Thesis for the Degree of Licentiate of Philosophy

Analysing Constraint Grammar with SAT

Inari Listenmaa

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

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

Analysing Constraint Grammar with SAT
Inari Listenmaa

Technical Report 154L
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 robust and language-independent formalism for part-of-speech tagging and shallow parsing. A grammar consists of disambiguation rules for initially ambiguous, morphologically analysed text: the correct analysis for the sentence is attained by removing improper readings from ambiguous words. Wide-coverage Constraint Grammars, containing some thousands of rules, generally achieve very high accuracy for free text: thanks to the flexible formalism, new rules can be written to address even the rarest phenomena, without compromising the general tendencies. Even with a smaller set of rules, CG-based taggers have been competetive with statistical taggers. In addition, CG has been used in more experimental settings: dependency syntax, dialogue system for language learning and anaphora resolution.

This thesis presents two contributions to the study of CG. Firstly, we model a parallel variant of CG as a Boolean satisfiability (SAT) problem, and implement a CG-engine by 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. Using SAT-solver as a backend, we can exploit decades of research into optimising SAT-solvers. In addition, we suggest two novel alternatives for conflict handling in the parallel scheme.

Secondly, we implement a SAT-based tool for analysing sequential CGs. Sequential CG is the most commonly used variant: rules are applied in order, and actions are irreversible. We apply symbolic evaluation, a common technique in software verification, and model the state of all possible sentences after applying each rule; this enables grammar writers to see whether some rule or rule set prevents others from applying, or request example sequences that trigger a given rule.

Keywords: *Constraint Grammar, Satisfiability, Grammar Analysis*

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, who have guided this work from an afternoon experiment to an actual thesis. I have learnt a lot about research, time management 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 Tino Didriksen, Tommi Pirinen and other nice people on `#hfst`. On the computational side of computational linguistics, Wen Kokke and Anssi Yli-Jyrä have sparked my interest in the expressivity of CG.

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* [28], 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 the analyses which are most adequate in the current context 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.

1.1 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 [28]

Throughout its history, CG has been very practically oriented, with few studies on its formal properties, or attempts to relate it to existing frameworks. Notable exceptions are Torbjörn Lager and Joakim Nivre [34], who model CG in logic, and Pasi Tapanainen [51], who defines the expressivity of rules. 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

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

the word forms are translated into Boolean variables, and the constraint rules into logical formulas operating on those variables. 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.

1.2 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 [55]

The most important contribution of this thesis is, however, practical. The SAT-encoding enables us to find out if there are rules that contradict other rules. We do this by using the generation property: for a given set of rules, we ask for a sentence that would pass through all of them and still be able to trigger the last one. If such sentence cannot be created, it means that some rule prevents the last rule from applying, regardless of the input sentence; for instance, if an earlier rule removes all verbs unconditionally, then any rule that removes verbs in some specific context may not apply.

In addition to finding conflicts, we have found the generation property to be helpful in the process of grammar writing, to give grammarians feedback on the rules under development. 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.

1.3 Structure of this thesis

The core of this thesis is an extension of two articles: "Constraint Grammar as a SAT problem" [35] and "Analysing Constraint Grammars with a SAT-solver" [36]. 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.

1.4 Contributions of the author

Both articles [35] and [36] are co-written by the author and Koen Claessen. In general, the ideas behind the publications were joint effort. For the first article, all of the implementation of SAT-CG is by the author, and all of the library SAT+ by Koen Claessen. The version appearing in this monograph, Chapter 3, is thoroughly rewritten by the author since the initial publication.

For the second article, the author of this work was in charge of all the implementation, except for the ambiguity class constraints, which were written by Koen Claessen. The version appearing in Chapter 4 is to large extent the same as the original article, with added detail and examples throughout the chapter.

Chapter 2

Background

In this chapter, we present the two components of this thesis: Constraint Grammar and SAT. Section 2.1 broadly introduces the CG formalism and some of the different variants, along with the historical context and related work. For a description of any specific CG implementation, we direct the reader to the original source: CG-1 [27, 28], CG-2 [50], VISL CG-3 [7, 17]. Section 2.4 is a brief and high-level introduction to SAT, aimed to a reader with no prior knowledge on SAT. For a more thorough description, we recommend [8].

2.1 Introduction to CG

Constraint Grammar (CG) is a formalism for disambiguating morphologically analysed text. It was first introduced by Fred Karlsson [27, 28], and has been used for many tasks in computational linguistics, such as part-of-speech tagging, surface syntax and machine translation [5]. CG-based taggers are reported to achieve F-scores of over 99 % for morphological disambiguation, and around 95-97 % for syntactic analysis [2, 3, 4]. CG disambiguates morphologically analysed input 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>"                    "<sleeps>"
       "the" det def               "sleep" noun pl
"<bear>"                           "sleep" verb pres p3 sg
       "bear" noun sg      "<.>"
       "bear" verb pres            "." sent
       "bear" verb inf
```

Figure 2.1: Ambiguous sentence *the bear sleeps.*

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 marker means *careful* context: e.g. `(-1C noun)` requires the previous word to be unambiguously noun. Such a 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*$_{@\text{DN>}}$ *little*$_{@\text{AN>}}$ *dog*$_{@\text{HEAD-N}}$. Using <TAG and TAG> can identify chunks, as long as they are not disconnected.

With the introduction of VISL CG-3 [17, 7], it is possible to add full-fledged dependency relations, such as `"<the>" "the" det def IND=1 PARENT=2`. However, for the remainder of 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 in particular are presented in Section 2.3.

Disambiguation using constraint rules dates back to 1960s and 1970s—the closest system to modern CG was Taggit [22], which was used to tag the Brown Corpus. Karlsson [28] lists various approaches to the disambiguation problem, including manual intervention, statistical optimisation, unification and Lambek calculus. For disambiguation rules based on local constraints, Karlsson mentions [25, 24].

CG itself was introduced in 1990. Around the same time, a related formalism was proposed: finite-state parsing and disambiguation system using constraint rules [30], which was later named Finite-State Intersection Grammar (FSIG) in [42]. Like CG, a FSIG grammar contains a set of rules which remove impossible readings based on contextual tests, in a parallel manner: a sentence must satisfy all individual rules in a given FSIG. Due to these similarities, the name Parallel Constraint Grammar was also suggested [31]. However, FSIG rules usually target syntactic phenomena, and the rules allow for more expressive contextual tests than CG. In this thesis, we use the name FSIG to refer to the framework that is aimed at producing a full syntactic analysis with the more expressive rules, and PCG to describe any CG-variant which happens to be parallel. Thus, we will call implementations such as [53] an *FSIG grammar* and a program that parses it an *FSIG parser*. Conversely, the CG parser in [35] and the "CG-like" system in [32] are instances of PCG. We return to the comparison with FSIG in Sections 2.3.3 and 2.3.4.

Brill tagging [11] is based on transformation rules: the starting point of an analysis is just one tag, the most common one, and subsequent rule applications transform one tag into another, based on contextual tests. Like CG, Brill tagging is known to be efficient and accurate. The contextual tests are very similar; [33] has automatically learned both Brill rules and CG rules, using the same system.

Similar ideas to CG have been also explored in other frameworks, such as finite-state automata [23, 21], logic programming [39, 32], constraint satisfaction [40], and dependency syntax [52]. In addition, there are a number of reimplementations of CG using finite-state methods [61, 26, 41].

2.3 Properties of Constraint Grammar

Karlsson [28] 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 description.

CG is a *reductionistic* system: the analysis starts from a list of alternatives, and removes those which are impossible or improbable. CG is 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 which apply their own heuristics, or remove the ordering completely, applying the rules in parallel.

In addition, we will discuss the *expressivity* of CG. We present earlier work on defining expressivity in CG and similar formalisms, and suggest a way to emulate generation; this would allow us to place CG in the Chomsky hierarchy, and better relate to other grammar formalisms.

2.3.1 Reductionistic

CG analysis starts from a set of alternative readings, 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—Karlsson [28] 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, Eckhard Bick and Tino Didriksen [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 expressivity 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 successfully disambiguate *en* in this case. 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 $en_{\text{Det Utr}}$ $bord_{\text{N Neutr}}$ is undeniably ungrammatical; however, the alternative $en_{\text{N Utr}}$ $bord_{\text{N Neutr}}$ would be even "more wrong" for the purpose of analysing the text.

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

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 individual lexemes 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. Karlsson [28] 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, Karlsson [28] argues CG to be a reasonable psycholinguistic hypothesis: in generative formalisms, rule applications create ambiguities, whereas in CG, rule applications reduce ambiguity. This means less processing load in order to achieve clarity.

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 for morphological disambiguation 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 removes a 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*). There is a template feature, present already in CG-1, which allows to list alternatives under a single name, such as "`det n`", "`det adj n`" and "`adj n`" under `NP`. But this is just syntactic sugar: one rule with a condition `IF -1 NP` is expanded into three rules, with the respective conditions `IF (-2 det) (-1 n)`, `IF (-3 det) (-2 adj) (-1 n)` and `IF (-2 adj) (-1 n)`.

Finally, some rules may be purely lexical, such as `REMOVE v IF (-1 "bear")`, and some rules may target the whole class of nouns, or something in between, such as all finite verb forms. 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 and execution of the rules

The previous properties of Constraint Grammar formalism and rules were specified in [28], 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 question is that*":

