UMEÅ UNIVERSITY

Towards Semantic Language Processing

Anna Jonsson

Licentiate Thesis, December 2018
Department of Computing Science
Umeå University
Sweden

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

aj@cs.umu.se

ISBN 978-91-7601-964-1
ISSN 0348-0542
UMINF 18.12

Cover illustration by Lina Lidmark.
Printed by UmU Tryckservice, Umeå University, 2018.

Abstract

The overall goal of the field of *natural language processing* is to facilitate the communication between humans and computers, and to help humans with natural language problems such as translation. In this thesis, we focus on *semantic* language processing. Modelling semantics – the *meaning* of natural language – requires both a *structure* to hold the semantic information and a *device* that can enforce rules on the structure to ensure well-formed semantics while not being too computationally heavy. The devices used in natural language processing are preferably *weighted* to allow for comparison of the alternative semantic interpretations outputted by a device.

The structure employed here is the *abstract meaning representation (AMR)*. We show that AMRs representing well-formed semantics can be generated while leaving out AMRs that are not semantically well-formed. For this purpose, we use a type of graph grammar called *contextual hyperedge replacement grammar (CHRG)*. Moreover, we argue that a more well-known subclass of CHRG – the *hyperedge replacement grammar (HRG)* – is not powerful enough for AMR generation. This is due to the limitation of HRG when it comes to handling co-references, which in its turn depends on the fact that HRGs only generate graphs of *bounded treewidth*.

Furthermore, we also address the N *best problem*, which is as follows: Given a weighted device, return the N best (here: smallest-weighted, or more intuitively, smallest-errored) structures. Our goal is to solve the N best problem for devices capable of expressing sophisticated forms of semantic representations such as CHRGs. Here, however, we merely take a first step consisting in developing methods for solving the N best problem for *weighted tree automata* and some types of *weighted acyclic hypergraphs*.

Acknowledgements

Thanks to:

- ⋆ Johanna Björklund for being the best supervisor there is and for teaching me that failing is okay (as long as there is a party afterwards).
- ⋆ my co-supervisor Frank Drewes for being meticulous at all times and for always having a bad joke at hand.
- ⋆ my reference person Lars Karlsson for making sure everything is in order.
- ⋆ Andreas Maletti for hosting a research visit in Stuttgart, providing machine translation data and answering my questions.
- ⋆ Martin Berglund for never denying me thesis formatting help (and for always bringing tricky problems).
- ⋆ Adam Dahlgren for travelling with me, making sure that we arrived at CIAA 2018 on time, and for being my colleague and friend.
- ⋆ Henrik, Niklas, Petter and Suna the role-model for lunch (and dinner) company, seminars and problem-solving sessions.
- ⋆ Lili for laughing at my little dances in the fika room and for repeatedly answering my curious questions about her food with "it is a long story".
- ⋆ the rest of my colleagues at the department for helping me with all sorts of things, but most importantly for shared laughs.
- ⋆ my family – your support means everything.
- ⋆ Gustav, Lina, Matilda, Martin and William for being my second family.
- ⋆ Micke for all the much needed love.

Finally, I want to dedicate his thesis to my grandparents (and for their sake, I will do so in Swedish). Till farmor Vera som lärde mig att göra grimaser, till farfar Åke som visade mig hur man löser korsord, och till mormor Margit som lärde mig att inte bitas med orden "Stopp stopp, var och en biter sig själv!"

This project was funded by Vetenskapsrådet, DNR 2012-04555.

List of papers

This thesis is based on the following papers:

Paper I Frank Drewes and Anna Jonsson. Contextual Hyperedge Replacement Grammars for Abstract Meaning Representations. In *13th International Workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+13)*, 102–111, Association for Computational Linguistics, 2017.

Paper II Johanna Björklund, Frank Drewes and Anna Jonsson. Finding the N Best Vertices in an Infinite Weighted Hypergraph. In *Theoretical Computer Science*, volume 682, 30–41, Elsevier, 2017.

Paper III Johanna Björklund, Frank Drewes and Anna Jonsson. A Comparison of Two N-Best Extraction Methods for Weighted Tree Automata. In *23rd International Conference on Implementation and Applications of Automata (CIAA)*, 97–108, Springer, 2018.

List of contributions

Here the contributions of the thesis author to the papers is stated. Naturally, the contributions include reading, commenting on and approving the papers.

Paper I Writing of first draft (which we then developed together).

Paper II Programming of and experiments on software.

Paper III Performing, describing and giving the results of the experiments.

Contents

Chapter 1
Introduction

Humans communicate in *natural language* such as English or Swedish. When humans talk to each other, it is acceptable for us to make grammatical mistakes since we will likely still understand each other. In other words: the meaning of what we are saying – the *semantics*[1] – will still come across, regardless of the quality of the *syntax* (specific formulation). A computer, on the other hand, would deem the sentence useless unless the syntax is perfectly correct; it can only interpret language that follows pre-specified rules strictly.

