Order-Preserving Graph Grammars

Petter Ericson

Doctoral Thesis, February 2019
Department of Computing Science
Umeå University
Sweden

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

pettter@cs.umu.se

Copyright © 2019 by Petter Ericson
Except for Paper I, © Springer-Verlag, 2016
Paper II, © Springer-Verlag, 2017

ISBN 978-91-7855-017-3
ISSN 0348-0542
UMINF 19.01

Front cover by Petter Ericson
Printed by UmU Print Service, Umeå University, 2019.

It is good to have an end to journey toward;
but it is the journey that matters, in the end.

URSULA K. LE GUIN

Abstract

The field of *semantic modelling* concerns formal models for semantics, that is, formal structures for the computational and algorithmic processing of *meaning*. This thesis concerns formal *graph languages* motivated by this field. In particular, we investigate two formalisms: Order-Preserving DAG Grammars (OPDG) and Order-Preserving Hyperedge Replacement Grammars (OPHG), where OPHG generalise OPDG.

Graph parsing is the practise of, given a graph grammar and a graph, to determine if, and in which way, the grammar could have generated the graph. If the grammar is considered fixed, it is the *non-uniform* graph parsing problem, while if the grammars is considered part of the input, it is named the *uniform* graph parsing problem. Most graph grammars have parsing problems known to be NP-complete, or even exponential, even in the non-uniform case. We show both OPDG and OPHG to have polynomial uniform parsing problems, under certain assumptions.

We also show these parsing algorithms to be suitable, not just for determining membership in graph languages, but for computing weights of graphs in *graph series*.

Additionally, OPDG is shown to have several properties common to regular languages, such as MSO definability and MAT learnability. We moreover show a direct correspondence between OPDG and the regular tree grammars.

Finally, we present some limited practical experiments showing that real-world semantic graphs appear to mostly conform to the requirements set by OPDG, after minimal, reversible processing.

Populärvetenskaplig sammanfattning

Inom språkvetenskap och datalingvistik handlar mycket forskning om att på olika sätt analysera *strukturer* inom språk; dels syntaktiska strukturer – vilka ordklasser som kommer före andra och hur de i satser och satsdelar kan kombineras för korrekt meningsbyggnad, och dels hur olika ords *mening* eller *semantik* kan relatera till varandra och till idéer, koncept, tankar och ting. Denna avhandling behandlar formella datavetenskapliga modeller för just sådan *semantisk modellering*. I vårt fall representeras vad en mening betyder som en *graf*, bestående av noder och kanter mellan noder, och de formella modeller som diskuteras är *grafgrammatiker*, som avgör vilka grafer som är korrekta och inte.

Att med hjälp av en grafgrammatik avgöra om och hur en viss graf är korrekt kallas för *grafparsing*, och är ett problem som generellt är beräkningsmässigt svårt – en grafparsingalgoritm tar i många fall exponentiell tid i grafens storlek att genomföra, och ofta påverkas dessutom körtiden av grammatikens sammansättning och storlek.

I den här avhandlingen beskrivs två relaterade modeller för semantiskt modellering – Ordningsbevarande DAG-grammatiker (OPDG) och Ordningsbevarande Hyperkantsomskrivningsgrammatiker (OPHG). Vi visar att grafparsingproblemet för OPDG och OPHG är effektivt lösbart, och utforskar vad som behöver gälla för att en viss grammatik skall vara en OPHG eller OPDG.

Acknowledgements

There are very few books that can be attributed solely to a single person, and this thesis is no exception to that rule. Listing all of the contributors is an exercise in futility, and thus my lie in the acknowledgement section of my licentiate thesis is exposed – this will by necessity if not by ambition be an incomplete selection of people who have helped me get to this point.

First off, my advisor Henrik Björklund and co-advisor Frank Drewes both deserve an enormous part of the credit that is due from this thesis. They have discussed, co-authored, taught, coaxed, calmed down, socialised, suggested, edited, and corrected, all as warranted by my ramblings and wanderings in the field and elsewhere, and all of that has acted to make me a better researcher and a better person both. Thank you.

Second, Linn, my better half, has helped with talk, food, hugs, kisses, walks, travels, schedules, lists (so many lists) and all sort of other things that has not only kept me (more or less) sane, but in any sort of state to be capable of finishing this thesis. There is no doubt in my mind that it would not have been possible, had I not had you by my side. You are simply the best.

Third, the best friend I could never deserve and always be thankful for, Philip, who has simply been there through it all, be it on stage, at the gaming table, or in the struggles of academia. Your help with this thesis made it at least $134(\pm 3)\%$ better in every single respect, and hauled me back from the brink, quite literally. You are *also*, somehow, the best.

I have also had the pleasure and privilege to be part of an awesome group of people known as the Foundations of Language Processing research group, which have at different points during my PhD studies contained people such as my co-authors Johanna and Florian, my bandmate Niklas, my PhD senior Martin, and juniors Anna, Adam and Yonas. Suna and Mike are neither co-authors nor common PhD student with me, but have been magnificient colleagues and inspirations nonetheless. I have enjoyed our friday lunches, seminars and discussion greatly, and will certainly try to set up something similar wherever I end up.

Many other colleagues have helped me stay on target in various ways, such as my partner in course crime Jan-Erik, the compulsive crosswords-solvers including Tomas, Carina, Helena, Niklas, Niclas, Helena, Lars, Johan, Mattias, Pedher, and many more coworkers of both the crossword-solving and -avoiding variety.

My family has also contributed massively with both help and inspiration during my PhD studies. My parents Curry and Lars both supplying rides, talks, jams, academic insights, dinners, berries, mushrooms, practically infinite patience, and much more.

My sister Tove and her husband Erik providing a proof-of-concept of Dr. Ericson, showing that it can indeed be done, and that it can be done while being and remaining some of the best people I know, and all this while my niece Isa and nephew Malte came along as well.

Too many friends and associations have brightened my days and evenings during these last years to list them all, but I will make the attempt. In no particular order, thanks to Olow, Diana, Renhornen, JAMBB, Lochlan, Eric, Jenni, Bengt, Birgit, Kelly, Tom, MusicTechFest, Björn, Linda, Erik, Filip, Hanna (who got me into this whole thing), Hanna (who didn't), Michela, Dubber (we'll get that paper written soon), Sofie, Alexander, Maja, Jonathan, Peter, Niklas, Oscar, Hanna (a recurring pattern), Mats, Nick, Viktor, Anna, Ewa, Snösvänget, Avstamp, Isak, Samuel, Kristina, Ivar, Helena, Christina, Britt, Louise, Calle, Berit, Thomas, Kerstin, Lottis, Ola, Mattias, Mikael, Mikael, Tomas, Mika, Benjamin, and the rest of the Umeå Hackerspace gang, Offer, André, Four Day Weekend, LESS, Mikael, Staffan, Lisa, Jesper, Henrik, John, Mats, Carlos, Valdemar, Axel, Arvid, Christoffer, Tomas, Becky, Jocke, Jennifer, Jacob, Jesper, Ellinor, Magne, Hanna (yes really), Johan, Lennart, LLEO, Jet Cassette, Calzone 70, Veronica, Johan, Container City, Malin, Sanna, Fredrik, Maja, Mats, and everyone that have slipped my mind for the moment and will remember seconds after sending this to print. Thank you all!

Preface

The following papers make up this Doctoral Thesis, together with an introduction.

Paper I	Henrik Björklund, Frank Drewes, and Petter Ericson. Between a Rock and a Hard Place – Parsing for Hyperedge Replacement DAG Grammars. In *10th International Conference on Language and Automata Theory and Applications (LATA 2016), Prague, Czech Republic*, pp. 521-532, Springer, 2016.
Paper II	Henrik Björklund, Johanna Björklund, and Petter Ericson. On the Regularity and Learnability of Ordered DAG Languages. In Arnaus Carayol and Cyril Nicaud, editors, *22nd International Conference on the Implementation and Application of Automata (CIAA 2017), Marne-la-Vallée, France,* volume 10329 of *Lecture Notes in Computer Science*, pp. 27-39, Springer 2017.
Paper III	Henrik Björklund, Johanna Björklund, and Petter Ericson. Minimisation and Characterization of Order-preserving DAG Grammars. *Technical Report UMINF 18.15 Dept. Computing Sci., Umeå University*, `http://www8.cs.umu.se/research/uminf/index.cgi`, 2018. Submitted
Paper IV	Henrik Björklund, Frank Drewes, and Petter Ericson. Uniform Parsing for Hyperedge Replacement Grammars. *Technical Report UMINF 18.13 Dept. Computing Sci., Umeå University*, `http://www8.cs.umu.se/research/uminf/index.cgi`, 2018. Submitted
Paper V	Henrik Björklund, Frank Drewes, and Petter Ericson. Parsing Weighted Order-Preserving Hyperedge Replacement Grammars. *Technical Report UMINF 18.16 Dept. Computing Sci., Umeå University*, `http://www8.cs.umu.se/research/uminf/index.cgi`, 2018.

Additionally, the following technical report and paper were completed during the course of the PhD program.

Paper I	Petter Ericson. A Bottom-Up Automaton for Tree Adjoining Languages. *Technical Report UMINF 15.14 Dept. Computing Sci., Umeå University*, `http://www8.cs.umu.se/research/uminf/index.cgi`, 2015.
Paper II	Henrik Björklund and Petter Ericson. A Note on the Complexity of Deterministic Tree-Walking Transducers. In *Fifth Workshop on Non-Classical Models of Automata and Applications (NCMA)*, pp. 69-84, Austrian Computer Society, 2013.

Contents

CHAPTER 1

Introduction

This thesis concerns the study of graphs, and grammars and automata working on graphs, with applications in the processing of human language as well as other fields. A new formalism with two variants is presented and various desirable features are shown to hold, such as efficient (weighted) parsing, and for the limited variant, MSO definability, MAT learnability, and a normal form.

It also presents novel, though limited, practical work using these formalisms, and aims to show their usefulness in order to motivate further study.

1.1 The Study of Languages

The *study of languages* has at minimum two very distinct meanings. This thesis concerns both.

The first, and probably most intuitive, sense of "study of language" concerns *human languages*, which is the kind of languages we humans use to communicate ideas, thoughts, concepts and feelings. The study of such languages is an immense field of research, including the whole field of linguistics, but also parts of informatics, musicology, various fields studying *particular* languages such as English or Swedish, philosophy of language, and many others. It concerns what human language is, how languages work, and how they can be applied. It also concerns how understanding of human language and various human language tasks such as translation and transcription can be formalised or computerised, which is where the work presented in this thesis intersects the subject.

Though many of the applications mentioned in this thesis make reference to *natural language processing* (NLP), it may be appropriate to mention that many of the techniques are in fact also potentially useful for processing *constructed languages* such as Klingon,[1] Quenya,[2] Esperanto,[3], Lojban[4] and toki pona[5]. The language of music also shares many of the same features, and musical applications are used later in this introduction to illustrate a practical use case of the results.

[1] From Gene Roddenberry's "Star Trek"
[2] From J.R.R. Tolkien's "The Lord of the Rings"
[3] An attempt at a "universal" language, with a grammar without exceptions and irregularities.
[4] A "logical" language where the grammar is constructed to minimise ambiguities.
[5] A "Taoist" language, where words are generally composites of a "minimal" amount of basic concepts.

The particular field that has motivated the work presented in this thesis is that of *semantic modelling*, which seeks to capture the semantics, or *meaning* of various language objects, e.g. sentences, in a form suitable for further computational and algorithmic processing. In short, we wish to move from representing some real-world or imaginary concept using *natural* language to representing the same thing using more *formal* language.

This bring us to the second meaning of "language" and its study. For this we require some mathematical and theoretical computer science background. In short, given a (usually infinite) universe (set) of objects, a (formal) *language* is any subset of the universe (including the empty set and the complete universe). The study of these languages, their definitions, applications, and various other properties, is, loosely specified, the field of *formal language theory*. Granted, the given definition is *very* wide, bordering on unusable, and we will spend Chapters 3 to 5 defining more specifically the field and subfield that is the topic of this thesis.

1.2 Organisation

The organisation of the rest of this thesis proceeds as follows: First, we present the field of semantic modelling through a brief history of the field, after which we introduce formal language theory in general, including the relevant theory of finite automata, logic, and various other fields. We then discuss the formalisms that form the major contributions in the papers included in this thesis. We briefly introduce a number of related formalisms, before turning to the areas and applications which we aim to produce practical results for, and the minor practical results themselves. We conclude the introduction with a section on future work, both for developing and using our formalisms. Finally, the five papers that comprise the major scientific contributions in this thesis are included.

CHAPTER 2

Semantic modelling

The whole field of human language is much too vast to give even a cursory introduction to in a reasonable way, so let us focus on the specific area of study that has motivated the theoretical work presented here.

From the study of human language in its entirety, let us first focus on the area of *natural language processing*, which can be very loosely described as the study of how to use computers and algorithms for natural language tasks such as translation, transcription, natural language interfaces, and various kinds and applications of natural language understanding. This, while being more specific than the whole field of human language, is thus still quite general.

The specific area within natural language processing that interests us is *semantic processing*, and even more specifically, *semantic modelling*. That is, we are more interested in the structure of *meaning*, than that of *syntax*. Again, this is a quite wide field, which has a long and varied history, but the problem can be briefly stated as follows: How can we represent the meaning of a sentence in a way that is both reasonable and useful?