```
"<what>"            "<question>"            "<is>"            "<that>"
       "what" det           "question" noun          "be" verb          "that" det
       "what" pron          "question" verb                             "that" rel
```

	Strict	**Heuristic**	**Unordered**
Sequential	CG-1 [27] VISL CG-3 [17] Peltonen 2011 [41] Yli-Jyrä 2011 [61] Huldén 2011 [26]	CG-2 [50] Weighted CG-3 [43]	–
Parallel	SAT-CG$_{\text{Ord}}$	SAT-CG$_{\text{Max}}$ *FSIG (Voutilainen) [53]* *Voting constraints [39]*	*Lager 1998 [32]* *FSIG (Koskenniemi) [30]*

Table 2.1: Combinations of rule ordering and execution strategy.

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 v IF (-1 det)` and `REMOVE n IF (-1 pron)` 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 a long list 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 n IF (-1 det) (1 v)`, even if this rule appears last in the file, it would be preferred to the shorter rules, because it is a more exact match. There are also methods that use explicit weights to favour certain rules, such as [43] for CG, and [53, 39, 49] for related formalisms.

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, CG-1 [27] and VISL CG-3 [17] follow the strict scheme, whereas CG-2 [50] is heuristic[2].

[2]Note that CG-2 is heuristic *within sections*: the rules in a given section are executed heuristically, but all of them will be applied before any rule in a later section.

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 v IF (-1 det)`.

In contrast, a *parallel execution* strategy disambiguates all the words at the same time, using their initial, ambiguous context. To give a minimal example, assume we have a single rule, `REMOVE verb IF (-1 verb)`, and the three words *can can can*, shown below. In parallel execution, both can_2 and can_3 lose their verb tag; in sequential only can_2.