The aim of the field of natural language processing is to make it possible for computers to communicate in and help humans with natural language (here, we limit ourselves to its written form). To do that, the computer must be able to extract the semantics even if the input sentence is slightly syntactically incorrect. Then it has to process the semantics to compute the desired result – also in the form of semantics. Finally, it should transform the result into a syntactically passable sentence and output it to the user. Thus, we need to handle both syntax and semantics when developing natural language processing tools.

As a starting point, we know that the strength of computers lies in their ability to store data and carry out formal directions (e.g. performing computations). We use this information to build language models for the computer. Such models normally contain the following two parts:

(1) a structure in which we can express the syntax *or* semantics of pieces of language (usually sentences), and

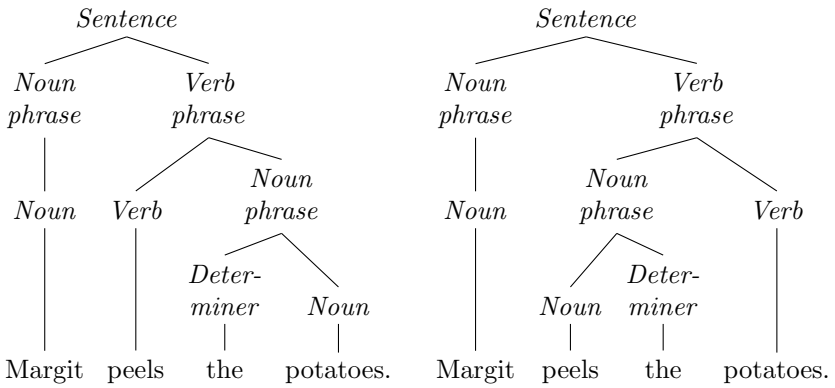
(2) a device that scores syntactic *or* semantic structures based on how well they follow a given set of rules, and outputs the ones that follow them well enough.

Let us start by looking at (1). The structure can for example simply be a string as in Figure 1.1(a) – in this case, it coincides with a sentence since a sentence is nothing but a string of natural language. It can also be a tree as seen in Figure 1.1(b) which contains more information about the sentence and is therefore more often used in natural language processing. A third option is to use a graph where the nodes hold the main concepts of the sentence and the edges are the relations given by the sentence (see Figure 1.1(d) for an example).

[1] We include pragmatics in this term since it is not helpful to separate them in this case.

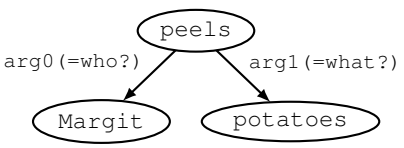
"Margit peels the potatoes."

(a) String representation (syntactic).



(b) Syntax tree representation.

(c) Incorrect syntax tree representation.



(d) Semantic graph representation.

Figure 1.1: Different representations of the same sentence.

The first two structures – strings and trees – are dependent on the syntax of the sentence and therefore said to be *syntactic representations*. In our example, the semantics might as well be expressed as "The potatoes are peeled by Margit" which would result in changed string and tree representations. The graph representation, however, would remain the same; it is therefore considered a *semantic representation*. When translating a sentence from one natural language to one or several others, it is valuable to have a structure that can represent the semantics regardless of syntax since the syntax changes between languages. When time comes to present the result to the user, however, a syntactic representation is needed. The ultimate goal of this research is to extract and process semantic structures. Important tasks are to use these structures efficiently in computations and to judge their quality. This is where (2) comes in.

Consider the tree in Figure 1.1(c). The sentence represented by the tree uses 'potatoes' as a verb and 'peels' as a noun, making it mean "Margit potatoes the peels". To humans, it is obvious that this is an example of incorrect natural language. How can we teach computers to differentiate between structures that represent correct and structures that represent incorrect natural language? Since computers are rule-followers, we use devices based on rules that

express desired properties of the structures. The devices considered here are *grammars* and *automata*. A grammar is a rule set that *generates* a structure. An automaton is defined by transition rules that *accept* a structure. Apart from using different mechanics, automata and grammars are interchangeable. The set of structures that can be generated by a grammar or accepted by an automaton is called a *formal language*, or *language* for short.

These devices are practical in the sense that they remove the necessity of saving entire languages in memory, allowing for infinitely large languages – and also for efficient processing. We can instead, given a structure, check whether or not the device generates or accepts it. This check is referred to as *parsing*. Unfortunately, depending on the expressiveness of the device, parsing can be a very difficult problem. An expressive device can handle more complex structures and rules, but the more expressive a device is, the harder is its parsing problem.

Both grammars and automata can be *weighted* in which case they provide an error score for the structure. Using the error score, we can compare different structures and thereby obtain a quality measure. For example, if we receive a set of translations in the form of an automaton, then we can pick the translation with the lowest weight. In this thesis, we only work with these two device types.

When we have both (1) and (2), we can model natural language using a formal language. Using the rules of the device, we can try to mimic the actual grammar of the natural language. The problem is that it is hard to find a device that is expressive enough whose parsing problem is solvable using a computer. Countless combinations of structures, devices and device expressibility have been explored, with varying results and usage areas. Yet another possibility is explored in this thesis.

