Thesis for the Degree of Licentiate of Philosophy

Analysing Constraint Grammar with SAT

If you have a ~~Koen~~ SAT-solver, everything looks like a SAT-problem

Inari Listenmaa

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2016

Analysing Constraint Grammar with SAT
If you have a ~~Koen~~ SAT-solver, everything looks like a SAT-problem
INARI LISTENMAA

Technical Report TODO: ...L
ISSN 1652-876X

Research groups: Functional Programming / Language Technology
Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
SE-412 96 Gothenburg
Sweden
Telephone +46 (0)31-772 1000

Typeset in Palatino and Monaco by the author using XeTeX
Printed at Chalmers
Gothenburg, Sweden 2016

Abstract

Constraint Grammar (CG) is a relatively young formalism, born out of practical need for a robust and language-independent method for part-of-speech tagging. This thesis presents two contributions to the field of CG. We model CG as a Boolean satisfiability (SAT) problem, and describe an implementation using a SAT solver. This is attractive for several reasons: formal logic is well-studied, and serves as an abstract language to reason about the properties of CG.

As a practical application, we use SAT-CG to analyse existing CG grammars, taking inspiration from software verification.

Acknowledgements

You should do it because it solves a problem, not because your supervisor has a fetish for SAT.
– Koen Claessen, 2016

The greatest thanks go to my supervisors Koen Claessen and Aarne Ranta. You have guided this work from an afternoon experiment (where I probably procrastinated marking student labs) to an actual thesis. I have learnt a lot about research, **[TODO:** life**]** and SAT. Can't avoid that.

Thanks for Eckhard Bick for suggesting CG analysis as a research problem, and subsequently being my discussion leader. Your remark at the CG workshop in Vilnius has led to many fun discoveries! In addition, I want to thank Francis Tyers for providing the invaluable real-life CG-writer perspective and tirelessly answering my questions. Same goes for Anssi Yli-Jyrä, Tino Didriksen, Tommi Pirinen and other nice people on #hfst. On the other side of the rule-based NLP community, Pepijn Kokke has contributed to the notion of expressivity of CGs.

Finally, I want to thank all the awesome people I've met at Chalmers and outside! My time at the department has been fantastic—thanks to `[ x | x <- people, friend inari x]` and everyone else I'm forgetting!

Finally finally, here's a random anecdote. In the beginning of my PhD studies, someone suggested our project a tagline "SMT meets SMT". While I'm not quite doing Satisfiability Modulo Theories nor Statistical Machine Translation, I'd say the spirit is there.

This work has been carried out within the REMU project — Reliable Multilingual Digital Communication: Methods and Applications. The project is funded by the Swedish Research Council (*Vetenskapsrådet*) under grant number 2012-5746.

Contents

Chapter 1

Introduction to this thesis

I wish a wish.

How do you know that the first *wish* is a verb and the second one is a noun? A computer can approach this from many angles; below we list a few of them.

1. We can gather a set of example sentences and annotate them manually. Based on the examples, the computer will learn to guess the part of speech for new instances of *wish* with a certain probability, depending on what other words appear close to it.

2. We can write a generative grammar to describe the English language, and use it to analyse the sentence in question. The only well-formed solution has the first *wish* as a verb and the second as a noun: alternative hypotheses are rejected, because they do not fit in the structure.

3. We can split the problem in two phases: lookup and disambiguation. The lookup component has a dictionary with all inflection forms, so it knows that *wish* can be a noun or a verb, but it knows nothing about context. The disambiguation component completes the job: $wish_V$ for the first one, and $wish_N$ for the second.

Method number 3 is known as *Constraint Grammar* [16], abbreviated as *CG* throughout this thesis. CG is a formalism for writing disambiguation rules for an initially ambiguous input. The rules describe the impossibilities and improbabilities of a given language, such that after a successful run of the rules, only correct analyses remain.

How does the disambiguation work? In the spirit of approach 1, we can look at the context words for help. But we have more powerful tools, from the world of approach 2: grammatical categories and abstraction. Instead of learning facts about the strings *a* and *wish*, or *my* and *house*, we can formulate a *constraint rule*: "If a word can be a noun or a verb, and is preceded by a determiner, then remove the verb analysis". Add some hundreds of these rules, and they expose the structure hidden behind the clutter.

In the grand scheme of grammar formalisms, CG is lightweight, yet accurate and efficient. The rules are self-contained and mutually independent, each describing small pieces of complex phenomena. These features make the formalism fast, because the input sentence can be processed in local units. Furthermore, CG is highly robust: even if some parts of the sentence contain errors or unknown words, the rest of the sentence will still be processed. This makes it possible to assign *some* structure to *any* input. Finally, a grammar can always be made more accurate: if there is one single exception in the whole language, we can address only that exception in a new rule, without modifying any of the more general rules.

However, the same properties also make it challenging to manage the grammars. Due to the bits-and-pieces nature of CG rules, grammars tend to grow large; up to several thousands of rules. At the same time, there are no inbuilt mechanisms to detect if a new rule contradicts an older one[1]. From these two factors, it is natural that errors creep in without anyone noticing.

This thesis aims to create methods for analysing CG rules and detecting conflicts in grammars, as well as contribute to the theoretical understanding of the formalism and its various implementations. The main contributions are summarised in the following two sections.

Theoretical understanding of CG

> This explicitly reductionistic approach does not seem to have any obvious counterparts in the grammatical literature or in current formal syntactic theory.
> – Karlsson, 1995

Throughout its history, CG has been very practically oriented, with little studies on its formal properties, or attempts to relate it to existing frameworks. Notable exeptions are [20], who model CG in logic, and [24], who evaluate different finite-state based implementations. This brings us to the yet unmentioned part of the title. We model CG in the framework of *Boolean satisfiability*, abbreviated *SAT*. The analyses of the word forms are translated into Boolean variables, and the constraint rules into logical formulas operating on those variables.

[1] A rule may also contradict itself: e.g. "choose noun for all the words that are *not* potentially nouns".

Applying a rule becomes a task of assigning truth values to different analyses, such that ultimately each word should have one true analysis.

This simple translation gives us properties that standard CG implementations do not have. Most importantly, the rules lose their independence: any conflict between two rules renders the formula unsatisfiable. To counter that, we have implemented a novel conflict handling scheme, using maximisation. However, the loss of independence between rules is most useful for detecting contradictions in the grammar.

We have also investigated more theoretical properties of CG. Our SAT-encoding allows us to *generate* text from CG rules. We can construct a *maximally ambiguous sentence*: unlike any real sentence, every word starts off with every analysis, and the CG rules are responsible for shaping the sentence into a possible one. In addition, this setup lets us approximate the notion of *expressivity* within the framework of CG.

Analysis and quality control of CG

> Another desirable facility in the grammar development environment would be a mechanism for identifying pairs of rules that contradict each other.
> – Voutilainen, 2004

The most important contribution of this thesis is, however, practical. The SAT-encoding enables us to track the effect of each rule, and in case of a conflict, we can find out which rules cause it. In addition, we have found practical use for the generation property, to give grammarians feedback on the rules they are writing. For instance, a grammar writer can take a set of rules and ask for a sequence that triggers some of them and not others. In Chapter 4, we describe a method and a working implementation on grammar analysis, as well as evaluation on real grammars.

Structure of this thesis The core of this thesis is an extension of two articles: [21] and [22]. Some of the content has been updated since the initial publication; in addition, the implementation is described in much more detail. The thesis is structured as a stand-alone read; however, a reader who is familiar with the background, may well skip Chapter 2.

Chapter 2 presents a general introduction to both CG and SAT, aimed for a reader who is unfamiliar with the topics. Chapter 3 discusses previous logical representations of CG, and describes our SAT-encoding in detail, complete with an appendix of different rule types as SAT-clauses. Chapter 4 presents the method of grammar analysis using our SAT-based implementation, along with evaluation on three different grammars. Chapter 5 concludes the thesis.

Chapter 2

Background

In this chapter, we present the two components of this thesis: Constraint Grammar, and SAT.

2.1 Introduction to CG

Constraint Grammar (CG) is a formalism for disambiguating morphologically analysed text. It was first introduced by [16], and has been used for many tasks in computational linguistics, such as POS tagging, surface syntax and machine translation [5], **[TODO:** and is reported to achieve F-scores of around 9X % (citation needed) for various languages.**]** CG disambiguates output by morphological analyser by using constraint rules which can select or remove a potential analysis (called *reading*) for a target word, depending on the context words around it. Together these rules disambiguate the whole text.

In the example below, we show an initially ambiguous sentence "the bear sleeps". It contains three word forms, such as "<bear>", each followed by its *readings*. A reading contains one lemma, such as "bear", and a list of morphological tags, such as *noun sg*. A word form together with its readings is called a *cohort*. A cohort is ambiguous, if it contains more than one reading.

```
"<the>"
        "the" det def
"<bear>"
        "bear" noun sg
        "bear" verb pres
```

```
        "bear" verb inf
"<sleeps>"
        "sleep" noun pl
        "sleep" verb pres p3 sg
```

We can disambiguate this sentence with two rules:

1. `REMOVE verb IF (-1 det)` 'Remove verb after determiner'
2. `REMOVE noun IF (-1 noun)` 'Remove noun after noun'

Rule 1 matches the word *bear*: it is tagged as verb and is preceded by a determiner. The rule removes both verb readings from *bear*, leaving it with an unambiguous analysis `noun sg`. Rule 2 is applied to the word *sleeps*, and it removes the noun reading. The finished analysis is shown below:

```
"<the>"
        "the" det def
"<bear>"
        "bear" noun sg
"<sleeps>"
        "sleep" verb pres p3 sg
```

It is also possible to add syntactic tags and dependency structure within CG. After disambiguating *bear* by part of speech (noun), it remains ambiguous whether it is a subject, object, adverbial or any other possible syntactic role. Disambiguating the syntactic role can be done in several ways. One option is to get both syntactic and morphological analyses from the initial parser—the analysis for *bear* would look like the following:

```
"<bear>"
        "bear" noun sg          @Subj
        "bear" noun sg          @Obj
        "bear" verb pres        @Pred
        "bear" verb inf
```

Morphological disambiguation rules would resolve *bear* into a noun, and syntactic disambiguation rules would be applied later, to resolve *bear*-noun into subject or object.

Alternatively, one can have the initial parser return only morphological tags, and use CG rules that map a syntactic tag to a reading (or a reading to a cohort). These rules are usually run after the morphological disambiguation is finished. The following rules could be applied to the example sentence. The letter 'C' after the position means *careful* context: e.g. `(-1C noun)` requires the previous word to be unambiguously noun. Such rule would not fire if the previous word has any other reading in addition to the noun reading.

```
MAP @Pred IF (0C verb) (-1C noun)
MAP @Subj IF (0C noun) (1C verb)
```

With both of the strategies, we would end up with the following output:

```
"<the>"
        "the" det def
"<bear>"
        "bear" noun sg          @Subj
"<sleeps>"
        "sleep" verb pres p3 sg  @Pred
```

The syntactic tags can also indicate some dependency structure. For instance, *the* could get a tag such as `DN>`, to indicate that it is a determiner whose head is to the right. A phrase such as *the little dog* could get tagged as $the_{@DN>}$ $little_{@AN>}$ $dog_{@HEAD-N}$. Using `<TAG` and `TAG>` can identify chunks, as long as they are not disconnected.