```
"<can1>"          "<can2>"          "<can3>"
      "can" noun        "can" noun        "can" noun
      "can" verb        "can" verb        "can" verb
```

The question of parallel execution becomes more complicated if there are multiple rules that apply for the same context. Both `REMOVE v IF (-1 det)` and `REMOVE n IF (-1 pron)` would match *question*, because the original input from the morphological analyser contains both determiner and pronoun as the preceding word. The result depends on various details: shall all the rules also act in parallel? If we allow rules to be ordered, then the result will not be any different from the same grammar in sequential execution; that is, the later rule (later by any metric) will not apply. The only difference is the reason why not: "context does not match" in sequential, and "do not remove the last reading" in parallel.

However, usually parallel execution is combined with *unordered* rules. In order to express the result of these two rules in an unordered scheme, we need a concept that has not been discussed so far, namely, disjunction: "the allowed combinations are *det*+*n* or *pron*+*v*". If we wanted to keep the purely list-based ontology of CG, but combine it with a parallel and unordered execution, then the result would have to inconclusive and keep both readings; both cannot be removed because that would leave *question* without any readings. The difference between the list-based and disjunction-based ontologies, corresponding to CG and FSIG respectively, is explained with further detail in [34].

Table 2.1 shows different systems of the constraint rule family, with rule order (strict vs. heuristic) on one axis, and execution strategy (sequential vs. parallel) on other. Traditional CG implementations are shown in a normal font; other, related systems in cursive font and lighter colour. SAT-$CG_{Max}$and SAT-CG_{Ord}refer to the systems by the author; they are presented in [35] and in Chapter 3 of this thesis.

2.3.4 Expressivity and complexity of CG

So far, we have approached CG from a practical and language-oriented point of view. But grammar formalisms are interesting also from a computational perspective. For the theoretically inclined, making a description of a formalism more precise is an end of its own. However, even a practically oriented reader should appreciate the applications: better understanding of a formalism may lead to novel uses and implementation techniques.

The standard measure of formal languages is the Chomsky hierarchy [12], with its four classes of grammars and languages, in order from most expressive to least expressive: *recursively enumerable* (Type 0), *context-sensitive* (Type 1), *context-free* (Type 2), and *regular* (Type 3). The notion of *expressive power*, "which constructs can we express in the language" is coupled with *parsing complexity*, "how much time and memory do we need to parse sentences in the language"; more expressive power corresponds to greater parsing complexity.

However, the Chomsky hierarchy is defined for generative grammar formalisms, hence it is difficult to place a reductionistic formalism such as CG into it. Especially the question of expressivity is problematic in the context of CG. A given grammar in CG does not describe a language, but a *relation* between an input language and an output language. In addition, we need to consider a number of implementation features: as we have learnt, there are many variants of CG, with a different take on rule ordering, execution strategy, and supported operations. Anssi Yli-Jyrä (personal communication) suggests that the expressivity should be defined separately for each grammar, and not for the complete formalism.

In contrast, parsing complexity can be easily defined for a given variant and implementation of CG. For instance, Pasi Tapanainen [51] and Måns Huldén [26] analyse their CG-2 parser implementations in terms of the sentence length n; the number of rules in the grammar G; and the maximal number of different readings per cohort k. Tapanainen's parser has the worst-case complexity $O(Gn^3k^2)$, and Huldén's parser $O(Gn^2k^2)$; even though both systems can run the same CG-2 style grammars[3], the complexities of the respective parsers are different, due to different implementation techniques.

Even if CG is not a generative formalism, the concepts around Chomsky hierarchy play an important role, for instance, when defining what a single rule can express. 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.

[3] We assume both systems give the same output for the same input—at least the opposite has not been reported.

Expressivity of a rule Tapanainen [51] defines the notion *finite-state constraint language*: a formalism for writing disambiguation rules, where the contextual tests 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 a single rule such as "remove verb if the previous segment contains only well-nested parentheses".

Tapanainen gives a hierarchy of four different constraint languages, in order of the expressivity in a single rule: Taggit [22] only allows rules to look at 1-2 tokens before or after the target; CG-1 [27] 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, for instance the notion of barrier. According to Tapanainen [51], a contextual test such as $(\Sigma - N) * A(\Sigma - V) * N$[4] is not supported in CG-1; in turn, CG-2 [50] can express it, but not e.g. repeated patterns, such as $(Det\ A\ N)+$. Finally, FSIG [30] allows the full expressivity of regular languages in the contextual tests.

Since CG-3 [17, 7] did not exist at the time, [51] does not include it in the hierarchy of the constraint formalisms. The formalism has added some new constructions to the morphological disambiguation rules, such as CBARRIER for requiring the barrier to be unambiguously tagged, or subreadings, for distinguishing different levels of tags within one reading. It is unclear whether these additions make the REMOVE and SELECT rules of VISL CG-3 strictly more expressive than those of CG-2, or if they just provide syntactic convenience. [5]

Expressivity of a grammar The expressivity of a single rule is well defined. However, the expressivity of the whole grammar is harder to define. To our knowledge, there is no work that tries to answer the latter question for CG; however, the question has been explored for other reductionist formalisms.

Anssi Yli-Jyrä [60] investigates the computational complexity of the related formalism FSIG; similarly to CG, it can be seen as a description of a relation rather than a language [31]. The main results are twofold: on the level of output, Yli-Jyrä shows that an individual FSIG grammar can approximate context-free grammars [58] and dependency grammars [59]. On the level of input, [57] shows that all the contextual tests for the English FSIG grammar [53] can be converted into *star-free regular expressions*; a more restricted set within the class of regular languages.

[4] IF (*1 A BARRIER N LINK *1 N BARRIER V)

[5] There are many new rule types in VISL CG-3, such as ADDCOHORT, REMCOHORT, MOVE, JUMP, which on the whole make the formalism uncomparable to CG-1 and CG-2. Same holds for the dependency features.

Input	Output
"<W>"	"<W>"
"a"	"a"
"b"	
"<W>"	"<W>"
"a"	"a"
"b"	
"<W>"	"<W>"
"a"	"b"
"b"	
"<W>"	"<W>"
"a"	"b"
"b"	

Figure 2.2: Recognising the context-free language $a^n b^n$ in CG.

As for another related formalism, Pasi Tapanainen [51] demonstrates the expressive power of Functional Dependency Grammar [52]. Instead of applying rules only to natural language texts, Tapanainen shows two examples of FDG grammars that can express structures beyond regular: the first grammar can successfully create a dependency structure for the context-free language of balanced parentheses, and the second grammar correctly parses the context-sensitive language $a^n b^n c^n$.

