obtain the sequents before the cut:

1. if either sequent is the result of (Id), then the other is already the result of the cut,

2. if \( \beta \vdash B \) is the result of a rule that did not introduce the main connective of \( B \), i.e. rule (\land L), (\land R), or (\cdot L), then there is a premise of form \( \beta' \vdash B \) of smaller size, which by induction hypothesis yields \( \alpha \beta' \gamma \vdash A \) in the cut-free calculus, and later \( \alpha \beta \gamma \vdash A \) by the same rule application that lead from \( \beta' \vdash B \) to \( \beta \vdash B \),

3. if \( \alpha B \gamma \vdash A \) is the result of a rule that did not introduce the main connective of \( B \), then there is a premise of form \( \alpha' B \gamma' \vdash A' \) of smaller size, which by induction hypothesis yields a cut-free proof of \( \alpha' B \gamma' \vdash A' \), and an application of the rule that lead from \( \alpha' B \gamma' \vdash A' \) to \( \alpha B \gamma \vdash A \) yields the result,

4. if \( B = C \circ D \) is the result of (\cdot R) and (\cdot L), and we can replace

\[
\begin{align*}
\beta' \vdash C & \quad \beta'' \vdash D \\
\alpha \beta' \beta'' \vdash C \circ D & \quad \alpha CD \gamma \vdash A \\
\text{by the proof} & \\
\beta' \vdash C & \quad \beta'' \vdash D \\
\alpha C \beta'' \gamma \vdash A & \quad \alpha CD \gamma \vdash A \\
\alpha \beta' \beta'' \gamma \vdash A & \text{by the rule (Cut)} \\
\end{align*}
\]

with both (Cut) applications are on smaller sequents, thus provable in the cut-free calculus by induction hypothesis,

5. if \( B = C / D \) is the result of (/ R) and (/ L), and we can replace

\[
\begin{align*}
\beta' \vdash C & \quad \beta'' \vdash D \\
\alpha \beta' \beta'' \gamma \vdash A & \quad \alpha C \gamma \vdash A \\
\text{by the proof} & \\
\beta' \vdash C & \quad \beta'' \vdash D \\
\alpha (C / D) \beta'' \gamma \vdash A & \quad \alpha C \gamma \vdash A \\
\alpha \beta' \beta'' \gamma \vdash A & \text{by the rule (Cut)} \\
\end{align*}
\]
by the proof

\[
\frac{\beta'' \vdash D \quad \beta' D \vdash C \quad \alpha C \gamma \vdash A}{\alpha \beta' \beta'' \gamma \vdash A} \text{ (Cut)}
\]

where both (Cut) applications are on smaller sequents, thus provable in the cut-free calculus by induction hypothesis.

6. if \( B = C \setminus D \) is the result of (\( \setminus R \)) and (\( \setminus L \)), the case is symmetric to case 5.

**Encoding Natural Deduction** The natural deduction rules (\( \setminus E \)), and (\( \bullet E \)) can be obtained as (the case of (\( /E \)) being symmetric to that of (\( \setminus E \))):

\[
\begin{align*}
\alpha & \vdash B \setminus A & \frac{\beta \vdash B}{\beta \alpha \vdash A} \text{ (\( \setminus L \))} \\
& & \frac{A \vdash A}{\beta \alpha \vdash A} \text{ (\( \setminus R \))} \\
\end{align*}
\]

Conversely, one can prove that the Lambek calculus is actually equivalent to its natural deduction presentation (see e.g. Retoré, 2005, Section 2.6).

### 7.2.3 Equivalence with Context-Free Grammars

Although the Lambek calculus is strictly more expressive than the two cancellation rules (\( \setminus E \)) and (\( /E \)) of \( AB \) categorial grammars, **Lambek grammars**, i.e. the categorial grammars that employ the (product-free) Lambek calculus for the derivability quasi ordering \( \vdash \), are not more expressive: they define exactly the context-free languages. This result was conjectured by Chomsky in the 1960s but remained open until the 1992 proof of Pentus.

We merely give a taste of the proof in the product-free case (Pentus, 1997). It defines the norm \( \| \gamma \| \) of a product-free type sequence \( \gamma \) in \((C(\Gamma))^+\) as its number of atomic type occurrences, i.e.

\[
\| p \| = 1, \quad \| C \setminus C' \| = \| C / C' \| = \| C \| + \| C' \|, \quad \| C_1 \cdots C_n \| = \| C_1 \| + \cdots + \| C_n \|.
\]

and uses it to define the finite sets of types and type sequences

\[
C_m(\Gamma) = \{ C \in C(\Gamma) \mid \| C \| \leq m \} \quad L_m(\Gamma) = \{ \gamma \in (C(\Gamma))^+ \mid \| \gamma \| \leq 2m \}
\]

for all \( m \geq 0 \). The \((m, \Gamma)\)-bounded Lambek calculus is then defined by the two rules

\[
\gamma \vdash C \quad (Ax) \quad \frac{\beta \vdash B \quad \alpha B \gamma \vdash A}{\alpha \beta \gamma \vdash A} \text{ (Cut)}
\]

where \( \gamma \vdash C \) in (\( Ax \)) is any sequent in \( L_m(\Gamma) \times C_m(\Gamma) \) provable in the product-free Lambek calculus (thus there are only finitely many such axioms for a fixed \((m, \Gamma)\) pair).

**Theorem 7.4** (Pentus, 1997). Let \( B_1, \ldots, B_n, A \) be types in \( C_m(\Gamma) \). If \( B_1 \cdots B_n \vdash A \) is provable in the product-free Lambek calculus, then it is also provable in the \((m, \Gamma)\)-bounded Lambek calculus.
Thus, given a Lambek categorial grammar $\mathcal{C} = \langle \Sigma, \Gamma, \vdash, \ell \rangle$, there exist $m \geq 0$ and $\Gamma' \subseteq \Gamma$ s.t. $S \in C_m(\Gamma')$ and $\ell(\Sigma) \subseteq C_m(\Gamma')$. We can construct a context-free grammar $\mathcal{G} = \langle C_m(\Gamma'), \Sigma, P, S \rangle$ with

$$P = \{ C \to \gamma \mid C \in C_m(\Gamma'), \gamma \in L_m(\Gamma'), \gamma \vdash C \text{ provable} \} \cup \{ A \to a \mid (a, A) \in \ell \} .$$

Context-free derivations then simulate the action of the (Cut) rule in the $(m, \Gamma')$-calculus.

**(∗∗)** Exercise 7.5. Prove using [Theorem 7.4](#) the equivalence of $\mathcal{C}$ and $\mathcal{G}$ as defined above.
Chapter 8

First-Order Semantics

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

8.1 Formal Semantics

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

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

Example 8.1. For instance, assume our vocabulary includes \( John^{(0)} \) as a constant denoting \( John \), along with \( apple^{(1)} \), \( red^{(1)} \), and \( eat^{(2)} \), we can associate the sentence

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

(8.1)

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

\[
a, j \in W \quad a \in red \quad a \in apple \\
j = John \quad (j, a) \in eat
\]

in which case the sentence is satisfiable using the assignment \( \{x \mapsto a\} \).

An interesting consequence of this analysis is that paraphrases are typically associated with the same semantics: \( (8.1) \) could for instance be the formalisation of

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

The kind of modelling that underlies Example 8.1 is a rather straightforward one: named entities (e.g. John, or the President) are interpreted as constants, properties (e.g. red, apple) as unary relations, and verbs as relations with an arity equal to the number of arguments present in their subcategorisation frames.

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

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

Using the approach of Example 8.1, we need to introduce several relations eat\(^{(i)}\) largely beyond the simple choice between the intransitive eat\(^{(1)}\) and transitive eat\(^{(2)}\) forms of eat:

\[
eat^{(1)}_1(\text{John})(0)
\]
\[
\exists x.\eat^{(2)}_2(\text{John}(0),x) \land \text{red}^{(1)}(x) \land \text{apple}^{(1)}(x)
\]
\[
\exists xy.\eat^{(3)}_3(\text{John}(0),x,y) \land \text{apple}^{(1)}(x) \land \text{park}^{(1)}(y)
\]
\[
\exists y.\eat^{(2)}_4(\text{John}(0),y) \land \text{park}^{(1)}(y)
\]
\[
\exists xy.\eat^{(4)}_5(\text{John}(0),x,y,\text{slowly}(0)) \land \text{red}^{(1)}(x) \land \text{apple}^{(1)}(x) \land \text{park}^{(1)}(y)
\]

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

How can we relate all the variations of eat so that e.g. (8.6) entails each of (8.2–8.5)? One possibility is to add explicit meaning postulates like

\[
\forall jxy.\eat^{(3)}_3(j,x,y) \supset \eat^{(2)}_2(j,x)
\]
\[
\forall jx.\eat^{(2)}_2(j,x,y) \supset \eat^{(1)}_1(j)
\]
\[
\ldots
\]

Similarly, we could treat slowly and the locative in as modal operators and rewrite (8.6) as

\[
\exists xy.\text{in}^{(2)}(\text{slowly}^{(1)}(\eat^{(2)}_2(\text{John}(0),x)),y) \land \text{red}^{(1)}(x) \land \text{apple}^{(1)}(x) \land \text{park}^{(1)}(y)
\]

along with the schemata

\[
\forall Py.\text{in}^{(2)}(P,y) \supset P
\]
\[
\forall P.\text{slowly}^{(1)}(P) \supset P
\]

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

\[
\exists xy.\text{slowly}^{(1)}(\text{location}^{(2)}(\eat^{(2)}_2(\text{John}(0),x),y)) \land \text{red}^{(1)}(x) \land \text{apple}^{(1)}(x) \land \text{park}^{(1)}(y)
\]
instead, and proving the equivalence of (8.10) and (8.13) would require yet more machinery. (We will however return to modal operators later in Section 8.3.)

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

\[
eat_1^{(1)}(j) \overset{\text{def}}{=} \exists y a . \eat_5^{(4)}(j, x, y, a)
\]

\[
eat_2^{(2)}(j, x) \overset{\text{def}}{=} \exists y a . \eat_5^{(4)}(j, x, y, a)
\]

\[
eat_3^{(3)}(j, x, y) \overset{\text{def}}{=} \exists a . \eat_5^{(4)}(j, x, y, a)
\]

\[
eat_4^{(2)}(j, y) \overset{\text{def}}{=} \exists y a . \eat_5^{(4)}(j, x, y, a)
\]

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

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

\[
\exists e . \eat_1^{(2)}(e, \text{John}^{(0)})
\]

\[
\exists e x . \eat_2^{(3)}(e, \text{John}^{(0)}, x) \land \text{red}^{(1)}(x) \land \text{apple}^{(1)}(x)
\]

\[
\exists e x y . \eat_3^{(3)}(e, \text{John}^{(0)}, x) \land \text{apple}^{(1)}(x) \land \text{park}^{(1)}(y) \land \text{location}^{(2)}(e, y)
\]

\[
\exists e y . \eat_4^{(2)}(e, \text{John}^{(0)}) \land \text{park}^{(1)}(y) \land \text{location}^{(2)}(e, y)
\]

\[
\exists e x y . \eat_5^{(3)}(e, \text{John}^{(0)}, x) \land \text{red}^{(1)}(x) \land \text{apple}^{(1)}(x) \land \text{park}^{(1)}(y) \land \text{location}^{(2)}(e, y)
\]

\[
\land \text{slowly}^{(1)}(e)
\]

This Davidsonian analysis succeeds in reducing the variations to the two main forms of eat. It also yields a rather more natural way of handling time and aspects modifiers like slowly. Note that the distinction between intransitive and transitive forms of verbs are better motivated than the ones between say (8.2) and (8.3): contrast for instance

I sank the Bismark.
I sank.

where the transitive usage does not imply the intransitive one.