With the introduction of CG-3, it is possible to add full-fledged dependency relations: number the words in the sentence and for each set a parent, such as `"<the>" "the" det def IND=1 PARENT=2`. However, these features are recent and not used in many of the grammars written in the community. In this introduction, we will illustrate the examples with the most basic operations, that is, disambiguating morphological tags. The syntactic operations are not fundamentally different from morphological: the rules describe an *operation* performed on a *target*, conditional on a *context*.

2.2 Related work

CG is one in the family of shallow and reductionist grammar formalisms. In this section, we briefly describe other formalisms of the same family, and provide a bit of historical context. In-depth details about CG are presented in Section 2.3.

Historical overview:

[TODO: Taggit (1970s or so)**]** — it's similar because it also looks at immediate context and discards analyses.

[TODO: Functional Dependency Grammar (1990s)**]** — it's similar because it also produces complex stuff from simple components.

[TODO: FSIG (1990s) – also called Parallel CG, more on that in the next section**]** — it's similar because it is basically the same thing, just parallel.

[TODO: All the other things that the standard CG-related papers/theses cite**]**

2.3 Properties of Constraint Grammar

[16] lists 24 design principles and describes related work at the time of writing. Here we summarise a set of main features, and relate CG to the developments in grammar formalism since the initial CG description.

All theories and implementations of CG a *reductionistic* system, designed primarily for analysis, not generation. Its task is to give correct analyses to the words in given sentences, not to describe a language as a collection of "all and only the grammatical sentences".

The syntax is decidedly *shallow*: the rules do not aim to describe all aspects of an abstract phenomenon such as noun phrase; rather, each rule describes bits and pieces with concrete conditions. The rules are self-contained and mutually independent—this makes it easy to add exceptions, and exceptions to exceptions, without changing the more general rules.

There are different takes on how *deterministic* the rules are. The current state-of-the-art CG parser VISL CG-3 executes the rules strictly based on the order of appearance, but there are other implementations that apply their own heuristics or remove the ordering completely, such as in finite-state or logic-based implementations.

In addition, we will discuss the *expressivity* of CG—while it is not a generative grammar and cannot be placed in the Chomsky hierarchy, we aim to provide parallels in what kind of output it can produce.

2.3.1 Reductionistic

CG is not a generative system. It starts from a set of alternative analyses, given by a morphological analyser, and eliminates the impossible or improbable ones using constraint rules. The remaining analyses are assumed to be correct; that is, everything that is not explicitly eliminated, is allowed. This kind of system is called *reductionistic*. It is contrasted with a *licencing* (or *generative*) system, where all constructions must be explicitly allowed, otherwise they are illegal. An empty reductionist grammar will accept any string, whereas an empty licencing grammar will accept no string.

According to the initial specification, CG is meant for analysis—[16] does not see it fit for generation:

> In practice, however, constraints are geared towards parsing, and Constraint Grammars are analysis grammars. It remains to be demonstrated that full-scale syntax can be done by one and the same reversible grammar.

Two decades later, [7] describe CG as "a declarative whole of contextual possibilities and impossibilities for a language or genre", which is nevertheless implemented in a low-level way: selecting and removing readings from individual words, without explicit connection between the rules. Bick and Didriksen argue that as a side effect, the rules actually manage to describe language.

We return to the questions of generation and declarativity in section 2.3.4. In chapter 4, we revisit these questions in the context of analysing CGs.

No enforcement of grammaticality CG does not define sequences as *grammatically correct*: it simply aims to add structure to any input. Even local phenomena, such as gender agreement, is not necessarily enforced. However, this is often desirable for practical purposes, because it makes the grammar more robust. Let us illustrate with an example in Swedish.

```
"<en>"
        "en" det indef utr sg
        "en" noun utr sg
"<bord>"
        "bord" noun neutr sg
```

The first word, *en*, is ambiguous between the indefinite determiner for *utrum* gender (masculine and feminine collapsed into one), or the noun 'juniper'. The second word, *bord*, is a neuter noun ('table')—we can assume that the most likely meaning is "a table" instead of "juniper table"[1], but the writer has mistaken the gender of 'table'. The correct form, which also would not be ambiguous, is "ett bord".

CG rules can be as fine-grained or broad as we want: `SELECT det IF (1 noun) (0 noun)` would match any determiner and any noun, and succesfully disambiguate *en* in this case.

[1] Compound words are written without a space: "juniper table" would be *enbord*.

We can also enforce grammaticality by writing SELECT det utr sg IF (1 noun utr sg) (0 noun utr sg). In that case, nothing in the grammatically incorrect phrase matches, and it is left ambiguous.

The previous example illustrates that the notions of *accept* and *reject* are not clear: if agreement was enforced by enumerating only legal combinations, the grammar would just refuse to disambiguate an ungrammatical sequence and leave all tags intact—in that case, its performance would not differ from a grammar that simply does not have many rules. The sequence en<det><utr> bord<noun><neutr> is undeniably ungrammatical; however, the alternative en<noun><utr> bord<noun><neutr> would be even "more wrong" for the purpose of analysing the text.

2.3.2 Shallow syntax

CG can be described as a shallow formalism for various reasons. The analysis is limited to concrete words; no invisible structure is postulated. CG rules usually operate on a low level of abstraction, and may target concrete words or word forms. Finally, the rules are independent: each rule describes its own piece of a phenomenon, and there is no inbuilt mechanism to guarantee that the rules are consistent with each other.

Low hierarchy We saw that the analysis starts from a list of alternatives for each word, and proceeds by eliminating impossible or unlikely candidates. These alternatives may include syntactic labels as well as morphological, but there are no invisible components in the analysis: all labels attach to a concrete token in the phrase. Syntactic or dependency labels are functionally the same as morphological: for example, @Subj is just one more tag in a reading. In fact, it is possible to add even more data to the readings, either from the initial morphological analysis, or with suitable post-processing. One could include semantic roles, or any other information that a lexicon may provide, such as animacy or frequency. As a result, readings may look like the following:

```
"<cat>"
        "cat" noun sg   @Subj £Agent §Animal
        "cat" verb pres @Pred §Computer §Rare
```

Moreover, any of these levels can be accessed in the same rule: e.g. SELECT noun IF (-1 "cat" §Animal) (NOT 0 §Rare). In other words, we may use syntactic or semantic features to help in morphological disambiguation, and vice versa.

This lack of deep structure is intentional in the design of CG. [16] justifies the choice with a number of theoretical principles: that language is open-ended and grammars are bound to leak, and that necessary information for the syntactic analysis comes from the morphology, hence morphology is "the cornerstone of syntax". In addition, [16] argues CG to be a reasonable psycholinguistic hypothesis; indeed, CG has been used in psycholinguistic research. **[TODO:** cite some paper by CLEAR people**]** find CG appealing especially because all levels are accessible at the same time, from lexicon and morphology to semantics.

Low abstraction As for the level of abstraction, CG rules lie somewhere between traditional grammar formalisms, such as HPSG, and purely statistical methods. The rules do not usually handle long-distance dependencies; they operate on a unit of a few words, nor do they abstract away from surface features such as word order. The rule that tells to remove verb reading after a determiner does not describe the whole abstract category "noun phrase"; we need a second rule to remove a verb after a determiner and an adjective (*the happy bear*) or an adjectival phrase (*the very happy bear*). Some rules may be purely lexical, and some rules may target the whole class of nouns or determiners; these decisions are motivated by what is needed for disambiguating real-life texts, rather than formulating the most elegant and concise rules possible.

Independence of rules The rules are self-contained and independent. On the one hand, this provides no guarantee that a grammar is internally consistent. On the other hand, these features provide flexibility that is hard to mimic by a deeper formalism. As we have seen in the previous sections, rules can target individual words or other properties that are not generalisable to a whole word class, such as verbs that express cognitive processes. Introducing a subset of verbs, even if they are used only in one rule, is very cheap and does not create a complicated taxonomy of different verb types.

Most importantly, the independence of rules makes CG highly robust. If one of the words is unknown or misspelt, a generative grammar would fail to produce any analysis. CG would, at worst, just leave that part ambiguous, and do as good a job it can elsewhere in the sentence.

2.3.3 Ordering of the rules

The previous properties of Constraint Grammar formalism and rules were specified in [16], and retained in further implementations. However, in the two decades following the initial specification, several independent implementations have experimented with different ordering schemes. In the present section, we describe the different parameters of ordering and execution: *strict vs. heuristic*, and *sequential vs. parallel*.

Throughout the section, we will apply the rules to the following ambiguous passage:

```
"<what>"
        "what" determiner
        "what" pronoun
 "<question>"
        "question" noun
        "question" verb
 "<is>"
        "be" verb
 "<this>"
        "this" determiner
```

Strict vs. heuristic aka. “In which order are the rules applied to a single cohort?”

An implementation with *strict order* applies each rule in the order in which they appear in the file. If a grammar contains the rules “remove verb after determiner” and “remove noun after pronoun” in the given order, the rule that removes the verb reading in *question* will be applied first. After it has finished, there are no verb readings available anymore for the second rule to fire.

How do we know which rule is the right one? There can be many rules that fit the context, but we choose the one that just happens to appear first in the rule file. A common design pattern is to place rules with narrow set of conditions first; only if they do not apply, then try a rule with less conditions. For a similar effect, a *careful mode* may be used: “remove verb after *unambiguous* determiner” would not fire on the first round, but it would wait for other rules to clarify the status of *what*.

An alternative solution to a strict order is to use a *heuristic order*: when disambiguating a particular word, find the rule that has the longest and most detailed match. Now, assume that there is a rule with a longer context, such as “select noun if -1 is determiner and +1 is verb”, even if this rule appears last in the file, it would be preferred to the shorter rules, because it is a more exact match.

Both methods have their strengths and weaknesses. A strict order is more predictable, but it also means that the grammar writers need to pay more thought to rule interaction.

A heuristic order frees the grammar writer from finding an optimal order, but it can give unexpected results, which are harder to debug. As for major CG implementations, [15] and [11] follow the strict scheme, whereas [25] is heuristic.

Sequential vs. parallel aka. "When does the rule application take effect?"

The input sentence can be processed in sequential or parallel manner. In *sequential execution*, the rules are applied to one word at a time, starting from the beginning of the sentence. The sentence is updated after each application. If the word *what* gets successfully disambiguated as a pronoun, then the word *question* will not match the rule "remove verb after determiner".