Finding a good structure and device combination is not enough, however. We must also be able to process language in all sorts of ways: automatically changing between natural language representations, translating natural language in a computer (machine translation), extracting language from images and videos and put it into language representations, analysing natural language correctness and much more. Here, we focus on a very specific problem which is motivated by an application in machine translation.

One way to implement machine translation is to use *cascade evaluation* (see Figure 1.2). Cascade evaluation is when a problem can be solved in several steps where the output of one step is the input of the next. When the intermediate

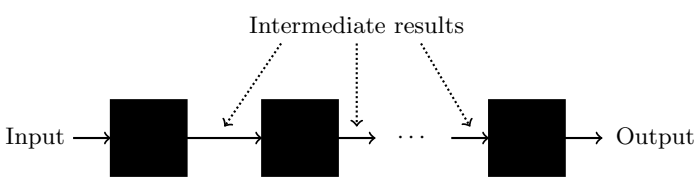


Figure 1.2: Cascade evaluation.

results are devices that describe infinite languages, we aggregate too much data to handle. This is the case for machine translation where we get a weighted tree automaton in each step. One approach to solving this is taking the result that is given the lowest error score by the device and propagate only that to the next step. This might not give us the optimal solution to the problem, but it is a good heuristics. An even better heuristics is achieved by propagating more that one result in each step. Finding the lowest-weighted structure is an easy problem for many devices whereas finding the $N > 1$ best structures is a harder one – the latter is a central topic of this thesis.

The outline of the remainder of this thesis follows: Chapter 2 lays out the theoretical background and Chapter 3 summarises the research results. My recommendation to the reader is to only skim Chapter 2, then move on to Chapter 3 and revisit Chapter 2 if necessary.

Chapter 2
Theoretical foundation

In the previous chapter, a number of concepts were introduced but not mathematically defined. Here, we provide the formal definitions and notations. These are for clarity divided into three sections: structures, devices and semirings.

2.1 Structures

We write $\mathbb{N}$ for the natural numbers, $\mathbb{N}_+$ for the natural numbers excluding zero and $\mathbb{R}^\infty$ for the non-negative real numbers including ∞. Moreover, let A be a set, and let $|A|$ be the number of elements it contains (its cardinality). Then, A^* is the set of finite sequences or strings over A, and $A^\circledast$ is the set of strings with no repeated elements over A. The power set of A, i.e., the set of all subsets of A, is denoted by 2^A. For simplicity, we let the symbol $\uplus$ denote disjoint union – the union of sets that have no common element. The domain of a function f is denoted by $\mathrm{dom}(f)$. Furthermore, we allow f to apply to sequences through the extension f^* defined as $f^*(a_0, \ldots, a_n) = f(a_0) \cdots f(a_n)$. A *labelling alphabet* is a finite set of symbols $\Sigma = \Sigma_V \uplus \Sigma_E \uplus \Sigma_N$ that has an arity function $\mathrm{arity} \colon \Sigma_E \uplus \Sigma_N \to 2^{\Sigma_V^*}$ defined on it. The arity function is only necessary in the definitions where it is explicitly used, but it is convenient to use the same alphabet definition in all cases. Elements that are labelled by symbols in Σ_N are called *nonterminals* and will have a special role in Section 2.2. For now, just consider them ordinary labels.

In Figure 1.1(d) we saw an example of a *labelled directed graph*, and although it is quite a self-explanatory structure, we will see its formal definition.

Definition 2.1 (Graph) A *labelled directed graph* (*graph*, for short) with labels from the labelling alphabet Σ is a tuple $G = (V, E, \mathrm{label}_V, \mathrm{label}_E)$ where

- V is a finite set of nodes.
- $E \subseteq \{(v, v') \mid v, v' \in V\}$ contains the edges, all directed from v to v'.
- the function $\mathrm{label}_V : V \to \Sigma_V$ labels the nodes.
- the function $\mathrm{label}_E : E \to \Sigma_E \uplus \Sigma_N$ labels the edges.

A *tree* is an *undirected graph* (see the above definition but disregard the edge directions) in which each node is only connected to any other node of the graph by a single path. One node is often assigned the *root* role which gives the tree a hierarchical structure of parents and children – the root is the only node that is never a child (though it can be a parent). In Figures 1.1(b) and 1.1(c), the roots of the trees are the nodes labelled 'Sentence'. For the interested, the formal definition of a tree used in Papers II and III follows.

Definition 2.2 (Tree) A *tree* labelled over a labelling alphabet Σ is a partial function $t\colon \mathbb{N}_+^* \to A$ such that the below conditions are fulfilled for $v \in \mathbb{N}_+^*$ and $i, j \in \mathbb{N}_+$ with $1 \leq j \leq i$.

- $\mathrm{dom}(t)$ is non-empty and finite.
- $vi \in \mathrm{dom}(t) \implies v \in \mathrm{dom}(t)$. (prefix-closedness)
- $vi \in \mathrm{dom}(t) \implies vj \in \mathrm{dom}(t)$. (left-closedness)
- $|\{i \mid vi \in \mathrm{dom}(t)\}| = \mathrm{arity}(t(v))$.