8.1.2 Thematic Roles

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

\[
\exists e . \eat_1^{(1)}(e) \land \text{agent}^{(2)}(e, \text{John}^{(0)})
\]

\[
\exists e x . \eat_1^{(1)}(e) \land \text{agent}^{(2)}(e, \text{John}^{(0)}) \land \text{patient}^{(2)}(e, x) \land \text{apple}^{(1)}(x)
\]

This is known as a neo-Davidsonian analysis (Parsons 1990).
Role | Typical use
---|---
agent | John eats
patient | John eats an apple.
experiencer | John regrets his actions.
cause | The crisis worries John.
theme | John asks a question.
beneficiary | John gives Mary a kiss.

Table 8.1: A basic set of thematic roles.

the subject and the verb:

\[ \exists e. \text{sink}^{(1)}(e) \land \text{agent}^{(2)}(e, I^{(0)}) \land \text{patient}^{(2)}(e, \text{Bismark}^{(0)}) \]  
(8.25)

\[ \exists e. \text{sink}^{(1)}(e) \land \text{patient}^{(2)}(e, I^{(0)}) \]  
(8.26)

The definition of a fixed set of thematic roles and how to classify the different uses are of course problematic; Table 8.1 proposes a very simple account.

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

### 8.2 A Dip into Description Logics

Let us make a short detour through a family of logics primarily developed for knowledge representation. Basic description logics, similarly to the modal logics we will see in Section 8.3, can be translated into first-order logic, so their use does not yield any additional expressive power. Their interest is rather that they force us into well-behaved fragments of FO, where we are able to draw inferences and reason automatically.

#### 8.2.1 A Basic Description Logic

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

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

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

where \( a \) ranges over \( A \) and \( r \) over \( R \). This syntax can be enriched by \( \bot \equiv \neg \top \), \( C \sqcup D \equiv \neg(\neg C \sqcap \neg D) \), and \( \forall r_j.C \equiv \neg \exists r_j.\neg C \). A concept defines a subset \( \mathcal{M}(C) \) of a model \( \mathcal{M} \):

\[
\begin{align*}
\mathcal{M}(\top) & \equiv W \\
\mathcal{M}(C \sqcap D) & \equiv \mathcal{M}(C) \cap \mathcal{M}(D) \\
\mathcal{M}(\exists r.C) & \equiv r^{-1}(\mathcal{M}(C)) \\
\mathcal{M}(\neg C) & \equiv W \setminus \mathcal{M}(C)
\end{align*}
\]
The basic questions one might ask on concepts are consistency ones, i.e. whether there exists a model \( \mathfrak{M} \) such that \( [C]^{\mathfrak{M}} \) is non-empty. An especially useful case is that of an inclusion \( C \subseteq D \), i.e. the inconsistency of \( C \cap \neg D \).

**Examples** Consider the sentence *Every man loves a woman*. Its most common semantic reading can be formalised in first-order logic as
\[
\forall y. \text{man}(1)(y) \supset \exists x. \text{woman}(1)(x) \land \text{love}(2)(y, x)
\] (8.27)

It can also be formalised as a consistency question in ALC:
\[
\text{Man} \sqsubseteq \exists \text{love. Woman} \tag{8.28}
\]

where the binary relation \( \text{love}(2) \) is translated as a role, and the unary predicates \( \text{man}(1) \) and \( \text{woman}(1) \) as atomic concepts. The sentence *A man eats an apple* is captured by the consistency of
\[
\text{Man} \sqcap \exists \text{eat. Apple} \tag{8.29}
\]

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

For instance, the sentence *John eats a red apple* can be checked by
\[
\{John\} \sqsubseteq \exists \text{eat.}(\text{Apple} \cap \text{Red}) \tag{8.30}
\]

and the sentence *Helen of Troy is loved by every man in Greece* by
\[
(\text{Man} \sqcap \exists \text{inhabit.}\{\text{Greece}\}) \sqsubseteq \exists \text{love.}\{\text{Helen of Troy}\} \tag{8.31}
\]

### 8.2.2 Translation into First-Order Logic

As hinted by the first-order and ALC formalisations in (8.27)–(8.28), there is a translation of ALC into first-order logic. Every nominal \( i \) is associated with a constant symbol \( i^{(0)} \), every atomic concept \( a \) with a unary predicate \( a^{(1)} \), and every role \( r \) with a binary relation \( r^{(2)} \). Then, a concept definition \( C \) is translated into a first-order formula \( ST_x(C) \) with a single free variable \( x \):

\[
\begin{align*}
\text{ST}_x(\top) & \overset{\text{def}}{=} x = x \\
\text{ST}_x(\{i\}) & \overset{\text{def}}{=} x = i^{(0)} \\
\text{ST}_x(\neg C) & \overset{\text{def}}{=} \neg \text{ST}_x(C) \\
\text{ST}_x(C \sqcap D) & \overset{\text{def}}{=} \text{ST}_x(C) \land \text{ST}_x(D) \\
\text{ST}_x(\exists r.C) & \overset{\text{def}}{=} \exists y. r^{(2)}(x, y) \land \text{ST}_y(C)
\end{align*}
\]

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

\[
\begin{align*}
\text{ST}(C) & \overset{\text{def}}{=} \exists x. \text{ST}_x(C) \\
\text{ST}(C \subseteq D) & \overset{\text{def}}{=} \forall x. \text{ST}_x(C) \supset \text{ST}_x(D)
\end{align*}
\]

These definitions result for instance in the following first-order semantics for (8.31):
\[
\forall y. (\text{man}(1)(y) \land \text{inhabit}(2)(y, \text{Greece}^{(0)}) \supset \text{love}(2)(y, \text{Helen of Troy}^{(0)}) \tag{8.32}
\]

Two important remarks can be made regarding this translation:

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

Each of these conditions is enough to yield decidability of ALC; see Section 8.4.
8.3 Modal Semantics

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

John is ___ happy.

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

8.3.1 Background: Modal Logic

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

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

\[
\varphi ::= p \mid \top \mid \neg \varphi \mid \varphi \lor \varphi' \mid \lozenge \varphi
\]

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

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

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

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

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

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

- In epistemic logic, we reason about knowledge of agents (mind the difference with beliefs): instead of writing \( \Box \varphi \) to denote the fact that “the agent knows \( \varphi \)”, we write \( K \varphi \). Epistemic logic is typically interpreted over transitive, symmetric, and reflexive frames, i.e. where \( R \) is an equivalence relation. If the knowledge of several agents is to be modelled, we can introduce multiple relations \( R_a \) and modalities \( K_a \), one for each agent \( a \).
In the basic temporal logic, \( \lozenge \varphi \) denotes that “at some future point, \( \varphi \) holds”, written \( F \varphi \). Its dual \( G \varphi \) means that in all future points, \( \varphi \) holds. Its converse \( P \) allows to reason about the past, and is defined by \( \mathfrak{M}, w \models P \varphi \) iff there exists \( w' R w \) s.t. \( \mathfrak{M}, w' \models \varphi \), with dual \( H \). One expects \( R \) to be a transitive, irreflexive relation. An important distinction arises between linear time and branching time frames: in the first case, there is a unique possible future, while in the second case there exist multiple different futures.

Exercise 8.1 (Basic Axiom). Show that \( \mathbf{K} : \square (\lozenge \varphi \lor \psi) \lor (\square \varphi \lor \square \psi) \) is valid, i.e. for any model \( \mathfrak{M} \) and any world \( w \) of \( W \), \( \mathfrak{M}, w \models \mathbf{K} \).

Exercise 8.2 (Transitive Frames). Show that, if \( R \) is transitive, then \( 4 : \lozenge \lozenge \varphi \lor \lozenge \varphi \) is valid.

Exercise 8.3 (Epistemic Frames). Prove the following implications for all modal formulæ \( \varphi \) when \( R \) is an equivalence relation:

\[
\begin{align*}
T & : \square \lozenge \varphi — \text{in epistemic logic, if indeed an agent really knows something, then it must be true—}, \\
4 & : \square \lozenge \varphi \lor \square \varphi — \text{in epistemic logic again, an agent has introspection about its own knowledge—}, \\
B & : \varphi \lor \square \lozenge \varphi — \text{in epistemic logic again, a truth is known by the agent as possibility compatible with her knowledge.}
\end{align*}
\]

Modal Languages As seen with our examples, the basic modal language can be extended to multiple modalities and underlying relations; in particular PDL defined in Section 4.3 is a modal language with an unbounded number of binary relations. A modal similarity type \( O \) is a ranked alphabet of modal operators \( \triangle \) of arity \( r(\triangle) \). A modal formula is then defined as

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

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

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

Exercise 8.4 (\( \text{ALC} \) as a Modal Language). Provide a consistency-preserving translation from \( \text{ALC} \) concepts into modal formulæ.

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

\[
\begin{align*}
\text{ST}_x(p) & \overset{\text{def}}{=} P_p(x) \\
\text{ST}_x(\top) & \overset{\text{def}}{=} (x = x) \\
\text{ST}_x(\neg \varphi) & \overset{\text{def}}{=} \neg \text{ST}_x(\varphi) \\
\text{ST}_x(\varphi \lor \varphi') & \overset{\text{def}}{=} \text{ST}_x(\varphi) \lor \text{ST}_x(\varphi') \\
\text{ST}_x(\Delta(\varphi_1, \ldots, \varphi_{r(\triangle)})) & \overset{\text{def}}{=} \exists x_1 \ldots x_{r(\triangle)}. R_\triangle(x, x_1, \ldots, x_{r(\triangle)}) \land \bigwedge_{i=1}^{r(\triangle)} \text{ST}_{x_i}(\varphi_i)
\end{align*}
\]
is a FO formula with a free variable $x$ equivalent to $\varphi$: $M, w \models \varphi$ iff $M |_{x \mapsto w} \models ST_x(\varphi)$. By reusing variables in the standard translation, we can use only $(n + 1)$ first-order variables if $\max_{\Delta \in O}(r(\Delta)) = n$.

**Bisimulations and Modal Invariance**

**Definition 8.2** (Bisimulations). Let $O$ be a modal similarity type and let $M = \langle W, (R_\Delta)_{\Delta \in O}, V \rangle$ and $M' = \langle W', (R'_\Delta)_{\Delta \in O}, V' \rangle$ be two $O$-models. A non-empty relation $Z \subseteq W \times W'$ is a **bisimulation** between $M$ and $M'$ if for all $w, w'$ s.t. $wZw'$,

1. $\{p \in A \mid w \in V(p)\} = \{p' \in A \mid w' \in V'(p')\}$,
2. if $(w, w_1, \ldots, w_{r(\Delta)}) \in R_\Delta$, then there are $w'_1, \ldots, w'_{r(\Delta)}$ in $W'$ s.t. $wZw'_i$ for all $1 \leq i \leq r(\Delta)$ and $(w', w'_1, \ldots, w'_{r(\Delta)}) \in R'_\Delta$, and
3. if $(w'_1, w'_2, \ldots, w'_{r(\Delta)}) \in R'_\Delta$, then there are $w_1, w_2, \ldots, w_{r(\Delta)}$ in $W$ s.t. $wZw'_i$ for all $1 \leq i \leq r(\Delta)$ and $(w, w_1, \ldots, w_{r(\Delta)}) \in R_\Delta$.