We can sketch some requirements for a similar experiment for CGs. 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 in the alphabet Σ. Figure 2.2 shows the input and output of a hypothetical CG, which takes cohorts of $\Sigma = \{a, b\}$ as an input, and removes b from the first n words, and a from the last n words. For strings of uneven length, the grammar would either leave it ambiguous, which would be interpreted as a sign of rejection, or use the REMCOHORT operation of VISL CG-3, and make the whole string empty.

Writing such grammars, or proving they cannot be written, is out of scope of this thesis. However, we wish to bring to attention the concept maximally ambiguous sentence—which will be made use of in Chapter 4—as a way of generating sequences with CG, and better answering the question about expressivity. The hypothesis is that if the input language is always $\Sigma*$, then we may be able to reason more easily about the grammar itself. If we can achieve more precise definition what CG grammars can express, it may open new possibilities: for instance, automatically deriving CGs from other grammar formalisms.

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 to live with 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 due to their restrictions, 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 cohabiting restrictions of each animal translate into *clauses*, such that "dog wants to live with cat" becomes an implication $dog \Rightarrow cat$. Under the hood, all of these implications are translated into even simpler constructs: lists of disjunctions. For instance, "dog wants to live with cat" as a disjunction is $\neg dog \vee cat$, which means "I don't buy the dog or I buy the cat". The representation as disjunctions is easier to handle algorithmically; however, for the rest of this thesis, we show our examples as implications, because they are easier to understand. The variables and the clauses are shown in Figure 2.3.

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 example of an *NP-complete* (Nondeterministic Polynomial time) problem [15]: for all such problems, a potential solution can be *verified* in polynomial time, but there is no known

Variable	Constraint	Explanation
	$ant \vee bat \vee cat \vee dog \vee fox$	"You want to buy at least one pet."
ant	$ant \Rightarrow \neg cat \wedge \neg dog \wedge \neg fox$	"Ant does not like four-legged animals."
bat	$bat \Rightarrow \neg ant \wedge \neg emu$	"Bat does not like herbivores."
cat	$cat \Rightarrow bat$	"Cat wants to live with bat."
dog	$dog \Rightarrow cat$	"Dog wants to live with cat."
emu	$emu \Rightarrow \neg ant \wedge \neg bat$	"Emu does not like small animals."
fox	$fox \Rightarrow \neg(bat \wedge emu)$	"Fox cannot live with two winged animals."

Figure 2.3: Animals' cohabiting constraints translated into a SAT-problem.

algorithm that would *find* such a solution, in general case, in sub-exponential time. This equivalence means that we can express any NP-complete problem as a SAT-instance, and use a SAT-solver to solve it. The class includes problems which are much harder than the animal example; nevertheless, all of them can be reduced into the same representation, just like $\neg bat \vee \neg emu$.

The first decades of SAT-research was concentrated on the theoretical side, with little practical applications. But things changed in the 90s: there was a breakthrough in the SAT-solving techniques, which allowed for scaling up and finding new use cases. As a result, modern SAT-solvers can deal with problems that have hundreds of thousands of variables and millions of clauses [37].

What was behind these advances? SAT, as a general problem, remains NP-complete: it is true that there are still SAT-problems that cannot be solved in sub-exponential time. However, there is a difference between a general case, where the algorithm must be prepared for any input, and an "easy case", where we can expect some helpful properties from the input. Think of a sorting algorithm: in the general case, it is given truly random lists, and in the "easy case", it mostly gets lists with some kind of structure, such as half sorted, reversed, or containing bounded values. The general time complexity of sorting is still $O(n \log n)$, but arguably, the easier cases can be expected to behave in linear time, and we can even design heuristic sorting algorithms that exploit those properties.

Analogously, the 90s breakthrough was due to the discovery of right kind of heuristics. Much of the SAT-research in the last two decades has been devoted to optimising the solving algorithms, and finding more efficient methods of encoding various real-life problems into SAT. This development has led to an increasing amount of use cases since the early 2000s

[13]. One of the biggest success stories for a SAT-application is model checking [48, 9, 10], used in software and hardware verification. Furthermore, SAT has been used in domains such as computational biology [14] and AI planning [46], just to pick a few examples. In summary, formulating a decision problem in SAT is an attractive approach: instead of developing search heuristics for each problem 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

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 Torbjörn Lager and Joakim Nivre [32, 34], but there has not been, to our knowledge, a full logic-based CG implementation; at the time, logic programming was too slow to be used for tagging or parsing. Since those works, SAT-solving techniques have improved significantly [37], and they are used in domains such as microprocessor design and computational biology—these problems easily match or exceed CG in complexity. In addition, SAT-solving brings us more practical tools, such as maximisation, which enables us to implement a novel conflict resolution method for parallel CG.

The content in this chapter is based on "Constraint Grammar as a SAT problem" [35]. As in the original paper, we present a translation of CG rules into logical formulas, and show how to encode it into a SAT-problem. This work is implemented as an open-source software SAT-CG[1]. It uses the high-level library SAT+[2], which is based on MiniSAT [18]. We evaluate SAT-CG against the state of the art, VISL CG-3. The experimental setup is the same, but we ran the tests again for this thesis: since the writing of [35], we have optimised our program and fixed some bugs; this makes both execution time and F-scores better than we report in the earlier paper.

[1] https://github.com/inariksit/cgsat
[2] https://github.com/koengit/satplus

3.1 Related work

Our work is inspired by previous approaches of encoding CG in logic [32, 34]. Lager [32] presents a "CG-like, shallow and reductionist system" translated into a disjunctive logic program. Lager and Nivre [34] build on that in a study which reconstructs four different formalisms in first-order logic. CG is contrasted with Finite-State Intersection Grammar (FSIG) [30] and Brill tagging [11]; all three work on a set of constraint rules which modify the initially ambiguous input, but with some crucial differences. On a related note, Yli-Jyrä [62] explores the structural correspondence between FSIG and constraint-solving problems. In addition, logic programming has been applied for automatically inducing CG rules from tagged corpora [19, 47, 33].

3.2 CG as a SAT-problem

In this section, we translate the disambiguation of a sentence into a SAT-problem. We demonstrate our encoding with an example in Spanish, shown in Figure 3.1: *la casa grande*. The first word, *la*, is ambiguous between a definite article ('the') or an object pronoun ('her'), and the second word, *casa*, can be a noun ('house') or a verb ('(he/she) marries'). The subsegment *la casa* alone can be either a noun phrase, $la_{\text{DET}}\ casa_{\text{N}}$ 'the house' or a verb phrase $la_{\text{PRN}}\ casa_{\text{V}}$ '(he/she) marries her'. However, the unambiguous adjective, *grande* ('big'), disambiguates the whole segment into a noun phrase: 'the big house'. Firstly, we translate input sentences into variables and rules into clauses. Secondly, we disambiguate the sentence by asking for a solution. Finally, we consider different ordering schemes and conflict handling.