We call $v \in \mathrm{dom}(t)$ a *node* in t, and $vi \in \mathrm{dom}(t)$ *children* of v (v is their *parent*). The last condition states that a node can only be labelled with a symbol whose arity is equal to the number of children of the node.

Sometimes it is useful to measure how much a graph resembles a tree. For that, we use the measurement *treewidth*. In the case where the graph is in fact a tree, the treewidth of the graph is 1. The definition of treewidth below is the one used in [CDG+18].

Definition 2.3 (Treewidth) This definition is done in three steps. First, a *tree decomposition* of a graph $G = (V, E, \mathrm{label}_V, \mathrm{label}_E)$ is a tree t labelled over $A \subseteq 2^V$ such that

- $\forall v \in V$, $\exists v' \in \mathrm{dom}(t) : v \in t(v')$,
- $\forall (v_0, v_1) \in E$, $\exists v' \in \mathrm{dom}(t) : \{v_0, v_1\} \subseteq t(v')$, and
- $\forall v \in V$, the subgraph of t created using the nodes $v' \in \mathrm{dom}(t) : v \in t(v')$ and the edges between them is connected.

Secondly, the *width* of a tree decomposition t is given by

$$\max_{v' \in \mathrm{dom}(t)} (|t(v')| - 1).$$

Finally, the *treewidth* of a graph G is the minimal width over all of its tree decompositions.

The purpose of the -1 in the width definition is to make sure that the treewidth of a tree is 1. Moreover, the treewidth of a cycle is 2. (For those who are familiar with k-cliques: a k-clique has treewidth $k - 1$.)

Next, we will see a structure that is a generalisation of the graphs we saw above, namely *hypergraphs*. The only difference between the two is that the latter uses *hyperedges*. A hyperedge is an edge that can connect an arbitrary number of nodes, and a hypergraph is a graph with hyperedges. More formally:

Definition 2.4 (Hypergraph) A *labelled hypergraph* (*hypergraph*, for short) over a labelling alphabet $\Sigma = \Sigma_V \uplus \Sigma_E \uplus \Sigma_N$ is a tuple $G = (V, E, \mathrm{att}, \mathrm{label}_V, \mathrm{label}_E)$ such that all of the below criteria are fulfilled.

- V is a finite set of nodes.
- E is a finite set of hyperedges.
- $\mathrm{att}\colon E \to V^{\circledast}$ is the *attachment of hyperedges to nodes*.
- $\mathrm{label}_V\colon V \to \Sigma_V$ is the *labelling of nodes*.
- $\mathrm{label}_E\colon E \to \Sigma_E \uplus \Sigma_N$ with $\mathrm{label}_V^*(\mathrm{att}(e)) \in \mathrm{arity}(\mathrm{label}_E(e))$ for all $e \in E$ is the *labelling of hyperedges*.

The hypergraph definition intuitively says that hyperedges that are connected to n nodes can only be labelled with symbols whose arity contains a sequence of length n such that all of the labels of the n nodes are represented in the sequence. Note that hyperedges that are connected to exactly two nodes are equivalent to ordinary directed edges. In Paper II, we use a slightly different but equivalent hypergraph definition: each hyperedge has a target node and zero or more source nodes. This is only for our convenience as it allows us to easily specify a desirable property of the input graphs.

2.2 Devices

The *derivation* of a grammar (or *run* of an automaton) is a way of assigning a weight to a structure using the internal rules of the device (see Section 2.3 for how the cost of a run is computed). The weight assignment only works if the structure is a member of the language of the device. If the device is *deterministic*, each structure has only one corresponding run. If the device is on the other hand *nondeterministic*, there can be several runs on one single structure. Nondeterminism complicates computations using the devices, and devices that describe natural language often contain nondeterminism.

A device that is used to describe (hyper)graph languages is the *hyperedge replacement grammar* (see [DKH97] for a survey). Its definition follows.

Definition 2.5 (Hyperedge replacement grammar) A *(weighted) hyperedge replacement grammar (HRG)* is a tuple $\mathcal{G} = (\Sigma, \mathcal{R}, Z)$ where

- Σ is the labelling alphabet $\Sigma = \Sigma_V \uplus \Sigma_E \uplus \Sigma_N$,
- $\mathcal{R}$ is a set of *context-free rules*, and

- Z is a start hypergraph labelled over Σ.

A context-free rule has the form (L, R, w) where the *left-hand side* L is a connected hypergraph with exactly one hyperedge that must be a nonterminal, the *right-hand side* R is any supergraph of the hypergraph we obtain when removing the nonterminal from L, and w is the weight of the rule.