In contrast, a *parallel execution* scheme disambiguates all the words at the same time, using their initial, ambiguous context. Both "remove verb after determiner" and "remove noun after pronoun" would match *question*, because the original input from the morphological analyser contains both determiner and pronoun as the preceding word. Then, the final decision depends on any another rule that resolves *what*.

To give a minimal example, assume we have a single rule, `REMOVE verb IF (-1 verb)`, and the three words *w1–w3*, shown below. In parallel execution, both *w2* and *w3* lose their verb tag; in sequential only *w2*.

```
"<w1>"
        "w1" noun
        "w1" verb
"<w2>"
        "w2" noun
        "w2" verb
"<w3>"
        "w3" noun
        "w3" verb
```

Which execution scheme is better? For most purposes, a sequential execution is preferred. Sequential execution—together with strict or heuristic rule order—makes the rule take action immediately, and updates the context for the following words. If we know the correct part-of-speech for *what*, then it is easier to disambiguate *question*: there are less possible rules that may apply. Parallel execution may have uses in more marginal cases: for instance, if we have a small rule set and we want to find out if the rules are consistent with each other. [17] states the following about parallel CG:

```
Input                       Output
"<w>"                       "<w>"
         "a"                         "a"
         "b"
"<w>"                       "<w>"
         "a"                         "a"
         "b"
"<w>"                       "<w>"
         "a"                         "b"
         "b"
"<w>"                       "<w>"
         "a"                         "b"
         "b"
```

Figure 2.1: Recognising the context-free language $a^n b^n$ in CG.

> Each constraint simply adds something to the discriminating power of the whole grammar. No constraint rule may ever forbid something that would later on be accepted as an exception. This, maybe, puts more strain for the grammar writer but gives us better hope of understanding the grammar we write.

Perhaps unsurprisingly, nearly all of the existing CG implementations are sequential. A notable exception is the system proposed in [17], known by the names Finite-State Intersection Grammar (FSIG) or Parallel Constraint Grammar (PCG). As per [18], we use the latter term. In addition, a variant of the CG engine in [21] is parallel, as well as the "CG-like" system in [19].

2.3.4 Expressivity

As we have learnt, CG is not a generative grammar— it needs a list of alternatives and finds the correct ones among them. The question "what can CG express?" depends on what is present in the initial input. Take the example of central embedding; a feature known to require a context-free grammar. If a given generative grammar would analyse correctly an arbitrary number of central embeddings (e.g. *the dog* the cat bit *died*), this would suggest that the grammar is (at least) context-free. In order to translate such criterion to CG, we have two questions to ask: 1. Can a CG grammar correctly disambiguate a sentence with an arbitrary number of central embeddings? 2. If such grammar can be written, does it mean that CG is context-free? In addition, there are different variants and implementations of CG: it is not a meaningful question to define the expressivity of "CG", but rather CG-1 [16], CG-2 [25], VISLCG **[TODO:** cite someone**]**, VISL CG-3 [7], or any other less mainstream implementation.

In the following, we look at existing approaches to define the expressivity of reductionist formalisms, and suggest a way to emulate generation within (a variant of) CG.

Expressivity of a rule vs. expressivity of a grammar As defined in [26], the rules in finite-state constraint languages may include "contextual tests that are expressed in terms of regular languages". Indeed, we can verify that in all implementations of CG, the contextual tests must be regular: we cannot write in a single rule "remove verb if the previous segment contains only well-nested brackets". Tapanainen gives a hierarchy of four different constraint languages, in order of the expressivity in a single rule: Taggit [14] only allows rules to look at 1-2 tokens before or after the target; CG-1 [15] allows unlimited context within the window (usually one sentence) and adds operators such as the Kleene star and complement, but it does not allow all combinations, such as $(\Sigma - N) * A(\Sigma - V) * N$ **[TODO:** check this rule in CG-2 and CG-3**]**; in turn, CG-2 [25] can express this, but not e.g. repeated patterns. Finally, FSIG [17] allows the full expressivity of regular languages. Since CG-3 [7] did not exist at the time, [25] does not include it in the hierarchy of the constraint formalisms.

The expressivity of a single rule is well defined. However, the expressivity of the whole rule set is harder to define; if we return to the question of central embedding, a CG has much more lightweight task than a generative grammar. The initial tokens, or symbols, come already with a list of alternatives. Assume we have a rule to disambiguate the word form *who* as a relative pronoun instead of interrogative in the context "N who V". Now, this rule will perform the correct disambiguation for the pronoun, even if the relative clause is centrally embedded. If all the other rules are successful, then the grammar has chosen a correct analysis for a sentence, which is not context-free. Still, all of the rules have only regular contextual tests.

The aforementioned task is easy partly because the input already contains so much information. The CG rules do not need to choose from arbitrary number of tags; it is safe to assume that *who* will only be analysed as an interrogative or **[TODO:** a? damn indoeuropean languages with your articles**]** relative pronoun. Furthermore, it does not need to generate only correct embeddings: as defined in Section 2.3.1, CG is aimed to provide analysis even for incorrect input.

This said, CG is still a grammar formalism. For a moment, we ignore the practical needs of natural language analysis, and ask the question: "Can it express languages beyond regular?"

We draw inspiration from [26], who demonstrates the expressive power of another reductionist parsing formalism, Functional Dependency Grammar **[TODO:** cite!!!**]**. Instead of applying rules only to natural language texts, Tapanainen shows a FDG grammar which can successfully create a dependency structure for the context-free bracketing language, and the context-sensitive $a^n b^n c^n$. Unfortunately, [26] does not present a similar method for CG, but we can make such leap quite easily. In order to do this, we introduce the concept of *maximally ambiguous sentence*: a finite sequence of cohorts, where every cohort starts off with every possible reading. We call the set of readings Σ. In addition, we may need some helper labels Σ', such as $\{even, odd, start, end\}$. We add the additional requirement that the output of the grammar may only contain readings in Σ.

[TODO: Should I explain here how the $a^n b^n$ grammar works? Or just say "if we can write such grammar in a variant of CG, we can argue that this variant of CG is at least context-free"?**]**

Effect of ordering and execution schemes We can show that the $a^n b^n$ grammar, at least in the form described above, requires sequential execution: in the beginning, the status of the positions is clear at the edges, and the certainty spreads towards the middle, with the help of the careful context. In a parallel execution, we have the initial context during the whole round. If we can do parallel with multiple runs, it should work. But not just with one round.

How about strict and heuristic?

Note on termination Running the grammar multiple times is desirable: especially the rules with a careful context may need other rules to run first and disambiguate some items. With the most basic operations, SELECT and REMOVE, we can ensure that applying a grammar to a text will finish. The text might not be fully disambiguated, but if there is a point where running the rules again doesn't make a change, then the execution is stopped.

If we allow the addition of arbitrary tags to readings, or arbitrary readings to cohorts, we cannot guarantee that running the grammar will terminate. It is possible to add the same reading over and over again, in which case every time we run the grammar, the argument is changed. The syntactic tags used by MAP rules have an additional property, that one reading can have at most one `@tag`, and if a reading has one already, it is substituted with the old one. One can end up in a loop substituting `@A` with `@B`. However, specific implementations can freely add heuristics to stop this kind of behaviour from happening.

2.4 Boolean satisfiability (SAT)

Imagine you are in a pet shop with a selection of animals: *ant*, *bat*, *cat*, *dog*, *emu* and *fox*.

These animals are very particular about each others' company. The dog has no teeth and needs the cat to chew its food. The cat, in turn, wants its best friend bat. But the bat is very aggressive towards all herbivores, and the emu is afraid of anything lighter than 2 kilograms. The ant hates all four-legged creatures, and the fox can only handle one flatmate with wings.

You need to decide on a subset of pets to buy—you love all animals, but realistically, you cannot have them all. You start calculating in your head: "If I take the ant, I cannot take cat, dog, nor fox. How about I take the dog, then I must take the cat and the bat as well." After some time, you decide on bat, cat, dog and fox, leaving the ant and the emu in the pet shop.

Definition This everyday situation is an example of *Boolean satisfiability (SAT)* problem. The animals translate into *variables*, and the lists of friends and enemies of each animal translate into *clauses*. These clauses are disjunctions of *literals*: either a variable, such as *cat*, or its negation, $\neg cat$. A positive literal *cat* means that the animal comes with you, and conversely, $\neg cat$ means it is left in the shop. The clause $\neg dog \vee cat$ means "I don't buy the dog or I buy the cat". All the clauses are shown in Figure 2.2.

The objective is to find a *model*: each variable is assigned a Boolean value, such that the conjunction of all clauses evaluates into true. A program called *SAT-solver* takes the set of variables and clauses, and performs a search, like the mental calculations of the animal-lover in the shop.

We can see that the assignment $\{ant = 0, bat = 1, cat = 1, dog = 1, emu = 0, fox = 1\}$ satisfies the animals' wishes. Another possible assignment would be $\{ant = 0, bat = 0, cat = 0, dog = 0, emu = 1, fox = 1\}$: you only choose the emu and the fox. Some problems have a single solution, some problems have multiple solutions, and some are unsatisfiable, i.e. no combination of assignments can make the formula true.

History and applications SAT-solving as a research area dates back to 1970s. Throughout its history, it has been of interest for both theoretical and practical purposes. SAT is a well-known NP-complete [9]. This means that we can express any problem in the complexity class NP, such as prime number factorisation, as a SAT-instance, and use a SAT-solver to solve it. As we know, problems in NP have an exponential worst-case complexity. However, a typical real-life scenario formulated as SAT-problem can be solved much more efficiently with proper heuristics [8].

Animal	Constraint	Constraint in conjunctive normal form				
ant	$ant \Rightarrow \neg cat \wedge \neg dog \wedge \neg fox$	$\neg ant \vee \neg cat$	$\wedge$	$\neg ant \vee \neg dog$	$\wedge$	$\neg ant \vee \neg fox$
bat	$bat \Rightarrow \neg ant \wedge \neg emu$	$\neg bat \vee \neg ant$	$\wedge$	$\neg bat \vee \neg emu$		
cat	$cat \Rightarrow bat$	$\neg cat \vee bat$				
dog	$dog \Rightarrow cat$	$\neg dog \vee cat$				
emu	$emu \Rightarrow \neg ant \wedge \neg bat$	$\neg emu \vee \neg ant$	$\wedge$	$\neg emu \vee \neg bat$		
fox	$fox \Rightarrow \neg(bat \wedge emu)$	foo	$\wedge$	bar	$\wedge$	baz

Figure 2.2: Animals' cohabiting constraints translated into a SAT-problem.

Since the 2000s, SAT-solving is widely used in practical applications [23]. Much of the SAT research in the last two decades has been devoted to heuristic solving algorithms and more efficient ways of encoding various real-life problems into SAT. One of the biggest success stories for SAT application is model checking **[TODO:** cite**]**, used in software and hardware verification. In addition, SAT has been used in domains such as computational biology **[TODO:** cite**]** and **[TODO:** example**]**.