3.2.1 Encoding the input

Original analysis	Variables	Default rule
`"<la>"`		"do not remove the last reading"
`"el" det def f sg`	la_{DET}	
`"lo" prn p3 f sg`	la_{PRN}	$la_{\text{DET}} \vee la_{\text{PRN}}$
`"<casa>"`		
`"casa" n f sg`	$casa_{\text{N}}$	
`"casar" v pri p3 sg`	$casa_{\text{V}}$	$casa_{\text{N}} \vee casa_{\text{V}}$
`"<grande>"`		
`"grande" adj mf sg`	$grande_{\text{ADJ}}$	$grande_{\text{ADJ}}$

Figure 3.1: Ambiguous segment in Spanish: translation into SAT-variables.

Reading The readings of the word forms make a natural basis for variables. We translate a combination of a word form and a reading, such as `"<la>" ["el" det def f sg]`, into a variable la_{DET}, which represents the possibility that *la* is a determiner. This example segment gives us five variables: $\{la_{\text{DET}}, la_{\text{PRN}}, casa_{\text{N}}, casa_{\text{V}}, grande_{\text{ADJ}}\}$, shown in 3.1.

Cohort As in the original input, the readings are grouped together in cohorts. We need to keep this distinction, for instance, to model SELECT rules and cautious context: `SELECT "casa" n` means, in effect, "remove $casa_{\text{V}}$", and `IF (-1C prn)` means "if la_{PRN} is true and la_{DET} false". Most importantly, we need to make sure that the last reading is not removed. Hence we add the default rule, "do not remove the last reading", as shown in the third column of 3.1. These disjunctions ensure that at least one variable in each cohort must be true.

Sentence In order to match conditions against analyses, the input needs to be structured as a sentence: the cohorts must follow each other like in the original input, indexed by their absolute position in the sentence. Thus when we apply `REMOVE v IF (-1 det)` to the cohort $2 \rightarrow [casa_{\text{N}}, casa_{\text{V}}]$, the condition will match on la_{DET} in cohort 1.

Rule Next, we formulate a rule in SAT. A single rule, such as `REMOVE v IF (-1 det)`, is a template for forming an implication; when given a concrete sentence, it will pick concrete variables by the following algorithm.

1. Match rule against all cohorts

   la: No target found

   casa: Target found in $casa_{\text{V}}$, match conditions to *la*

   - Condition found in la_{DET}
   - Create a clause: $la_{\text{DET}} \Rightarrow \neg casa_{\text{V}}$ 'if *la* is a determiner, *casa* is not a verb'

   grande: No target found

2. Solve with all clauses: $\{\overbrace{la_{\text{DET}} \vee la_{\text{PRN}},\ casa_{\text{N}} \vee casa_{\text{V}},\ grande_{\text{ADJ}}}^{\text{given by the default rule}},\ \overbrace{la_{\text{DET}} \Rightarrow \neg casa_{\text{V}}}^{\texttt{REMOVE v IF (-1 det)}}\}$

In Appendix 3.4 , we have included a translation of all the rule types that SAT-CG supports: REMOVE and SELECT rules, with most of the operations from CG-2 [50], and a couple of features from VISL CG-3 [17]. The following examples in this section do not require reading the appendix.

3.2.2 Applying a rule

Finally, we have all we need to disambiguate the segment: the sentence and the constraints encoded as SAT-variables and clauses. The SAT-solver returns a model that satisfies all the clauses presented in step 2. We started off with all the variables unassigned, and required at least one variable in each cohort to be true. In addition, we gave the clause $la_{\text{DET}} \Rightarrow \neg casa_{\text{V}}$. We can see with a bare eye that this problem will have a solution; in fact, multiple ones, shown in Figure 3.2. The verb analysis is removed in the first two models, as required by the presence of la_{DET}. However, the implication may as well be interpreted "if $casa_{\text{V}}$ may not follow la_{DET}, better remove la_{DET} instead"; this has happened in Models 3–4. We see a third interpretation in Model 5: $casa_{\text{V}}$ may be removed even without the presence of la_{DET}. This is possible, because $la_{\text{DET}} \Rightarrow \neg casa_{\text{V}}$ is only an implication, not an equivalence.

Model 1	**Model 2**	**Model 3**	**Model 4**	**Model 5**
la_{DET}	la_{DET}			
	la_{PRN}	la_{PRN}	la_{PRN}	la_{PRN}
$casa_{\text{N}}$	$casa_{\text{N}}$	$casa_{\text{N}}$		$casa_{\text{N}}$
		$casa_{\text{V}}$	$casa_{\text{V}}$	
$grande_{\text{ADJ}}$	$grande_{\text{ADJ}}$	$grande_{\text{ADJ}}$	$grande_{\text{ADJ}}$	$grande_{\text{ADJ}}$

Figure 3.2: Possible models for `REMOVE v IF (-1 det)`.

It seems like SAT-CG does worse than any standard CG implementation: the latter would just remove the verb, not give 5 different interpretations for a single rule. In fact, the rule `REMOVE v IF (-1 det)` alone behaves exactly like `REMOVE det IF (1 v)`. But there is power to this property. Now, we add a second rule: `REMOVE n IF (-1 prn)`, which will form the clause $la_{\text{PRN}} \Rightarrow \neg casa_{\text{N}}$. The new clause prohibits the combination $la_{\text{PRN}}\ casa_{\text{N}}$, which rules out three models out of five. The disambiguation is shown in Figure 3.3.

After two rules, we only have two models: one with $la_{\text{DET}}\ casa_{\text{N}}$ and other with $la_{\text{PRN}}\ casa_{\text{V}}$. In fact, we have just implemented parallel CG (PCG), introduced in Section 2.3.3: the rules act in parallel, and if the sentence cannot be fully disambiguated, the remaining uncertainty is modelled as a disjunction of all possible combinations of readings. In contrast, a sequential CG (SCG) engine applies each rule individually, and it cannot handle disjunction; its only operation is to manipulate lists of readings in a cohort. The SCG engine would have just applied one of the rules—say, the first one, removed the verb and stopped there. If another rule later in the sequence removes the determiner, there is no way to restore the verb.

Model 1	Model 2
la_{DET}	
	la_{PRN}
$casa_{\text{N}}$	
	$casa_{\text{V}}$
$grande_{\text{ADJ}}$	$grande_{\text{ADJ}}$

Figure 3.3: Possible models for `REMOVE v IF (-1 det)` and `REMOVE n IF (-1 prn)`.

To finish our PCG example, let us add one more rule: `REMOVE v IF (1 adj)`, and the corresponding clause $grande_{\text{ADJ}} \Rightarrow \neg casa_{\text{V}}$. This clause will rule out Model 2 of Figure 3.3, and we will get Model 1 as the unique solution. We can see another benefit in allowing connections between rules: none of the three rules has targeted *la*, still it has become unambiguous.

3.2.3 Solving conflicts in the parallel scheme

As described in Section 2.3.3, PCG behaves differently from SCG: the rules are dependent on each other, and the order does not matter. This prevents too hasty decisions, such as removing $casa_{\text{V}}$ before we know the status of *la*. However, ignoring the order means that we miss significant information in the rule set. The truth is that pure PCG is very brittle: each and every rule in the set must fit together, without the notion of order. The rule sequence in Figure 3.4, taken from a Dutch grammar[3], will be well-behaved in an SCG with strict rule order. The grammar will behave as intended also in a heuristic variant of SCG, because the rules with a longer context are matched first. But in PCG, the rule set will definitely cause a conflict, rendering the whole grammar useless.

The order clearly demonstrates the course of action: "If a potential imperative starts a sentence and is followed by an object pronoun, select the imperative reading; then, move on to other rules; finally, if any imperative is still ambiguous, remove the imperative reading." Comparing the success of SCG to PCG in practical applications, one may speculate that the sequential order is easier to understand—undeniably, its behaviour is more transparent. If two rules target the same cohort, the first mentioned gets to apply, and removes the target. When the first rule has acted, the second rule is not even considered, because it would remove the last reading.

[3] https://svn.code.sf.net/p/apertium/svn/languages/apertium-nld/apertium-nld.nld.rlx