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

**Proposition 8.3** (Invariance for Bisimulation). Let $O$ be a modal similarity type, and $M$ and $M'$ be $O$-models. Then, for every $w$ in $W$ and $w'$ in $W'$ with $w \leftrightarrow w'$, and every modal formula $\varphi$, $M, w \models \varphi$ iff $M', w' \models \varphi$.

**Proof.** The proof proceeds by induction on $\varphi$. The case where $\varphi$ is an atomic proposition is a consequence of (1) in Definition 8.2, the case where $\varphi$ is $\top$ is trivial, and the cases of Boolean connectives follow from the induction hypothesis. For a formula of form $\Delta(\varphi_1, \ldots, \varphi_{r(\Delta)})$:

$M, w \models \Delta(\varphi_1, \ldots, \varphi_{r(\Delta)})$

implies $\exists w_1, \ldots, w_{r(\Delta)} \in W.(w, w_1, \ldots, w_{r(\Delta)}) \in R_\Delta \land \forall 1 \leq i \leq r(\Delta).M, w_i \models \varphi_i$

implies $\exists w'_1, \ldots, w'_{r(\Delta)} \in W'.(w', w'_1, \ldots, w'_{r(\Delta)}) \in R'_\Delta \land \forall 1 \leq i \leq r(\Delta).M', w'_i \models \varphi_i$

(by ind. hyp. and (2))

implies $M', w' \models \Delta(\varphi_1, \ldots, \varphi_{r(\Delta)})$,

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

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

$$\{(w_1, \ldots, w_{r(\Delta)}) \mid (w, w_1, \ldots, w_{r(\Delta)}) \in R_\Delta\}$$

is finite, the converse holds: let us define the **modal equivalence** relation $w \leftrightarrow w'$ as holding iff $w$ and $w'$ are indistinguishable, i.e.

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

**Theorem 8.4** (Hennessy-Milner Theorem). Let $O$ be a modal similarity type, and $M$ and $M'$ be $O$-models with finite image. If $w \leftrightarrow w'$, then $w \leftrightarrow w'$.
Proof. Let us prove that modal equivalence is a bisimulation relation. Condition (1) holds since a difference in labelling would be witnessed by propositional formulæ. For condition (2), assume \( w \sim w' \) and \((w, w_1, \ldots, w_{r(\Delta)}) \in R_{\Delta} \) and assume that there do not exist \( w'_1, \ldots, w'_{r(\Delta)} \) satisfying (2). The image set \( S' = \{(w'_1, \ldots, w'_{r(\Delta)}) \mid (w', w'_1, \ldots, w'_{r(\Delta)}) \in R'_{\Delta} \} \) is finite, and non-empty since otherwise \( \mathfrak{M}, w \models \Delta(T_1, \ldots, T) \) but \( \mathfrak{M}', w' \not\models \Delta(T_1, \ldots, T) \). Thus \( S' \) is a finite set \( \{(w'_1, \ldots, w'_{1,r(\Delta)}), \ldots, (w'_{n,1}, \ldots, w'_{n,r(\Delta)}) \} \) where, by assumption, for every \( 1 \leq j \leq n \), there exists \( 1 \leq i \leq r(\Delta) \) s.t. \( w_i \leftrightarrow w'_j,i \), i.e. there exists a formula \( \varphi_{j,i} \) s.t. \( \mathfrak{M}, w_i \models \varphi_{j,i} \) but \( \mathfrak{M}', w'_j,i \not\models \varphi_{j,i} \). But then

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

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

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

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

**Theorem 8.5 (van Benthem Characterisation Theorem).** Let \( \psi(x) \) be a first-order formula in \( \text{FO}(R_{\Delta})_{\Delta \in \text{O}}, (P_p)_{p \in A} \) with one free variable \( x \). Then \( \psi(x) \) is invariant for bisimulation iff it is equivalent to the standard translation of a modal formula.

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

### 8.3.2 First-Order Modal Logic

In order to work with both modal operators and first-order semantics as in Section 8.1, we introduce a mixed logic, **first-order modal logic** (FOML). For simplicity we give the definitions for the basic modal operator and not the fully general modal logic. The syntax of the logic over a vocabulary \( \langle (R_i)_{i} \rangle \) of \( k_i \)-ary symbols is

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

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

We consider structures \( \mathfrak{M} = \langle W, R, D, I \rangle \) where \( \langle W, R \rangle \) is a frame, \( D \) is a domain function from \( W \) to non-empty sets, and \( I \) is an interpretation function mapping each \( R_i \) with arity \( k_i \) > 0 and world \( w \) from \( W \) into a \( k_i \)-ary relation \( I(R_i)(w) \) over \( D(w) \) (constants are handled similarly). The domain of the model is \( \mathcal{D} = \bigcup_{w \in W} D(w) \). A valuation is a partial mapping from variables in \( \mathcal{X} \) to the domain.
The satisfaction of a formula by a model $M$ at a world $w$ for a valuation $\nu$ is defined inductively by
\[
M, w | = \nu x = y \quad \text{iff} \quad \nu(x) = \nu(y)
\]
\[
M, w | = \nu R_i(x_1, \ldots, x_k) \quad \text{iff} \quad (\nu(x_1), \ldots, \nu(x_n)) \in I(R_i)(w)
\]
\[
M, w | = \nu \neg \varphi \quad \text{iff} \quad M, w \not|= \nu \varphi
\]
\[
M, w | = \nu \varphi \land \varphi' \quad \text{iff} \quad M, w |= \nu \varphi \quad \text{and} \quad M, w |= \nu \varphi'
\]
\[
M, w | = \nu \exists x. \varphi \quad \text{iff} \quad \exists e \in D(w). M, w | = \nu [x \leftarrow e] \varphi.
\]

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

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

**Example 8.6** (First-order temporal logic). Let us consider some very simple examples in the temporal extension of first-order logic: we can model the meaning of the following sentence

John will eat an apple.

as
\[
\exists a. \text{apple}^{(1)}(a) \land F(\text{eat}^{(2)}_2(\text{John}^{(0)}_0, a)).
\]

(8.33)

Observe however that, in an actualist view, this reading implies the existence of the apple John will eventually eat in the current instant; the formula might not be satisfied by the model if no appropriate object $a$ on which $\text{apple}(a)$ holds can be found. Another reading would be
\[
F(\exists a. \text{apple}^{(1)}(a) \land \text{eat}^{(2)}_2(\text{John}^{(0)}_0, a)).
\]

(8.34)

### 8.4 Decidability

In modern terms, the Entscheidungsproblem or classical decision problem of Hilbert asks, given a first-order formula $\psi$, whether it is satisfiable. Church and Turing famously proved in the 1930s that the problem is undecidable, and a long line of research has established the decidability status of many fragments of first-order logic. Notably, the decidability status is known for all the prefix classes for formulæ in prenex normal form. For instance, the semantic reading
\[
\exists x. \text{woman}^{(1)}(x) \land \forall y. \text{man}^{(1)}(y) \supset \text{love}^{(2)}(y, x)
\]

(8.35)

for Every man loves a woman—to be contrasted with (8.27)—belongs to the $\exists^* \forall^*$ class shown decidable by Bernays and Schönfinkel and $\text{NExpTime}$-complete by Lewis (1980). It also belongs to the two-variable fragment $\text{FO}^2$, which was shown decidable by Mortimer and $\text{NExpTime}$-complete by Grädel, Kolaitis, and Vardi (1997). The standard translations of $\text{ALC}$ and of basic modal logic also yield $\text{FO}^2$ formulæ, and they are therefore decidable (they are actually $\text{PSPACE}$-complete).
8.4.1 The Guarded Fragment

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

Let \( \mathcal{X} \equiv \{x_1, \ldots, x_n\} \) be the set of variables. A \textit{guarded formula} over a vocabulary \((R_i^{(k_i)}), i\) is defined syntactically by

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

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

For example, the formula (8.27) is in GFO\(^2\): \textit{man}(1)(y) guards the universal quantification and \textit{love}(2)(y,x) guards the existential quantification. By contrast, (8.35) is not in GFO\(^2\): the universal quantification \( \forall y.\textit{man}(1)(y) \supset \textit{love}(2)(y,x) \) is not guarded. Observe more generally that the standard translations of \textit{ALC} or basic modal formulae are in GFO\(^2\).

Guarded Bisimulations

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

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

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

As in the modal case, we write \( \mathcal{M} \models_k \mathcal{M}' \) if there exists a guarded-\( k \)-bisimulation between \( \mathcal{M} \) and \( \mathcal{M}' \). We also write \( \mathcal{M} \equiv_k \mathcal{M}' \) if for all GFO\(^k\) sentences \( \psi \), \( \mathcal{M} \models \psi \) iff \( \mathcal{M}' \models \psi \). Proposition 8.3 can be extended to the case of guarded-\( k \)-bisimilarity:

**Proposition 8.7.** Let \( \mathcal{M} \) and \( \mathcal{M}' \) be two relational structures over the vocabulary \((R_i)\). If \( \mathcal{M} \models_k \mathcal{M}' \), then \( \mathcal{M} \equiv_k \mathcal{M}' \).

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

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

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

The interesting case is that of an existential quantification $\psi(x) = \exists y. \alpha(x, y) \land \varphi(x, y)$. Since $\mathcal{M} \models x \to a \psi(x)$, there exists $b$ in $\mathcal{M}$ such that $\mathcal{M} \models x \to a \exists y \to b \alpha(x, y) \land \varphi(x, y)$. Suppose toward a contradiction that there exists $f$ in $I$ that maps $a$ to $a'$. By condition (1), since $a \cup b$ is guarded by $\alpha(x, y)$, there exists $g$ in $I$ that maps $a$ to $a'$ and $b$ to $b'$. Then $\mathcal{M}' \models x \to a' \exists y \to b' \alpha(x, y)$ since $g$ is a partial isomorphism, which entails that $\mathcal{M}' \not\models x \to a' \exists y \to b' \varphi(x, y)$, which together with the existence of $g$ contradicts the induction hypothesis on $\varphi$. □

Models of Bounded Treewidth

An important model-theoretic property of $\mathcal{ALC}$ and the basic modal language is that they enjoy the tree model property: if a formula is satisfiable, then it has a tree model. In the case of GFO$^k$, we can generalise this idea to models of treewidth bounded by $k - 1$, see Proposition 8.8. In the case where $k = 2$ (which is the case of $\mathcal{ALC}$ and the basic modal logic), we find again the tree model property.

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

1. for every guarded set $X$ in $\mathcal{M}$ there exists a position $u$ in $\text{dom} \ t$ with $X \subseteq t(u)$, and
2. for every element $a$ in $\mathcal{M}$, the set of nodes $\{u \in \text{dom} \ t \mid b \in t(u)\}$ is connected in $t$ using the child relation $\downarrow$.

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

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

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

Observe that the tree $t'$ is a tree decomposition of $\mathcal{M}'$. This entails that $\mathcal{M}'$ has treewidth at most $k - 1$. Furthermore, $\{f_u \mid u \in \text{dom} \ t\}$ is a non-empty (note that the root $e$ gives rise to the empty isomorphism) set of partial isomorphisms.
between $\mathcal{M}$ and $\mathcal{M}'$, which satisfies the conditions of a guarded-$k$-bisimulation: $\mathcal{M} \leftrightarrow^k \mathcal{M}'$. Hence, by Proposition 8.7.

**Proposition 8.8.** If a sentence $\psi$ in $\text{GFO}^k$ has a model, then it has a model of treewidth at most $k - 1$.