What kind of problems could benefit from an implementation in SAT? Formulating a decision problem in SAT is an attractive approach: instead of developing search heuristics independently, one can transform the problem into a SAT-instance and exploit decades of research into SAT-solving.

2.5 Summary

In this section, we have presented the theoretical background used in this thesis. We have introduced Constraint Grammar: a formalism for writing disambiguation rules; and Boolean satisfiability: methods for solving whether a given propositional formula can evaluate to true.

In the following chapters, we will connect the two branches. First, we look into existing research in connecting CG to logic, and formulate a SAT-instance for two varieties of CG. After establishing a SAT-encoding, we use it, much in the spirit of software verification, to show interesting properties of a given grammar.

Chapter 3

CG as a SAT-problem

3.1 Introduction

In this chapter, we present CG as a Boolean satisfiability (SAT) problem, and describe an implementation using a SAT solver. This is attractive for several reasons: formal logic is well-studied, and serves as an abstract language to reason about the properties of CG. Constraint rules encoded in logic capture richer dependencies between the tags than standard CG.

Applying logic to reductionist grammars has been explored earlier by [19, 20], but it was never adopted for use. Since those works, SAT solving techniques have improved significantly [23], and they are used in domains such as microprocessor design and computational biology—these problems easily match or exceed CG in complexity. Thanks to these advances, we were able to revisit the idea and develop it further.

Our work is primarily inspired by [19], which presents constraint rules as a disjunctive logic program, and [20], which reconstructs four different formalisms in first-order logic. Other works combining logic to CG include [13] and [1], both using Inductive Logic Programming to learn CG rules from a tagged corpus.

3.1.1 Previous work: Encoding in logic

[19] represents a CG-like, shallow and reductionist system in logic. [20] builds on that in a study which reconstructs four formalisms in logic. CG is contrasted with Finite-State Intersection Grammar (FSIG) and Brill tagging; all three work on a set of constraint rules which modify the initially ambiguous input, but with some crucial differences.

The rules and analyses are represented as clauses, and if it is possible to build a model which satisfies all clauses, then there is an analysis for the sentence. Take the unordered case first. The authors define predicates *word* and POS, and form clauses as follows. The variable P is used to denote any word, and the index n its position.

$$
\begin{aligned}
\text{word(P, the)} &\Rightarrow \textsc{pos}\text{(P, det)} \\
\text{word(P, bear)} &\Rightarrow \textsc{pos}\text{(P, verb)} \vee \textsc{pos}\text{(P, noun)} \\
\textsc{pos}(\text{P}_n, \text{det}) \wedge \textsc{pos}(\text{P}_{n+1}, \text{verb}) &\Rightarrow
\end{aligned}
$$

The first clauses read as "if the word is *the*, it is a determiner" and "if the word is *bear*, it can be a verb or a noun". The third clause represents the rule which prohibits a verb after a determiner. It normalises to $\neg\textsc{pos}(\text{P}_n, \text{det}) \vee \neg\textsc{pos}(\text{P}_{n+1}, \text{verb})$, and we know that POS(P, det) must be true for the word *the*, thus the verb analysis for *bear* must be false.

This representation models FSIG, where the rules are logically unordered. For CG, the authors introduce a new predicate for each rule, $\textsc{pos}^i$, where i indicates the index of the rule in the sequence of all rules. Each rule is translated into two clauses: 1) the conditions hold, the target has >1 analysis[1], and the targetet reading is selected or removed; and 2) the conditions don't hold or the target has only one analysis, and the target is left untouched. The general form of the clauses is shown below:

$$
\begin{aligned}
\textsc{pos}^i(\text{P, T}) \wedge (\ \ \text{conditions\_hold} \wedge |\text{T}| > 1) &\Rightarrow \textsc{pos}^{i+1}(\text{P, T} \setminus [\text{target}]) \\
\textsc{pos}^i(\text{P, T}) \wedge (\neg\text{conditions\_hold} \vee |\text{T}| = 1) &\Rightarrow \textsc{pos}^{i+1}(\text{P, T})
\end{aligned}
$$

To show a concrete example, the following shows the two rules in the specified order:

```
REMOVE verb IF (-1 det) ;

REMOVE noun IF (-1 noun) ;
```

.

[1] In [21], the default rules are given to the SAT solver as separate clauses; for every word w, at least one of its analyses $\{w_1,w_2,...,w_n\}$ must be true. Then the clauses for each rule don't need to repeat the "only if it doesn't remove the last reading" condition.

$$
\begin{aligned}
\textsc{pos}^1(\mathrm{P}, [\mathrm{det}]) &\Leftarrow \mathrm{word}(\mathrm{P}, \mathrm{the}) \\
\textsc{pos}^1(\mathrm{P}, [\mathrm{verb},\mathrm{noun}]) &\Leftarrow \mathrm{word}(\mathrm{P}, \mathrm{bear}) \\
\textsc{pos}^1(\mathrm{P}, [\mathrm{verb},\mathrm{noun}]) &\Leftarrow \mathrm{word}(\mathrm{P}, \mathrm{sleeps})
\end{aligned}
$$

$$
\begin{aligned}
\textsc{pos}^2(\mathrm{P}_n, \mathrm{T} \setminus [\mathrm{verb}]) &\Leftarrow \textsc{pos}^1(\mathrm{P}_n, \mathrm{T}) \wedge (\ \textsc{pos}^1(\mathrm{P}_{n-1}, [\mathrm{det}]) \wedge \mathrm{T} \setminus [\mathrm{verb}] \neq []) \\
\textsc{pos}^2(\mathrm{P}_n, \mathrm{T}) &\Leftarrow \textsc{pos}^1(\mathrm{P}_n, \mathrm{T}) \wedge (\neg\textsc{pos}^1(\mathrm{P}_{n-1}, [\mathrm{det}]) \vee \mathrm{T} \setminus [\mathrm{verb}] = [])
\end{aligned}
$$

$$
\begin{aligned}
\textsc{pos}^3(\mathrm{P}_n, \mathrm{T} \setminus [\mathrm{noun}]) &\Leftarrow \textsc{pos}^2(\mathrm{P}_n, \mathrm{T}) \wedge (\ \textsc{pos}^2(\mathrm{P}_{n-1}, [\mathrm{noun}]) \wedge \mathrm{T} \setminus [\mathrm{noun}] \neq []) \\
\textsc{pos}^3(\mathrm{P}_n, \mathrm{T}) &\Leftarrow \textsc{pos}^2(\mathrm{P}_n, \mathrm{T}) \wedge (\neg\textsc{pos}^2(\mathrm{P}_{n-1}, [\mathrm{noun}]) \vee \mathrm{T} \setminus [\mathrm{noun}] = [])
\end{aligned}
$$

The logical reconstruction helps to provide some clarity when comparing CG to FSIG. In FSIG, the predicate *pos* is a statement of an analysis of a word; in case of uncertainty, disjunction is used to present all possible analyses. In CG, uncertainty is modelled by sets of analyses, and the predicate pos^i is a statement of the set of analyses of a word at a given stage of the rule sequence. The final result is obtained by composition of these clauses in the order of the rule sequence.

3.2 CG as a SAT problem

Let us demonstrate our approach with the following example in Spanish.

```
"<la>"
        "el" det def f sg
        "lo" prn p3 f sg
"<casa>"
        "casa" n f sg
        "casar" v pri p3 sg
        "casar" v imp p2 sg
```

The ambiguous passage can be either a noun phrase, *la*`<det>` *casa*`<n>` 'the house' or a verb phrase *la*`<prn>` *casa*`<v><pri><p3>` '(he/she) marries her'. We add the following rules:

```
REMOVE prn IF (1 n) ;
REMOVE det IF (1 v) ;
```

Standard CG will apply one of the rules to the word *la*; either the one that comes first, or by some other heuristic. The other rule will not fire, because it would remove the last reading. If we use the cautious mode (`1C n` or `1C v`), which requires the word in the context to be fully disambiguated, neither of the rules will be applied. In any case, all readings of *casa* are left untouched by these rules.

The SAT solver performs a search, and starts building possible models that satisfy both constraints. In addition to the given constraints, we have default rules to emulate the CG principles: an analysis is true if no rule affects it, and at least one analysis for each word is true—the notion of "last" is not applicable.

With these constraints, we get two solutions. The interaction of the rules regarding *la* disambiguates the part of speech of *casa* for free, and the order of the rules does not matter.

```
1) "<la>"
           "el" det def f sg
   "<casa>"
           "casa" n f sg

2) "<la>"
           "lo" prn p3 f sg
   "<casa>"
           "casar" v pri p3 sg
           "casar" v imp p2 sg
```

The most important differences between the traditional and the SAT-based approach are described in the following sections.

3.2.1 Rules disambiguate more

Considering our example phrase and rules, the standard CG implementation can only remove readings from the target word (`prn` or `det`). The SAT-based implementation interprets the rules as "determiner and verb together are illegal", and is free to take action that concerns also the word in the condition (`n` or `v`).

This behaviour is explained by simple properties of logical formulae. When the rules are applied to the text, they are translated into implications: `REMOVE prn IF (1 n)` becomes *casa*<n> ⇒ ¬*la*<prn>, which reads "if the n reading for *casa* is true, then discard the prn reading for *la*". Any implication $a \Rightarrow b$ can be represented as a disjunction $\neg a \lor b$; intuitively, either the antecedent is false and the consequent can be anything, or the consequent is true and the antecedent can be anything. Due to this property, our rule translates into the disjunction ¬*casa*<n> ∨ ¬*la*<prn>, which is also equivalent to another implication, *la*<prn> ⇒ ¬*casa*<n>. This means that the rules are logically flipped: REMOVE prn IF (1 n) translates into the same logical formula as REMOVE n IF (-1 prn). A rule with more conditions corresponds to many rules, each condition taking its turn to be the target.

3.2.2 Cautious context is irrelevant

Traditional CG applies the rule set iteratively: some rules fire during the first iteration, either because their conditions do not require cautious context, or because some words are unambiguous to start with. This makes some more words unambiguous, and new rules can fire during the second iteration.

In SAT-CG, the notion of cautious context is irrelevant. Instead of removing readings immediately, each rule generates a number of implications, and the SAT solver tries to find a model that will satisfy them.

Let us continue with the earlier example. We can add a word to the input:

la casa grande 'the big house'

and a rule that removes verb reading, if the word is followed by an adjective:

```
REMOVE v IF (1 adj) ;
```

The new rule adds the implication *grande*<adj> ⇒ ¬*casa*<v>, which will disambiguate *casa* to a noun[2]. As the status of *casa* is resolved, the SAT solver can now discard the model where *casa* is a verb and *la* is a pronoun and we get a unique solution with det n adj.