```
SECTION

   # Zeg me
   SELECT Imp IF (-1 BOS) (1 (prn obj)) ;

   # . Heb je
   SELECT (vbhaver pres p2 sg) IF (-1 BOS) (1 (prn obj uns p2 mf sg)) ;

   [--]

SECTION

   # remove all imperative readings that have not been explicitly selected
   REMOVE Imp ;

   # remove informal 2nd person singular reading of "heb"
   REMOVE (vbhaver pres p2 sg) ;
```

Figure 3.4: Example from a Dutch grammar

Ideally, both ways of grammar writing should yield similar results: sequential CG rules are more imperative, and parallel CG rules are more declarative. But the problem of conflicts in PCG still remains. In the following, we present two solutions: in the first one, we emulate ordering in choosing which clauses to keep, and in the second one, we maximise the number of rule applications.

Emulating order We keep the parallel base, but use ordering as information for solving conflicts. This means that all the benefits of parallel execution still hold: the three rules, which all target *casa*, may still disambiguate *la*, without *la* ever being the target. If all the rules play well together, or if the earlier rules do not match any cohorts, then no rule applications need to be removed. However, if we have the grammar from Figure 3.4, and imperative is the right analysis for a given context, then the clauses created by `REMOVE Imp` would be ignored, in favour of the clauses that are created by `SELECT Imp IF (-1 BOS) (1 (prn obj))`.

In this modified scheme, we introduce the clauses to the SAT-solver one by one, and attempt to solve after each clause. If the SAT-problem after the 50^{th} rule has a solution, we accept all the clauses created by rule 50. If rule 51 causes a conflict, we prioritise the previous, well-behaving subset of 50 rules, and discard the conflicting clauses created by rule 51.

If a rule matches multiple cohorts, it creates a separate clause for each instance. Thus, it is no problem if the rule causes a conflict in only one cohort—say, we have another potential imperative in the sentence, but there is no other rule which targets its other readings. We can discard only the conflicting instances: we prevent `REMOVE Imp` from applying to *Zeg* in the sequence *# Zeg me*, but it still may apply to other ambiguous tokens with imperative reading.

Let us demonstrate the procedure with the Spanish segment *la casa grande*. Assuming our rule set is {`REMOVE v IF (-1 det)`, `REMOVE v IF (1 adj)`, `REMOVE n`}, the revised algorithm goes as follows:

1. Apply `REMOVE v IF (-1 det)`
   - Create a clause: $la_{\text{DET}} \Rightarrow \neg casa_{\text{V}}$
   - Solve with previous clauses: $\{\overbrace{la_{\text{DET}} \vee la_{\text{PRN}},\ casa_{\text{N}} \vee casa_{\text{V}},\ grande_{\text{ADJ}}}^{\text{default rule}},\ \overbrace{la_{\text{DET}} \Rightarrow \neg casa_{\text{V}}}^{\texttt{REMOVE v IF (-1 det)}}\}$
   - Solution found: add new clause to the formula
2. Apply `REMOVE v IF (1 adj)`
   - Create a clause: $grande_{\text{ADJ}} \Rightarrow \neg casa_{\text{V}}$
   - Solve with previous clauses: $\{..., la_{\text{DET}} \Rightarrow \neg casa_{\text{V}},\ \overbrace{grande_{\text{ADJ}} \Rightarrow \neg casa_{\text{V}}}^{\texttt{REMOVE v IF (1 adj)}}\}$
   - Solution found: add new clause to the formula
3. Apply `REMOVE n`
   - Create a clause: $\neg casa_{\text{N}}$
   - Solve with previous clauses: $\{..., la_{\text{DET}} \Rightarrow \neg casa_{\text{V}},\ grande_{\text{ADJ}} \Rightarrow \neg casa_{\text{V}},\ \overbrace{\neg casa_{\text{N}}}^{\texttt{REMOVE n}}\}$
   - No solution: discard clause

With this procedure, we use ordering to decide which clauses to include, and then apply all of them in parallel. After going through all the rules, the final formula to the SAT-solver will contain the clauses $la_{\text{DET}} \vee la_{\text{PRN}}$, $casa_{\text{N}} \vee casa_{\text{V}}$, $grande_{\text{ADJ}}$, $la_{\text{DET}} \Rightarrow \neg\ casa_{\text{V}}$ and $grande_{\text{ADJ}} \Rightarrow \neg\ casa_{\text{V}}$.

Maximisation Solving conflicts means that we have multiple rules that target the same reading, and we must choose which rule to apply. Strict ordering substitutes the question with a simpler one: "which rule comes first in the grammar?" Heuristic rule order asks "out of all the rules that target this cohort, which one has the best matching context?" If the competitors are `REMOVE n IF (-1 prn)` and `REMOVE v IF (-1 det) (1 adj)`, then the second one will win. However, if the rules are both as good a match, which happens in Figure 3.3, we need to resort to mere guessing, or fall back to ordering.

However, we can ask yet another 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*?" As opposed to heuristic or weighted approaches [53, 39], here all the individual rule applications are equally important; we just want to find the largest possible subset of rule applications that can act together without conflict. We will explain the procedure in the following.

Each rule application to a concrete cohort produces a clause, and the whole rule set applied to the whole sentence produces a large formula. In an ideal case, all the rules are well-behaved, and the whole formula is satisfiable. However, if the whole formula is unsatisfiable, we may still ask for an assignment that satisfies the maximum number of the clauses; that is, rule applications. 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".

We keep the Spanish segment and the rule set {`REMOVE v IF (-1 det)`, `REMOVE v IF (1 adj)`, `REMOVE n` }. Now the procedure goes as follows:

1. Apply `REMOVE v IF (-1 det)`
   - Create a clause: $la_{\text{DET}} \Rightarrow \neg casa_{\text{V}}$
2. Apply `REMOVE v IF (1 adj)`
   - Create a clause: $grande_{\text{ADJ}} \Rightarrow \neg casa_{\text{V}}$
3. Apply `REMOVE n`
   - Create a clause: $\neg casa_{\text{N}}$
4. Solve with all clauses: $\{\overbrace{la_{\text{DET}} \vee la_{\text{PRN}}, \ldots}^{\text{default rule}}, \overbrace{la_{\text{DET}} \Rightarrow \neg casa_{\text{V}}}^{\texttt{REMOVE v IF (-1 det)}}, \overbrace{grande_{\text{ADJ}} \Rightarrow \neg casa_{\text{V}}}^{\texttt{REMOVE v IF (1 adj)}}, \overbrace{\neg casa_{\text{N}}}^{\texttt{REMOVE n}}\}$
5. No solution for all clauses: try to find a solution that satisfies maximally many rule applications; however, default rule cannot be overridden.

Similarly to the previous, order-based scheme, we create a clause for each instance of the rule application. In the case of a conflict, we can only discard the clauses targeting the offending cohort, but the rule may apply elsewhere in the sentence.

The problem of satisfying maximum amount of clauses is known as *Maximum Satisfiability* (MaxSAT). Whereas SAT is a decision problem, MaxSAT is an optimisation problem. However, optimisation can be expressed as a sequence of decision problems: first, we compute a solution, then we add a constraint "the solution must be better than the one just found", and ask for another one. This process repeats until a better solution cannot be found; then we accept the latest solution.

Now let us define how is one solution "better" than other, by using a simple trick. For each clause c, we create a new variable v. Instead of the original clause, we give the SAT-solver an implication $v \Rightarrow c$. This means that if v is false, the SAT-solver can ignore the actual clause c—the part that comes from the rule application. Conversely, if v is true, then the SAT-solver must handle the original clause. Then, we ask for a solution where maximally many of these vs are true, and the question for improvement becomes "can we make any more of the vs true"? The method of maximising the variables is described in [20].

As a alternative to creating a helper variable, we could also separate the variables into contexts and targets, and maximise the set of contexts: for $la_{\text{DET}} \Rightarrow \neg casa_{\text{V}}$ and $grande_{\text{ADJ}} \Rightarrow \neg casa_{\text{V}}$, maximise the set of $\{la_{\text{DET}}, grande_{\text{ADJ}}\}$. This variant would bring back the distinction between targets and contexts; given the design of most actually used CGs, it may be better suited for a practical implementation.

3.3 Experiments

In this section, we report experiments on the two modifications to the parallel scheme, presented in the previous section. We evaluate the performance against VISL CG-3 in accuracy and running time; in addition, we offer some preliminary observations on the effect of grammar writing.

For these experiments, we implemented another variant for both schemes: we force all the literals that are not targeted by the original rules to be true. This modification destroys the "rules may disambiguate their conditions" property, which we hypothesised to be helpful; however, turns out that this variation slightly outperforms even VISL CG-3 in terms of precision and recall. Conversely, the original SAT-CG scheme, where all variables are left unassigned, fares slightly worse for accuracy. Overall, the accuracy is very low with the

tested grammars, thus it is hard to draw conclusions. As for execution time, all variants of SAT-CG perform significantly worse than VISL CG-3—depending on the sentence length and the number of rules, SAT-CG ranges from 10 times slower to 100 times slower. The results are presented in more detail in the following sections.

3.3.1 Performance against VISL CG-3

	19 rules			**261 rules**		
	F-score		Time	F-score		Time
SAT-CG$_{\text{Max}}$	80.22$^{\text{U}}$ %	83.28$^{\text{F}}$ %	4s	78.15$^{\text{U}}$ %	79.03$^{\text{F}}$ %	8s
SAT-CG$_{\text{Ord}}$	81.14$^{\text{U}}$ %	83.12$^{\text{F}}$ %	4s	79.03$^{\text{U}}$ %	79.17$^{\text{F}}$ %	8s
VISL CG-3	81.52 %		0.39s	78.83 %		0.64s

Table 3.1: F-scores and execution times for the subset of the Spanish grammar, tested on a gold standard corpus of 20,000 words.

We took a manually tagged corpus[4] containing approximately 22,000 words of Spanish news text, and a small constraint grammar[5] from the Apertium repository. We kept only SELECT and REMOVE rules, which left us 261 rules. In order to test the grammars, we produced an ambiguous version of the tagged corpus: we discarded the tags, and analysed the text again with the Apertium morphological analyser and the morphological dictionary for Spanish. Ideally, we should have used the same analyser, but since the gold standard was produced, there have been some changes in the dictionary. Due to these differences, we had to discard some 2000 words from the gold standard—in most cases, there was a change of just one word, but we excluded the whole sentence nevertheless. With this setup, we ran both SAT-CG and VISL CG-3 on the newly ambiguous corpus. In addition, we wrote a small grammar of 19 rules, optimised to be very compact and disambiguate effectively, taking advantage of the parallel execution. Some of the rules were selected from the original grammar, and some were written by the author.