2.1 History

Arguably, this field has predecessors all the way back to the beginning of the Enlightenment with the attempts of Francis Lodwick, John Wilkins and Gottfried Wilhelm Leibniz among others to develop a "philosophical language" which would be able to accurately represent facts of the world without the messy abstractions, hidden contexts and ambiguities of natural language. This was to be accomplished in Wilkins conception [Wil68] through, on the one hand, a "scientifically" conceived writing system based on the anatomy of speech, on the other hand, an unambiguous and regular syntax with which to construct words and sentences, and on the gripping hand,[1] a well-known and unambiguous way to refer to things and concepts, and their relations. Needless to say these attempts, though laudable and resulting in great insights, ended with no such language in use, and the worlds of both reality and imagination had proven to be much more complex and fluid than the strict categories and forty

[1] This is a somewhat oblique reference to the science fiction novel "A Mote in Gods Eye" by Larry Niven and Jerry Pournelle. The confused reader may interpret this as "on the third hand, and most importantly".

all-encompassing groupings (or genera) of Wilkins.

Even though the intervening years cover many more interesting and profound insights, the next major step that is relevant to this thesis occurs in the mid-twentieth century, with the very first steps into implementing general AI on computers. Here, various approaches to *knowledge representation* could be tested “in the field”, by attempting to build AI systems using them. Initially, projects like the General Problem Solver [NSS59] tried to, once again, deal with the world and realm of imagination in a complete, systematic way. However, the complexities and ambiguities of the real world once again gradually asserted themselves to dissuade this approach. Instead, *expert systems* became the norm, where the domain was very limited, e.g. to reasoning about medical diagnoses. Here, the meaning of sentences could be reduced to simple logical formulae and assertions, which could then be processed to give a correct output given the inputs.

In parallel, the field of natural language processing was in its nascent stages, with automated translation being widely predicted to be an easy first step to fully natural language interfaces being standard for all computers. This, too, was quickly proven to be an overly optimistic estimation of the speed of AI research, but led to an influx of funding, kick-starting the field and its companion field of *computational linguistics*. Initial translator systems dispensed with any pretence at semantic understanding or representation, and used pure lexical word for word correspondences between languages to translate text from one language to another, with some special rules in place for certain reorderings, deletions and insertions. Many later attempts were built on similar principles, but sometimes used an intermediate representation, or *interlingua*, as a step between languages. This can be seen as a kind of semantic representation.

After several decades of ever more complex rules and hand-written grammars, the real world once again showed itself to be much too complex to write down in an exact way. Meanwhile, an enormous amount of data had started to accumulate in ever more computer accessible formats, and simple statistical models trained on such data started seeing success in various NLP tasks. Recent examples of descendants of such models include all sorts of neural network and “deep learning” approaches.

With the introduction of statistical models, we have essentially arrived at a reasonable picture of the present day, though the balance between data-driven and handwritten continues to be a difficult one, as is the balance between complexity and expressivity of the chosen models. An additional balance that has recently come to the fore is the balance between efficiency and *explainability* – that is, if we construct a (usually heavily data-driven) model and use it for some task, how easy is it so see *why* the model behaves as it does, and gives the results we obtain? All of these separate dimensions of course interact, sometimes in unexpected ways.

2.2 Issues in Semantic Modelling

With this history in mind, let us turn to the practicalities of representing semantics. The two keywords in our question above is “reasonable” and “useful” – they require some level of judgement as to for whom and what purpose the representation should be reasonable and useful. As such, this question has much in common with other questions in NLP research. Should we focus on making translations that seem good to professional translators and bilinguals, or should we focus on making the translations explicit, showing the influence of each part of the input to the relevant parts of the output? Is it more important to use data structures that mimic what "really" is going on in the mind, or should we use abstractions that are easier to reason about, even though they may be faulty or inflexible, or should we just use "whatever works best", for some useful metric?

Thus the particular semantic representation chosen depends very much on the research question and on subjective valuations. In many instances, the semantics are less important than the function, and thus a relatively opaque representation may be chosen, such as a simple vector of numbers based on word proximity (word embeddings). In others, the computability and manipulation of semantics is the central interest, and thus the choice is made to represent semantics using logical structures and formulae.

In this thesis, we have chosen to explore formalisms for semantic *graphs*, that represent the meaning of sentences using connections between concepts. However, to properly define and discuss these requires a bit more background of a more formal nature, which will be provided in the following chapters.

CHAPTER 3
String languages

With some background established in the general field of natural language processing, let us turn to theory.

To define *strings*, we first need to define *alphabets*: these are (usually finite) sets of distinguishable objects, which we call *symbols*. A *string over the alphabet* Σ is any sequence of symbols from the alphabet, and a *string language* (over the same) is any set of such strings. A set of languages is called a *class* of languages.

Thus the string *"aababb"* is a string over the alphabet $\{a,b\}$ (or any alphabet containing a and b), while the string *"the quick brown fox jumps over the lazy dog"* is a string over the alphabet of lowercase letters a to z and a space character (or any superset thereof).

Further, any finite set of strings, such as $\{$"*a*","*aa*","*aba*"$\}$ is a string language, but we can also define infinite string languages such as "all strings over the alphabet $\{a,b\}$ containing an even number of a's", or "any string over the English alphabet (plus a space character) containing the word *supercalifragilisticexpialidocious*".

For classes of languages, we can again look at finite sets of languages, though it is generally more interesting to study infinite sets of infinite languages. Let us look closer at such a class – the well-studied class of *regular string languages* (REG). We define this class first inductively using *regular expressions*, which were first described by Stephen Kleene in [Kle51]. In the following, e and f refer to regular expressions, while a is a symbol from the alphabet.

- A symbol a is a regular expression defining the language "a string containing only the symbol a".
- A concatenation $e \cdot f$ defines the language "a string consisting of first a string from the language of e, followed by a string from the language of f". We often omit the dot, leaving ef.
- An alternation (or union) $(e)|(f)$ defines the language "either a string from the language of e, or one from the language of f"
- A repetition e^* defines the language "zero or more concatenated strings from the language of e" [1]

[1] This is generally called a Kleene star, from Stephen Kleene.

Thus, the regular expression $(a)|(aa)|(aba)$ defines the finite string language $\{"a","aa","aba"\}$, while $(b^*ab^*a)^*b^*$ is one way of writing the language “all strings over $\{ab\}$ with an even number of *a*’s”.

3.1 Automata

Let us now turn to one of the most important, well-studied, and, for lack of a better word, modified structures of formal language theory – the finite automaton, which in its general principles were defined and explored by Alan Turing in [Tur37].

In short, a finite automaton is an idealised machine that can be in any of a finite set of *states*. It reads an input string of symbols, and, upon reading a symbol, moves from one state to another. Using this simple, mechanistic framework, we can achieve great complexity. As great, it turns out, as when using regular expressions (Kleene’s Theorem [Kle51]). Let us formalise this:

Finite automaton A *finite (string) automaton* is a structure $A = (\Sigma, Q, q_0, F, \delta)$, where

- Σ is the *input alphabet*
- Q is the set of *states*
- $q_0 \in Q$ is the *initial state*
- $F \subset Q$ is the set of *final states*, and
- $\delta : (\Sigma \times Q) \to 2^Q$ is the *transition function*

A *configuration* of an automaton A is an element of $(Q \times \Sigma^*)$, that is, a state paired with a string or, equivalently, a string over $Q \cup \Sigma$ where only the first symbol is taken from Q. The *initial configuration* of A on a string w is the configuration q_0w, a *final configuration* is q_f, where $q_f \in F$, and a *run* of A on w is a sequence of configurations $q_0w = q_0w_0, q_1w_1, \ldots, q_kw_k = q_k$, where for each i, $w_i = cw_{i+1}$ for some $c \in \Sigma$, and $q_{i+1} \in \delta(q_i, c)$. If a run ends with a final configuration, then the run is *successful*, and if there is a successful run of an automaton on a string, the automaton *accepts* that string. The *language* of an automaton is the set of strings it accepts.

An automaton that accepts the finite language $\{"a","aa","aba"\}$ is, for example, $A = (\{a,b\}, \{q_0,q_1,q_2,q_f\}, q_0, \{q_1,q_f\}, \delta)$ where

$$\delta = \{(q_0,a) \to \{q_1\}, (q_1,a) \to \{q_f\}, (q_1,b) \to \{q_2\}, (q_2,a) \to \{q_f\},\}$$

depicted in Figure 3.1, and one for the language ”all strings with an even number of *a*’s in Figure 3.2.

As mentioned, there are many well-known and well-studied modifications of the basic form of a finite state automaton,[2] such as augmenting the processing with an unbounded stack (yielding push-down automata), or an unbounded read-write tape

[2] Finite automata are perhaps more correctly described as being derived by restricting the more general Turing machines, which was the initially defined machine given in [Tur37].

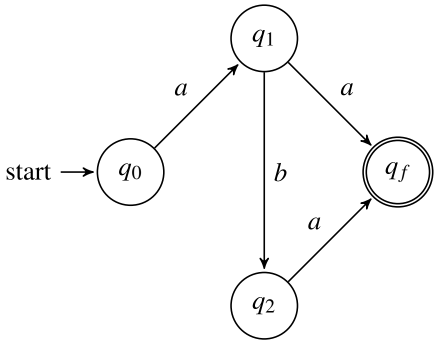

Figure 3.1: An automaton for the language $\{"a","aa","aba"\}$.

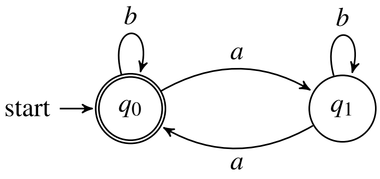

Figure 3.2: An automaton for the language of all strings over $\{a,b\}$ with an even number of a's

(yielding Turing machines), or restricting it, for example by forbidding loops (restricting the use to finite languages). Another common extension is to allow the automaton to write to an *output* tape, yielding string-to-string *transducers*

Another restriction which is more relevant to this thesis is to require the transition function be single-valued, i.e. instead of yielding a set of states, it yields a single state. The definition of runs is amended such that instead of requiring that $q_{i+1} \in \delta(q_i, c)$, we require that $q_{i+1} = \delta(q_i, c)$. Such an automaton is called *deterministic*, while ones defined as above are *nondeterministic*. Both recognise exactly the regular languages, but deterministic automata may require exponentially many more states to do so. Additionally, while nondeterministic automata may have several runs on the same string, deterministic automata have (at most) one.

3.2 Grammars

Finite automata is a formalism that works by recognising a given string, and regular expressions are well-specified descriptions of a languages. Grammars, in contrast, are *generative* descriptions of a languages. In general, grammars work by some kind of *replacement* of *nonterminals*, successively generating the next configuration, and ending up with a finished string of *terminals*. Though, again, many variants exist, and the practise of string replacement has a long history, the most studied and used type of grammar is the *context-free grammar*.

Context-free grammar A *context-free grammar* is a structure $G = (\Sigma, N, S, P)$ where

- Σ and N are finite alphabets of *terminals* and *nonterminals*, respectively
- $S \in N$ is the *initial nonterminal*, and
- P is a set of *productions*, on the form $A \to w$ where $A \in N$ and $w \in (\Sigma \cup N)^*$.

Intuitively, for a grammar, such as the context-free, we employ replacement by taking a string uAv and applying a production rule $A \rightarrow w$, replacing the left-hand side by the right-hand side, obtaining the new string uwv. The language of a grammar is the set of terminal strings that can be obtained by starting with the string S containing only the initial nonterminal, and then applying production rules until only terminal symbols remain.

Analogously to automata, we may call any string over $\Sigma \cup N$ a *configuration*. A pair w_i, w_{i+1} of configurations such that w_{i+1} is obtained from w_i by applying a production rule is called a *derivation step*. A sequence of configurations $w_0, w_1 \ldots, w_k$, starting with the string $w_0 = S$ containing only the initial nonterminal and ending with a string $w_k = w$ over Σ, where each pair w_i, w_{i+1} is a derivation step is a *derivation*.

While the context-free grammars are the most well-known and well-studied of the grammar formalisms, they do *not* correspond directly to regular expressions and finite automata in terms of computational capacity. That is, there are languages that can be defined by context-free grammars that no finite automaton recognises. Instead, CFG correspond to *push-down automata*, that is automata that have been augmented with a stack which can be written to and read from as part of the computation.

The grammar formalism that generates regular languages is named, unsurprisingly the *regular grammars*, and is defined in the same manner as the context-free, except that the strings on the right-hand side of productions are required to consist only of terminal symbols, with the exception of the *last* symbol. Without sacrificing any expressive power, we can even restrict the right-hand sides to be exactly aA where a is a terminal symbol and A is an optional nonterminal. If all rules in a regular grammars are on this form, we say that it is on *normal form*. Compare this to finite automata, where we start processing in one end and continue to the other, processing one symbol after the other.

The *parsing problem* for a certain class of grammars is the task of finding, given a grammar and a string, one or more derivation of the grammar that results in the string, if any exists. For regular grammars, it amounts, essentially, to restructuring the grammar into an automaton and running it on the string.

For a well-written introduction to many topics in theoretical computing science, and in particular grammars and automata on strings, see Sipser [Sip06].

3.3 Logic

Though the connection between language and logic is mostly thought of as concerning the semantic meaning of statements and analysing self-contradictions and implications in arguments, there is also a connection to formal languages. Let us first fix some notation for how to express logic.