Contrast this with the behaviour of the standard CG. With the new rule, standard CG will also remove the verb reading from *casa*, but it is in no way connected to the choice for *la*. It all depends on the order of the two rules; if the det reading of *la* is removed first, then we are stuck with that choice. If we made the first rules cautious, that is, keeping the determiner open until *casa* is disambiguated, then we get the same result as with the SAT solver. Ideally, both ways of grammar writing should yield similar results; traditional CG rules are more imperative, and SAT-CG rules are more declarative.

[2] Assuming that adj is the only reading for *grande*, it must be true, because of the restriction that at least one analysis for each word is true. Then the implication has a true antecedent (*grande*<adj>), thus its consequent (¬*casa*<v>) will hold.

3.2.3 Rules can be unordered

As hinted by the previous property, the SAT solver does not need a fixed order of the rules. Applying a rule to a sentence produces a number of clauses, and those clauses are fed into the SAT solver. However, in the unordered scheme, some information is lost: the following rule sets would be treated identically, whereas in the traditional CG, only the first would be considered as a bad order.

```
1) SELECT v ;
   REMOVE v IF (-1 det) ;

2) REMOVE v IF (-1 det) ;
   SELECT v ;
```

Without order, both of these rule sets will conflict, if applied to an input that has sequence `det v`. The SAT solver is given clauses that tell to select a verb and remove a verb, and it cannot build a model that satisfies all of those clauses. To solve this problem, we create a variable for every instance of rule application, and request a solution where maximally many of these variables are true. If there is no conflict, then the maximal solution is one where all of these variables are true; that is, all rules take action.

In case of a conflict, the SAT solver makes it possible to discard only minimal amount of rule applications. Continuing with the example, it is not clear which instances would be discarded, but if the rules were part of a larger rule set, and in the context the REMOVE rule was the right one to choose, it is likely that the interaction between the desired rules would make a large set of clauses that fit together, and the SELECT rule would not fit in, hence it would be discarded.

This corresponds loosely to the common design pattern in CGs, where there is a number of rules with the same target, ordered such that more secure rules come first, with a catch-all rule with no condition as the last resort, to be applied if none of the previous has fired. The order-based heuristic in the traditional CG is replaced by a more holistic behaviour: if the rules conflict, discard the one that seems like an outlier.

We can also emulate order with SAT-CG. To do that, we enter clauses produced by each rule one by one, and assume the solver state reached so far is correct. If a new clause introduces a conflict with previous clauses, we discard it and move on to the next rule. By testing against gold standard, we see that this scheme works better with ready-made CGs, which are written with ordering in mind. It also runs slightly faster than the unordered version.

These three features influence the way rules are written. We predict that less rules are needed; whether this holds in the order of thousands of rules remains to be tested. On the one hand, getting rid of ordering and cautious context could ease the task of the grammar writer, since it removes the burden of estimating the best sequence of rules and whether to make them cautious. On the other hand, lack of order can make the rules less transparent, and might not scale up for larger grammars.

3.3 Evaluation

For evaluation, we measure the performance against the state-of-the-art CG parser VISL CG-3. SAT-CG fares slightly worse for accuracy, and significantly worse for execution time. The results are presented in more detail in the following sections.

3.3.1 Performance against VISL CG-3

We took a manually tagged corpus[3] containing approximately 22,000 words of Spanish news text, and a small constraint grammar[4], produced independently of the authors. We kept only SELECT and REMOVE rules, which left us 261 rules. With this setup, we produced an ambiguous version of the tagged corpus, and ran both SAT-CG and VISL CG-3 on it. Treating the original corpus as the gold standard, the disambiguation by VISL CG-3 achieves F-score of 82.6 %, ordered SAT-CG 81.5 % and unordered SAT-CG 79.2 %. We did not test with other languages or text genres due to the lack of available gold standard.

We also tested whether SAT-CG outperforms traditional CG with a small rule set. With our best performing and most concise grammar[5] of only 19 rules, both SAT-CG and VISL CG-3 achieve a F-score of around 85 %. This experiment is very small and might be explained by overfitting or mere chance, but it seems to indicate that rules that work well with SAT-CG are also good for traditional CG.

3.3.2 Execution time

The worst-case complexity of SAT is exponential, whereas the standard implementations of CG are polynomial, but with advances in SAT solving techniques, the performance in the

[3] https://svn.code.sf.net/p/apertium/svn/branches/apertium-swpost/apertium-en-es/es-tagger-data/es.tagged
[4] https://svn.code.sf.net/p/apertium/svn/languages/apertium-spa/apertium-spa.spa.rlx
[5] https://github.com/inariksit/cgsat/blob/master/data/spa_smallset.rlx

# rules	SAT-CG$_u$	SAT-CG$_o$	VISL CG-3
19	39.7s	22.1s	4.2s
99	1m34.1s	1m14.9s	6.1s
261	2m54.1s	2m31.6s	10.7s

Table 3.1: Execution times for 384,155 words.

average case in practice is more feasible than in the previous works done in 90s–00s. We used the open-source SAT solver MiniSat [12].

We tested the performance by parsing Don Quijote (384,155 words) with the same Spanish grammars as in the previous experiment. Table 3.1 shows execution times compared to VISL CG-3; SAT-CG$_u$ is the unordered scheme and SAT-CG$_o$ is the ordered. From the SAT solving side, maximisation is the most costly operation. Emulating order is slightly faster, likely because the maximisation problems are smaller. In any case, SAT does not seem to be the bottleneck: with 261 rules, the maximisation function was called 147,253 times, and with 19 rules, 132,255 times, but the differences in the execution times are much larger, which suggests that there are other reasons for the worse performance. This is to be expected, as SAT-CG is currently just a naive proof-of-concept implementation with no optimisations.

3.4 Applications and future work

Instead of trying to compete with the state of the art, we plan to use SAT-CG for grammar analysis[6]. There has been work on automatic tuning of hand-written CGs [6], but to our knowledge no tools to search for inconsistencies or suboptimal design.

The sequential application of traditional CG rules is good for performance and transparency. When a rule takes action, the analyses are removed from the sentence, and the next rules get the modified sentence as input. As a downside, there is no way to know which part comes directly from the raw input and which part from applying previous rules.

A conflict in an ordered scheme can be defined as a set of two or more rules, such that applying the first makes the next rules impossible to apply, regardless of the input. We can reuse the example from Section 3.2.3:

```
SELECT v ;
REMOVE v IF (-1 det) ;
```

[6] We thank Eckhard Bick for the idea.

The first rule selects the verb reading everywhere and removes all other readings, leaving no chance for the second rule to take action. If the rules are introduced in a different order, there is no conflict: the REMOVE rule would not remove verb readings from all possible verb analyses, so there is a possibility for the SELECT rule to fire.

Ordered SAT-CG can be used to detect these conflicts without any modifications, as a side effect of its design. After applying each rule, it stores the clauses produced by the rule and commits to them. In case of a conflict, the program detects the particular rule that violates the previous clauses, with the sentence where it is applied. Thus we get feedback which rule fails, and on which particular word(s).

Unordered SAT-CG with maximisation-based conflict solving is not suitable for this task: the whole definition of conflict depends on ordering, and the unordered scheme deliberately loses this information. On a more speculative note, an unordered formalism such as Finite-State Intersection Grammar [17] might benefit from the maximisation-based technique in conflict handling.

Finally, we intend to test for conflicts without using a corpus. Let us illustrate the idea with the same two rules, `SELECT v` and `REMOVE v IF (-1 det)` in both orders. Assume we have the tag set `{det, n, v}`, and we want to find if there exists an input such that both rules, applied in the given order, remove something from the input. There are no inputs that satisfy the requirement with the first order, but several that work with the second, such as the following:

```
"<w1>"
        det
        v
"<w2>"
        n
        v
```

Thus we can say that the first rule order is conflicting, but the second one is not. Implementing and testing this on a larger scale is left for future work.

3.5 Conclusions

SAT-solvers are nowadays powerful enough to be used for dealing with Constraint Grammar. A logic-based approach to CG has possible advantages over more traditional approaches;

a SAT solver may disambiguate more words, and may do so more precisely, capturing more dependencies between tags. We experimented with both ordered and unordered rules, and found the ordered scheme to work better with previously written grammars. For future direction, we intend to concentrate on grammar analysis, especially finding conflicts in constraint grammars.

Appendix: SAT-encoding

```
"<sobre>"
        "sobre" pr
        "sobre" n m sg
"<una>"
        "uno" prn tn f sg
        "uno" det ind f sg
        "unir" v prs p3 sg
        "unir" v prs p1 sg
        "unir" v imp p3 sg
"<aproximación>"
        "aproximación" n f sg
"<más>"
        "más" adv
        "más" adj mf sp
"<científica>"
        "científico" adj f sg
        "científico" n f sg
```

Figure 3.A.1: Ambiguous segment in Spanish.

In this appendix, we show the SAT-encoding of different rule types in detail. A reader who wants a general overview can read Section **[TODO:** write that section**]**. The operations in this appendix are independent of the implementation of the software SAT-CG; it does not describe how the sentence is processed in order to find the variables. The only purpose is to demonstrate what kind of clauses are constructed.

3.A SAT-encoding of sentences

We demonstrate the SAT-encoding with a concrete segment in Spanish: *sobre una aproximación más científica*. It has the virtue of being ambiguous in nearly every word: for instance, *sobre* is either a preposition ('above' or 'about') or a noun ('envelope'); *una* can be a pronoun, determiner or a verb. The full analysis, with the initial ambiguities, is shown in Figure 3.A.1. We transform each reading into a SAT-variable:

$$
\begin{array}{ccccc}
sobre_{Pr} & una_{Prn} & aproximación_N & más_{Adv} & científica_{Adj} \\
sobre_N & una_{Det} & & más_{Adj} & científica_N \\
 & una_{PrsP3} & & & \\
 & una_{PrsP1} & & & \\
 & una_{ImpP3} & & &
\end{array}
\tag{3.1}
$$

CG rules may not remove the last reading, even if the conditions hold otherwise. To ensure that each cohort contains at least one true variable, we add the clauses in 3.2. The word *aproximación* is already unambiguous, thus the clause $aproximación_N$ is a unit clause, and the respective variable is trivially assigned $True$. The final assignment of the other variables depends on the constraint rules.

$$\begin{gathered} sobre_{Pr} \vee sobre_{N} \\ una_{Prn} \vee una_{Det} \vee una_{PrsP3} \vee una_{PrsP1} \vee una_{ImpP3} \\ aproximación_{N} \\ más_{Adv} \vee más_{Adj} \\ científica_{Adj} \vee científica_{N} \end{gathered} \tag{3.2}$$

3.B SAT-encoding of rules

In order to demonstrate the SAT-encoding, we show variants of REMOVE and SELECT rules, with different contextual tests. We try to craft rules that make sense for this segment; however, some variants are not likely encountered in a real grammar, and for some rule types, we modify the rule slightly. We believe this makes the encoding overall more readable, in contrast to using more homogenous but more artificial rules and input.