Proposition 8.8 is instrumental in the proof of Grädel and Walukiewicz (1999) that the satisfiability problem for $\text{GFO}^k$ is in $\text{ExpTime}$. More precisely, the idea is to reduce the problem to a modal $\mu$-calculus satisfiability question over (infinite, countable) trees: given a $\text{GFO}^k$ formula $\psi$, one can construct a modal $\mu$-calculus formula $\varphi$ which describes a tree decomposition of a model of $\psi$ of treewidth at most $k - 1$. The complexity then follows by adapting the results of Vardi (1998) on the emptiness problem for 2ATAs over infinite trees.

**Limitations & Extensions**

Although the guarded fragment includes many formulae of interest in formal semantics, it is not comprehensive: (8.35) is an example of an unguarded formula. We can furthermore show that there is no equivalent formula in GFO. Observe that the two structures depicted in Figure 8.1 are guarded-2-bisimilar for the following set $I$ of partial isomorphisms

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

Because every guarded set in $\mathcal{M}$ is in the domain of one of the partial isomorphisms in $I$, every guarded set in $\mathcal{M}'$ is in the range of at least one of the partial isomorphisms in $I$, and all the partial isomorphisms in $I$ agree, this is indeed a guarded-2-bisimulation. Therefore by Proposition 8.7, $\mathcal{M}$ and $\mathcal{M}'$ are undistinguishable through guarded formulae over the vocabulary $\{\text{man}^{(1)}, \text{woman}^{(1)}, \text{love}^{(2)}\}$. However, $\mathcal{M} \not| \exists x. \text{woman}^{(1)}(x) \land (\forall y. \text{man}^{(1)}(y) \supset \text{love}^{(2)}(y,x))$ but $\mathcal{M}' \models \exists x. \text{woman}^{(1)}(x) \land (\forall y. \text{man}^{(1)}(y) \supset \text{love}^{(2)}(y,x))$. In particular, no $\text{ALC}$ formula can express (8.35).

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

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

or by

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

is not guarded—nor in $\text{FO}^2$. In fact, the two-variable guarded fragment without equality and only a handful of transitive relations is already undecidable.

There are extensions on the guarded fragment that retain most of its model- and complexity-theoretic properties, e.g. the guarded negation fragment of Bárány, ten Cate, and Segoufin (2011). Those extensions do not solve the issues pointed here.
This is an issue when considering epistemic or temporal modal logics, where transitivity is assumed; thankfully, decidability can be recovered when restricting transitive relations to occur solely in guards (e.g., Ganzinger et al., 1999; Michaliszyn, 2009).
Chapter 9

Tree Patterns

In this chapter, we consider formulæ called patterns from severely restricted fragments of first-order logic over trees. These provide concise means to define tree languages while avoiding the non-elementary complexity of full first-order logic over finite trees (e.g. Reinhardt, 2002). More precisely, we use patterns to define finite tree languages, which are then used as elementary trees in a grammar (Section 9.2) or as possible semantic readings in ambiguous sentences (Section 9.3).

9.1 Background: Existential First-Order Logic

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

$$\varphi_M \overset{\text{def}}{=} \exists x_1 \ldots x_n. \chi^+_M(x_1, \ldots, x_n) \tag{9.1}$$

where the formula $\chi^+_M$ is its positive diagram and consists of the conjunction of all the positive relational atomic formulæ true of $M$:

$$\chi^+_M(x_1, \ldots, x_n) \overset{\text{def}}{=} \bigwedge_i \bigwedge (w_{i_1}, \ldots, w_{i_k}) \in R_i R_i^L(x_{i_1}, \ldots, x_{i_k}) \tag{9.2}$$

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

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

$$\alpha ::= x = y \mid R^{(k)}(x_1, \ldots, x_k) \quad \text{(atomic formulæ)}$$

$$\varphi ::= \alpha \mid \neg \alpha \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \exists x. \varphi \quad \text{(existential formulæ)}$$

where $x, y, x_1, \ldots, x_k$ range over $X$, the set of variables, and $R$ over the vocabulary $\sigma$.

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

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

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

\[
\varphi \equiv \bigvee_i \psi_i 
\]  
(9.3)

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

\[
\psi = \exists x. \bigwedge_j \beta_j(x_j) 
\]  
(9.4)

where the \( \beta_j \)'s are atoms or negated atoms and \( x_j \) is a subvector of \( x \). (If additionally \( \varphi \) was positive, then each \( \psi_i \) is primitive positive and the \( \beta_j \)'s are atoms.) Observe finally that any atom of the form \( x = y \) in some \( \psi \) can be eliminated by identifying the two variables \( x \) and \( y \) in \( \psi \):

\[
\exists x_1 x_2 \ldots x_n. \chi \land x_1 = x_2 \equiv \exists x_2 \ldots x_n. \chi\{x_1 \leftarrow x_2\} 
\]  
(9.5)

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

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

\[
\text{Mod}_{\leq n}(\psi) \equiv \{ \mathcal{M} = (W, \sigma) \mid |W| \leq n \land \mathcal{M} \models \psi \} . 
\]  
(9.6)

If \( \psi \) is positive, and positive equality atoms of the form \( x = y \) have been eliminated as explained just before (thus only positive relational atoms appear in \( \psi \)), then \( \psi \) has a canonical model \( \mathcal{M}_\psi \) with domain \( \{w_1, \ldots, w_n\} \) and a tuple \( (w_{i_1}, \ldots, w_{i_k}) \) in a \( k \)-ary relation \( R \) iff \( R^{(k)}(x_{i_1}, \ldots, x_{i_k}) \) is an atom in \( \psi \). Clearly, \( \mathcal{M}_\psi \models \psi \), and furthermore the canonical sentence associated with \( \mathcal{M}_\psi \) is \( \psi \) itself.

\[\text{(*) Exercise 9.1 (Canonical Model). Given an existential conjunctive sentence } \psi \text{ without positive equality atoms (but possibly with some negated atom of the form } \neg R^{(k)}(x_{i_1}, \ldots, x_{i_k}) \text{ or } x_i \neq x_j \text{), we distinguish its positive part } \psi^+, \text{ which contains only the positive relational atoms of } \psi. \text{ Show that, if } \psi \text{ is satisfiable, then } \mathcal{M}_\psi^+ \models \psi. \]

### 9.1.1 Characterisations over Finite Models

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

The Łoś-Tarski theorem states that a first-order sentence is preserved under extensions over the class of all (finite and infinite) structures if and only if it is equivalent to an existential sentence. If we work on a particular class of structures, the theorem might fail, but one direction remains correct:
Proposition 9.1. Let \( C \) be a class of structures. If \( \varphi \) is equivalent to an existential sentence over \( C \), then it is preserved under extensions over \( C \).

Proof. Let \( \mathfrak{M} = \langle W, (R_i)_{i \in I} \rangle \) and \( \mathfrak{M}' = \langle W', (R'_i)_{i \in I} \rangle \) be two structures in \( C \) with \( \mathfrak{M} \models \varphi \) and \( \mathfrak{M} \subseteq_i \mathfrak{M}' \). Write \( \varphi \) as a finite disjunction of \( \text{ECFO} \) sentences as in (9.3): there exists a disjunct \( \psi \) such that \( \mathfrak{M} \models \psi \). More precisely, \( \psi \) can be put in prenex normal form as \( \psi \equiv \exists x_1 \ldots x_n. \chi \) where \( \chi \) is a conjunction of atoms and negated atoms, and \( \mathfrak{M} \models \nu \chi \) for some valuation \( \nu : \{x_1, \ldots, x_n\} \rightarrow W \). Consider the substructure \( \mathfrak{M}_\nu \) (not necessarily in \( C \)) induced by the subset \( \nu(\{x_1, \ldots, x_n\}) \subseteq W \) in \( \mathfrak{M} \): then \( \mathfrak{M}_\nu \models \nu \chi \) and \( \mathfrak{M}_\nu \subseteq_i \mathfrak{M} \subseteq_i \mathfrak{M}' \). We can easily check that \( \mathfrak{M}' \models \nu \chi \) and the result follows. \( \square \)

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

Lemma 9.2. Let \( C \) be a class of finite structures closed under induced substructures. If \( \varphi \) is equivalent to an existential sentence over \( C \), then \( \varphi \) has finitely many minimal models in \( C \).

Proof. Using again the disjunctive normal form equivalent to \( \varphi \), it suffices to show that there are finitely many minimal models for a disjunct \( \psi \) in \( \text{ECFO} \). Let \( \psi \equiv \exists x_1 \ldots x_n. \chi \), \( \mathfrak{M} \) be a minimal model of \( \psi \) and \( \nu \) be a valuation such that \( \mathfrak{M} \models \nu \chi \). Then \( \nu \) induces as in the proof of Proposition 9.1 a substructure \( \mathfrak{M}_\nu \subseteq_i \mathfrak{M} \) with \( \mathfrak{M}_\nu \models \nu \chi \). Because \( C \) is closed under induced substructures, \( \mathfrak{M}_\nu \) also belongs to \( C \), and because \( \mathfrak{M} \) was assumed minimal, this in turn entails that \( \mathfrak{M}_\nu \) and \( \mathfrak{M} \) are isomorphic, and thus that \( \mathfrak{M} \) has at most \( n \) elements.

In other words, if \( \mathfrak{M} \) is a minimal model in \( C \), then

\[
\mathfrak{M} \in \text{Mod}_{\leq n}(\psi) .
\]  

(9.7)

(Note that this is not directly implied by Exercise 9.1 because \( \mathfrak{M}_\psi \) might not be in \( C \).) We conclude by noting that \( \text{Mod}_{\leq n}(\psi) \) is finite for every \( n \), and that \( n \) itself is bounded by the quantifier depth of \( \varphi \). \( \square \)

Exercise 9.2 (Diagrams). Let \( \mathfrak{M} = \langle W, (R_i)_{i \in I} \rangle \) be a finite structure with \( W = \{w_1, \ldots, w_n\} \). We define its diagram as the conjunction of the atomic and negated atomic formulae it satisfies under the valuation \( \nu : x_j \mapsto w_j \):

\[
\chi_{\mathfrak{M}} \overset{\text{def}}{=} \bigwedge_{1 \leq j \leq k \leq n} x_j \neq x_k \wedge \bigwedge_i \left( \bigwedge_{(w_{i_1}, \ldots, w_{i_k}) \in R_i} R_{i}^{(i_k)}(x_{i_1}, \ldots, x_{i_k}) \wedge \bigwedge_{(w_{i_1}, \ldots, w_{i_k}) \notin R_i} \neg R_{i}^{(i_k)}(x_{i_1}, \ldots, x_{i_k}) \right) .
\]  

(9.8)

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

Exercise 9.3 (Converse of Lemma 9.2). Let \( C \) be a class of finite structures and let \( \varphi \) be a first-order sentence preserved under extensions on \( C \). Show that, if \( \varphi \) has finitely many minimal models in \( C \), then it is equivalent to an existential sentence over \( C \).

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

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

An issue with the class of trees is that it is not closed under induced substructures. For instance, the proof of Lemma 9.2 is incorrect for trees, e.g. the sentence

$$\exists xyz. P_a(x) \land P_b(y) \land P_c(z) \land x \downarrow^* y \land x \not\downarrow z \land x \not\downarrow z \quad (9.9)$$

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

```
\[ \begin{array}{c}
a \\
$ \\
\downarrow \\
/ \\
c \end{array} \]
```

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

**Theorem 9.3** (Koller et al., 2001). Satisfiability of ECFO $\langle (\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a \in A} \rangle$ sentences is $\text{NP}$-complete.