We use $\neg$ for logical inversion, in addition to the standard logical binary connectives: $\wedge, \vee, \Leftrightarrow, \rightarrow$, as in Table 3.1

We can form logical *formulae* by combining facts (or *atoms*), with logical connectives, for example claiming that “It is raining” $\rightarrow$ “the pavement is wet”, or “This is an ex-parrot” $\wedge$ “it has ceased to be”. Each of these facts can be true or false, and

A	B	$A \wedge B$	$A \vee B$	$A \Leftrightarrow B$	$A \to B$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	F	T
F	F	F	F	T	T

Table 3.1: Standard logical connectives, T and F stands for "true" and "false", respectively

the truth of a formula is most often dependent on the truth of its component atoms. However, if we have some fact P, then the truth of the statements $P \vee \neg P$ and $P \wedge \neg P$ are both independent of P. We call the first formula (which is always true) a *tautology*, and the second (which is always false) a *contradiction*. Much has been written on this so-called *propositional logic*, both as a tool for clarifying arguments and as formal methods of proving properties of systems in various contexts.

We can be more precise about facts. Let us first fix a *domain* – a set of things that we can make statements about, such as "every human who ever lived", "the set of natural numbers", or "all strings over the alphabet Σ". We can then define subsets of the domain for which a certain fact is true, such as `even`, which is true of every even natural number, `dead`, which is true of all dead people, or `empty`, which is true of the empty string. A fact, then is for example `dead(socrates)`, which is true, or `dead(petter)`, which as of this writing is false.

We can generalise this precise notion of facts to not just be claims about the properties of single objects, but claims of *relations*. For example, we could have a binary `father` relation, which would be true for a pair of objects where the first is the father of the second, or a trinary `concatenation` relation, which is true if the first argument is the concatenation of the two following, as in `concatenation(aaab, a, aab)`. The facts about properties discussed previously are simply monadic relations.

A domain together with a set of relations (a *vocabulary*) and their definitions on the domain of objects is a logical *structure*, or *model*, and *model checking* is the practise of taking a model and a formula and checking whether or not the model *satisfies* the formula – that is, if the formula is true, given the facts provided by the model.

Up until now, we have discussed only propositional, or *zeroth-order* logic. Let us introduce *variables* and *quantification*, to yield *first-order* logic: We add to our logical symbols an infinite set $X = \{x, y, z \dots\}$ of *variables*, disjoint from any domain, and the two symbols $\exists$ and $\forall$ that denote *existential* and *universal* quantification, respectively. Briefly, we write $\exists x\ \phi$ for some formula ϕ containing x to mean that "there exists some object x in the domain such that ϕ holds". Conversely, we write $\forall x\ \phi$ to mean "for *all* objects x in the domain, ϕ holds".

Quantification is thus a tool that allows us to express things like $\forall x\ \texttt{human}(x) \to (\texttt{alive}(x) \vee \texttt{dead}(x) \wedge \neg(\texttt{alive}(x) \wedge \texttt{dead}(x)))$, indicating that we have no vampires (who are neither dead nor alive) or zombies (who are both) in our domain, or at least that they do not count as human. Moreover, $\exists x\ \texttt{even}(x) \wedge \texttt{prime}(x)$ holds thanks to the number 2, assuming the domain is the set of natural numbers and `even` and `prime` are given reasonable definitions.

Second-order logic uses the same vocabulary as first-order, but allows us to use variables not only for objects in the domain, but also for *relations*. Restricting ourselves to quantification over relations of arity one yields *monadic second-order* (MSO) logic, which is a type of logic with deep connections to the concept of regularity as defined using regular grammars or finite automata, see e.g. [Büc60].

Let our domain be the set of positions of a string, and define the relations $\mathtt{lab}_a(x)$ for "the symbol at position x has the label a", and $\mathtt{succ}(x,y)$ for "the position y comes directly after the position x". With these predicates, each string s has, essentially, one single reasonable logical structure $\mathscr{S}_s$ that encodes it. We can then use this encoding to define languages of strings, using logical formulae, saying that the language $\mathscr{L}(\phi)$ of a formula ϕ over the vocabulary of strings over Σ (i.e. using facts only on the form $\mathtt{lab}_a(x)$ and $\mathtt{succ}(x,y)$ for $a \in \Sigma$) is the set of strings s such that $\mathscr{S}_s$ satisfies ϕ.

For example, if we let $\phi = \forall x\,(\neg\exists z\,(\mathtt{succ}(z,x)) \to \mathtt{lab}_{\mathtt{a}}(x))$, we capture all strings that start with an a in our language. We arrive at a new hierarchy of language classes, where first-order logic captures the *star-free* languages, and MSO logic captures the regular languages (Büchi's Theorem). The proof is somewhat technical, but for the direction of showing that all regular languages are MSO definable, it proceeds roughly as follows: We are going to assign each position of the string to a set, representing the various states that the automaton or grammar could have at that position. We do this by assigning the first position to the set representing the initial state, and then having formulae that represent transitions, checking that, for each pair of positions x,y, if $succ(x,y)$, then x is assigned to the proper set (say, $Q_q(x)$), that $\mathtt{lab}_a(x)$, and then claiming that y is assigned to the proper set (say, $Q_{q'}(y)$). By finally checking whether or not the final position of the string belongs to any of the sets representing final states, we can ensure that the formula is only satisfied for structures $\mathscr{S}_s$ such that s is in the target language.

3.4 Composition and decomposition

We have thus far defined regularity in terms of automata, grammars and logic. We now introduce another perspective on regularity. First, let us define the universe of strings over an alphabet Σ, once again, this time algebraically. In short, the set of all strings over Σ is also named the *free monoid* over Σ, that is, the monoid with concatenation as the binary operation, and the empty string as the identity element.

We can moreover compose and decompose strings into *prefixes* and *suffixes*.

Given a string language L, we can define *prefix equivalence in relation to L* as the following: Two strings w and v are prefix equivalent in relation to L, denoted $\equiv_L$, if and only if for all strings $u \in \Sigma^*$, $uw \in L$ iff $uv \in L$. For each language, $\equiv_L$ is an equivalence relation on Σ^* – that is, it is a relation that is reflexive, symmetric and transitive. As such, it partitions Σ^* into equivalence classes, where all strings in an equivalence class are equivalent according to the relation. The number of equivalence classes for an equivalence relation is called its *index*, and we let the index of a language be the index of its prefix equivalence.

With these definitions in hand, we can define regular languages in a different way: The string languages with finite index are exactly the regular languages (Myhill-Nerode theorem) [Ner58].

3.5 Regularity, rationality and robustness

The regular languages have several other definitions and names in the literature, such as the *recognisable* languages, or the *rational languages*. What all these disparate definitions of regular languages have in common is that they are relatively simple, and for want of a better word, “natural”, and though they come from radically different contexts (viz. algebra, formal grammars/automata, logic), they all describe the *same* class of languages. This is a somewhat unusual, though highly desirable property called *robustness*.

3.6 Learning of regular languages

Using finite automata or MSO logic, we can determine whether or not a string (or set of strings) belongs to a specific regular language. With regular grammars or expressions, we can, for a given language, generate strings taken from that language. However, sometimes we have no such formal description of a language, but we *do* have some set of strings we know are in the language, and some set of strings that are not. If we wish to *infer* or *learn* the language from these examples, we say we are trying to solve the problem of *grammatical inference* or *grammar induction*.

There are many different variants of this problem, such as learning only from positive examples or from a positive and negative set, these set(s) of examples being finite or infinite, having some coverage guarantees of the (finite) sets of examples, or having more or less control over which examples are given. The learning paradigm relevant to this thesis is one where not only is this control rather fine-grained, but in fact usually envisioned as a *teacher*, having already complete knowledge of the target language. More specifically, we are interested in the *minimally adequate teacher* (MAT) model of Angluin [Ang87], where the teacher can answer two types of queries:

- **Membership:** Is this string a member of the language?
- **Equivalence:** Does this grammar implement the target language correctly?

Membership queries are easily answered, but equivalence queries require not only an up-or-down boolean response, but a *counterexample*, that is, a string that is *misclassified* by the submitted grammar.[3]

Briefly, using membership and equivalence queries, the learner builds up a set of representative strings for each equivalence class of the target language, and a set of

[3] There are variants of MAT learning that avoids this type of equivalence queries using various techniques or guarantees on an initial set of positive examples.

distinguishing suffixes (or prefixes), such that for any two representatives w_1, w_2, there is some suffix s such that either $w_1 s$ is in the language while $w_2 s$ is not, or vice versa.

While many learning paradigms are limited to some subset of the regular languages, MAT learning can identify any regular language using a polynomial number of queries (in the number of equivalence classes of the target language).

3.7 Weights

We can augment our grammars and automata with *weights*,[4] yielding regular string series, more well known as recognisable series. There is a rich theory in abstract algebra with many results and viewpoints concerning recognisable series, though most of these are not relevant for this thesis. Refer to [DKV09] for a thorough introduction to the subject. Briefly, we augment our grammars and automata with a *weight function*, which gives each transition or production a weight taken from some semiring. The weight of a run or derivation is the product of the weights of all its constituent transitions/productions, and the weight of a string, the *sum* of all its runs/derivations. Deterministic weighted grammars and automata are those where for each string there is only a single run or derivation of non-zero weight.

[4] There are several candidates for putting weights on logical characterisations, e.g. [DG07], but thus far no obviously superior one.

CHAPTER 4

Tree languages

Though useful in many contexts, regular string languages are generally not sufficiently expressive to model natural human language. For this, we require at minimum context-free languages.[1] An additional gain from moving to the context-free languages is the ability to give some structure to the generation of strings – we can for example say that a *sentence* consists of a *noun phrase* and a *verb phrase*, and encode this in a context-free grammar using the rule $S \to NP\ VP$, and then have further rules that specify what exactly constitutes a noun or verb phrase. That is, we can give *syntactic rules*, and have them correlate meaningfully to our formal model of the natural language.

Of course, with the move from regular to context-free languages we lose a number of desirable formal properties, such as MSO definability and closure under complement and intersection, as well as the very simple linear-time parsing achievable using finite automata.

However, though context-free production rules and derivations are more closely related to how we tend to think of natural language syntax, working exclusively with the output strings is not necessarily sufficient. Ideally, we would like to reason not only about the strings, but about the syntactic structures themselves. To this end, let us define a *tree* over an alphabet Σ as being either

- a symbol a in Σ, or
- a formal expression $a[t_1,\dots,t_k]$ where a is a symbol in Σ, and $t_1,\dots,t_k$ are trees

A specific tree such as $a[b,c[d]]$ can also be shown as in Figure 4.1. We call the position of a in this case the *top* or *root*, while b and d are *leaves* that together make up the *bottom*, or *frontier*. We say that a is the *parent* of its *direct subtrees* b and $c[d]$. Further, $a[b,c[d]], b, c[d]$ and d are all the *subtrees* of $a[b,c[d]]$. Note that each tree is a subtree of itself, and each leaf is a subtree as well.

The *paths* in a tree is the set of sequences that start at the root and then move from parent to direct subtree down to some node. In the example that would be the set $\{a, ab, ac, acd\}$, optionally including the empty sequence.

[1] There are some known structures in natural language that even context-free languages are insufficient for. As the more expressive context-sensitive languages are prohibitively computationally expensive, there is an active search among several candidates for an appropriate formalism of *mildly context-sensitive languages*. See my licentiate thesis [Eri17] for my contributions to that field.

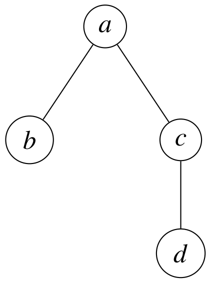

Figure 4.1: The tree $a[b,c[d]]$

4.1 Tree grammars

We can now let our CFG produce not strings, but trees, by changing each rule $A \to a_1 a_2 \dots a_k$ into $A \to A[a_1, a_2, \dots, a_k]$, with replacement working as in Figure 4.2. This gives us a *tree grammar* that instead of generating a language of strings generates a language of trees. In fact, these grammars are a slight restriction of *regular tree grammars*.[2] Unsurprisingly, many of the desirable properties that hold for regular string languages also hold for regular tree languages, but for technical reasons, this is easier to reason about using *ranked* trees, for which we require ranked alphabets:

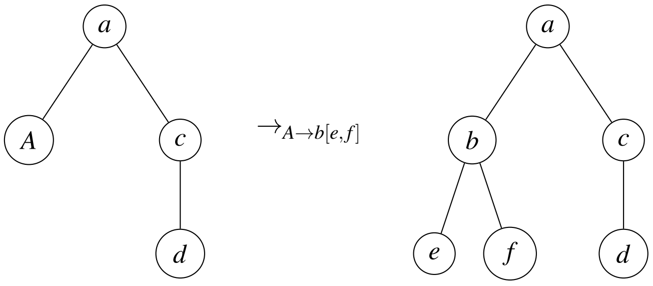

Figure 4.2: A tree replacement of A by $b[e,f]$.

Ranked alphabet A *ranked alphabet* $(\Sigma, rank)$ is an alphabet Σ together with a ranking function, $rank : \Sigma \to \mathbb{N}$, that gives a rank $rank(a)$ to each symbol in the alphabet. When clear from context, we identify $(\Sigma, rank)$ with Σ. We let Σ_k denote the largest subset of Σ such that $rank(a) = k$ for all $a \in \Sigma_k$, i.e. Σ_k is all the symbols $a \in \Sigma$ such that $rank(a) = k$.

We can now define the set T_Σ of ranked trees over the (ranked) alphabet Σ inductively as follows:

- $\Sigma_0 \in T_\Sigma$

[2] Specifically, over unranked trees.

- For $a \in \Sigma_k$ and $t_1 \ldots t_k \in T_\Sigma$, $a[t_1, \ldots, t_k] \in T_\Sigma$

We can immediately see that we can modify our previous CFG translations by letting all terminal symbols have rank 0 and creating several copies of each nonterminal A, one for each $k = |w|$ where $A \to w$ is a production. This also requires several copies of each production where A appears in the right-hand side. Though this may require an exponential number of new rules (in the maximum width of any right-hand side), the construction is relatively straightforward to prove correct and finite.