3.B.1 Unordered scheme

We begin by introducing the unordered scheme. For basic rules, this corresponds to the encoding in [19]; each analysis is given a single variable, and the rule application is unordered.

No conditions

The simplest rule types remove or select a target in all cases. A rule can target one or multiple readings in a cohort. We demonstrate the case with one target in rules 3.3–3.4, and multiple target in rules 3.5–3.6.

REMOVE adj Unconditionally removes all readings which contain the target.

$$\begin{gathered} \neg más_{Adj} \\ \neg científica_{Adj} \end{gathered} \tag{3.3}$$

SELECT adj Unconditionally removes all other readings which are in the same cohort with the target.

$$\begin{gathered} más_{Adj} \wedge \neg más_{Adv} \\ científica_{Adj} \wedge \neg científica_{N} \end{gathered} \tag{3.4}$$

REMOVE verb As 3.3, but matches multiple readings in a cohort. All targets are negated.

$$\neg una_{PrsP3} \wedge \neg una_{PrsP1} \wedge \neg una_{ImpP3} \tag{3.5}$$

SELECT verb As 3.4, but matches multiple readings in a cohort. At least one of the target readings is true, all other readings in the same cohort are negated.

$$(una_{PrsP3} \vee una_{PrsP1} \vee una_{ImpP3}) \wedge (\neg una_{Det} \wedge \neg una_{Prn}) \tag{3.6}$$

Positive conditions

Rules with contextual tests apply to the target, if the conditions hold. This is naturally represented as implications. We demonstrate both REMOVE and SELECT rules with a single condition in 3.7; the rest of the variants only with REMOVE. All rule types can be changed to SELECT by changing the consequent.

$$\begin{array}{ll} \textbf{REMOVE adj IF (1 adj)} & \textbf{SELECT adj IF (1 adj)} \\ \mathit{científica}_{Adj} \implies \neg \mathit{más}_{Adj} & \mathit{científica}_{Adj} \implies \mathit{más}_{Adj} \wedge \neg \mathit{más}_{Adv} \end{array} \tag{3.7}$$

REMOVE adj IF (1 adj) (-1 n) Conjunction of conditions.

$$\mathit{científica}_{Adj} \wedge \mathit{aproximación}_{N} \implies \neg \mathit{más}_{Adj} \tag{3.8}$$

REMOVE adj IF ((1 adj) OR (-1 n)) Disjunction of conditions (template).

$$\mathit{científica}_{Adj} \vee \mathit{aproximación}_{N} \implies \neg \mathit{más}_{Adj} \tag{3.9}$$

REMOVE adj IF (1C adj) Careful context. Condition must be must be unambiguously adjective.

$$\mathit{científica}_{Adj} \wedge \neg \mathit{científica}_{N} \implies \neg \mathit{más}_{Adj} \tag{3.10}$$

REMOVE adj IF (-1* n) Scanning. Any noun before the target is a valid condition.

$$sobre_N \vee aproximación_N \implies \neg más_{Adj} \tag{3.11}$$

REMOVE adj IF (-1* n BARRIER det) Scanning up to a barrier. Any noun before the target, up to a determiner, is a valid condition: $sobre_N$ is a valid condition only if una_{Det} is false.

$$(sobre_N \wedge \neg una_{Det}) \vee aproximación_N \implies \neg más_{Adj} \tag{3.12}$$

REMOVE adj IF (-1* n CBARRIER det) Scanning up to a careful barrier. Any noun before the target, up to an unambiguous determiner, is a valid condition. The variable una_{Det} fails to work as a barrier, if any of the other analyses of *una* is true. Let una_{Any} denote the disjunction $una_{Prn} \vee una_{PrsP3} \vee una_{PrsP1} \vee una_{ImpP3}$.

$$(sobre_N \wedge una_{\text{Any}}) \vee aproximación_N \implies \neg más_{Adj} \tag{3.13}$$

REMOVE adj IF (-1C* n) Scanning with careful context

$$(sobre_N \wedge \neg sobre_{Pr}) \vee aproximación_N \implies \neg más_{Adj} \tag{3.14}$$

REMOVE adj IF (-1C* n BARRIER det) Scanning with careful context, up to a barrier.

$$(sobre_N \wedge \neg sobre_{Pr} \wedge \neg una_{Det}) \vee aproximación_N \implies \neg más_{Adj} \tag{3.15}$$

REMOVE adj IF (-1C* n CBARRIER det) Scanning with careful context, up to a careful barrier.

$$(sobre_N \wedge \neg sobre_{Pr} \wedge una_{\text{Any}}) \vee aproximación_N \implies \neg más_{Adj} \tag{3.16}$$

Inverted conditions

In the following, we demonstrate the effect of two inversion operators. The keyword NOT inverts a single contextual test, uch as **IF (NOT 1 noun)** , as well as linked conditions, such as **IF (-2 det LINK NOT *1 noun)** . The keyword NEGATE inverts a whole conjunction of contextual tests, which may have any polarity: **IF (NEGATE -2 det LINK NOT 1 noun)** means “there may not be a determiner followed by a not-noun”; thus, “det noun” would be fine, or “prn adj”, but not “det adj”. Inversion cannot be applied to a BARRIER condition. If one wants to express **IF (*1 foo BARRIER ¬bar)** , that is, “try to find a *foo* until you see the first item that is not *bar*", a set complement operator must be used: **(*) - bar** .

There is a crucial difference between matching positive and inverted conditions. If a positive condition is out of scope or the tag is not present in the initial analysis, the rule simply does not match, and no clauses are created. For instance, the conditions '10 adj' or '-1 punct', matched against our example passage, would not result in any action. In contrast, when an inverted condition is out of scope or unapplicable, that makes the action happen unconditionally. As per VISL CG-3, the condition **NOT 10 adj** applies to all sentences where there is no 10th word from target that is adjective; including the case where there is no 10th word at all. If we need to actually have a 10th word to the right, but that word may not be an adjective, we can, again, use the set complement: **IF (10 (*) - adj)** .

REMOVE adj IF (NOT 1 adj) Single inverted condition.

$$\neg científica_{Adj} \implies \neg más_{Adj} \tag{3.17}$$

REMOVE adj IF (NOT 1 adj) (NOT -1 n) Conjunction of inverted conditions.

$$\neg científica_{Adj} \wedge \neg\ aproximación_{N} \implies \neg más_{Adj} \tag{3.18}$$

REMOVE adj IF (NEGATE -3 pr LINK 1 det LINK 1 n) Negation of a conjunction of conditions.

$$\neg sobre_{Pr} \vee \neg una_{Det} \vee \neg\ aproximación_{N}) \implies \neg más_{Adj} \tag{3.19}$$

REMOVE adj IF (NOT 1C adj) Negated careful context. Condition cannot be unambiguously adjective.

$$científica_{N} \implies \neg más_{Adj} \tag{3.20}$$

REMOVE adj IF (NOT -1* n) Scanning. There must be no nouns before the target.

$$\neg sobre_{N} \wedge \neg\ aproximación_{N} \implies \neg más_{Adj} \tag{3.21}$$

REMOVE adj IF (NOT -1* n BARRIER det) Scanning up to a barrier. There must be no nouns before the target, up to a determiner.

$$(\neg sobre_N \lor una_{Det}) \land \neg\, aproximación_N \quad \Longrightarrow \quad \neg más_{Adj} \tag{3.22}$$

REMOVE adj IF (-1* n CBARRIER det) Scanning up to a careful barrier.

$$(\neg sobre_N \lor una_{\mathrm{Any}}) \land \neg\, aproximación_N \quad \Longrightarrow \quad \neg más_{Adj} \tag{3.23}$$

Non-matching inverted conditions

Here we demonstrate a number of inverted rules, in which the contextual test does not match the example sentence. As a result, the action is performed unconditionally.

REMOVE adj IF (NOT 1 punct) Single inverted condition, not present in initial analysis.

REMOVE adj IF (NOT 10 adj) Single inverted condition, out of scope.

REMOVE adj IF (NOT 1 adj) (NOT -10 n) Conjunction of inverted conditions, one out of scope.

REMOVE adj IF (NEGATE -3 pr LINK 1 punct) Negation of a conjunction of conditions, some not present in initial analysis.

$$\neg más_{Adj} \tag{3.24}$$

Solving and conflict handling

[TODO: Move some of this long explanation to the main chapter]

The unordered scheme has a strong advantage compared to the ordered one: it can start disambiguating, even if none of the rules on its own was able to act. In addition,

However, ignoring the order means that we miss significant information. In the unordered scheme, the different rule orders in Figure 3.B.1 would be treated the same. Without order, they would be both applied to the word *más*, and that would cause a conflict: we cannot remove both. With order, the problem is trivial: whichever is mentioned first, gets to apply, and removes the target. After that, the second one would not even be considered, because of the default rule, which prohibits removing the last reading.

Can we use an alternative way to handle conflicts in the unordered scheme? Recall the notion of heuristic rule order from Section 2.3.3: the rules are chosen based on how well they match the context. A longer rule is a more exact match than a shorter rule, and thus preferred.

```
REMOVE adv IF (-1 n) ;          REMOVE adj IF (-1 n) ;
REMOVE adj IF (-1 n) ;          REMOVE adv IF (-1 n) ;
```

Figure 3.B.1: Two rules that target the same reading, in different orders.

The heuristic rule order is still an order, though; it does not tell us what to do, when the two rules have already created clauses.

Heuristic order asks the question "out of all the rules that target this cohort, which one is the most exact match?" If the competitors are **REMOVE adj IF (-1 n)** and **REMOVE adv IF (-1 n) (1 adj)** , then the second one will win. However, if the rules are both as good a match, which happens in Figure 3.B.1, we need to resort to mere guessing, or picking the one which is mentioned first in the rule set.

For an unordered scheme, we can ask a more complex question: "out of all the rules that target this cohort, which one is a best fit *with other rules that will apply to this whole sentence*?" With a SAT-solver, we can answer this question. Recall that each rule application produces a formula, which is a conjunction of clauses, and in an ideal case, the whole formula is satisfiable. However, if the whole formula is unsatisfiable, we may still ask the question: give an assignment that satisfies the maximum number of the clauses. If the grammar is good, we hope that the interaction between the appropriate rules would make a large set of clauses that fit together, and the inapplicable rule would not "fit in". In the SAT-world, this means that the largest number of satisfiable clauses would include the group of well-fitting rules, and leave the odd rule out.

[TODO: Difference between "condition didn't hold in the first place" and "two rules have the same condition and conflicting action"**]**

3.B.2 Ordered scheme

We continue with the ordered scheme. In contrast to the unordered scheme, ordering of the rules makes a difference: the two rule orders in Figure 3.B.1 produce a different result.

In the unordered scheme, we simply translated every applicable rule into a clause, and asked the SAT-solver to find a model that satisfies all the clauses. Now, we need to model a state of the sentence. The first rule accesses the initial input by the morphological analyser, and the n^{th} rule accesses the sentence after the first $n-1$ rules have been run. Similarly to [20], we model this as a composition of clauses, each accessing the state produced by the previous clause.