9.2 Meta-Grammars

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

9.2.1 Diathesis Alternation

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

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

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

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

where each of the basic formulæ ActiveVerb, PassiveVerb, etc. is the canonical positive primitive formula of the corresponding tree in Figure 9.1. For instance, the conjunction

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

(9.10)
gives rise to the unique minimal model of Figure 9.2.

### 9.2.2 Complexity

Observe that we only used the ↓, →, and →⁺ axes in our examples in Section 9.2. One might hope that this fragment of E⁺FO would have a polynomial-time sat-
is satisfiability problem, but it turns out to be NP-hard already for primitive positive sentences with only \( \rightarrow \) and \( \rightarrow^+ \):

**Proposition 9.4.** Satisfiability of \( E^+ \cdot \text{CFO}(\rightarrow, \rightarrow^+, (P_0)_a) \) sentences is NP-complete.

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

We reduce for this from the Shortest Common Supersequence Problem (SSSP), c.f. (Räihä and Ukkonen, 1981). An instance of SSSP is an integer \( k \) and a set of strings \( S = \{s_i = a_{i_1} \cdots a_{i_{r_i}}\}_{1 \leq i \leq p} \) over some finite alphabet \( \Sigma \). The instance is positive if there exists a string \( s \) of length at most \( k \), which is simultaneously a supersequence of every string in \( S \), i.e. for every \( i \), there exist strings \( s_i^0, \ldots, s_i^{\ell_i+1} \) such that \( s = s_i^0a_{i_1} \cdots a_{i_{r_i}}s_i^{\ell_i+1} \). Importantly, if \( s \) is such a witness, then any supersequence of \( s \) over some alphabet that includes \( \Sigma \) and of length exactly \( k \) is also a witness.

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

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

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

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

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

\[
\varphi_S(z, z') \overset{\text{def}}{=} \bigwedge_{1 \leq i \leq p} \exists y_1 \ldots y_{r_i}. z \rightarrow^+ y_1 \land y_{r_i} \rightarrow^+ z' \land \left( \bigwedge_{1 \leq j \leq r_i} P_{a_{ij}}(y_j) \right)
\]

\[
\land \left( \bigwedge_{1 \leq r < r_i} y_r \rightarrow^+ y_{j+1} \right).
\]

\( \square \)

9.3 Underspecified Semantics

9.3.1 Scope Ambiguities

An pervasive issue in semantic representations is related to scope ambiguities. Linguistic expressions are often semantically ambiguous (i.e. they have several possible readings that are mapped to different meaning representations) but fail to reflect this ambiguity syntactically (e.g. they have a single syntactic analysis).

For instance, the sentence *Every man loves a woman* accepts two readings

\[
\exists y. \text{woman}(y) \land \forall x. \text{man}(x) \supset \exists e. \text{love}(e) \land \text{agent}(e, x) \land \text{patient}(e, y) \quad (9.11)
\]

\[
\forall x. \text{man}(x) \supset \exists y. \text{woman}(y) \land \exists e. \text{love}(e) \land \text{agent}(e, x) \land \text{patient}(e, y) \quad (9.12)
\]

depending on whether we are talking about one single woman or not; there is no clear reason why we should provide the sentence with different syntactic analyses.

Assuming we view meaning construction as a relation from one syntactic representation to several semantic ones, the number of readings can grow exponentially.
with the number of scope-bearing operators (quantifiers, modal operators, etc.),
and simply enumerating the possible readings quickly turns impossible.

For instance, the sentence

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

(Poesio, 1994)

is reputed as having several thousand readings. Arguably, not all these readings
are born equal: some might be implied by others (just like (9.11) implies (9.12)),
and some downright impossible. However there can still remain a considerable
number of incomparable readings. A naive approach to counting the number of
possible readings is to consider all the permutations of quantifiers in a sentence:
for a sentence with \( n \) quantifiers this will yield \( n! \) different readings. Hobbs and
Shieber (1987) for instance refine this approach and show how the sentence

Every representative of a company saw most samples.

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

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

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

### 9.3.2 Hole Semantics

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

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

(hole formulæ)

where \( l \) ranges over \( L \), \( a^{(r)} \) over \( A_r \), \( x, x_1, \ldots, x_r \) over \( L \not\in H \), and \( h \) over \( H \). As
with ECFO formulæ, hole formulæ \( \gamma \) can be put in prenex normal form

\[
\gamma \equiv \exists l_1 \ldots l_n \exists h_1 \ldots h_m \land \gamma_p .
\]

(9.13)
Hole formulæ $\gamma$ are interpreted in ECFO$((\downarrow_i)_{i<k}, \downarrow^*, (P_a)_{a \in A})$ by associating a formula $[\gamma]$

$$[\gamma] = \exists l_1 \ldots l_n h_1 \ldots h_m. \bigwedge_{1 \leq i < j \leq n} l_i \neq l_j \land \bigwedge_p \gamma_p$$  \hspace{1cm} (9.14)

where we interpret

$$l : a^{(r)}(x_1, \ldots, x_r) \overset{\text{def}}{=} P_a(l) \land \bigwedge_{i=1}^r l \downarrow_{i-1} x_i.$$ \hspace{1cm} (9.15)

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

1. in every $h \downarrow^* l$ subformula, $l$ is a root of $\gamma$,

2. every hole appears exactly once as a child of a $l : a^{(r)}(x_1, \ldots, x_r)$ subformula, and thus cannot be a root,

3. every label should appear at most once as a parent and at most once as a child in a $l : a^{(r)}(x_1, \ldots, x_r)$ subformula. This excludes for instance $l' : f^{(2)}(l, l), l : f^{(2)}(l_1, l_2) \land l : f^{(2)}(l'_1, l'_2)$, or $l_1 : g^{(1)}(l) \land l_2 : g^{(1)}(l)$.

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

**Theorem 9.5** \cite{Althaus et al., 2003}. Satisfiability of normal hole formulæ is in $P$.

### Constructive Satisfiability

The issue with our interpretation of hole formulæ into ECFO is that not every model $\mathbb{M}$ over $A$ is suitable as a SRL formula. For instance, there could be extra points in the model not constrained by $\gamma$, or conversely several labels could be mapped to a single node. An alternative notion of model is needed in practice. Consider a hole formula in prenex conjunctive normal form as in (9.13). Then a plugging $P$ is an injective function from holes $\{h_1, \ldots, h_m\}$ to labels $\{l_1, \ldots, l_n\}$. A model $\mathbb{M} = (\text{dom}(t), (l_i)_{i<k}, h^*, (P_a)_{a \in A})$ of $\gamma$ is a plugged model for a plugging $P$ if its domain is in bijection with the set of labels (we write $\text{dom}(t) = \{l_1, \ldots, l_n\}$) and $\mathbb{M} \models \nu \gamma$ where the valuation $\nu$ is defined by

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

(9.16)

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

**Example 9.6.** Let us extend the syntax of hole formulæ by allowing larger tree segments:

$$\begin{align*}
\gamma &::= l : a^{(r)}(\theta_1, \ldots, \theta_r) \mid h \downarrow^* l \mid \gamma \land \gamma \mid \exists x. \gamma \\
\theta &::= a^{(r)}(\theta_1, \ldots, \theta_r) \mid h
\end{align*}$$ (hole formulæ)

(tree formulæ)
∀x ⊃ man x □ ∃y ∧ woman y □ love x y

Figure 9.3: Underspecified formula for (9.11) and (9.12). Dominance relations are indicated through dotted arrows and holes by boxes.

and translating back into hole formulæ by defining

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

\[ l : a^{(r)}(\theta_1, \ldots, \theta_r) \overset{\text{def}}{=} l : a^{(r)}(x_{\theta_1}, \ldots, x_{\theta_r}) \]

\[ a^{(r)}(\theta_1, \ldots, \theta_r) \overset{\text{def}}{=} \exists l_{\theta_1} l_{\theta} : a^{(r)}(x_{\theta_1}, \ldots, x_{\theta_r}) \]

A hole semantic formula that models the two readings (9.11) and (9.12) is the following (see also Figure 9.3):

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

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

**Theorem 9.7.** Constructive satisfiability of normal hole formulæ is NP-complete.

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

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

\[ \mathfrak{M} = \{ \{ \hat{l}_1, \ldots, \hat{l}_n \}, \downarrow_{i<k}, (P_a)_{a \in A} \} ; \quad (9.17) \]

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

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

\[ \mathfrak{M}' = \{ \{ \hat{l}_1, \ldots, \hat{l}_n \}, \downarrow_{i<k}, \downarrow^*, (P_a)_{a \in A} \} \]

(9.18) still of polynomial size, and

3. verifying that \( \mathfrak{M}' \) is a model of the existentially conjunctive formula \([ \gamma ]\) for the assignment \( \nu \) defined in (9.16) (this is in P).

For the NP lower bound, we exhibit a reduction from the 3-Partition Problem. An instance of this problem is given by a finite multiset \( A = \{ a_1, \ldots, a_{3m} \} \) of integers and a bound \( B \), all in \( \mathbb{N} \) and encoded in unary, such that \( \frac{B}{3} < a_i < \frac{B}{2} \) for all \( i \) and \( \sum_{i=1}^{3m} a_i = mB \). The instance is positive if there exists a partition \( A_1 \uplus A_2 \uplus \cdots \uplus A_m \) Polynomial-time processing can be recovered if we further restrict hole formulæ; see Koller et al. (2003).
of $A$ s.t. for all $j$, $|A_j| = 3$ and $\sum_{a \in A_j} a = B$. We can assume $B > 0$ (or $a_i = 0$ for all $i$).

We construct from an instance $(A, B)$ a hole formula over the ranked alphabet $\{l_1^{(a_i+1)}, g^{(m)}, h^{(0)} \mid 1 \leq i \leq 3m\}$:

$$\exists l_{f_1} \ldots l_{f_{3m}} l_g^{1,1} \ldots l_b^{B+1,1} h_{g}^{1,1} h_{g}^{0} h_{g}^{m,B+1} l_{f_1}^{a_1+1} h_{f_1}^{1} \ldots h_{f_{3m}}^{a_{3m}+1}.$$  

$$\forall l_{f_1} \ldots l_{f_{3m}} l_g^{1,1} \ldots l_b^{B+1,1} h_{g}^{1,1} h_{g}^{0} h_{g}^{m,B+1} l_{f_1}^{a_1+1} h_{f_1}^{1} \ldots h_{f_{3m}}^{a_{3m}+1}.$$  

$$\forall l_{f_1} \ldots l_{f_{3m}} l_g^{1,1} \ldots l_b^{B+1,1} h_{g}^{1,1} h_{g}^{0} h_{g}^{m,B+1} l_{f_1}^{a_1+1} h_{f_1}^{1} \ldots h_{f_{3m}}^{a_{3m}+1}.$$

$$\forall l_{f_1} \ldots l_{f_{3m}} l_g^{1,1} \ldots l_b^{B+1,1} h_{g}^{1,1} h_{g}^{0} h_{g}^{m,B+1} l_{f_1}^{a_1+1} h_{f_1}^{1} \ldots h_{f_{3m}}^{a_{3m}+1}.$$

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

Conversely, assume there is a plugging $P$ from holes to labels and let $\mathfrak{N}$ be the corresponding plugged model using valuation $v$. For every $1 \leq j \leq m$, consider the set $A_j$ of integers $a_i$ such that $f_i$-rooted fragments are plugged below $h_{g}^{j}$, i.e. $A_j \equiv \{a_i \mid \exists \mathfrak{N} \models h_{g}^{j} \downarrow l_{f_i}\}$. Note that $A_1 \cup \ldots \cup A_m$ forms a partition of $A$. Because a plugging is injective from holes to labels, each $f_i$-rooted fragment requires $a_i + 1$ labels, $h_{g}^{j}$ requires one, and $|A_j| + B + 1$ are available using the $f_i$- and $b$-rooted fragments, we get that $1 + |A_j| + \sum_{a \in A_j} a_i \leq |A_j| + B + 1$, hence $\sum_{a \in A_j} a_i \leq B$ for every $1 \leq j \leq m$. Because $\sum_{a \in A_j} a = mB$, there is no choice and $\sum_{a \in A_j} a = B$.