We now generalise our tree grammars slightly to obtain the regular tree grammars, as follows:

Regular tree grammar A *regular tree grammar* is a structure $G = (\Sigma, N, S, P)$ where

- Σ and N are finite ranked alphabets of *terminals* and *nonterminals*, respectively, where additionally $N = N_0$,
- $S \in N$ is the *initial nonterminal*, and
- P is a set of *productions*, on the form $A \to t$ where $A \in N$ and $t \in T_{(\Sigma \cup N)}$.

Now, the connection to context-free string grammars is obvious, but it is perhaps less obvious why these tree grammars are named *regular* rather than context-free. There are several ways of showing this, but let us stay in the realm of grammars for the moment.

Consider the way we would encode a string as a tree – likely we would let the first position be the root, and then construct a *monadic* tree, letting the string grow "downwards". The string $abcd$ would become the tree $a[b[c[d]]]$, and some configuration of a regular string grammar $abcA$ would become $a[b[c[A]]]$. Now, a configuration $abAc$ of a *context-free* grammar would, by the same token, be encoded as the tree $a[b[A[c]]]$, but note that the nonterminal A would need to have rank 1 for this to happen – something which is disallowed by the definition of regular tree grammars, where $N = N_0$. This correlates to the restriction that regular grammars have only a single nonterminal at the very end of all right-hand sides.[3]

Another way to illustrate the connection is through the *path languages* of a tree grammar – the string language that is defined by the set of paths of any tree in the language. For monadic trees, this would coincide with the string translation used in the previous paragraph, but even allowing for wider trees, this is regular for any regular tree grammar. Moreover, as implied in the above sketch, each regular string language is the path language of some regular tree grammar.

4.2 Automata

As in the string case, the connection between automata and grammars is relatively straightforward. First we restrict the grammar to be on normal form, which in the tree case means that the right-hand sides should all be on the form $a[A_1, \ldots, A_k]$ for $a \in \Sigma_k$ and $A_i \in N$ for all i.

[3] Context-free tree grammars, analogously, let nonterminals have any rank.

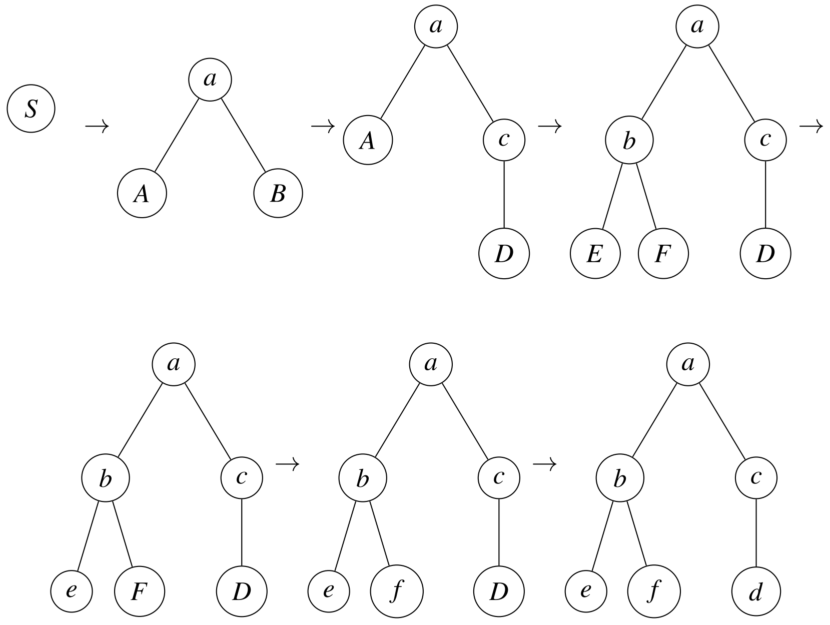

Figure 4.3: A derivation of a tree grammar.

Now, consider a derivation of a grammar on that form that results in a tree t, as in Figure 4.3. We wish to design a mechanism that, given the tree t, accepts or rejects it, based on similar principles as the grammar. Intuitively, we can either start at the bottom, and attempt to do a "backwards" generation, checking at each level if and how well a subtree matches a specific rule, or we start at the top and try to do a "generation" matching what we see in t. These are informal descriptions of *bottom-up* and *top-down finite tree automata*, respectively. Let us first formalise the former:

Bottom-up finite tree automaton A *bottom-up finite tree automaton* is a structure $A = (\Sigma, Q, F, \delta)$, where

- Σ is the ranked *input alphabet*
- Q is the set of *states*
- $F \subset Q$ is the set of *final states*, and
- $\delta : \bigcup_k (\Sigma_k \times Q^k) \to 2^Q$ is the *transition function*

In short, to compute the next state(s) working bottom-up through a symbol a of rank k, we take the k states $q_i, i \in \{1, 2, \ldots, k\}$ computed in its direct subtrees, and return $\delta(a, q_1, \ldots, q_k)$. Note that we have no initial state – instead the transition function for symbols of rank 0 (i.e. leaves) will take only the symbol as input and produce a set of states from only reading that. Compare this to the top-down case, defined next:

Top-down finite tree automaton A *top-down finite tree automaton* is a structure $A = (\Sigma, Q, q_0, \delta)$, where

- Σ is the ranked *input alphabet*
- Q is the set of *states*
- $q_0 \in Q$ is the *initial state*
- $\delta : \bigcup_k (\Sigma_k \times Q) \to 2^{Q^k}$ is the *transition function*

Here, we instead have an initial state q_0, but no *final states*, as the transition function, again for symbols of rank 0, will either be undefined (meaning no successful run could end thus), or go to the empty sequence of states λ.

Runs and derivations for tree automata naturally become more complex than for the string case, though configurations are, similar to the string case, simply trees over an extended alphabet with some restrictions. In particular, in derivations of regular string grammars there is at most one nonterminal symbol present in each configuration, making the next derivation step relatively obvious. For regular tree grammars, in contrast, there may in any given configuration be *several* nonterminals, any of which could be used for the next derivation step. Likewise, for string automata one simply proceeds along the string, while the computation proceeds in parallel in several different subtree in runs for both top-down and bottom-up tree automata. Let us for the moment ignore this difficulty, as for the unweighted case *the order is irrelevant*, and the run will be successful or not, the derivation result in the same tree *regardless* of what particular order we choose to compute or replace particular symbols in, as long as the computation or replacement is possible.

Bottom-up finite tree automata recognise exactly the class of regular tree languages in both its deterministic and nondeterministic modes, but for top-down finite tree automata, only the nondeterministic variant does so. This asymmetry comes from, essentially, the inability of top-down deterministic finite tree automata to define the language $\{f[a,b], f[b,a]\}$, as it can designate that it expects either an a in the left subtree and a b in the right, or vice versa, but not both at the same time without also expecting $f[a,a]$ or $f[b,b]$. For a slightly aged, but still excellent introduction to the basics of tree formalisms, including this and many other results, see [Eng75].

4.3 Logic

Logic over trees works in much the same way as logic over strings – we let the domain be the set of positions in the tree, and the relations be, essentially, the successor relation, but with several successors. More specifically, let the vocabulary for a tree over the ranked alphabet Σ with maximum rank k be, for each $a \in \Sigma$, the unary relation $\mathtt{lab}_{\mathtt{a}}$, and for each $i \in \{1, 2, \ldots, k\}$, the binary relation $\mathtt{succ}_i$.

It should come as no surprise that the set of MSO definable tree languages are exactly the regular tree languages [TW68]. As for the string case, the proof is quite technical, but is built on similar principles that apply equally well.

4.4 Composition and decomposition

Translating prefixes, suffixes and algebraic language descriptions to the tree case is slightly more complex than the relatively straightforward extension of grammars, automata and logic. In particular, while both prefixes and suffixes in the string case, for trees we must introduce the notion of *tree contexts*, which are trees "missing" some particular subtree. More formally, we introduce the symbol $\square$ of rank 0, disjoint from any particular alphabet Σ under discussion, and say that the set C_Σ of *contexts* over the alphabet Σ is the set of trees in $T_{\Sigma\cup\{\square\}}$ such that $\square$ occurs exactly once.

We can then define *tree concatenation* as the binary operation $concat : (C_\Sigma \times T_\Sigma) \to T_\Sigma$, written $s = concat(c,t) = c[t]$ where s is the tree obtained by replacing $\square$ in c by t. As for string languages, this gives us the necessary tools to define *context equivalence relative to L* where L is a tree language: trees t and s are context equivalent in relation to L if $c[t] \in L$ iff $c[s] \in L$ for all $c \in C_\Sigma$, written $s \equiv_L t$. As in the string case, the regular tree languages are exactly those for which $\equiv_L$ has finite index.[4]

4.5 Weights

Augmenting the transitions and productions of tree automata and grammars with weights taken from some semiring, and computing weights in a similar manner to the string case yields *recognisable tree series*.[5] In the case where a tree has a single run or derivation, the computation is simple - multiply all the weights of all the transitions/productions used – but if we have some tree $f[a,b]$ with a derivation

$$S \to f[A,B] \to f[a,B] \to f[a,b]$$

then we could also derive the same tree as

$$S \to f[A,B] \to f[A,b] \to f[a,b]$$

.

However, considering exactly the same derivation steps have been made, with no interaction between the reordered parts, it would seem silly to count these derivations as distinct in order to compute the weight of the tree.

We can formalise the distinction between necessarily different derivations and those that differ only by irrelevant reorderings by placing our derivation steps into a *derivation tree*. This is a special kind of ranked tree where the alphabet is the set of productions of a grammar, and the rank of a production $A \to t$ is the number ℓ of nonterminals in t. We call this the *arity* of the production. A derivation tree for a grammar $G = (\Sigma, N, S, P)$, then, is a tree over the ranked alphabet $(P, arity)$, where *arity* is the arity function that returns the arity of a rule. However, note that not *every* tree over this alphabet is a proper derivation trees, as we have more restrictions on derivations than just there being *a* nonterminal in the proper place.

[4] See [Koz92] for a quite fascinating history of, and accessible restatement and proof of this result.
[5] See, once again, [DKV09], specifically [FV09], for a more thorough examination of the topic.

Instead, a derivation tree is one where the labels of nonterminals is respected, in the sense that (i) the root is marked with some production that has the initial nonterminal S as its left-hand side, and (ii) for each subtree $(A \to t)[s_1, \ldots, s_\ell]$ such that the ℓ nonterminals in t are A_1 to A_ℓ we require that for each s_i, its root is $(A_i \to t_i)$ for $A_i \to t_i \in P$. The derivation tree of the example derivation would then be

$$(S \to f[A,B])[(A \to a),(B \to b)]$$

We then define the weight of a tree t according to a weighted regular tree grammar G to be sum of the weights of all its distinct derivation trees, where the weight of a derivation tree is the product of the weights of all its constituent productions.

CHAPTER 5

(Order-Preserving) Graph Languages

Shifting domains once again, let us discuss graphs and graph languages, with the background we have established for strings and trees. Graphs consist of *nodes* and *edges* that connect nodes. As such, graphs generalise trees, which in turn generalise strings. However, while the generalisation of regularity from strings to trees is relatively painless, several issues make the step to graphs significantly more difficult.

Let us first define what type of graphs we are focusing on. First of all, our graphs are *marked*, in the sense that we have a number of designated nodes that are *external* to the graph. Second, our graphs are *directed*, meaning that each edge orders its attached nodes in sequence. Third, our edges are not the standard edges that connect just a pair of nodes, but the more general *hyperedges*, that connect any (fixed) number of nodes. Fourth, our edges have *labels*, taken from some ranked alphabet, such that the rank of the label of each edge matches the number of attached nodes.

Thus, from now on, when we refer to graphs we refer to *marked, ranked, directed, edge-labelled hypergraphs*, and when we say edges, we generally refer to hyperedges.

Formally, we require some additional groundwork before being able to properly define our graphs. Let LAB, $\mathscr{V}$, and $\mathscr{E}$ be disjoint countably infinite supplies of labels, nodes, and edges, respectively. Moreover, for a set S, let $S^{\circledast}$ be the set of *non-repeating* strings over S, i.e. strings over S where each element in S occurs at most once. Let S^{+} be the set of strings over S excluding the empty string ε, and $S^{\oplus}$ be the same for non-repeating strings.

Marked, directed, edge-labelled hypergraph A *graph* over the ranked alphabet $\Sigma \subset LAB$ is a structure $g = (V, E, lab, att, ext)$ where

- $V \subset \mathscr{V}$ and $E \subset \mathscr{E}$ are the disjoint sets of *nodes* and *edges*, respectively
- $lab : E \to \Sigma$ is the *labelling*,
- $att : E \to V^{\oplus}$ is the *attachment*, with $rank(lab(e)) = |att(e)| - 1$ for all $e \in E$
- $ext : V^{\oplus}$ is the sequence of *external nodes*

Further, for $att(e) = vw$ with $v \in N$ and $w \in V^{\circledast}$ we write $src(e) = v$ and $tar = w$ and say that v is the *source* and w the sequence of *targets* of the edge e. This implies a directionality on edges from the source to the targets. Graphs, likewise have sources and targets, denoted as $v = \dot{g}$ and $w = g_{\cdot\cdot}$ for $ext = vw$. The rank $rank(x)$ of an edge or a graph x is the length of its sequence of targets. The *in-degree* of a node v is the size of the set $\{e : e \in E, v \in tar(e)\}$, and the *out-degree* the size of the set $\{e : e \in E, v = src(e)\}$. Nodes with out-degree 0 are leaves, and with in-degree 0, roots. We subscript the components and derived functions with the name of the graph they are part of, in cases where it may otherwise be unclear ($E_g, src_g(e)$ etc.).