The following is not the only possible SAT-encoding to model an ordered execution of the rules. In fact, [21] uses a different encoding; however, we changed into the one described in the following, because it generalises to grammar analysis (see Chapter 4).

Creating the sentence

The main difference is that the sentence starts off with all variables assigned $True$. Contrast 3.25 with 3.2;

$$\begin{gathered} \mathit{sobre}_{Pr} \wedge \mathit{sobre}_{N} \\ \mathit{una}_{Prn} \wedge \mathit{una}_{Det} \wedge \mathit{una}_{PrsP3} \wedge \mathit{una}_{PrsP1} \wedge \mathit{una}_{ImpP3} \\ \mathit{aproximación}_{N} \\ \mathit{más}_{Adv} \wedge \mathit{más}_{Adj} \\ \mathit{científica}_{Adj} \wedge \mathit{científica}_{N} \end{gathered} \tag{3.25}$$

At each rule application, we create a new variable for each targeted reading. The new variable is $True$ iff

(a) the old variable was $True$, and (b) the rule cannot apply: this can be because
- its conditions do not hold, or
- it would remove the last reading.

For subsequent rule applications, only the variables from the latest round are accessible: the n^{th} rule may only access the variables that are created, or left unchanged, by the $n-1^{th}$ rule.

[TODO: Koen, you can stop reading here, this is still crap!**]**

$$\begin{aligned} \text{condsHold} &::= c1 \vee c2 \vee c3 \ldots \\ \text{someTrgIsTrue} &::= r1 \vee r2 \vee r3 \ldots \\ \text{noOtherIsTrue} &::= \neg r4 \wedge \neg r5 \ldots \\ \text{onlyTrgLeft} &::= \text{someTrgIsTrue} \wedge \text{noOtherIsTrue} \\ \text{cannotApply} &::= \neg\text{condsHold} \vee \text{onlyTrgLeft} \\ \text{New variable } trg' &::= trg \wedge \text{cannotApply} \end{aligned} \tag{3.26}$$

No conditions

REMOVE adj Unconditionally removes all readings which contain the target.

$$\begin{gathered}\text{New variable } \textit{más}'_{Adj} \\ \text{New variable } \textit{científica}'_{Adj} \\ \neg \textit{más}'_{Adj} \\ \neg \textit{científica}'_{Adj}\end{gathered} \tag{3.27}$$

SELECT adj SELECT variant. Unconditionally removes all other readings that appear in the same cohort with the target.

$$\begin{gathered}\textit{más}_{Adj} \wedge \neg \textit{más}_{Adv} \\ \textit{científica}_{Adj} \wedge \neg \textit{científica}_{N}\end{gathered} \tag{3.28}$$

Positive conditions

Chapter 4

Grammar analysis using SAT

In the previous chapter, we have seen the SAT encoding of CG used to create a CG engine. We evaluated our engine against the state-of-the-art VISL CG-3, using the same grammar and same gold standard corpus. Unsurprisingly, we got worse results when using the SAT-based implementation on grammars that were written for an imperative and sequential CG engine. Given that most real grammars out there are written in such way, using SAT in the CG engine offers little practical use.

On the other hand, SAT-based implementation offers benefits that are out of reach for the standard CG implementations. By design, the effect of each rule is retained, because it makes a difference whether to execute a rule that appears later. This means that we can use our implementation for analysing the grammar.

In the implementation described in the previous paragraph, we analysed some real input sentences, and generated clauses of the rules that applied to those particular sentences. If there is a word that is analysed as n or v, it can only match rules that target those analyses (and is surrounded by appropriate context).

Now, we operate on *symbolic sentences*. We start from a situation where each word in the sentence can have any analysis: this means that every rule potentially applies to every word. This combination of rule application starts narrowing down the potential sentence. The rules are interpreted as more abstract and declarative: `REMOVE verb IF -1 det` does not just check if a particular word is verb, it prohibits a combination of determiner followed by verb *anywhere*. The restriction can show in various ways, and must be in sync with other restrictions.

If it turns out that there is no symbolic sentence that can satisfy a number of rules, this means that there is a conflict among the rules. In the following chapter, we will give examples

```
SELECT Inf IF (-1 Para OR De) (0C V) ;
SELECT Inf IF (-1 Prep) (0C V) ;
SELECT Inf IF (-1C Vai) ;
SELECT Inf IF (-1C Vbmod) (0C V) ;
SELECT Inf IF (-1C Ter/de) ;
SELECT Inf IF (-1C Vbmod) (0 Ser) ;
```

Figure 4.1: Rules to select infinitive in Portuguese.

of such conflicts and describe how to detect them.

Former LREC paper starts here CGs are valuable resources for rule-based NLP, especially for lesser resourced languages. They are robust and can be written without large corpora—only morphological analysis is needed. The formalism is lightweight and language-independent, and resources can be shared between related languages [4, 3]. Mature CGs contain some thousands of rules, but even small CGs are shown to be effective [2].

By design, CG is a shallow and robust formalism. There is no particular hierarchy between lexical, morphological, syntactic or even semantic tags: individual rules can be written to address any property, such as "verb", "copula verb in first person singular", or "the word form *sailor*, preceded by *drunken* anywhere in the sentence". This makes it possible to treat very particular edge cases without touching the more general rule: we would simply write the narrow rule first ("if noun AND *sailor*"), and introduce the general rule ("if noun") later.

However, this design is not without problems. As CGs grow larger, it gets harder to keep track of all the rules and their interaction. Our tools will help grammar writers and users to find conflicting rules, diagnose problems and improve their grammars. We expect two major use cases: first, to test the effect of new rules while writing a grammar, and second, to take a complete grammar and analyse it as a whole, to find conflicts or dead rules.

Given the rules in figure **??**, a grammar writer may ask the following questions while writing a grammar.

- Are all the rules distinct? (e.g. `Para` and `De` may be included in `Prep`)
- Could two or more rules be merged? (e.g. `SELECT Inf IF -1 Prep OR Vai OR Vbmod ...`)
- What is the best order for the rules?
- Generate a sequence that triggers rule(s) R but not rule(s) R'.

For the second use case, here are examples of conflicts that our tools will detect.

- If a rule appears twice, the second occurrence will be disabled by the first
- R selects something in a context, R' removes it
- R removes something from the context of R', so R' can never apply
- R has an internal conflict, such as non-existent tag combination, or contradictory requirements for a context word

R can also be a set of rules: for instance, if one rule removes a verb in context C, and another in context $\neg C$, together these rules remove a verb in all possible cases, disabling any future rule that targets verbs.

While rule-internal conflicts can be detected by simpler means, taking care of rule interaction requires a *semantic* rather than a *syntactic* analysis. In order to find effects of rule interaction, we must keep track of the possible sentences at each step. After each rule, we have two possibilities: the rule fires, or it does not fire. In case the rule does not fire, we have again two options: either its conditions are not met, or its target is the only remaining analysis.

We express these requirements as a *Boolean satisfiability problem* (SAT). SAT problems consist of two components: a set of Boolean variables, and a set of clauses on those variables. For instance, let the set be $\{a, b\}$ and the formulas $\{a \vee b, \neg a\}$. A program called *SAT solver* will try to find a solution, where all the variables have a value. For this particular problem, the unique solution is $a = False, b = True$. It is also possible for a SAT problem to have no solution, or multiple solutions.

4.1 Implementation

In this section, we describe the implementation of the tool. The SAT-encoding we use is similar to the one introduced in [21], with one key difference: in this paper, we operate on *symbolic sentences* instead of concrete sentences from a corpus. The idea is that the SAT-solver is going to find the concrete sentence for us.

Preliminaries Our analysis operates on one rule R, and is concerned with answering the following question: "Does there exist an input sentence S that can trigger rule R, even after passing all rules R' that came before R?"

Before we can do any analysis any of the rules, we need to find out what the set of all possible readings of a word is. We can do this by extracting this information from a lexicon, but there are other ways too. In our experiments, the number of readings has ranged from about 300 to about 6000.

Furthermore, when we analyse a rule R, we need to decide the *width* $w(R)$ of the rule R: How many different words should there be in a sentence that can trigger R? Most often, $w(R)$ can be easily determined by looking at how far away the rule context indexes in the sentence relative to the target. For example, in the rule mentioned in the introduction, the width is 2.

If the context contains a *, we may need to make an approximation of $w(R)$ which may result in false negatives later on in the analysis.

Symbolic sentences We start each analysis by creating a so-called *symbolic sentence*, which is our representation of the sentence S we are looking for. A symbolic sentence is a sequence of *symbolic words*; a symbolic word is a table of all possible readings that a word can have, where each reading is paired up with a SAT-variable.

The number of words in the symbolic sentence we create when we analyse a rule R is $w(R)$. For the rule in the introduction, we have $w(R) = 2$ and a symbolic sentence may look as follows:

word1	word2	reading
v_1	w_1	det def
v_2	w_2	noun sg
v_3	w_3	noun pl
v_4	w_4	verb sg
v_5	w_5	verb pl

Here, v_i and w_j are SAT-variables belonging to word1 and word2, respectively. We can also see that the possible number of readings here was 5.

The SAT-solver contains extra constraints about the variables. Input sentences should have at least one reading per word, so we add the following two constraints:

$$v_1 \vee v_2 \vee v_3 \vee v_4 \vee v_5,$$
$$w_1 \vee w_2 \vee w_3 \vee w_4 \vee w_5$$

Any solution to the constraints found by the SAT-solver can be interpreted as a concrete sentence with $w(R)$ words that each have a set of readings.

Applying a rule Next, we need to be able to apply a given rule R' to a symbolic sentence, resulting in a new symbolic sentence.

For example, if we apply the rule from the introduction to the symbolic sentence above, the result is the following symbolic sentence:

word1	word2	reading
v_1	w_1	det def
v_2	w_2	noun sg
v_3	w_3	noun pl
v_4	w'_4	verb sg
v_5	w'_5	verb pl

The example rule can only affect readings of word2 that have a "verb" tag, so we create only two new variables w'_4 and w'_5 for the result, and reuse the other variables. We add the following constraint for w'_4:

$$w'_4 \Leftrightarrow [w_4 \wedge \neg(v_1 \wedge (w_1 \vee w_2 \vee w_3))]$$

In other words, after applying the rule, the reading "verb sg" (represented by the variable w'_4) can only be in the resulting sentence exactly when (1) "verb sg" was a reading of the input sentence (so w_4 is true) and (2) the rule has not been triggered (the rule triggers when v_1 is true and at least one of the non-verb readings $w_1 \ldots w_3$ is true). We add a similar constraint for the new variable w'_5:

$$w'_5 \Leftrightarrow [w_5 \wedge \neg(v_1 \wedge (w_1 \vee w_2 \vee w_3))]$$

Putting it all together Once we know how to apply one rule R' to a symbolic sentence, we can apply all rules preceding the rule R that is under analysis. We simply apply each rule to the result of applying the previous rule. In this way, we end up with a symbolic sentence that represents all sentences that could be the result of applying all those rules.