Furthermore, $|A_j| \geq 3$:

- $|A_j| \neq 0$ since $B > 0$, and $B + 1$ fragments rooted by $b$ must be plugged somewhere below the single hole $h_{g}^{j}$;

- $|A_j| \neq 1$ since a single $f_i$-rooted fragment provides $a_i + 1 < \frac{B}{2} < B + 1$ holes,

- $|A_j| \neq 2$ since a pair $\{a_x, a_y\}$ provides $a_x + a_y + 2 - 1 < B + 1$ holes.

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

\[ \text{(***)} \]

Exercise 9.4 (Tree Automata for Hole Formulae). The set of constructive models of a constraint is clearly a regular tree language. Provide a construction for a regular tree automaton $\mathcal{A}_{e}$ that recognizes exactly the constructive models of a normal hole formula $\gamma$.

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

The size of the automaton constructed in Exercise 9.4 is exponential in the size of the formula. This is unavoidable, as there exist normal formulae $\gamma_n$ of size $O(n)$ s.t. any automaton recognizing the set of plugged models of $\gamma_n$ requires at least $2^n$ states: let

$$A_n \equiv \{a^{(0)}, g_1^{(1)}, \ldots, g_n^{(1)}\}$$

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

$$L_n = \{g_{\pi(1)}(\Box) \cdot g_{\pi(2)}(\Box) \cdots g_{\pi(n)}(a) \mid \pi \text{ a permutation of } \{1, \ldots, n\}\}.$$  \hfill (9.21)

**Lemma 9.8.** Any finite tree automaton for $L_n$ requires at least $2^n$ states.

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

$$C_K \overset{\text{def}}{=} g_{i_1}(\Box) \cdots g_{i_{|K|}}(\Box)$$ \hfill (9.22)

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

$$t_K \overset{\text{def}}{=} C_{\bar{K}} \cdot C_K \cdot a$$ \hfill (9.23)

is in $L_n$.

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

$$C_{\bar{K}} \cdot C_K \cdot a \Rightarrow^* C_{\bar{K}} \cdot q \Rightarrow^* q_f$$ \hfill (9.24)

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

$$C_{\bar{K'}} \cdot C_K \cdot a \Rightarrow^* C_{\bar{K'}} \cdot q \Rightarrow^* q_f$$ \hfill (9.25)

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

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

Lemma 9.8 shows that there might be exponential succinctness gains from the use of hole formulae rather than tree automata for the description of semantic representations. One might object that the classes of tree languages obtained at the output of the linear higher-order tree functions of Section 10.1.4 are context-free tree languages and not necessarily regular ones, with potential exponential gains in succinctness. However, note that $L_n$ is basically a string language, and the exponential lower bounds on the size of any context-free string grammar for permutation languages (see e.g. [Filmus, 2011]) also apply to CFTGs for $L_n$. 
Chapter 10

Higher-Order Semantics

In this last chapter, we consider the use of higher-order functions in natural language semantics. We first motivate the need for such functions in Section 10.1 in order to define the interface between syntax and semantics. We then observe that, more generally, ‘increasing the order’ allows for elegant solutions to some difficulties like intensionality phenomena and many-world semantics.

10.1 Compositional Semantics

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

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

(Parthe et al., 1990, Chapter 13)

Let us illustrate this principle on Example 8.1: by associating a semantic representation to each meaningful word in the sentence, i.e. if we define \([John]\), \([eats]\) and so on, then the semantics of each intermediate structure like a red apple or John eats a red apple can be systematically computed as a function of its parts, based on the syntactic structures. Note that these structures play a crucial role, as otherwise John loves Mary and Mary loves John would not be distinguishable as naive ‘functions of their parts.’

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

Leaving these questions aside and adopting a modelling viewpoint, compositionality is a rather strenuous requirement: for instance, assuming \([John] = John^{(0)}\) and \([a \text{ red apple}] = \exists x. apple^{(1)}(x) \land red^{(1)}(x)\), it is not so clear how one should combine everything and obtain (8.1) or more involved representations like (8.24). Moreover any solution will be dependent on the specific syntactic analysis.

10.1.1 Background: Simply Typed Lambda Calculus

One of the best-studied ways to implement compositional semantics for natural
languages is to use lambda expressions as semantic values associated with each component (Montague, 1970, 1973). As Church’s simple theory of types provides an elegant setting for model-theoretic higher-order semantics (see Section 10.3), we favour a presentation that uses the simply typed $\lambda$-calculus over the untyped one.

**Lambda Terms** Given an infinite countable set $X$ of variables, and $C$ a countable set of constants, the set $\Lambda(C)$ of $\lambda$-terms is defined by
\[
L ::= c \mid x \mid LL \mid \lambda x.L
\]
where $c$ is a constant in $C$ and $x$ a variable in $X$.

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

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

We assume the usual definitions for $\alpha$, $\beta$, and $\eta$ reductions:
\[
\begin{align*}
\lambda x.L & \rightarrow_\alpha \lambda y.(L\{x \leftarrow y\}) \\
(\lambda x.L)M & \rightarrow_\beta L\{x \leftarrow M\} \\
\lambda x.(Lx) & \rightarrow_\eta L
\end{align*}
\]
(where substitutions have to avoid name clashes and $x \notin \text{FV}(L)$ for $\eta$-reductions), and recall that $\beta\eta$-reductions are Church-Rosser: if $L \Rightarrow^{*_{\beta\eta}} M$ and $L \Rightarrow^{*_{\beta\eta}} N$, then there exists $L'$ s.t. $M \Rightarrow^{*_{\beta\eta}} L'$ and $N \Rightarrow^{*_{\beta\eta}} L'$, which implies that $\beta\eta$ reductions define unique normal forms, noted $\Downarrow^{*_{\beta\eta}}$.

**Types** Assume we are provided with some non-empty countable set of atomic types $A$; then types in $T_A$ are terms defined inductively by
\[
\tau ::= a \mid \tau \rightarrow \tau
\]
where $a$ ranges over $A$. By convention we consider $\rightarrow$ to be right-associative, i.e. we write $\rho \rightarrow \sigma \rightarrow \tau$ for $\rho \rightarrow (\sigma \rightarrow \tau)$. The order of a type $\tau$ is defined inductively as
\[
\begin{align*}
\text{ord}(a) &= 1 \\
\text{ord}(\sigma \rightarrow \tau) &= \max(\text{ord}(\sigma) + 1, \text{ord}(\tau))
\end{align*}
\]
A higher-order signature is a triple $\Sigma = \langle A, C, \tau \rangle$ where $A$ is a set of atomic types, $C$ a countable set of constants and $\tau : C \rightarrow T_A$ a typing of the constants. Given a higher-order signature, each $\lambda I$-term of $\Lambda(C)$ can be assigned a type in $T_A$ by the deduction rules
\[
\frac{\Gamma \vdash x : \sigma}{\Gamma \vdash \lambda x. L : \sigma \rightarrow \tau} \quad (\rightarrow I) \\
\frac{\Gamma, x : \sigma \vdash L : \tau}{\Gamma \vdash \lambda x. L : \sigma \rightarrow \tau} \quad (\rightarrow E)
\]
\[
\frac{\Gamma \vdash L : \sigma \rightarrow \tau \quad \Delta \vdash M : \sigma}{\Gamma, \Delta \vdash LM : \tau} \quad (\rightarrow E) 
\]
\[
\frac{\Gamma \vdash c : \tau(c)}{(\text{Cons})} \\
\frac{x : \tau \vdash x : \tau}{(\text{Var})} \\
\frac{\Gamma, x : \sigma \vdash L : \tau}{\Gamma \vdash \lambda x. L : \sigma \rightarrow \tau} \quad (\rightarrow I)
\]
where the type contexts $\Gamma, \Delta$ are type assignments from free variables to $T_A$; in $\nrightarrow^E$ the two assignments have to be compatible, i.e. assign the same types to common variables. For linear lambda terms, this compatibility requirement is useless as $\text{FV}(L) \cap \text{FV}(M) = \emptyset$. We can extend the typing system to any $\lambda$-term instead of $\lambda I$-terms if we additionally allow $\langle \cdot \rangle$ to work on the premise $\Gamma \vdash_T L : \tau$ where $x$ is not among $\text{FV}(L)$ nor in the domain of $\Gamma$.

Example 10.1 (B combinator). Define $B \overset{\text{def}}{=} \lambda xyz.x(yz)$. It can be typed by:

$$
\begin{align*}
x : a \to b \vdash_T x \colon a \to b \\
y : c \to a \vdash_T y \colon c \to a \\
z : c \vdash_T z \colon c \\
\frac{x : a \to b, y : c \to a, z : c \vdash_T x(yz) : b}{\vdash_T \lambda z.x(yz) : c \to b}
\end{align*}
$$

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

Proposition 10.2 (Subject Reduction). If $\Gamma \vdash_T L : \tau$ and $L \Rightarrow^*_{\beta\eta} M$ then $\Gamma \vdash_T M : \tau$.

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

Proposition 10.3 (Subject Expansion). If $\tau$ is a linear $\lambda$-term, $\Gamma \vdash T L : \tau$, and $M \Rightarrow^*_\beta L$, then $\Gamma \vdash_T M : \tau$.

Exercise 10.1. Prove Proposition 10.2 and Proposition 10.3.

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

Theorem 10.4 (Strong Normalisation). If $L$ is a typable $\lambda$-term, then every $\beta\eta$-reduction starting at $L$ is finite.

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

Theorem 10.5 (Statman [1979b]). Simple type inhabitation is PSPACE-complete.

10.1.2 Ground Terms over Second-Order Signatures

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

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

for atomic $a_i$'s in $A$. 
Exercise 10.2 (Normalised Typing System). Consider the normalised typing system with a single rule

\[
\frac{\tau(c) = a_1 \to \cdots \to a_n \to a_0 \quad \vdash_\Sigma t_1 : a_1 \ldots \vdash_\Sigma t_n : a_n}{\vdash_\Sigma ct_1 \cdots t_n : a_0} \quad \text{(App)}
\]

We want to show that, for all ground terms \( t \) and atomic types \( a \), \( \vdash_\Sigma t : a \) if and only if \( \vdash_\Sigma t : a \).

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

2. Show that, if \( \vdash_\Sigma t : a \) for a ground term \( t \) and type \( \alpha \), then \( t = ct_1 \cdots t_i \) for some constant \( c \) with \( \tau(c) = a_1 \to \cdots \to a_n \to a_0, 0 \leq i \leq n, \) and some ground terms \( t_1, \ldots, t_i \) such that \( \alpha = a_{i+1} \to \cdots \to a_n \to a_0 \) and \( \vdash_\Sigma t_j : a_j \) for \( 0 < j \leq i \) for some atomic types \( a_j \)'s.