Note that we in the above define edges to have *exactly* one source and any (fixed) number of targets. This particular form of graphs is convenient for representing trees and tree-like structures, as for example investigated by Habel et. al. in [HKP87]. Moreover, natural language applications tend to use trees or tree-like abstractions for many tasks.

Even with this somewhat specific form of graphs, however, we can immediately identify a number of issues that prevent us from defining a naturally “regular” formalism, most obviously the lack of a clear order of processing, both in the sense of “this comes before that, and can be processed in parallel”, and as in “this must be processed before that, in sequence”. The formalisms presented in this thesis solves both of these problems, but at the cost of imposing even further restrictions on the universe of graphs.

5.1 Hyperedge replacement grammars

We base our formalism on the well-known *hyperedge replacement grammars*, which are context-free grammars on graphs, where we, as in context-free string grammars, continuously replace atomic nonterminal items with larger structures until we arrive at an object that contains no more nonterminal items. While nonterminal items are single nonterminal symbols in the string case, now they are hyperedges labelled with nonterminal symbols. To this end, we partition our supply of labels LAB into countably infinite subsets LAB_T and LAB_N of terminal and nonterminal labels, respectively.

Hyperedge replacement grammar A *hyperedge replacement grammar (HRG)* is a structure $G = (\Sigma, N, S, P)$ where

- $\Sigma \subset LAB_T$ and $N \subset LAB_N$ are finite ranked alphabets of *terminals* and *nonterminals*, respectively
- $S \in N$ is the *initial nonterminal*, and
- P is a set of *productions*, on the form $A \to g$ where $A \in N$ and $rank(A) = rank(g)$.

Hyperedge replacement, written $g = h[\![e : f]\!]$, is exactly what it sounds like – given a graph h we replace a hyperedge $e \in E_h$ with a graph f, obtaining the new graph g by identifying the source and targets of the replaced edge with the source and targets of the graph we replace it with. See Figure 5.1 for an example. As in the string and tree cases, replacing one edge with a new graph fragment according to a production rule is a derivation step. The set of terminal graphs we can derive using a sequence of derivations steps starting with just the initial nonterminal is the language of the grammar. For a thorough treatment of hyperedge replacement and HRG, see [DHK97].

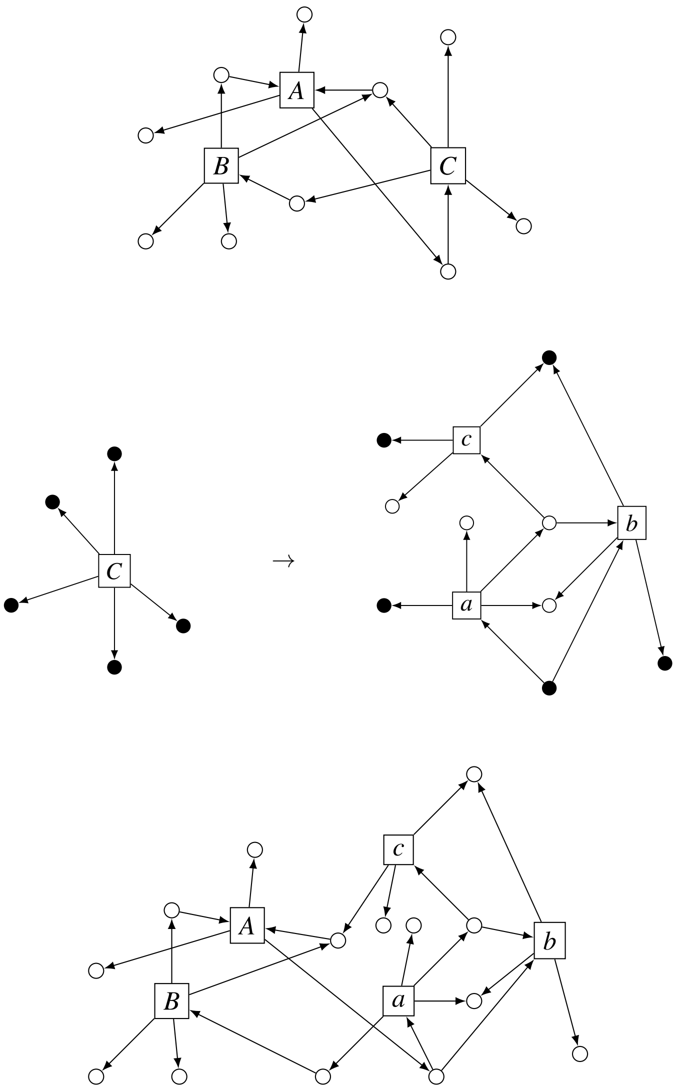

Figure 5.1: A hypergraph, replacement rule, and the result of applying the latter to the former

5.2 Issues of decomposition

A major difficulty in restricting HRG to something “regular” is, as mentioned, that there is no clear sense of “direction” in its derivations. Where strings go from one end to the other, and trees go from top to bottom (or bottom to top), we have no a priori obvious place in the right-hand sides of HRG to restrict our nonterminals to. Indeed, we have little sense of “position” and “direction” in graphs the first place, considering all the various connections that may occur. This is a major problem when it comes to efficient parsing of the graphs languages, as we would ideally like to be able to tell in “which end” to start processing, and how to choose a “next part” to continue with. Further, we would like these parts to be strictly nested, in the sense that we can immediately, from the structure of the graph, determine what larger subgraph of the graph the current subgraph is part of, in the same way that a specific subtree is not simultaneously a direct subtree of two different roots.

A natural first step in defining regular graph languages, then, is to simply decide that there is exactly one node that is the initial point from where everything starts generating, and that, moreover, the rest of the graph is at least *reachable* from that node. For us, the initial node is already clear – the source of the graph. Somewhat elided in the previous discussion on HRG, this remains the source of the graph even after replacing one of its nonterminals, and thus it is an ideal choice of a stable point.

In order to define reachability, let a *path* be a sequence $v_0, e_1, v_1, \ldots, e_k, v_k$ of nodes and edges such that for all $i \in \{1, 2, \ldots, k\}$, v_{i-1} is the source of e_i, and v_i is among the targets of e_i. We may optionally exclude the initial or the final node, or both. A path where $v_0 = v_k$ is a *cycle*, and path where v_0 is $\dot{g}$, a *source path*. Any node or edge on any path starting at a node v or edge e is *reachable from v (e)*. In a graph g with the source $\dot{g}$ we say that nodes or edges reachable from $\dot{g}$ are *reachable in g*. A graph g where $V_g \cup E_g$ are all reachable in g is *reachable*.

We now come to our first restriction on HRG, and on our universe of graphs – we require that all graphs are reachable.[1] Placing this restriction on all right-hand sides of our grammars ensures that all the graphs they generate will conform to this restriction as well (as the source of all nonterminals are reachable, so is the source of any graphs we replace them with, and these graphs in turn are all reachable from said source). Moreover, checking that an input graph has this property is easy.

The next restriction pertains to limiting the impact of a single nonterminal replacement. Consider a replacement like the one in Figure 5.2. From the final graph, there is no way to distinguish if the subgraphs having the external nodes as sources were part of the original graph, or if it was introduced in the replacement. For this reason, among others, we require that all targets of a right-hand side are leaves.

Together, these restriction provide an important property in that our grammars now are *path preserving* in the sense that if we have a replacement $g = h[\![e : f]\!]$ and two nodes $u, v \in V_h$, then u is reachable from v in g if and only if the same is true in h. Intuitively, this is because we no longer can introduce new paths between the targets of e, while all paths that pass $src_h(e)$ can still exit through the proper targets, as a result of us requiring f (and thus $f_{\bullet\bullet}$) to be reachable. Note that this property does *not*

[1] This also requires that the graphs are all connected.

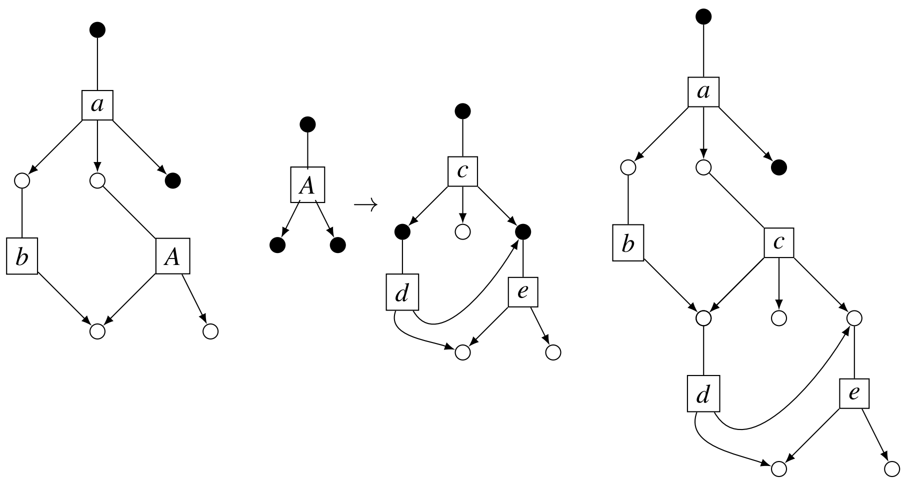

Figure 5.2: A hypergraph, replacement rule, and the result of applying the latter to the former

hold for all nodes in f, as there may be paths and cycles involving any combination of nodes of ext_f, including $\dot{f}$.

5.2.1 Reentrancies

We come now to a central concept in our formalisms – that of *reentrant nodes*. Briefly, the reentrant nodes of an edge or node x are the "first" nodes that can be reached from the root by passing x, but also using some path *avoiding* x. Let us make this more specific:

Reentrant nodes Let g be a graph. For $x \in V_g \cup E_g$, let $\hat{x}$ be x if $x \in V$, and $src_g(x)$ if $x \in E_g$. Moreover, let E_g^x be the set of all reachable edges e such that all source paths to e pass x, and let $TAR_g(E)$ for $E \subset E_g$ be the set of all targets of edges in E.

The set of *reentrant nodes of x in g* is

$$reent_g(x) = (TAR_g(E_g^x) \setminus \{\hat{x}\}) \cap (TAR_g(E_g \setminus E_g^x) \cup ext_g)$$

Looking at the final graph in Figure 5.2, the reentrant nodes of the root, and of the edge marked a, is simply the external node that is the third target of that same edge. For the edge marked e, only its leftmost target is reentrant, while for all other edges, their sets of reentrant nodes coincide with their targets.

We now introduce our third restriction on the grammars: That for every nonterminal edge e, $tar_g(e) = reent_g(e)$. The change from the previous is relatively small – this essentially means that all targets of nonterminals either have in-degree greater than 1, or are external nodes (or both). However, with all these restriction, our grammars are now *reentrancy preserving*,[2] in the following sense:

Reentrancy preservation Let $g = h[\![e : f]\!]$ be a replacement of a nonterminal e in h by f, resulting in g. The replacement is *reentrancy preserving* if, for all edges and nodes $x \in E_h \cup V_h$, $reent_g(x) = reent_h(x)$, and for all edges and nodes $x \in E_f \cup V_f \setminus ext_f$, $reent_g(x) = reent_f(x)$.

The precise claim is that if we have a grammar G with right-hand sides f conforming to the above restrictions, i.e.

- f is reachable
- All nodes in $f_{\bullet\bullet}$ are leaves
- For all nonterminal edges $e \in E_f$, $tar_f(e) = reent_f(e)$

Then all derivation steps of all derivations of G are reentrancy preserving. This is shown (with superficial differences) as Lemma 4.3 in Paper IV.

5.2.2 Subgraphs

Finally, with the above restrictions and concepts in place, we can define a hierarchy of subgraphs of each graph, delineated by each node and edge x and their reentrancies (see Lemma 3.4 of Paper IV). Moreover, with a reentrancy preserving grammar, we know that these subgraphs will be stable under derivation, and thus our parsing algorithm can assume that the reentrancies computed on the input will be useful throughout the parsing.

More formally, we let the *subgraph of g induced by x*, for $x \in V_g \cup E_g$ be the graph $g{\downarrow}_x = (E, V, lab, att, ext)$ where

- $E = E_g^x$
- $V = \{\hat{x}\} \cup TAR_g(E)$
- *lab* and *att* are the proper restrictions of lab_g and att_g to E, respectively
- *ext* is $\hat{x}$ followed by $reent_g(x)$ in some as of now unspecified order

Though a regular formalism is still some distance away, we now have an important set-piece: The ability to take an input graph and divide it into a tree-like hierarchy of subgraphs that can be processed one after the other, with clear lines between which graphs are direct subgraphs, and which are parallel lines of computation.

Let us look at a small example. In Figure 5.3 we have a small graph which has been generated by an OPDG. We have marked the "bottom-level" subgraphs using dashed, coloured lines. Note that these, as mentioned, can be found looking only at the structure of the input graph, with no reference to the grammar. The leftmost subgraph (indicated in green) has no reentrant nodes, and thus it must have come from a nonterminal of rank 0. In contrast the rightmost, indicated in purple, has two reentrant nodes, meaning that *it* must have come from a nonterminal of rank 2. Drawing in the "higher" subgraphs is left as an exercise for the reader.