Finally, we can take a look at the rule R we want to analyse. Here is an example:

```
REMOVE det IF (1 verb) ;
```

If we take the symbolic sentence above as input, we want to ask whether or not it can trigger the rule R. We do this by adding some more constraints to the SAT-solver.

First, the context of the rule should be applicable, meaning that the second word should have a reading with a "verb" tag:

$$w'_4 \vee w'_5$$

Second, the rule should be able to remove the "det" tag, meaning that the first word should have a reading with a "det" tag, and there should be at least one other reading:

$$v_1 \land (v_2 \lor v_3 \lor v_4 \lor v_5)$$

If the SAT-solver can find a solution to all constraints generated so far, we have found a concrete sentence that satisfies our goal. If the SAT-solver cannot find a solution, it means that there are no sentences that can ever trigger rule R. This means that there is something wrong with the grammar.

Creating realistic readings Earlier we have shown an example with 5 readings ("det def", "noun sg", ...). In a realistic case, we operate between hundreds and thousands of readings. In order to find the set of readings, we expand a morphological lexicon[1], ignore the word forms and lemmas, and take all distinct analyses. However, many grammar rules target a specific lemma or word form. A simple solution is to retain the lemmas and word forms only for those entries where it is specified in the grammar, and otherwise leave them out. For example, the Dutch grammar contains the following rule:

```
REMOVE ("zijn" vbser) IF (-1 Prep) (1 Noun) ;
```

This hints that there is something special about the verb *zijn*, compared to the other verbs. Looking at the lexicon, we find *zijn* in the following entries:

```
zijn:zijn<det><pos><mfn><pl>
zijn:zijn<det><pos><mfn><sg>
zijn:zijn<vbser><inf>
zijn:zijn<vbser><pres><pl>
```

Thus we add special entries for these: in addition to the anonymous <det><pos><mfn><pl> reading, we add <"zijn"><det><pos><mfn><pl>. The lemma is treated as just another tag.

However, for languages with more readings, this may not be feasible. For instance, Spanish has a high number of readings, not only because of many inflectional forms, but because it is possible to add 1–2 clitics to the verb forms. The number of verb readings without clitics is 213, and with clitics 1572. With the previously mentioned approach, we would have to double 1572 entries for each verb lemma. Even ignoring the clitics, each verb lemma would still result in 213 new readings.

[1] We used the lexica from Apertium, found in https://svn.code.sf.net/p/apertium/svn/languages/.

In our experience, even ignoring the clitics doubles the amount of readings.

Another note: the readings in grammar can be underspecified (e.g. `verb sg`), whereas the lexicon only gives us fully specified (`verb pres p2 sg`) readings. We tried a version where we took the tag combinations specified in the grammar as readings, and we could insert them into the symbolic sentences as well, but this was not an ideal solution: turns out that the tag lists in the grammars contain often errors (e.g. replacing OR with an AND; using a nonexistent tag; using a wrong level in a subreading), and if we accept those lists as readings, we will generate symbolic sentences that are impossible, and won't discover the bug in the grammar.

However, if we only want to find rule interaction effects, then using the underspecified readings from the grammar makes the task faster, and it will still catch potential interaction errors.

Creating realistic ambiguities In the previous section, we have created realistic *readings*, by simply hardcoding legal tag combinations into variables. The next step in creating realistic ambiguities is to constrain what readings can go together. For instance, the case of *zijn* shows us that "determiner or verb", is a possible ambiguity. In contrast, there is no word form in the lexicon that would be ambiguous between an adjective and a comma, hence we don't want to generate such ambiguity in our symbolic sentences.

	n nt sg	n f pl	vblex sep inf	det pos mfn
uitgaven	0	1	1	0
toespraken	0	1	1	0
haar	1	0	0	1

We solve the problem by creating *ambiguity classes*: groups of analyses that can be ambiguous with each other. We represent the expanded morphological lexicon as a matrix, as seen in figure **[TODO:** make a nice picture with rows and columns**]**: word forms on the rows and analyses on the columns. Each distinct row forms an ambiguity class. For example, the ambiguity class `[1257,496,16]` contains masculine plural adjectives and masculine plural past participles. Then we form SAT clauses that allow or prohibit certain combinations. These clauses will interact with the constraints created from the rules, and the end result will be closer to real-life sentences.

Our approach is similar to [10], who use ambiguity classes instead of distinct word forms, in order to reduce the number of parameters in a Hidden Markov Model. They take advantage of the fact that they don't have to model "bear" and "wish" as separate entries, but they can just reduce it to "word that can be ambiguous between noun and verb", and use it as a parameter in their HMM. We can do a similar thing by saving a list of words with each ambiguity class. For example, we map the ambiguity class "feminine plural noun or a separable verb infinitive" to the list of word forms {"uitgaven", "toespraken"}, and then, if we generate such reading for our symbolic word, we can give one of these words as an example word.

There are two advantages of restricting the ambiguity within words. Firstly, we can create more realistic example sentences, which should help the grammar writer. Secondly, we can possibly detect some more conflicts. Assume that the grammar contains the following rules:

```
REMOVE adj IF (-1 aux) ;
REMOVE pp  IF (-1 aux) ;
```

With our symbolic sentence, these rules will be no problem; to apply the latter, we only need to construct a target that has a realistic ambiguity with a past participle; the adjective will be gone already. However, it could be that past participles (pp) only ever get confused with adjectives—in that case, the above rules would be in conflict with each other. By removing the adjective reading, the first rule selects the past participle reading, making it an instance of "R selects something in a context, R' removes it". The additional constraints will prevent the SAT-solver from creating an ambiguity outside the allowed classes, and such a case would be caught as a conflict.

Bibliography

[1] Andrei Sfrent. 2014. *Machine Learning of Rules for Part of Speech Tagging*. M.Phil. thesis, Imperial College London, United Kingdom.

[2] Antonsen, Lene, & Trosterud, Trond. 2011. Next to nothing – a cheap South Saami disambiguator. *In: Proceedings of the Constraint Grammar workshop at NODALIDA*.

[3] Antonsen, Lene, Wiechetek, Linda, & Trosterud, Trond. 2010. Reusing grammatical resources for new languages. *In: Proceedings of the International conference on Language Resources and Evaluation LREC2010*.

[4] Bick, Eckhard. 2006. A Constraint Grammar Parser for Spanish. *In: Proceedings of TIL 2006 - 4th Workshop on Information and Human Language Technology*.

[5] Bick, Eckhard. 2011. Constraint grammar applications. *In: Proceedings of the Constraint Grammar workshop at NODALIDA*.

[6] Bick, Eckhard. 2013. ML-Tuned Constraint Grammars. *In: Proceedings of the 27th Pacific Asia Conference on Language, Information and Computation*.

[7] Bick, Eckhard, & Didriksen, Tino. 2015. CG-3 – Beyond Classical Constraint Grammar. *In: Proceedings of NODALIDA*.

[8] Claessen, Koen, Een, Niklas, Sheeran, Mary, Sörensson, Niklas, Voronov, Alexey, & Åkesson, Knut. 2009. SAT-Solving in Practice, with a Tutorial Example from Supervisory Control. *Discrete Event Dynamic Systems*, **19**(4), 495–524.

[9] Cook, Stephen A. 1971. The Complexity of Theorem-proving Procedures. *In: Proceedings of the Third Annual Symposium on Theory of Computing*.

[10] Cutting, Douglas R., Kupiec, Julian, Pedersen, Jan O., & Sibun, Penelope. 1992. A Practical Part-of-Speech Tagger. *Pages 133–140 of: ANLP*.

[11] Didriksen, Tino. 2014. *Constraint Grammar Manual*. Institute of Language and Communication, University of Southern Denmark.

[12] Eén, Niklas, & Sörensson, Niklas. 2004. An Extensible SAT-solver. *In:* Giunchiglia, Enrico, & Tacchella, Armando (eds), *Theory and Applications of Satisfiability Testing*. Lecture Notes in Computer Science, vol. 2919. Springer Berlin Heidelberg.

[13] Eineborg, Martin, & Lindberg, Nikolaj. 1998. Induction of Constraint Grammar-rules using Progol. *In:* Page, David (ed), *Inductive Logic Programming*. Lecture Notes in Computer Science, vol. 1446. Springer Berlin Heidelberg.

[14] Greene, Barbara B., & Rubin, Gerald M. 1971. *Automatic Grammatical Tagging of English*. Tech. rept. Department of Linguistics, Brown University.

[15] Karlsson, Fred. 1990. Constraint Grammar as a Framework for Parsing Running Text. *Pages 168–173 of: Proceedings of the 13th Conference on Computational Linguistics - Volume 3*. COLING '90. Stroudsburg, PA, USA: Association for Computational Linguistics.

[16] Karlsson, Fred, Voutilainen, Atro, Heikkilä, Juha, & Anttila, Arto. 1995. *Constraint Grammar: a language-independent system for parsing unrestricted text*. Vol. 4. Walter de Gruyter.

[17] Koskenniemi, Kimmo. 1990. Finite-state Parsing and Disambiguation. *In: Proceedings of the 13th Conference on Computational Linguistics - Volume 2*. COLING '90. Association for Computational Linguistics.

[18] Koskenniemi, Kimmo. 1997. Representations and finite-state components in natural language. *Finite-state language processing*, 99–116.

[19] Lager, Torbjörn. 1998. Logic for part of speech tagging and shallow parsing. *In: Proceedings of the 11th Nordic Conference on Computational Linguistics*.

[20] Lager, Torbjörn, & Nivre, Joakim. 2001. Part of speech tagging from a logical point of view. *In: Proceedings of LACL, Logical Aspects of Computational Linguistics*.

[21] Listenmaa, Inari, & Claessen, Koen. 2015. Constraint Grammar as a SAT problem. *In: Proceedings of the Constraint Grammar workshop at NODALIDA*.

[22] Listenmaa, Inari, & Claessen, Koen. 2016. Analysing Constraint Grammars with a SAT-solver. *In: Proceedings of the TODO TODO LREC*.

[23] Marques-Silva, João. 2010. *Boolean Satisfiability Solving: Past, Present & Future*. Presentation given at the Microsoft Research International Workshop on Tractability, Cambridge, UK, July 5–6.

[24] Nemeskey, Dávid Márk, Tyers, Francis, & Hulden, Mans. 2014 (August). Why Implementation Matters: Evaluation of an Open-source Constraint Grammar Parser. *In: Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics*.

[25] Tapanainen, Pasi. 1996. The Constraint Grammar Parser CG-2.

[26] Tapanainen, Pasi. 1999. *Parsing in two frameworks: finite-state and functional dependency grammar*. Ph.D. thesis, University of Helsinki.