So how do they work? Say that you have a starting hypergraph Z which contains the nonterminal e. Moreover, say that you have a rule (L, R, w) for which the single nonterminal in L is e. Then you can apply the rule to Z with cost w by first removing e from Z and then inserting R in its place – the rule specifies exactly how the attachment of R is to be done. See Figure 2.1 for a rule example and Figure 2.2 for an application of the rule. The intuition is that we can apply these context-free rules to our hypergraph over and over again until no nonterminals remain, thereby *generating* an entirely new hypergraph that follows the rules of the grammar. The nonterminals can be seen as placeholders that are attached to the nodes that they want to be able to incorporate in future changes to the hypergraph.

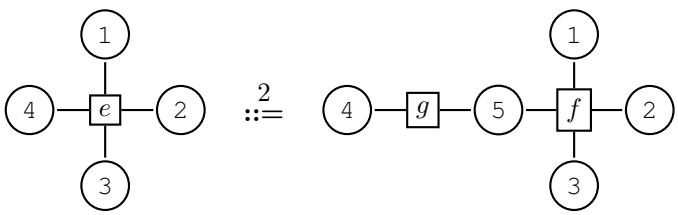


Figure 2.1: A context-free rule. The circles are nodes, the squares are nonterminals, and ::= separates the left-hand and the right-hand side. Above ::=, we see the weight of the rule, which is 2. Thus, the rule specifies a replacement of the nonterminal e with cost 2.

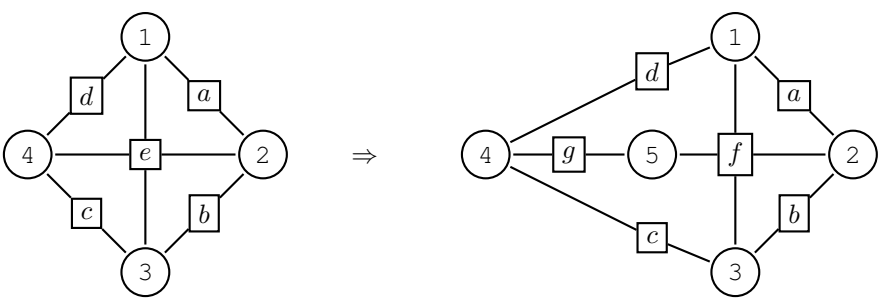


Figure 2.2: An application of the rule in Figure 2.1 to the leftmost hypergraph, resulting in the rightmost hypergraph. Note that the order of the attachment (1234) is preserved.

The next grammar type is a generalisation of HRG which allows changes to be made to parts of the hypergrah that are not connected to any nonterminal. This extended HRG is called *contextual HRG* [DH15], and its definition follows.

Definition 2.6 (Contextual hyperedge replacement grammar) A *(weighted) contextual hyperedge replacement grammar (CHRG)* is an HRG whose rules are *contextual rules*. A contextual rule is defined in the same way as a context-free rule, apart from the fact that the left-hand side L does not have to be connected. Thus, the rules of a CHRG are always contextual, but not all of them are context-free.

Intuitively, this expansion allows addition of hyperedges to nodes that have previously been released, i.e., nodes that no longer have any nonterminals connected to them. This generalisation turns out to be more suitable for semantic generation since it models the usage of e.g. pronouns better (see Section 3.1).

Until now, we have only seen graph grammars. Now, we will see a type of automata that describes tree languages (as defined in Papers II and III).

Definition 2.7 (Tree automaton) A *weighted tree automaton* (*tree automaton*, for short) is a tuple $\mathcal{M} = (Q, \Sigma, \mathcal{R}, Q_f)$ for which we have the following.

- Q is a finite set of *states*.
- Σ is a labelling alphabet disjoint with Q.
- $\mathcal{R}$ is a finite set of *transition rules* of the form $f[q_0, \ldots, q_{k-1}] \xrightarrow{w} q$ where $f \in \Sigma$, $\text{arity}(f) = k$, $q_0, \ldots, q_{k-1}, q \in Q$, and $w \in \mathbb{R}^\infty$.
- $Q_f \subseteq Q$ is a set of *final states*.

The rules of an automaton can be seen as consumers of symbols. For example, the rule $a[] \to q_0$ consumes the symbol a that does not have any children and puts the automaton in state q_0. If a has the parent f, which in its turn has no other children than a, then it is possible to apply the rule $f[q_0] \to q_1$ to consume f and put the automaton in state q_1. Similarly to the grammar case, we apply the rules repeatedly until no more rules can be applied – at the latest, this happens when the root of the tree is reached. If the automaton is in a final state when it has reached the root of the tree, then the automaton *accepts* (or *recognises*) the tree. (Not that the difference is that automata take structures as input whereas grammars generate structures.)

Both automata and grammars can be processed in the opposite direction: for a tree automaton, we can figure out what trees it accepts, and for a graph grammar, we can find out if a certain graph is generated by the grammar. Running automata in the usual direction and grammars in the opposite direction corresponds to *parsing*.

Definition 2.8 (Parsing) Let $\mathcal{D}$ be a device. Then $\mathcal{L}(\mathcal{D})$ denotes the language generated or accepted by the device. The *parsing* problem is solved by answering the question: Given a structure s of type S and a device $\mathcal{D}$ that describes structures of type S, does s belong to $\mathcal{L}(\mathcal{D})$?