Table 3.1 shows F-scores and execution times for these two grammars. SAT-CG$_{\text{Max}}$ is the parallel scheme with maximisation, and SAT-CG$_{\text{Ord}}$ is the parallel scheme where rule applications are introduced one by one; superscript U refers to the unassigned variant and F to the version where we force non-targeted literals true. The original grammar performs very poorly, with an F-score between 78–79 % for all three setups—even with an empty grammar, the F-score would be around 70 %. Due to the unrepresentative example, we cannot

[4] https://svn.code.sf.net/p/apertium/svn/branches/apertium-swpost/apertium-en-es/es-tagger-data/es.tagged
[5] https://svn.code.sf.net/p/apertium/svn/languages/apertium-spa/apertium-spa.spa.rlx

draw many conclusions. We had access to other grammars of the same size range, such as Portuguese, Dutch and Russian; however, no sufficiently large gold standard corpora were available.

The experiment with the 19-rule grammar was more interesting. We intended to write a grammar that would perform exceptionally well for the parallel execution; it was heavily tuned to the gold standard corpus, and tested only with SAT-CG (leaving all variables unassigned) during development. To our surprise, this small grammar turned out to outperform the original, 261-rule grammar, even when run with VISL CG-3. The rule ordering was not optimised by automatic means, and sections were not used; hence it is possible that with another order, the numbers could be even better for VISL CG-3 and the ordered scheme.

3.3.2 Execution time

	Spanish (380k words)		Finnish (19k words)
	19 rules	**261 rules**	**1185 rules**
SAT-CG$_{\mathbf{Max}}$	25s	1m 5s	4m 56s
SAT-CG$_{\mathbf{Ord}}$	19s	1m 2s	4m 17s
VISL CG-3	2.7s	4.8s	4.3s

Table 3.2: Execution times for Spanish and Finnish grammars of different sizes, disambiguating Don Quijote (384,155 words) and FinnTreeBank (19,097 words).

Number of rules vs. number of words In addition to the 20,000-word corpus of news text, we tested the performance by parsing Don Quijote (384,155 words) with the same Spanish grammars as in the previous experiment. More importantly, we wanted to test a grammar of a realistic size, so we took a Finnish grammar, originally written in 1995 for CG-1, and updated into CG-3 by Tommi Pirinen [43]. We discarded rules and constructions that SAT-CG does not support, and ended up with 1185 rules. Then, we parsed around 19,000 words of text from FinnTreeBank [56]. Table 3.2 shows the results for both Finnish and Spanish tests.

For both systems, the number of rules in the grammar affects the performance more than the raw word count. However, the performance of SAT-CG gets worse faster than VISL CG-3s. From the SAT-solving side, maximisation is the most costly operation. Emulating order performs slightly faster, but still in the same range: maximisation is not needed, but the solve function is performed after each clause—this gets costlier after each rule. However,

	261 rules		1185 rules	
	283 tokens	Split	251 tokens	Split
SAT-CG$_{Max}$	2.26s	2.06s	5.34s	2.36s
SAT-CG$_{Ord}$	2.20s	1.99s	4.70s	2.36s
VISL CG-3	0.05s	0.03s	0.04s	0.04s

Table 3.3: Experiments with sentence length. On the left, Spanish 261-rule grammar parsing the complete 283-token sentence, parsed as one unit vs. split at semicolons into four parts. On the right, Finnish 1185-rule grammar parsing an artificially constructed 251-token sentence.

we believe that SAT is not necessary the bottleneck: VISL CG-3 is a product of years of engineering effort, whereas SAT-CG is still, despite the improvements from [35], a rather naive implementation, written in Haskell and BNFC.

Sentence length In addition, we explored the effect of sentence length further. When split by just full stops, the longest sentence in Don Quijote consists of 283 tokens, including punctuation. In Table 3.3, we see this sentence parsed, on the right as a single unit, and on the left, we split it in four parts, at the semicolons. In Table 3.2, it was already split at semicolons. We tested the effect of sentence length also for the Finnish grammar. All the sentences in FinnTreeBank were very short, so we created an 251-token "sentence" by pasting together 22 short sentences and replacing sentence boundaries with commas.

The results in Table 3.3 are promising: the execution time does not grow unbearably, even with the unusually long sentence. To give some context, let us consider the performances of the sequential CG-2 and the parallel FSIG in 1998. [54] reports the performance of a 3,500-rule CG: "On a 266 MHz Pentium running Linux, EngCG-2 tags around 4,000 words per second". In contrast, a 2,600-rule FSIG grammar is unable to find a correct parse in the allowed time, 100 seconds per sentence, for most sentences longer than 15 words. As another CG result from the late 1990s, [51] reports an average time of 5 seconds for parsing 15-word sentences in CG-2.

Compared to the parsing times of sequential CG and FSIG nearly 20 years ago, our results demonstrate a smaller difference, and much less steep curve in the execution time related to sentence length. Of course, this remark should be taken with caution: to start with, the described experiments were conducted on different systems and different centuries[6]. Importantly, the parallel participants in the experiments were using different grammars—the FSIG grammar used in [54] contains much more difficult operations, which makes the constraint

[6]Eckhard Bick (personal communication) points out that CG-2 run on a modern system is around 6 times faster than VISL CG-3.

problem larger. Our experiments have just shown the lack of blowup for a small grammar. Nevertheless, we feel that this experiment could be interesting to conduct properly: try a SAT-based approach for executing the FSIG grammars from the 1990s, and compare the results to the current state of the art.

3.3.3 Effect on grammar writing

Throughout the implementation of SAT-CG, we have predicted that the more declarative features would influence the way rules are written. We hoped to combine the best parts of SCG and PCG: the rules would be more expressive and declarative, but we could still fall back to heuristics: eventual conflicts would not render the whole grammar useless. 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. Without an actual CG writer, it is hard to say whether this prediction holds or not.

As stated earlier, we got the best F-score with a modification which prevents the rules from disambiguating their context: in the case with *la casa grande* and the three rules which all target *casa*, we would assign both la_{DET} and la_{PRN} true. We tried also a second variant, where we would assign true to only those variables which are neither targets nor conditions to any rule. Table 3.4 shows in more detail what happens to the precision and recall with all the three variants, with the 19-rule grammar.

We notice that the SAT-based solution loses severely in recall, when all variables are left unassigned. This is understandable: given that the grammar is extremely small, only 19 rules, the majority of the tokens will never appear in the clauses, neither as targets nor conditions. So the SAT-solver is free to arbitrarily choose the readings of those free words, whereas VISL CG-3 does nothing to them. There is no universal behaviour what to do if a variable is not included in any of the clauses; it is likely that many solvers would just assign false by default. In our case, we still require at least one reading of every cohort to be true; this applies even for cohorts which do not appear in the clauses. Given the lack of other information, the SAT-solver is likely to just choose the first one in alphabetical order. It would not be difficult to modify the scheme based on another heuristic, such as the most likely unigram reading, or dispreference for analyses with many compound splits. However, we did not add any heuristics; the results just show the SAT-solver performing its default actions.

	NoAss		NoAff		NoTar	
	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.
SAT-CG$_{Max}$	79.97	80.47	77.24	82.94	80.36	86.41
SAT-CG$_{Ord}$	80.96	81.31	78.42	84.07	80.22	86.28
VISL CG-3	78.29	85.02	78.29	85.02	78.29	85.02

Table 3.4: Showing the effect of three variants on the 19-rule grammar: (NoAss) Leave all variables unassigned; (NoAff) Assign true to variables which do not appear in any rule; (NoTar) Assign true to variables which are not targeted by any rule.

The precision in SAT-CG is better already in the unassigned scheme. The percentage is calculated by overall number of analyses, but even at the level of different words, the advantage is at SAT-CG. Out of the gold standard corpus, VISL CG-3 differs in the analyses of some 4,200 words, whereas SAT-CG, in the unassigned variant, disagrees only on 3700-3900 words. Of course, the differences are still very small, and this effect could also be due to randomness: sometimes the SAT-solver happens to disambiguate correctly just by removing a random analysis.

We move on to the second variant, where the completely unaffected variables are assigned true. Unsurprisingly, the recall grows: we have just added some thousands of true variables, some of which would have been otherwise randomly removed by the SAT-solver. The ordered variant resembles most the behaviour of VISL CG-3, both in precision, recall and the amount of words where the analysis disagrees; now the number is around 4,200 for both.

The third variant, where all non-targeted analyses are assigned true, performs the best. High recall is expected; now we force some more thousands of variables true, among them must be some actually true readings. But at the same time, this limits the power of rules: they may only disambiguate targets, not conditions—why does precision go up? The most plausible explanation for this phenomenon, aside from mere chance, is the particular features of the tested grammar. When tested with the original 261-rule grammar, shown in Table 3.5, the pattern does not repeat: assigning more variables true just brings the performance closer to VISL CG-3.

	NoAss		NoAff		NoTar	
	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.
SAT-CG$_{Max}$	77.03	79.32	70.85	86.46	71.91	87.72
SAT-CG$_{Ord}$	77.91	80.18	70.79	86.40	71.78	87.62
VISL CG-3	71.17	88.34	71.17	88.34	71.17	88.34

Table 3.5: Evaluation of the three variants, for the original 261-rule grammar.