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

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

Given a second-order signature \( \Sigma \) and a distinguished atomic type \( s \), we define the ground tree language

\[
\mathcal{G}(\Sigma, s) \overset{\text{def}}{=} \{ t \in T(C) \mid \vdash_\Sigma t : s \text{ where } t \text{ is ground} \}.
\]

Example 10.6. Consider the second-order signature \( \Sigma_0 \) with atomic types \( A_0 = \{np, s, c\} \), constants \( C_0 = \{Alice, believe, left, someone, that\} \), and typing

\[
\begin{align*}
\tau_0(Alice) &= np \\
\tau_0(believe) &= c \to np \to s \\
\tau_0(left) &= np \to s \\
\tau_0(someone) &= np \\
\tau_0(that) &= s \to c
\end{align*}
\]

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

Then the set of trees in \( \mathcal{G}(\Sigma_0, s) \) is recognised by a tree automaton \( A = \langle Q, \mathcal{F}_0, \delta, I \rangle \) with \( Q = A_0, I = \{s\} \), and rules

\[
\delta = \{(np,Alice^{(0)}), (s,believe^{(2)}, c, np), (s,left^{(1)}, np), (np, someone^{(0)}), (c, that^{(1)}, s)\}.
\]

Exercise 10.3 (Local Tree Automata). Let \( \mathcal{F} \) be a ranked alphabet. A deterministic top-down tree automaton \( A = \langle Q, \mathcal{F}, \delta, \{q_0\} \rangle \) is local if there exists a function \( \ell : \mathcal{F} \to Q \) such that the rules in \( \delta \) are all of the form \( \ell(f^{(n)}), f^{(n)}, q_1, \ldots, q_n \).

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

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

### 10.1.3 Higher-Order Homomorphisms

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

**Definition 10.7 (Higher-Order Homomorphism).** A higher-order homomorphism from a set of constants $C$ to a set of constants $C'$ is generated by a function $[.]$ mapping constants in $C$ to closed $\lambda$-terms in $\Lambda(C')$. We lift $[.]$ to a homomorphism from $\Lambda(C)$ to $\Lambda(C')$ by $[x] = x$, $[LM] = [L][M]$, and $[\lambda x. L] = \lambda x.[L]$.

**Example 10.8.** Continuing with Example 8.1, Figure 10.1 presents two syntactic analyses (the dependency one could for instance be obtained from the constituent one through head percolation analysis or as the derivation tree of a TAG). For the constituent analysis of Figure 10.1, we have

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

and

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

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

(ignoring tree nodes with a single child, for which we set e.g. \( \llbracket \text{NN}^{(1)} \rrbracket = \lambda x_1. x_1 \)).

The first-order variables \( u \) and \( e \) could be considered as constants of arity 0 in \( C' \), but this causes some naming issues; an alternative would be treat \( \exists x. \varphi \) as \( \exists \lambda x. \varphi \).

This definition results successively in

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

which is the \( \lambda \)-term encoding of (8.24).

\((*)\) Exercise 10.4. Propose similarly a higher-order homomorphism from the dependency structure of Figure 10.1 into its semantics.

### 10.1.4 Tree Transductions

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

**Definition 10.9** (Typed Homomorphism). A *typed homomorphism* between two signatures \( \Sigma = \langle A, C, \tau \rangle \) and \( \Sigma' = \langle A', C', \tau' \rangle \) extends a higher-order homomorphism \( \llbracket \rrbracket \) between \( C \) and \( C' \) by mapping each atomic type of \( A \) into a type of \( T_A \) s.t. \( \vdash_{\Sigma'} [c] : [\tau(c)] \) is a valid typing judgement for all \( c \) in \( C \).

**Example 10.10** (Higher-Order Tree Functions). Let us see how this definition can be exercised to define tree transductions. We define the generic tree signature over a ranked alphabet \( F \) as \( \Sigma_F \overset{\text{def}}{=} \langle \{o\}, F, \tau_F \rangle \) where for every \( f^{(n)} \) in \( F \),

\[
\tau_F(f^{(n)}) \overset{\text{def}}{=} o \rightarrow \cdots \rightarrow o \rightarrow o = o^n \rightarrow o.
\]

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

\[
T(t_1) = t_2 \text{ iff } \vdash_\Sigma t_1 : s \land [t_1] \Rightarrow^*_\beta\eta t_2.
\] (10.1)

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

The semantic construction of Example 10.8 is a higher-order tree function when setting \(\Sigma_C\) and \(\Sigma'_{C'}\) as input and output signatures and if we consider \(e\) and \(v\) as nullary constants in \(C'\).

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

**Definition 10.11** (Abstract Categorial Grammar). An abstract categorial grammar (ACG) is a tuple \(G = \langle \Sigma, \Sigma', [\cdot], s \rangle\) where \(\Sigma = \langle A, C, \tau \rangle\) and \(\Sigma' = \langle A', C', \tau' \rangle\) are two signatures, \([\cdot]\) is a linear typed homomorphism, and \(s\) in \(A\) is a distinguished atomic type. The abstract language \(\mathcal{A}(G)\) of \(G\) is

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

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

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

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

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

\[
\downarrow_\beta\eta \mathcal{O}(G) = \downarrow_\beta\eta \{[t] \in \Lambda_\ell(C') \mid t \text{ ground } \in \mathcal{A}(G)\}.
\] (10.2)

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

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

**Exercise 10.5** (Tree Languages of ACG\(_{2,1}\)). Given an ACG \(G = \langle \Sigma, \Sigma_F, [\cdot], s \rangle\) with a second-order abstract signature \(\Sigma = \langle A, C, \tau \rangle\) and a generic tree signature \(\Sigma_F\) over some ranked alphabet \(F\) as object signature, we define its tree language as

\[
\mathcal{T}(G) \overset{\text{def}}{=} \{t \in T(F) \mid t \text{ ground } \in \mathcal{O}(G)\}.
\] (10.3)

Assume that \(\max_{a \in A} \text{ord}([a]) = 1\). Show that such ACGs generate exactly the set of regular tree languages.
More generally, the expressiveness of second-order ACGs has been studied by Kanazawa (2010): their object languages correspond to the tree languages of context-free hyperedge replacement grammars, which are also equivalent to attributed context-free grammars (Engelfriet and Heyker, 1992) and outputs of restricted forms of MTTs (Engelfriet and Maneth, 2000). This means that we could also implement the tree transformations defined by second-order ACGs using more classical tree transductions. However, this would be at the expense of the ability to view the translation as one into higher-order semantics, as we will do in Section 10.3. In that situation, we will no longer work with ground object terms.

10.2 Intensionality

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

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

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

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

Given an infinite countable set of object variables \(O\) and an infinite countable set of intension variables \(I\), FOIL formulae follow the syntax

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

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

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

\[
\mathfrak{M}, w \models_{o} \exists f. \varphi \quad \text{iff} \quad \exists i \in D_I(w). \mathfrak{M}, w \models_{v[f \leftarrow i]} \varphi
\]

\[
\mathfrak{M}, w \models_{o} [\lambda x. \varphi](f) \quad \text{iff} \quad \mathfrak{M}, w \models_{v[\lambda xx'^\prime. \varphi](g)[w]} \varphi.
\]

**Example 10.12** (Morning Star). Let us consider again the sentence *The morning star is the evening star* and associate \(f\) to the intension *the morning star* and \(g\) to the intension *the evening star*. Then \(\lambda xx'. x = x'\)(\(f, g\)) is correct in the real world \(w\), where \(f\) and \(g\) are associated to the same object \(\nu(f)(w) = \nu(g)(w)\) in \(D_O\), namely Venus. In an epistemic setting, the de dicto reading \(K[\lambda xx'. x = x']\(f, g\)) can be falsified if we find another state of knowledge \(w'\) compatible with the real world \(w\) where this information is missing, i.e. where \(\nu(f)(w') \neq \nu(g)(w')\)—this could be the case in the sentence *John knows that the morning star is the evening star* if John is unaware of the both being Venus. By contrast, the de re reading \([\lambda xx'. f, g\)](x = x')(\(f, g\)) is always satisfied in \(w\) because in any state of knowledge compatible with the real world, \(f\) and \(g\) have received the same extension \(\nu(f)(w) = \nu(g)(w)\).

**Example 10.13** (King of England). The treatment of the sentence *The King of England was the head of the Church of England* is similar: consider the intensions \(f\) for the King of England, \(g\) for the head of the Church of England, and a point in time \(w\). Then \(P[\lambda xx'. x = x']\(f, g\)) could be invalidated if there is no past time \(w' < w\) where the denotations \(\nu(f)(w')\) and \(\nu(g)(w')\) were the same—i.e. before the 1538 secession from the Roman Church—, but is valid in time points \(w\) after the secession. The de re reading does not make any sense: \([\lambda xx'. f, g\)](x = x')(\(f, g\)) holds iff \(\nu(f)(w) = \nu(g)(w)\) at the time of interest, regardless of past times where equality is evaluated.

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

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

\[
\forall x. \exists f. D(f, x) \quad (10.4)
\]
to hold. This is however a reasonable restriction; let us check for instance the following equivalence under the hypothesis of \((10.4)\):
\[
\exists x. \varphi \equiv \exists f. [\lambda x. \varphi](f) .
\] (10.5)

Indeed, for all \(\mathcal{M}, w, \nu\) and \(\varphi\),
\[
\mathcal{M}, w \models_{\nu} \exists f. [\lambda x. \varphi](f)
\]
iff \(\exists i \in D_{\mathcal{M}}, w \models_{\nu[f \leftarrow i]} [\lambda x. \varphi](f)\)
iff \(\exists i \in D_{\mathcal{M}}, w \models_{\nu[f \leftarrow i(x(w))] \varphi}\)
iff \(\exists e \in D_{\mathcal{M}}, w \models_{\nu[x \leftarrow e]} \varphi\) \hspace{1cm} \text{(by \((10.4)\) when choosing \(i(w) = e\))}
iff \(\mathcal{M}, w \models_{\nu} \exists x. \varphi\).