At the beginning of this section, we briefly touched upon nondeterminism and that it makes things more difficult. For automata, we however have the option of *determinising* it. *Determinisation* is the process of taking a nondeterministic automaton and making it deterministic. This is convenient when you, e.g., want to use an algorithm for extracting the best runs to find the best trees, but not all automata can be determinised. Most often, however, they can be partly determinised as we go such that we only determinise the part of the automaton that we actually need – this is called *on-the-fly determinisation*.

2.3 Semirings

When working with weighted structures and devices, we have to define what their weights can be and how the weights should be combined in computations using the devices. For this, we normally use a *semiring*.

A semiring $\mathcal{A}$ is a non-empty set with operations defined on it. More formally, we have $\mathcal{A} = (A, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ where A is a set, $\oplus$ is an operator with $\mathbb{0}$ as its identity element, and $\otimes$ is an operator with $\mathbb{1}$ as its identity element. The two operators $\oplus$ and $\otimes$ are both associative, but only the first has to be commutative (so $a \oplus \mathbb{0} = \mathbb{0} \oplus a = a$ for every $a \in A$). If $\otimes$ is also commutative, we say that the semiring itself is commutative, and this is the case in all of the semirings we consider here. $\mathcal{A}$ is *idempotent* if $a \oplus a = a$ for all $a \in A$; this is the case if $\oplus$ is for example the min function. For a semiring $\mathcal{A}$ to be *finitely generated*, there has to be a finite subset B of A whose elements can form all of the elements of A by applying the operations $\oplus$ and $\otimes$ to them.

Then, how do we compute the weight of a structure using a device that is weighted over a semiring $\mathcal{A} = (A, \oplus, \otimes, \mathbb{0}, \mathbb{1})$? As we saw in the previous section, each rule of the device has a weight – a cost imposed on using the rule. To say that a device is weighted over $\mathcal{A}$ implies that all of its rules have weights from A. The weight of a structure is computed by applying $\otimes$ to all of the weights of rules used in a run on (or a derivation of) the structure, and then $\oplus$ is used to summarise over all runs. For example, let $\otimes = +$ and $\oplus = \min$. Then we only have to look at the smallest-weighted run on a structure to know its cost because of the min operator. Thus, the weight of the structure is simply the sum of the weights of the rules used in the smallest run on the structure. This is the case for the semiring we will see next.

The *tropical semiring* (used in e.g. N best trees extraction, see Section 3.2) is defined as $\mathcal{T} = (\mathbb{R}^\infty, \min, +, \infty, 0)$. Here we see that ∞ is the identity element of min and 0 is the identity element of $+$ since $\min(\infty, a) = \min(a, \infty) = a$ and $0 + a = a + 0 = a$ for all $a \in \mathbb{R}^\infty$. Moreover, min and $+$ are both commutative, thus $\mathcal{T}$ is commutative; $\mathcal{T}$ is also idempotent since $\min(a, a) = a$ for all $a \in \mathbb{R}^\infty$.

We can also define classes of semirings. A *nice* semiring is idempotent, finitely generated, and has $\mathbb{1}$ as its smallest element. This is the semiring class used for the N best nodes extraction in Paper II, as we will see in Section 3.2.

Chapter 3

Research questions and contributions

This chapter provides a summary of the papers in this thesis and outlines directions for future work. The summary is divided into the two main topics: semantic modelling and finding the N best structures given a weighted device.

3.1 Semantic modelling

To work with semantic models, we must be able to extract semantics from a natural language sentence and put it into a semantic model. There are many studies on this topic using various semantic representations combined with different approaches; some of them follow. In [DM18] an existing method based on neural networks for annotating a natural language sentence with syntactic information is extended to allow for the addition of semantic information to the sentence. Rule-based devices can also be used for this purpose: in [ALZ15] semantic information is extracted using grammars. An important key to training any model is the access to high-qualitative semantic data. There are many efforts to create more such data, one of which is [OKM+16]. Contributing to solving this problem is however outside the scope of this thesis. Here, we instead focus on semantic models and devices that can describe them.

In Chapter 1, we saw that graphs can be used to model semantics. To have a fully working model, however, we need to specify how to build the graphs: what concepts that are allowed and what their relations can be. An example of such a graph-based semantic representation is the *abstract meaning representation* (AMR) [BBC+13]. An AMR is a directed acyclic graph with node and edge labels that follows the AMR specification[1] – the nodes represent *concepts* and the edges give us the *relations* between the concepts. The graph in Figure 1.1(d) is an AMR consisting of the concepts `peels`, `Margit` and `potatoes`, and the relations `arg0` and `arg1` that point us to the agent and patient of a verb, respectively. Here, AMR is the semantic model of choice.

[1] `https://github.com/amrisi/amr-guidelines/blob/master/amr.md`

The next step is to find a device that can describe *complete* AMR languages over a pre-specified domain of concepts and relations. The term complete implies that the language described by the device should contain all of the AMRs over the domain that represent well-formed semantics (it should optimally also leave out the ones that represent non-well-formed semantics). The device should also have a polynomial-time-solvable parsing problem, otherwise it is not practical since many applications, e.g. machine learning, require repeated parsing of large numbers of structures.