[2] Note that this is not quite the same restrictions as used in Paper IV. Specifically, we lack restriction P2 of Definition 4.2, obviating the need for duplication rules, but this is not used in the relevant theorems and lemmas: Lemma 3.4, and Lemmas 4.3-4.5

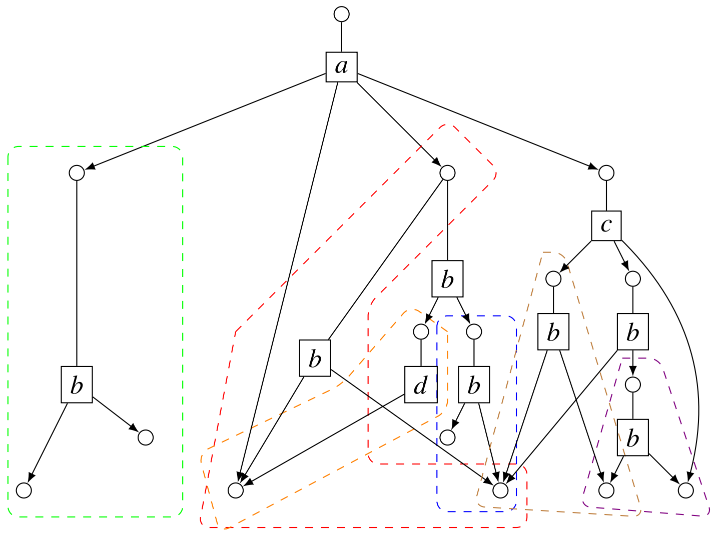

Figure 5.3: A graph with some of its subgraphs indicated.

5.3 Issues of order

An important consideration during the research that led to this thesis has been *efficient parsing*, rather than regularity itself. In particular, we have been searching for graph formalisms with efficient parsing in the *uniform* case. A thorough treatment of complexity theory is beyond the scope of this introduction,[3] but a major contributor to increased algorithmic complexity is the need to guess any particular node or edge, or even worse, a combination of nodes. Guessing the order of a set of nodes is another operation we wish to avoid, and in particular the need to remember an unbounded number of such guesses during the course of parsing.

Thus, while the hierarchy of subgraphs we have defined above is a *necessary* ingredient for efficient parsing, it is not yet *sufficient*.

In Paper IV, we investigate a set of requirements that lead to efficient parsing. One part is the reentrancy preservation that we have already discussed. The other is the property of *order preservation*. In the paper, this is a highly generalised set of requirements on the order itself (it needs to be efficiently computable, and order the targets of nonterminals according to the attachment), combined with a restriction on the rules that they preserve said order in a similar sense as the above on reentrancy preservation;

Order preservation Let $g = h[\![e : f]\!]$ be a replacement of a nonterminal e in h by f, resulting in g and let $\preceq_f, \preceq_h$ and $\preceq_g$ be ordering relations on the nodes of f, h and g, respectively. The replacement is *order preserving* if the restriction of $\preceq_g$ to V_f (V_h) is equal to $\preceq_f$ ($\preceq_h$).

[3] See e.g. [Pap03] for a useful textbook on the topic.

This is a slight restatement of Definition 4.7 of Paper IV. Let us now assume that our grammar is order preserving and reentrancy preserving. With structure and the order of nodes stable and discernible over derivations, we lack only a way to disambiguate or disregard the ordering of *edges*. In particular, while edges impose an order on their targets, no such ordering is imposed by a node on the edges that have it as their source. For this reason, we choose to disregard rather than disambiguate the order of edges, when it is ambiguous or undefined.

The restrictions that we now impose are thus intended to make sure that if we have some node with out-degree greater than 1, then we can parse it without referring to the order among the edges that are sourced there. We accomplish this by splitting our supply of rules into two sets: One where we limit the out-degrees of nodes to at most 1, and one a set of *duplication rules* with two edges, where nonterminal labels and attachments are required to be identical for both edges.[4] In Section 5.1 of Paper IV we show how duplication rules are parsed efficiently.

Finally, with all these restrictions, we have arrived at the formalism – Order-Preserving Hyperedge Replacement Grammars (OPHG) – that we present in Paper IV, and further investigate in Paper V. More properly, it is a *family* of formalisms, each with its own order. We show one example in Section 6 of Paper IV. It is still not known to be "regular", but does have efficient parsing in the uniform case.

5.4 A Regular Order-Preserving Graph Grammar

OPHG generalise an even more restricted type of graph grammar that was the initial object of study for this thesis – Order-preserving DAG grammars (OPDG). These can be seen as regular tree grammars augmented with (i) the ability to form connections through reentrant leaf nodes, and (ii) the possibility to create parallel structures through "cloning" a nonterminal.

In order to reason about the connections between OPDG and RTG, we need to be able to see trees as graphs. To this end, let us define a transformation from trees that yield graphs with the same structure in the following way: For a tree $t = a[t_1,\dots,t_k]$ we create a graph g with nodes v_a and v_{t_i} for each $i \in \{1,2\dots,k\}$, an edge e_a with $att_g(e_a) = v_a, v_{t_1},\dots,v_{t_k}$, and $lab(e_a) = a$. We then repeat the construction for each subtree in a similar way (substituting v_{t_i} for v_a where appropriate). See Figure 5.4 for an example. Note that when implementing a "regular tree grammar" as a HRG with the right-hand sides transformed in this way, the nonterminals will still appear only at the bottom of these graphs.

With this in place, we define our two extensions: First, we allow for *clone rules*, i.e. duplication rules where the two right-hand side nonterminals have the same symbol as the left-hand side. Second, we allow for leaves as targets, and moreover allow for leaves (but only leaves) with in-degree more than 1. In the style of OPHG, we require that the targets of nonterminals be reentrant, which in this context means that they must be leaves, and either external or the target of some other edges.

[4] We also require that for duplication rules $A \to f$ where $rank(A) = rank(f)$, that both edges in f have the label A.

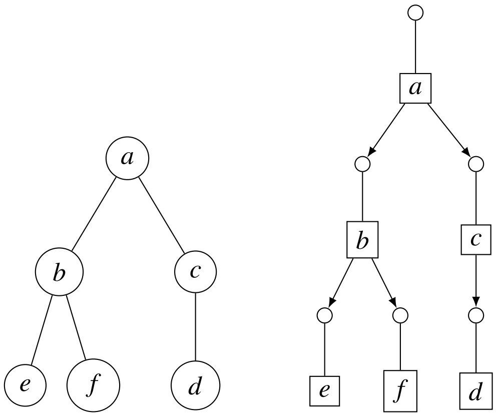

Figure 5.4: A tree and its translation into a graph.

Our grammars are still very close to regular tree grammars (with the exception of clone rules), and can be parsed strictly bottom-up in much the same fashion, as long as the order of leaves is accounted for. Let us look more closely at the way this is done in Papers I-III.

Closest common ancestor order If a node or edge x occurs on a source path ending at a node or edge y, we call x an *ancestor* of y, and y a *descendant* of x. If x is also an ancestor of z, it is a *common ancestor* of y and z. If no descendant of x is a common ancestor of y and z, then x is a *closest common ancestor*.

If x is an edge with targets $v_1,\dots,v_k$ and it is a closest common ancestor of y and z, then there is a node v_i that is an ancestor of y, and moreover, that for all other nodes v_j that are ancestors of y, that $i < j$. Likewise we have a node v_k that is the first of the targets that are ancestors of z. If $i < k$ we say that x *orders* y *before* z.

We say that a graph g is *ccae-ordered* if, for all pairs of leaves y, z, all common ancestor edges of y and z either order y before z or z before y. We write $y \preceq_g z$.

Essentially, what we require from our right-hand sides f is that they are ccae-ordered, and that moreover, if a target y of f comes before another target z, that $y \preceq_f z$. This ensures that leaves are not reordered (or made to be unordered) by hyperedge replacement, and that all derivation steps are order-preserving for ccae-ordering. In Paper I (Theorem 8), we establish that disregarding ordering constraints leads to intractable parsing in the uniform setting.

5.4.1 Normal forms

In Paper I, we also show that we can put all OPDG into *normal form*, reminiscent of regular tree grammars – a normal form rule is either a clone rule, or contains a single terminal edge. As nonterminals are positioned only above leaves, this results in a graph consisting of a single terminal and an optional “layer” of nonterminals, with three “layers” of nodes – the source, the targets of the terminal, and the leaves.[5]

This, together with the close relation to regular tree grammars leads us to call OPDG a regular formalism. In particular, by "flipping" the arrow of normal form rules, we get something closely resembling tree automata, whereas this is not obviously the case for OPHG.

5.4.2 Composition and decomposition

Though the above discussion seems to indicate some sort of regular properties for OPDG, they can, in contrast to regular string and tree grammars, not generate the full universe of graphs, or even the universe of connected graphs. Indeed, even the universe of acyclic, single-rooted, reachable graphs, with only leaves having in-degree 1, is strictly larger than the union of all OPDG languages, due to cloning and ordering constraints.

The proper definition of the universe of graphs that OPDG *can* generate is somewhat technical, and is given in two different ways in Papers II and III. Briefly, we define a set of typed concatenation operators, each essentially matching a normal-form rule. Their input is a number of graphs, each with rank equal to the matching "nonterminal" in the operator, and the output is simply the graph obtained by replacing each such "nonterminal" with the corresponding input graph. The base case is all the graphs consisting of a single terminal symbol, possibly with some of its targets marked. The universe of graphs is the set of graphs that can be constructed using such concatenation operators.

With the universe thus defined, we again take inspiration from the tree case to define "prefixes" and "suffixes" – The universe of graph contexts over a specific alphabet is, as in the tree case, graphs "missing" a subgraph, which we can express as follows: We add to the set of "base" graphs the graphs $\square_k$ for each k, which consist of a single $\square$-labelled edge with its source and k targets, all external. The concatenation that constructs a graph context uses exactly one of these graphs in its construction.

Graph concatenation is only defined if the graph being inserted into the context has the same rank as the $\square$ edge, but is otherwise similar to the tree case. This includes the opportunity to define equivalence relative to some language for graphs, which unsurprisingly has finite index for OPDG languages. In Paper II we establish this fact, and use it to prove that OPDG are MAT learnable.

It is not known at this time if OPHG languages are MAT learnable.

5.4.3 Logic

Logic over graphs has two distinct implementations in the literature,[6] with quantification either over both nodes and edges or just over nodes, with edges implemented as relations in the structure. Using the former, we can define an MSO formula that identifies the graphs that conform to our OPDG restrictions. We show in Paper III that OPDG languages are MSO definable, but, crucially, not the reverse. That is, we have not shown that for every MSO formula that picks out a subset of our universe of graphs, we have an OPDG that defines the same language.

[5] Though this is an imperfect picture, as a node may be both a leaf and a target of the terminal.

[6] See e.g. pioneering work by Courcelle [Cou90]

This is a weakness in our claim that OPDG is a regular formalism, and one that we conjecture can be rectified in a future article. Future articles may also investigate the logical characterisation, if any, of OPHG.

5.5 Weights

As with the step from string to tree formalisms, when taking the step from tree grammars to OPDG and OPHG in terms of introducing weights, much is familiar, but some new complications are introduced, in particular in terms of what derivations are supposed to be seen as distinct for the purpose of the weight computation. Recall that we moved from linear derivations to derivation *trees*, which are trees over the ranked alphabet of *productions* with some additional restrictions. Two derivation trees are distinct if their root label differs or if any of their subtrees differ. For non-duplication rules, we can simply use this same framework, but as we explore in Papers III and V, this leads to unwanted outcomes for duplication rules.

More specifically, as both edges on the right-hand side of a duplication have the same source, and the same label, we do not care about their identity (i.e. which is e_1 and which e_2) for the purposes of keeping derivations distinct. In Papers III and V we make precise what this means for the equivalence of derivation trees for OPDG and OPHG, respectively.

Let us take a deeper look at what we actually mean by two derivations being distinct, which we can use to inform our understanding of weighted grammars more generally. For some entirely generalised grammar – not necessarily working with either strings, trees or graphs, let p_1 and p_2 be two productions. We, as usual, say that two derivation steps are *independent* if $x \rightarrow_{p_1} x' \rightarrow_{p_2} y$ and $x \rightarrow_{p_2} x'' \rightarrow_{p_1} y$. But by the discussion above on duplication rules, this is not sufficient for determining if two derivations are distinct, as this would, for example, not count duplicated edges as equivalent for the purposes of derivations.

Now, let us consider two full derivations $d_1 = x_0 \rightarrow_{p_1} x_1 \rightarrow_{p_2} x_2 \ldots x_k$ and $d_2 = y_0 \rightarrow_{q_1} y_1 \rightarrow_{q_2} y_2 \ldots \rightarrow_{q_k} y_k$ for the purpose of how to distinguish them. Generally, our notion of independent derivation steps lets us consider d_1 and d_2 equivalent if the independent derivation steps of d_2 can be rearranged to produce $d_2' = y_0 \rightarrow_{q_1'} y_1' \rightarrow_{q_2'} y_2' \ldots \rightarrow_{q_k'} y_k$, where for each i, $y_i' = x_i$. If we instead change this requirement to be that for each i, y_i' is to *isomorphic* to x_i, our concept of distinct derivations more closely matches our intuition.

In particular, this new concept ensures that, for OPHG and OPDG derivations, which edge of a particular cloning we choose to apply further productions on is going to be irrelevant – the results will be isomorphic.

CHAPTER 6

Related work

OPDG and OPHG are far from the only attempts at defining or implementing grammars for working with semantic graphs in general, or even the specific one we have used to motivate our work – abstract meaning representations (AMR). Other parallel work has been looking at the specific theoretical properties we have attempted to achieve, such as polynomial parsing or a "natural" extension of the regular languages to graphs or DAGs. For semantic graphs, two major strains can be discerned in recent research; one focusing, like the present work, on hyperedge replacement and restrictions and extensions of same, and one focusing on various properties, restrictions and extensions of *DAG automata*.