\((*)\) Exercise 10.6. Show the following equivalence when \((10.4)\) holds:
\[
\exists f. \Diamond [\lambda x. \varphi](f) \equiv \Diamond (\exists x. \varphi) .
\] (10.6)

Example 10.14 (Unicorn). The sentence \textit{John finds a unicorn} could be associated with the semantics
\[
\exists x. \text{find}^{(1)}(x) \wedge \text{agent}^{(2)}(x, \text{John}) \wedge \text{patient}^{(2)}(e, x) \wedge \text{unicorn}^{(1)}(x) \tag{10.7}
\]
but it is better to treat \textit{unicorn} as an intension in the formula
\[
\exists u. [\lambda x. \exists e. \text{find}^{(1)}(e) \wedge \text{agent}^{(2)}(e, \text{John}) \wedge \text{patient}^{(2)}(e, x) \wedge \text{unicorn}^{(1)}(x)](u),
\] (10.8)
equivalent to \((10.7)\) in totally intensional models according to \((10.5)\). Then we better see the connection with the sentence \textit{John seeks a unicorn}: its \textit{de dicto} semantics would be
\[
\exists u. \text{TRY}(\text{John}, [\lambda x. \exists e. \text{find}^{(1)}(e) \wedge \text{patient}^{(2)}(e, x) \wedge \text{unicorn}^{(1)}(x)](u)) \tag{10.9}
\]
\[
\equiv \text{TRY}(\text{John}, \exists x. \text{find}^{(1)}(e) \wedge \text{patient}^{(2)}(e, x) \wedge \text{unicorn}^{(1)}(x)) \hspace{1cm} \text{(by \((10.6)\))}
\]
and its \textit{de re} semantics
\[
\exists u. [\lambda x. \text{TRY}(\text{John}, \exists e. \text{find}^{(1)}(e) \wedge \text{patient}^{(2)}(e, x) \wedge \text{unicorn}^{(1)}(x)](u) \tag{10.10}
\]
\[
\equiv \exists x. \text{TRY}(\text{John}, \exists e. \text{find}^{(1)}(e) \wedge \text{patient}^{(2)}(e, x) \wedge \text{unicorn}^{(1)}(x)) \hspace{1cm} \text{(by \((10.5)\))}
\]
and if the interpretation of \textit{unicorn} is the same in all worlds accessible through the \textit{TRY} modality,
\[
\equiv \exists x. \text{unicorn}^{(1)}(x) \wedge \text{TRY}(\text{John}, \exists e. \text{find}^{(1)}(e) \wedge \text{patient}^{(2)}(e, x)) .
\]

10.3 Higher-Order Logic

Most of the discussion on semantic representations can be recast in the framework of higher-order logic. This allows in particular to view the higher-order operations of Section 10.1 not as a technical means to generate \(\lambda\)-terms viewed as trees (which in turn can be interpreted in some logic), but instead to interpret these terms directly in the higher-order logic. They become the semantics of the sentences under consideration, with associated models.

10.3.1 Background: Church’s Simple Theory of Types

Higher-order semantics are typically expressed in simply typed lambda calculus as defined in Section 10.1.1. As we want not just to manipulate typed \(\lambda\)-terms, but also to be able to infer truths, we need to introduce a set of \textbf{logical constants} and the associated \textbf{logical rules}.

\[\text{See Church (1940) and the entry in the Stanford Encyclopedia of Philosophy.}\]
Higher-Order Signature  In Church’s simple theory of types, we use a signature \( \Sigma = \langle A, C, t \rangle \) where \( A = \{ \iota, o \} \) is set of atomic types, where \( \iota \) denotes entities and \( o \) truths. The logical constants are \( C = \{ \bot, \supset, (\forall_\tau) \tau \in T(A) \} \) with types \( t(\bot) = o, t(\supset) = o \rightarrow o \rightarrow o \), and \( t((\forall_\tau) = (\tau \rightarrow o) \rightarrow o \) for each type \( \tau \in T(A) \).

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

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

\[
\begin{align*}
\Gamma, L &\vdash L \quad (Ax) \\
\Gamma, L &\vdash M \implies M \quad (\supset I) \\
\Gamma &\vdash L \implies M \quad (\supset E) \\
\Gamma &\vdash L \quad x \not\in \text{FV}(\Gamma) \quad (\forall I) \\
\Gamma &\vdash (\forall_\tau) x.L =_\beta M \quad (\forall E) \\
\Gamma &\vdash (\forall_\tau) x.L x = M x \implies (L = M) \quad (\lambda X) \\
\Gamma &\vdash (L \equiv M) \implies (L = M) \quad (\equiv X)
\end{align*}
\]

As their name indicates, the extensionality axioms make the simple theory of types unable to deal with intensional phenomena directly; a solution we will see in Section 10.3.2 will be to introduce an new atomic type \( s \) ranging over worlds.

Higher-order logic can express a form of set theory: view the set comprehension \( \{ x \mid P \} \) as \( \lambda x.P \), or \( e \in E \) as \( Ee \). In fact, [Church (1940)](#church1940) shows how to implement Peano’s arithmetic in the simple theory of types, from which we can deduce the incompleteness of higher-order logic.

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

### 10.3.2 Type-Logical Semantics

Many classical modelings of natural language semantics in higher-order logic posit an additional type \( s \) of worlds in order to account for modalities and intensionality phenomena. The idea is to always treat truth values (of type \( o \)) as relativized with respect to a possible world of evaluation. Thus we will consider a
Observe that we introduced an explicit type for worlds in the logic: this can be avoided if we use intensional models as in Muskens (2007). Recall that Church’s simple type theory verifies the extensionality axioms!

Table 10.1: Some constants and their possible types.

<table>
<thead>
<tr>
<th>syntactic category</th>
<th>examples</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>intransitive verbs</td>
<td>walk, talk, eat, ...</td>
<td>( t \to s \to o )</td>
</tr>
<tr>
<td>transitive verbs</td>
<td>eat, love, ...</td>
<td>( t \to t \to s \to o )</td>
</tr>
<tr>
<td>common nouns</td>
<td>apple, man, woman, ...</td>
<td>( t \to s \to o )</td>
</tr>
<tr>
<td>adjectives</td>
<td>red, ...</td>
<td>( t \to s \to o )</td>
</tr>
<tr>
<td>determiners</td>
<td>every, a, the, no, ...</td>
<td>( (t \to s \to o) \to (t \to s \to o) \to s \to o )</td>
</tr>
<tr>
<td>proper nouns</td>
<td>John, Mary, ...</td>
<td>( t )</td>
</tr>
<tr>
<td>modal adverbs</td>
<td>necessarily, possibly, ...</td>
<td>( s \to o ) ( s \to o )</td>
</tr>
<tr>
<td>modal verbs</td>
<td>know, believe, ...</td>
<td>( (s \to o) \to t \to s \to o )</td>
</tr>
<tr>
<td>negation</td>
<td>not</td>
<td>( (s \to o) \to s \to o )</td>
</tr>
</tbody>
</table>

Table 10.2: Examples of semantics associated with lexical elements.

\[
\begin{align*}
[\text{walk}] &= \text{walk}_{t \to s \to o} \\
[\text{eat}] &= \text{eat}_{t \to s \to o} \\
[\text{apple}] &= \text{apple}_{t \to s \to o} \\
[\text{red}] &= \lambda P_{t \to s \to o} x w, \text{red}_{t \to s \to o} x w \land P x w \\
[\text{every}] &= \lambda P_{t \to s \to o} P'_{t \to s \to o} w, \forall x.(P x w \supset P' x w) \\
[\text{a}] &= \lambda P_{t \to s \to o} P'_{t \to s \to o} w, \exists x.(P x w \land P' x w) \\
[\text{no}] &= \lambda P_{t \to s \to o} P'_{t \to s \to o} w, \forall x.(P x w \supset \neg P' x w) \\
[\text{the}] &= \lambda P_{t \to s \to o} P'_{t \to s \to o} w, \exists x.(P' x w \land \forall y.(P x w \equiv x = y)) \\
[\text{John}] &= \text{John}_t \\
[\text{necessarily}] &= \lambda p_{s \to o} w, \forall x w'. (R_{s \to s \to o} w w' \supset p w') \\
[\text{possibly}] &= \lambda p_{s \to o} w, \exists x w'. (R_{s \to s \to o} w w' \land p w') \\
[\text{know}] &= \lambda p_{s \to o} x w, \forall x w'. (K_{s \to s \to s \to o} x w w' \supset p w') \\
[\text{believe}] &= \lambda p_{s \to o} x w, \forall x w'. (B_{s \to s \to s \to o} x w w' \supset p w') \\
[\text{not}] &= \lambda p_{s \to o} w, \neg p w
\end{align*}
\]

higher signature \( \Sigma = \{ A, \{ \bot, \supset, (\forall_T)_{T \in T(A)} \} \cup C, t \} \) as in the simple theory of types, where \( A = \{ s, t, o \} \) and \( C \) denotes additional non-logical constants. To simplify matters, we avoid explicit events from Section 1.2.

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

\[
\text{walks John }.
\]

In order to obtain the appropriate type, a possibility is to set \( t(\text{walks}) = t \to s \to o \) and \( t(\text{John}) = t \). Looking at more complex examples (for instance Example 10.8), we arrive at the types of Table 10.1. The semantics of a sentence can then be computed by a higher-order homomorphism as in Section 10.1, but there will be no need to translate back from \( \lambda \)-terms to first-order terms in order to reason about the semantics: the \( \lambda \)-term is a meaning representation with full-fledged model theory. See Table 10.2 for some examples of semantic values.

In this table, the semantics of alethic and epistemic modal logics have been implemented directly using the \( R, K, \) and \( B \) constants with types \( s \to s \to o \),
ι \rightarrow s \rightarrow s \rightarrow o, and \ i \rightarrow s \rightarrow s \rightarrow o \ \text{respectively. The desired properties of these relations can also be enforced; for instance } \forall_{s, w, w'} R_{w/w'} \ \text{forces } R \ \text{to be total.}
Notations

We use the following notations in this document. First, as is customary in linguistic texts, we prefix agrammatical or incorrect examples with an asterisk, like *ationhospitalmis or *sleep man to is the.

These notes also contain some exercises, and a difficulty appreciation is indicated as a number of asterisks in the margin next to each exercise—a single asterisk denotes a straightforward application of the definitions.

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

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

We lift $\cdot$ to subsets of $M$ by $L_1 \cdot L_2 = \{m_1 \cdot m_2 \mid m_1 \in L_1, m_2 \in L_2\}$. Then for $L \subseteq M$, $L^0 = \{1_M\}$ 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 $\Sigma$ is a relation $R \subseteq \Sigma^* \times \Sigma^*$. The elements $(u, v)$ of $R$ are called string rewrite rules and noted $u \rightarrow v$. The one step derivation relation generated by $R$, noted $R^1$, is the relation over $\Sigma^*$ defined for all $w, w'$ in $\Sigma^*$ by $w \xrightarrow{R} w'$ iff there exist $x, y$ in $\Sigma^*$ such that $w = xuv, w' = xvy$, and $u \rightarrow v$ is in $R$. The derivation relation is the reflexive transitive closure $R^*$.

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

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

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

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

See Comon et al. (2007) for missing definitions and notations.
The set \( T(\Sigma, \mathcal{X}) \) of terms over \( \Sigma \) and \( \mathcal{X} \) is the smallest set s.t. \( \Sigma_0 \subseteq T(\Sigma, \mathcal{X}) \), \( \mathcal{X} \subseteq T(\Sigma, \mathcal{X}) \), and if \( n > 0 \), \( f \) is in \( \Sigma_n \), and \( t_1, \ldots, t_n \) are terms in \( T(\Sigma, \mathcal{X}) \), then \( f(t_1, \ldots, t_n) \) is a term in \( T(\Sigma, \mathcal{X}) \). The set of terms \( T(\Sigma, \emptyset) \) is also noted \( T(\Sigma) \) and is called the set of ground terms.

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

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

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

Term Rewrite Systems. A term rewrite system over some ranked alphabet \( \Sigma \) is a set of rules \( R \subseteq (T(\Sigma, \mathcal{X}))^2 \), each noted \( t \rightarrow t' \) (also noted \( t \xrightarrow{\text{r}} t' \)), with \( t, t' \) in \( T(\Sigma, \mathcal{X}_n) \), the associated one-step rewrite relation over \( T(\Sigma) \) is \( \Rightarrow = \{ C[t\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}], C[t'\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}] \mid C \in C(\Sigma), t_1, \ldots, t_n \in T(\Sigma) \} \). We write \( \frac{R_1 \cup R_2}{R_1, R_2} \Rightarrow \) for \( \frac{R_1}{R_1} \Rightarrow \frac{R_2}{R_2} \Rightarrow \), and \( \frac{R}{R} \Rightarrow \) for \( \bigcup_{r \in R} \frac{r}{r} \Rightarrow \).


143