A device that has been investigated for this purpose is hyperedge replacement grammar (HRG, see Definition 2.5) [CAB+13, DHM15, DHM17, BDE16]. The usage of hyperedges as nonterminals allows us to implement the control structures that reside in natural language. For example, the verb 'try' implies that the person who is doing the trying is the one who should do what is tried as well – "Margit tries to Vera peels the potatoes" makes no sense whereas "Margit tries to peel the potatoes" has valid semantics. In [CAB+13], the authors use the motivation of semantic models to develop a new and more efficient parsing algorithm for HRGs. The improved efficiency is based on the assumption that the graphs have bounded treewidth (see Definition 2.3), which in practice means that they cannot have too many cross-references. Even though a general solution cannot rely on bounded treewidth, putting a limit on the treewidth can be motivated by real-life AMR data sets [CDG+18] (the ones examined in this particular study have treewidth of at most four).

Moreover, predictive top-down parsing [DHM15] and restricted directed acyclic graph grammar parsing [BDE16] are both efficient parsing algorithms for subclasses of HRG – these are compared with respect to suitability for AMR parsing in [Jon16]. The result of the comparison is that none of the subclasses is suitable for AMR parsing in general and it is hypothesised that the reason is that their superclass HRG is in itself not powerful enough.

In Paper I, we study this question and conclude that the languages resulting from hyperedge replacement, as they are of bounded treewidth, are to narrow to handle coreferences resulting from using pronouns. Furthermore, we show that the contextual hyperedge replacement grammar (CHRG, see Definition 2.6) seems to be a good option for AMR generation. This is because CHRG provides the same mechanisms as HRG to control how concepts and relations are added while it has the option to add additional relations to already added nodes, rendering it unnecessary to keep track of all of the nodes already generated. In slightly more technical terms, we show that CHRGs are more suitable than HRGs for AMR generation by showing that they can generate complete AMR languages over a given domain of concepts and relations. In addition, if we have a language that can be described using both grammar types, the CHRG turns out to be both smaller and simpler than the HRG.

However, it remains to see if all AMR languages are parsable with respect to CHRGs and what time complexity the parsing has. Recent results indicate that at least a subclass of HRG is parsable [DHM17], but are the AMR languages contained in that subclass?

3.2 The *N* best problem

The N best problem is defined as follows: Given a device $\mathcal{D}$ weighted over a semiring $\mathcal{A}$, extract the N pairwise distinct elements from $\mathcal{L}(\mathcal{D})$ that are optimal with respect to $\mathcal{A}$. For example, if $\mathcal{A}$ is the tropical semiring (recall that its $\oplus$ operator is min), we want N structures of minimal weight (none of which is equivalent to any of the other structures).

Our long-term goal is to solve the N best problem for devices that, like CHRGs, can describe semantic languages. Then we will be able to extract, e.g., the N most likely meanings of a sentence. This far, we have however only worked with devices whose resulting structures are not typically used to represent semantics. We consider the work summarised here to be a step towards our long-term goal.

The motivation for solving the N best problem is, as previously mentioned in Chapter 1, handling infinite amounts of intermediate data in cascade evaluations (see e.g. [MKV10, Mal10, Mal11] for cascade evaluations in machine translation). In the case where the intermediate data consists of strings, the problem is solved by the algorithm Mohri and Riley present in [MR02] which uses on-the-fly determinisation on the input weighted string automaton (e.g. over the tropical semiring). This string algorithm was generalised to work for trees in [BDZ15], resulting in the algorithm Best Trees that finds the N best trees given a tree automaton (see Definition 2.7) weighted over the tropical semiring. To keep the efficiency high, Best Trees makes use of *pruning* which is to at every step only consider the trees that can possibly be part of the solution and discard the others. The implementation of Best Trees is available on GitHub.[2]

In Paper II, we compare the running time of Best Trees with its running time without the pruning; as expected, the pruning makes the algorithm much faster. However, we do not only want to make sure that the algorithm is efficient; we also want to compare it with other methods. Therefore, in Paper III, we compare Best Trees with the state of the art manner of finding the N best trees in practice: first finding the $k \geq N$ best runs and then filtering the list, removing duplicates of higher weight, until the N best trees remain (done in e.g. [Que17]). We call this method Filtered Runs. The k best runs are preferably found using the tree automata toolkit Tiburon [MK06] (that implements the algorithm in [HC05]), and removing duplicates can be done with a simple script. A weakness of Filtered Runs in the current setting is that k has to be guessed, and if Tiburon returns less than N trees, we need to increase k and run it all over again. Even though this can be mitigated using an implementation that outputs all of the runs until we stop it, the running time still depends on how many runs that have to be found before we have seen all N trees, which is hard to predict. This reasoning aligns well with the result of the comparison which shows that Best Trees is faster when the input automaton contains a high degree of nondeterminism but slower when there is less

[2] `https://github.com/tm11ajn/besttrees`

nondeterminism. Thus, Best Trees is both more predictable with respect to running time and better on more nondeterministic automata.