6.1 Regular DAG languages

DAG automata define the regular DAG languages, and were originally introduced as straightforward generalisations of tree automata by Kamimura and Slutzki in [KS81]. They are usually defined as working on node-labelled directed acyclic graphs with node labels taken from a doubly ranked alphabet giving the in-degrees and out-degrees of the nodes. They have been further developed and investigated by Quernheim and Knight [QK12]. In [Chi+18], an investigation into the characteristics of regular DAG languages is made with an eye to semantic processing, with some encouraging, and some discouraging results. In particular, the parsing problem is shown to be NP-complete even in the non-uniform case, but a parsing algorithm is provided with relatively benign exponents.

Recent work by Vasiljeva, Gilroy and Lopez [VGL18] seems to indicate that DAG automata may be a suboptimal choice for weighted semantic models, if the intent is to have a probabilistic distribution. An interesting quirk of regular DAG languages is that they in general have regular path languages, but if the input graphs are required to be *single-rooted* this no longer holds. The specifics are explored in [BBD17]. See also [Dre+17] for a recent survey of results and open problems for DAG languages.

6.2 Unrestricted HRG

Major steps have been taken recently in making general HRG parsing more efficient in practise, see e.g. [Chi+13], and much seems to point to HRG being able to capture the structure of AMR in a reasonable way. However, numerous parameters in the input graphs – notably node degree – may contribute exponentially to the running time of the graph parsing. The same is also true of several grammar characteristics.

6.3 Regular Graph Grammars

In [Cou90; Cou91], Courcelle investigates several closely related classes of hypergraph languages – the languages of hyperedge replacement grammars (HRL),[1] the class of recognisable graph languages (REC), where the (context) equivalence relation has finite index, and the class of MSO definable graph languages (DEF). In particular, it is established that while DEF is properly included in REC, both are incomparable with HRL. Specifically, the languages that are recognisable but not in HRL are exactly the recognisable languages with unbounded treewidth.

The languages that are in the intersection of HRL and DEF are semantically defined as the *strongly context-free* graph languages, but a constructive definition is also given of a strongly context free class of graph languages: the *regular graph grammars*. This is reintroduced in [Gil+17] for use as an AMR formalism, and a parsing algorithm with slightly better, though still exponential complexity than for general HRG is given in [GLM17].

6.4 Predictive Top-Down HRG

Another restriction of HRG to obtain more favourable parsing complexity is *predictive top-down parsing* [DHM15], which is analogous to SLL(1) parsing for strings and uses *look-ahead* to limit the amount of backtracking that might be required during the parsing of a graph. The specific effects on the possible graphs and grammars is somewhat complex, and unfortunately [Jon16] seems to indicate that they may not be favourable for use with AMR.

6.5 S-Graph Grammars

S-graph grammars [Kol15] are able to produce the same graph languages as HRG, but does so in a quite different way, using an initial regular tree grammar to generate derivation trees for a constructive algebra. They have been applied to the AMR graph parsing problem and compared favourably to the unrestricted HRG parsing mentioned above. See [GKT15] for details.

[1] In [Cou91] HRL are referred to as the context-free graph languages, CF.

6.6 Contextual Hyperedge Replacement Grammars

While we argue that HRG are too powerful mainly for parsing complexity reasons, Drewes and Jonsson argue in [DJ17] that *extending* HRG to *contextual* HRG [DH15], while resulting in worse parsing complexity in general, captures more exactly and succinctly the actual structures of AMR over a set of concepts, thus resulting in more efficient parsing in practise.

CHAPTER 7

OPDG in Practise

The papers included in this thesis are almost exclusively built on theoretical results. To explore if these also carries practical usefulness, we conducted some very basic experiments, described below. Briefly, graphs were taken from two separate domains – abstract meaning representations and modular synthesis – and ingested into a hypergraph context with minimal transformations. In particular, the graphs were minimally and reversibly made to conform to the general structure of OPDG graphs, while order was ignored. Then, the order was checked for consistency according to ccae-order. Likewise, the sets of reentrant nodes for edges sharing a source were checked for equality. If a graph passes all these checks, then it can be generated by an OPDG, and if a large proportion of the graphs encountered in a bank are of that form, then we conjecture that a useful OPDG grammar could be inferred or constructed. The proportion of graphs where the order or reentrant sets was inconsistent is given below.

7.1 Abstract Meaning Representations

As briefly mentioned in the previous chapter, the development of OPHG and OPDG was motivated by *Abstract Meaning Representations (AMR)* [Ban+13]. These represent the semantics of actual sentences using graphs on the form shown in Figure 7.1. The construction of AMR graphs from sentences is currently done by humans, in order to build a graph bank of known correct examples which can be used as input for training a graph grammar. There is a lengthy specification [Ban+18] of how this is to be accomplished, which can be seen as an example of the "complex, rule-based" approaches mentioned above. However, note that this specification is intended to be used by *humans*, not computers, and thus it includes certain judgements that would otherwise be hard to encode.

The AMR process is thus, in some sense, a combination of the data-driven and the rule-based approaches mentioned at the end of Chapter 2 – use complex, hand-written rules in combination with human domain knowledge to build a large data bank, which then is used to train a relatively simple model. This is not unprecedented in NLP, as many syntactic treebanks have been built on the same principle, and for a similar purpose. The approach is less widespread in the context of semantics, however, with major projects starting mainly in the last decade.

We can immediately see from Figure 7.1 that both OPHG, and in particular our

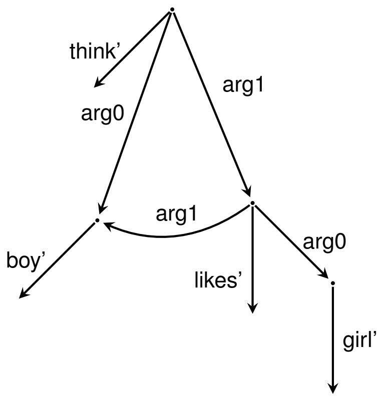

Figure 7.1: An AMR graph for "The boy thinks that the girl likes him"

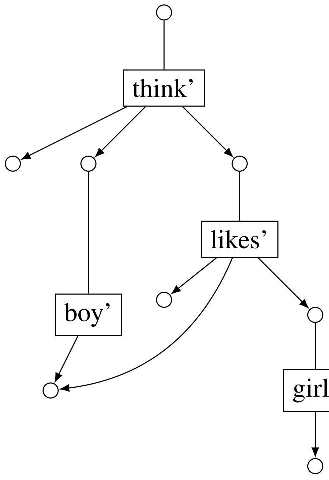

Figure 7.2: A "translation" of the AMR graph in Figure 7.1.

known learnable grammar OPDG may not be able to generate all AMR graphs (there are non-leaves with in-degree greater than 1, and no obvious control over the order). This is not ideal, as ostensibly, AMR is the motivation for their existence. However, let us take a closer look on the structure of an AMR.

We have a single root, and can assume that the graphs are acyclic. Each node has an identity, and at least one edge of rank 0 connected to it that shows what kind of thing or concept the node represents. There are also, depending on the type, a certain number of labelled edges pointing to arguments of whatever concept the node represents, and finally, potentially an unknown number of modifiers ("thinking deeply", is "think" modified with a "deeply" modifier). Our strategy, then, is to translate each such node into a hyperedge, labelled with the type, and with $k+2$ targets, where k is the number of arguments of the node. One of the extra nodes is used to collect all modifiers. For the case where we have a node of in-degree greater than 1, we choose one of its parents to be the primary one, and aim the other at the second extra target. In sum, the translation looks something like in Figure 7.2. Issues clearly remain, especially pertaining to order, but the experiments in Section 7.7 make this approach seem worthy of further study.

7.2 Modular synthesis

Briefly, a *modular synthesizer* is a synthesizer consisting of modules. That is, it is a mechanism for generating sounds and control signals from basic components such as sound sources, filters, sequencers, clock dividers, sample players, effects units, physical triggers, signal modifiers etc. Each module has a number of inputs and outputs, and these can be connected to each other in almost any configuration. The network that defines a specific sound (a *patch*) can be seen as a hypergraph, with the modules being the hyperedges, and the connection points being the nodes.

7.3 Typed Nodes

In modular synthesis it is important to consider what *type* a particular connection point has – broadly, either audio or control – and further that no more than one thing connected to said point is an *output*, whereas all other connections are *inputs*. Likewise, we also have semantic types and categories in AMR that carry important information, such as the object argument of an action concept actually being capable of that action.

Placing these typed nodes in a single well-specified order, we can reduce the potential reordering in a graph derivation (or parsing) to be only between nodes of the same type. As shown in Paper I, this does not, unfortunately, reduce our parsing problem from being NP-complete, but the intuition is that such reorderings are rare, or in the worst case, with properly chosen types, relatively meaningless.

7.4 Primary and Secondary References

Though our formalism has been generalised to handle graphs where reentrancies occur not only in the leaves, but anywhere in the graph (subject to reentrancy and order preservation), the more limited OPDG are still significantly easier to work with in several respects. As such, it is useful to have techniques to arrive at a leaf-reentrant DAG from a general rooted graph.

A simple way to do this is, if given a general rooted graph, for each source v of a hyperedge e with in-degree greater than one, we can designate one single edge as the *primary* referrer to e (and thus v). This done, we create a new target v' of e, and replace v by v' in all other edges targeting it. We call these others *secondary* referrers. With minor modifications (making no edge a primary referrer to the root), this can create a leaf-reentrant DAG from any rooted graph.

For AMR this is especially simple as, in the syntax of AMR, the nodes are initially given as nodes of a tree, giving the primary references directly, while the secondary references are given by further connections among nodes in the AMR. In modular synthesis graphs it is not as clear-cut which connection should be given the distinction of being the primary one.

7.5 Marbles

The software used for the following practical tests is originally a Tree Automata Workbench, written by the author in the course of completing the Master's Thesis [Eri12]. It is a modular system written in Scala with some useful abstractions and general conventions suitable for implementing and experimenting with automata and grammars on trees. Extending the functionality to the types of graphs discussed in this thesis proved relatively straightforward.

7.6 Modular Synthesis Graphs

Axoloti [Tae18] is an open source hard- and software project for modular synthesis, where modules are implemented with typed in- and outputs, mostly specified in XML files. Patches are also specified in XML files, and are thus relatively easy to parse into a manipulable form.

As an open and available dataset for modular synths patches, we have used the user-generated `contrib` [TC18] repository of the Axoloti project, containing about 560 complete patches, which, separating subpatches to separate graphs yields around 1400 graphs.

7.6.1 Translation

In our tests we have assumed that any module not connected to a global output can be disregarded, as these will not impact the final sound or behaviour of the patch. We also create a new root hyperedge that is connected to all global outputs. This results in a rooted graph, which we than transform into a leaf-reentrant DAG by the above construction, choosing the first occurring edge as the primary referrer.

7.7 Semantic Graphs

As described in Chapter 7.1 Abstract Meaning Representations are graphs representing the semantics of a single sentence, and the main motivation for the research included in this thesis. The specification of AMRs in its actual syntax is particularly well suited to our formalism in that each graph is essentially described as a tree of nodes with either a name and type, or a reference to a node defined elsewhere. The specific dataset we have tested with is the freely available Little Prince corpus [al18].

7.7.1 Translation

The tree structure of AMR mentioned above gives us the relations of primary referrals in a natural way, while any additional references are secondary. No further translations have been made. In particular, no ordering attempts have been made based on the types mentioned above.

7.8 Results

Due to time and resource constraints, the experiments are rather limited both in scope and rigour. As a shorthand for whether a specific input graph is 'covered' by the OPDG formalism, we have chosen to make a naive ingestion into the system, and then testing if the resulting graph is (i) rooted and reachable, and (ii) ordered. If the graph is reachable from the designated root (generally true by construction), and if there are no two edges ordering two nodes (leaves) in opposing ways, then the graph is on a structure that can be generated by an OPDG. We also checked that all edges sharing a source had the same set of reentrant nodes.

For the AMR Little Prince corpus, this fails only in 30 out of 1562 sentences. However, there is a natural explanation for this low number: In order for a graph to qualify as unordered there need to be two nodes that have two separate edges as closest common ancestors, and this occurs only in 80 graphs in the whole corpus. In a way, this is an encouraging finding, as the reordering of nodes is by far the most obvious difficulty in applying our grammars to real-world data. Thus, even though the naive ordering has obvious difficulties with ordering nodes consistently, it is not necessarily a major problems, as it is not something that comes into play very often, at least in this limited AMR corpus. Moreover, recall the translation given above, where we create a new target v' of each edge e. If we disregard e for the purposes of ordering inconsistencies involving v', the amount of unordered AMR graphs in the Little Prince corpus drops to a scant 12.

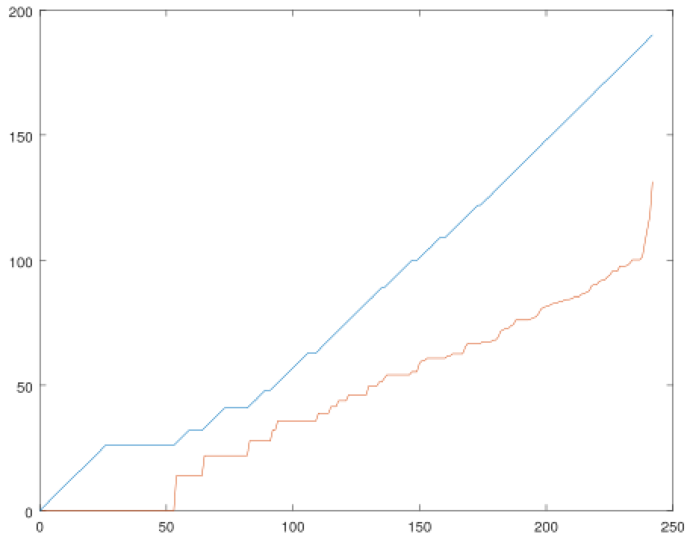

Figure 7.3: The cumulative sum of unordered graphs (blue), out of all potentially unordered modular synth graphs, ordered by number of potential unordered node pairs (red).

In the modular synthesis graph, ordering problems occur in 190 cases out of 1345 total tested graphs, that is, 14% of the graphs are unordered directly after the changes mentioned above. Here, only 242 graphs have of potential ordering problems, meaning that out of the potentially unordered graphs, around three fourths of them are actually unordered. This is likely due to there being no analogue to the tree structure in the specification of AMR, while the modules, in contrast, have a well-defined order on its connections in the Axoloti XML files already. As such, if one was to work more significantly with Axoloti graphs, it would be prudent to find some other way of ordering the nodes. A closer look at the statistics reveal further that, not unsurprisingly, graphs with few potential unorderings are less likely to be unordered than ones where many pairs of nodes have many shared closest common ancestor edges; see Figure 7.3, where the X axis represents the 242 graphs with potential reorderings, sorted by the amount of such reorderings. The blue line is the cumulative sum of unordered graphs. The red line is a scaled logarithm of the amount potentially unordered pairs of nodes in each graph. As can be plainly seen, almost every ordered graph is on the left side of the scale, i.e. has very few potential ordering problems.

There was no graph, in either corpus, where two edges sharing a source had different sets of reentrant nodes.

CHAPTER 8
Future work

The most immediate theoretical extension of the current work is to attempt to find corresponding proofs of “regular” properties for OPHG as already exists for OPDG, such as MSO definability, finite congruences for some reasonable definition, a Myhill-Nerode theorem, etc. For OPDG, similarly, a number of well-known results for regular tree languages should be relatively easy to prove for OPDG, such as various closedness and decidability results.

The most *interesting* direction to take for theoretical investigations may be to explore how well OPDG and OPHG match other semantic graph classes, as well as other graph representations. Another interesting topic is that of regularity for graphs, more generally – What *would* a regular formalism for graphs look like? Is it even possible to define, given the various difficulties we have encountered in doing so?

In terms of practical explorations, the first priority would be to properly implement both OPDG and OPHG into one or more systems suitable for testing. Second would be to do comparative testing against other already implemented systems for AMR parsing, and other graph parsing and generation problems.

Further practical explorations could focus on learning, either of weights in e.g. a expectation-minimisation model, or a practical implementation of a MAT oracle, either through human experts or some other model.

CHAPTER 9
Included Articles

9.1 Order-Preserving DAG Grammars

9.1.1 Paper I: Between a Rock and a Hard Place – Parsing for Hyperedge Replacement DAG Grammars

This paper is the first investigation into order-preservation, and thus uses various arguments, definitions and terminology that has been subsequently replaced by more elegant and descriptive terms. Nevertheless, the restrictions required for OPDG are introduced, together with a parsing algorithm. Additionally, a normal form is defined and proven to be relatively simple to construct for any given language. Furthermore, the restrictions of OPDG are motivated by giving several NP-completeness results for parsing variant grammars where one or more of the restrictions of OPDG are relaxed or removed. In particular, the ordering constraint is shown to be critical for polynomial uniform parsing.

9.1.2 Paper II: On the Regularity and Learnability of Ordered DAG Languages

Here we for the first time use the terminology of the rest of this thesis, and further develope the theory of OPDG. In particular, we define the universe of graphs in a way distinct from OPDG, and show that we have a Myhill-Nerode theorem for OPDG. This is used to instantiate an abstract MAT learner, which is proven correct using previous results. This algorithm is also a minimization algorithm for the deterministic unweighted case.

9.1.3 Paper III: Minimisation and Characterisation of Order-Preserving DAG Grammars

We deepen the connections between OPDG and regular tree grammars, showing their algebraic representations to be isomorphic. This is used to develop MAT learning, and thus minimisation, for OPDG in the weighted case. We also develop *concatenation schema*: a new way to conceive of the algebra generating the universe of graphs, as well as show OPDG to be MSO definable, giving additional credence to the idea of OPDG as a regular formalism. In addition, we replicate some results from Paper II using these new ideas.

9.2 Order-Preserving Hyperedge Replacement Grammars

9.2.1 Paper IV: Uniform Parsing for Hyperedge Replacement Grammars

The first paper on OPHG is also the first to properly identify the distinction between reentrancy preservation and order preservation, and to show the need for both for a uniformly polynomial parsing algorithm. Much of the paper is devoted to proving the nesting properties of reentrancies and subgraphs, and that OPHG are reentrancy preserving. The parsing algorithm is also defined and proven to run in polynomial time under certain conditions. We also give a concrete example of a suitable order, and additional restrictions under which OPHG preserve that order.

9.2.2 Paper V: Parsing Weighted Order-Preserving Hyperedge Replacement Grammars

The final paper builds mainly on results from Paper IV, introducing weights to the derivations of OPHG and making precise which derivations are to be seen as distinct for the purposes of computing weights for a graph. We also amend the parsing algorithm from Paper IV with weight computations and show it to be correct and remain efficient under the assumption that the weight operations are efficient.

9.3 Author's Contributions

Paper I: Conceptualization of formalism and parsing algorithm, translation of AMR

Paper II: Algebraic characterization, equality of graph classes, context definition

Paper III: Introducing weights, initial logical characterization

Paper IV: Generalisation parameters, separating reentrancy preservation from order preservation,

Paper V: Initial draft, derivation trees

CHAPTER 10
Articles not included in this Thesis

The papers described below were written during the course of the PhD program, though their content is significantly removed from the topic of order-preservation and graph grammars, and thus not included in this thesis.

10.1 Mildly Context-sentitive Languages

10.1.1 A Bottom-up Automaton for Tree Adjoining Languages

This technical report investigates the tree languages of the tree-adjoining grammars, in particular their path languages, and presents a bottom-up automaton for recognising them – essentially a regular bottom-up tree automaton augmented with an unbounded stack, and the opportunity to propagate the stack from any single subtree.

10.1.2 A Note on the Complexity of Deterministic Tree-walking Transducers

While tree adjoining languages are one of the weaker candidates for mildly context-sensitive languages, linear context-free rewriting systems (LCFRS) are one of the stronger. Here we investigate the precise parameterised complexities of LCFRS and an equally powerful formalism: deterministic tree-walking transducers. Unfortunately, the complexity results are for the most part negative.

Bibliography

[al18] Kevin Knight et al. *AMR Download page*. 2018. URL: `https://amr.isi.edu/download.html` (visited on 09/10/2018).

[Ang87] Dana Angluin. "Learning Regular Sets from Queries and Counterexamples". In: *Information and Computation* 75 (1987), pp. 87–106.

[Ban+13] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider. "Abstract Meaning Representation for Sembanking". In: *Proc. 7th Linguistic Annotation Workshop, ACL 2013*. 2013.

[Ban+18] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, Hermjakobj U., K. Knight, P. Koehn, M. Palmer, and N. Schneider. *Abstract Meaning Representation (AMR) 1.2.5 Specification*. 2018. URL: `https://github.com/amrisi/amr-guidelines/blob/master/amr.md` (visited on 12/27/2018).

[BBD17] Martin Berglund, Henrik Björklund, and Frank Drewes. "Single-Rooted DAGs in Regular DAG Languages: Parikh Image and Path Languages". In: *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms*. 2017, pp. 94–101.

[Büc60] J Richard Büchi. "Weak second-order arithmetic and finite automata". In: *Mathematical Logic Quarterly* 6.1-6 (1960), pp. 66–92.

[Chi+13] David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. "Parsing graphs with hyperedge replacement grammars". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2013, pp. 924–932.

[Chi+18] David Chiang, Frank Drewes, Daniel Gildea, Adam Lopez, and Giorgio Satta. "Weighted DAG automata for semantic graphs". In: *Computational Linguistics* 44.1 (2018), pp. 119–186.

[Cou90] Bruno Courcelle. "The monadic second-order logic of graphs. I. Recognizable sets of finite graphs". In: *Information and computation* 85.1 (1990), pp. 12–75.

[Cou91] Bruno Courcelle. "The monadic second-order logic of graphs V: On closing the gap between definability and recognizability". In: *Theoretical Computer Science* 80.2 (1991), pp. 153–202.

[DG07] Manfred Droste and Paul Gastin. “Weighted automata and weighted logics”. In: *Theoretical Computer Science* 380.1 (2007), p. 69.

[DH15] Frank Drewes and Berthold Hoffmann. “Contextual hyperedge replacement”. In: *Acta Informatica* 52.6 (2015), pp. 497–524.

[DHK97] Frank Drewes, Annegret Habel, and Hans-Jörg Kreowski. “Hyperedge Replacement Graph Grammars”. In: *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations*. Ed. by G. Rozenberg. World Scientific, 1997. Chap. 2, pp. 95–162.

[DHM15] Frank Drewes, Berthold Hoffmann, and Mark Minas. “Predictive top-down parsing for hyperedge replacement grammars”. In: *International Conference on Graph Transformation*. Springer. 2015, pp. 19–34.

[DJ17] Frank Drewes and Anna Jonsson. “Contextual Hyperedge Replacement Grammars for Abstract Meaning Representations”. In: *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms*. 2017, pp. 102–111.

[DKV09] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

[Dre+17] Frank Drewes et al. “On DAG languages and DAG transducers”. In: *Bulletin of EATCS* 1.121 (2017).

[Eng75] J Engelfriet. “Tree automata and tree grammars”. In: (1975).

[Eri12] Petter Ericson. *Prototyping the Tree Automata Workbench Marbles*. 2012.

[Eri17] Petter Ericson. “Complexity and expressiveness for formal structures in Natural Language Processing”. PhD thesis. Umeå Universitet, 2017.

[FV09] Zoltán Fülöp and Heiko Vogler. “Weighted tree automata and tree transducers”. In: *Handbook of Weighted Automata*. Springer, 2009, pp. 313–403.

[Gil+17] Sorcha Gilroy, Adam Lopez, Sebastian Maneth, and Pijus Simonaitis. “(Re)introducing Regular Graph Languages”. In: *Proceedings of the 15th Meeting on the Mathematics of Language*. 2017, pp. 100–113.

[GKT15] Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. “Graph parsing with s-graph grammars”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Vol. 1. 2015, pp. 1481–1490.

[GLM17] Sorcha Gilroy, Adam Lopez, and Sebastian Maneth. “Parsing Graphs with Regular Graph Grammars”. In: *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017)*. 2017, pp. 199–208.

[HKP87] Annegret Habel, Hans-Jörg Kreowski, and Detlef Plump. “Jungle evaluation”. In: *Workshop on the Specification of Abstract Data Types*. Springer. 1987, pp. 92–112.

[Jon16] Anna Jonsson. “Generation of abstract meaning representations by hyperedge replacement grammars–a case study”. MA thesis. Umeå University, 2016.

[Kle51] Stephen Cole Kleene. *Representation of events in nerve nets and finite automata*. Tech. rep. RAND PROJECT AIR FORCE SANTA MONICA CA, 1951.

[Kol15] Alexander Koller. “Semantic construction with graph grammars”. In: *Proceedings of the 11th International Conference on Computational Semantics*. 2015, pp. 228–238.

[Koz92] Dexter Kozen. “On the Myhill-Nerode theorem for trees”. In: *Bulletin of the EATCS* 47 (1992), pp. 170–173.

[KS81] Tsutomu Kamimura and Giora Slutzki. “Parallel and two-way automata on directed ordered acyclic graphs”. In: *Information and Control* 49.1 (1981), pp. 10–51.

[Ner58] Anil Nerode. “Linear automaton transformations”. In: *Proceedings of the American Mathematical Society* 9.4 (1958), pp. 541–544.

[NSS59] Allen Newell, John C Shaw, and Herbert A Simon. *Report on a general problem solving program*. Tech. rep. RAND PROJECT AIR FORCE SANTA MONICA CA, 1959.

[Pap03] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[QK12] Daniel Quernheim and Kevin Knight. “Towards probabilistic acceptors and transducers for feature structures”. In: *Proceedings of the Sixth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Association for Computational Linguistics. 2012, pp. 76–85.

[Sip06] Michael Sipser. *Introduction to the Theory of Computation*. Vol. 2. Thomson Course Technology Boston, 2006.

[Tae18] Johannes Taelman. *Axoloti website*. 2018. URL: `http://www.axoloti.com/` (visited on 12/29/2018).

[TC18] Johannes Taelman and Axoloti Contributors. *Axoloti contrib repository*. 2018. URL: `https://github.com/axoloti/axoloti-contrib` (visited on 09/10/2018).

[Tur37] Alan M Turing. “On computable numbers, with an application to the Entscheidungsproblem”. In: *Proceedings of the London mathematical society* 2.1 (1937), pp. 230–265.

[TW68] James W. Thatcher and Jesse B. Wright. “Generalized finite automata theory with an application to a decision problem of second-order logic”. In: *Mathematical systems theory* 2.1 (1968), pp. 57–81.

[VGL18] Ieva Vasiljeva, Sorcha Gilroy, and Adam Lopez. “The problem with probabilistic DAG automata for semantic graphs”. In: *arXiv preprint arXiv:1810.12266* (2018).

[Wil68] John Wilkins. *An Essay Towards a Real Character and a Philosophical Language*. 1668.