In Paper II, we also develop a generalisation of Best Trees that allows us to find the best nodes in a weighted acyclic hypergraph that can also be infinite. Thus, the hypergraph is the weighted device and the smallest-weighted nodes are what we want to extract. Recall that we use an alternative hypergraph definition here where each hyperedge has exactly one target node and any number of source nodes (a hyperedge is directed from the sources to the target). The weight of each node is computed by a function over the weights of the incoming hyperedges which in their turn depend on the source nodes of each hyperedge: i.e., the weight function is inductive. Solving the N best trees problem was done for weighted tree automata over the tropical semiring. Here, we do not only generalise with respect to the device, but with respect to the semiring as well: we compute the N best nodes of a hypergraph weighted over nice semirings. (Note that the tropical semiring is strictly speaking not nice but only idempotent with $\mathbb{1}$ as its smallest element. However, the part of the semiring that is actually used for computations within the automaton is finitely generated and thereby nice.) If we let each node of the hypergraph represent a tree, and the weight of the node be the weight of the tree, we have roughly the N best problem for weighted tree automata translated into the hypergraph generalisation. (This translation does however not take the states of the automaton into account.)

Currently, we are working on an extension of the best nodes algorithm that allows cycles. Moreover, we have developed a new version of Best Trees, and preliminary results show that it is clearly faster than the old one. The next step is to compare the new version with Tiburon on actual machine translation data resulting from [Que17]. Furthermore, we want to do the comparison not only for extracting the N best trees, but also for finding the N best runs to see whether or not Best Trees can compete with Tiburon on its home turf.

References

[ALZ15] Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. Broad-coverage CCG semantic parsing with AMR. In *Proc. of the Conference on Empirical Methods in Natural Language Processing*, pages 1699–1710, 2015.

[BBC+13] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proc. of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, 2013.

[BDE16] 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*, 2016.

[BDZ15] Johanna Björklund, Frank Drewes, and Niklas Zechner. An efficient best-trees algorithm for weighted tree automata over the tropical semiring. In *Proc. 9th Intl. Conf. on Language and Automata Theory and Applications*, volume 8977 of *LNCS*, pages 97–108, 2015.

[CAB+13] David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. Parsing graphs with hyperedge replacement grammars. In *Proc. of the 51st Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 924–932, 2013.

[CDG+18] David Chiang, Frank Drewes, Daniel Gildea, Adam Lopez, and Giorgio Satta. Weighted dag automata for semantic graphs. *Computational Linguistics*, 44(1):119–186, 2018.

[DH15] Frank Drewes and Berthold Hoffmann. Contextual hyperedge replacement. *Acta Informatica*, 52(6):497–524, 2015.

[DHM15] Frank Drewes, Berthold Hoffmann, and Mark Minas. Predictive top-down parsing for hyperedge replacement grammars. In *Proc. of the 8th International Conference on Graph Transformation*, 2015.

[DHM17] Frank Drewes, Berthold Hoffmann, and Mark Minas. Predictive shift-reduce parsing for hyperedge replacement grammars. In *Proc. 10th Intl. Conf. on Graph Transformation*, Lecture Notes in Computer Science, 2017.

[DKH97] Frank Drewes, Hans-Jörg Kreowski, and Annegret Habel. Hyperedge replacement graph grammars. In *Handbook of Graph Grammars and Computing by Graph Transformation*. 1997.

[DM18] Timothy Dozat and Christopher D Manning. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 484–490, 2018.

[HC05] Liang Huang and David Chiang. Better k-best parsing. In *Proc. of the Conference on Parsing Technology 2005*, pages 53–64. Association for Computational Linguistics, 2005.

[Jon16] Anna Jonsson. Generation of abstract meaning representations by hyperedge replacement grammars – a case study. Master's thesis, June 2016.

[Mal10] Andreas Maletti. Survey: Tree transducers in machine translation. In *NCMA*, pages 11–32, 2010.

[Mal11] Andreas Maletti. Survey: Weighted extended top-down tree transducers part II – application in machine translation. *Fundamenta Informaticae*, 112(2-3):239–261, 2011.

[MK06] Jonathan May and Kevin Knight. Tiburon: A weighted tree automata toolkit. In *International Conference on Implementation and Application of Automata*, pages 102–113. Springer, 2006.

[MKV10] Jonathan May, Kevin Knight, and Heiko Vogler. Efficient inference through cascades of weighted tree transducers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1058–1066, 2010.

[MR02] Mehryar Mohri and Michael Riley. An efficient algorithm for the n-best-strings problem. In *Proc. of the Conference on Spoken Language Processing*, 2002.

[OKM+16] Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Zdeňka Urešová. Towards comparability of linguistic graph banks for semantic parsing. In *10th International Conference on Language Resources and Evaluation*, pages 3991–3995, 2016.

[Que17] Daniel Quernheim. *Bimorphism Machine Translation*. PhD thesis, 2017.