```
# la casa (en que vivo)
SELECT:i01_s_det Det IF (NOT 1C VerbFin) ;

# (el sacerdote) la casa (con el novio)
SELECT:i08_s_pro PrnIndep IF (1C VerbFin) ;
```

Figure 3.5: Excerpt from the 19-rule Spanish grammar.

Now, what are these features of the tiny grammar? One reason may be that many of the rules come in pairs. Figure 3.5 shows an excerpt from the grammar: due to these rules, both la_{DET} and la_{PRN} would be targeted, and remain unassigned. Another peculiarity in the 19-rule grammar is the high number of SELECT rules—this means that all the other readings in the cohort become the actual targets. Thus, if a rule addresses some phenomenon, most of the readings in the target cohort are saved from the forced assignment.

To conclude, this experiment has been very inconclusive, due to the extremely low coverage of the tested grammars. A high-quality grammar would target significantly more individual words in the test corpus; this means less chances for the SAT-solver to randomly decide the values for untargeted readings. We would hope that the power of rules disambiguating their condition would help such grammar to go that extra mile; however, based on the experiments so far, this is mostly speculation.

It would be interesting to repeat the experiment in writing a serious, large-scale grammar. Some questions to address would be (a) do we need less rules; (b) do we need careful context; (c) do we write more SELECT rules; and (d) will a grammar that performs well with parallel CG also perform well with sequential CG?

3.4 Summary

In this section, we started from the known parallels between CG and logic: the rules express what is true (SELECT) and false (REMOVE), under certain conditions (IF). Expressing this as a SAT-problem, we get a working CG implementation, which executes the rules in a parallel and unordered manner. However, a straight-forward parallel encoding means that the rules of a given grammar should contain no contradictions. To alleviate the brittleness of PCG, we developed two approaches to discard conflicting rule applications. The first one is based on order; introducing clauses one by one, and assuming that the previous clauses are true, and the second one is based on maximisation; trying to make as many clauses as possible apply.

As a conclusion, SAT-solver provides an interesting alternative for a CG engine, but loses in practicality: performance and lack of transparency for grammar writers. We see more potential in contributing to the FSIG community, with a practical implementation and novel ways to get around conflicts. It would be interesting to test an actual syntactic FSIG grammar encoded in SAT, to see if a SAT-based solution is powerful enough for the more complex constraints.

Appendix: SAT-encoding

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

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

In this appendix, we show the SAT-encoding for all the rule types that SAT-CG supports: REMOVE and SELECT rules, with the contextual tests and set operations described in CG-2 [50], except for the operators ** (continue search if linked tests fail) and @ (absolute position). In addition, the software SAT-CG supports a couple of new constructions from VISL CG-3: CBARRIER, which requires the barrier tag to be unambiguous in order to function as a barrier; NEGATE, which inverts a whole chain of contextual tests; and subreadings, which make it possible to include e.g. a clitic in the same cohort, but distinguish it from the main reading.

The encoding in this appendix is 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 purpose of this appendix is to give a full description of the different rule types; the method is introduced with more pedagogical focus in Section 3.2.

3.A SAT-encoding of sentences

As in Section 3.2, 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:

$$\begin{array}{lllll} sobre_{\text{PR}} & una_{\text{PRN}} & aproximación_{\text{N}} & más_{\text{ADV}} & científica_{\text{ADJ}} \\ sobre_{\text{N}} & una_{\text{DET}} & & más_{\text{ADJ}} & científica_{\text{N}} \\ & una_{\text{PRSP3}} & & & \\ & una_{\text{PRSP1}} & & & \\ & una_{\text{IMPP3}} & & & \end{array} \tag{3.1}$$

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; later referred to as the "default rule". The word *aproximación* is already unambiguous, thus the clause $aproximación_{\text{N}}$ is a unit clause, and the respective variable is trivially assigned true. The final assignment of the other variables depends on the constraint rules.

$$\begin{gathered} sobre_{\text{PR}} \vee sobre_{\text{N}} \\ una_{\text{PRN}} \vee una_{\text{DET}} \vee una_{\text{PRSP3}} \vee una_{\text{PRSP1}} \vee una_{\text{IMPP3}} \\ aproximación_{\text{N}} \\ más_{\text{ADV}} \vee más_{\text{ADJ}} \\ científica_{\text{ADJ}} \vee científica_{\text{N}} \end{gathered} \tag{3.2}$$

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 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.

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

SELECT adj Unconditionally removes all other readings which are in the same cohort with the target. We do not add an explicit clause for $más_{\text{ADJ}}$ and $científica_{\text{ADJ}}$: the default rule 3.2 already contain $más_{\text{ADV}} \vee más_{\text{ADJ}}$ and $científica_{\text{ADJ}} \vee científica_{\text{N}}$, and the combination $\neg más_{\text{ADV}}$ and $más_{\text{ADV}} \vee más_{\text{ADJ}}$ implies that $más_{\text{ADJ}}$ must be true.

$$\begin{array}{c} \neg más_{\text{ADV}} \\ \neg científica_{\text{N}} \end{array} \tag{3.4}$$

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

$$\neg una_{\text{PRSP3}} \wedge \neg una_{\text{PRSP1}} \wedge \neg una_{\text{IMPP3}} \tag{3.5}$$

SELECT verb As 3.4, but matches multiple readings in a cohort. All other readings in the same cohort are negated. As previously, we need no explicit clause for $una_{\text{PRSP3}} \vee una_{\text{PRSP1}} \vee una_{\text{IMPP3}}$: the default rule 3.2 guarantees that at least one of the target readings is true.

$$\neg una_{\text{DET}} \wedge \neg una_{\text{PRN}} \tag{3.6}$$

3.B.2 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 from $\neg más_{\text{ADJ}}$ to $\neg más_{\text{ADV}}$.

$$\begin{array}{ll} \textbf{REMOVE adj IF (1 adj)} & \textbf{SELECT adj IF (1 adj)} \\ científica_{\text{ADJ}} \Longrightarrow \neg más_{\text{ADJ}} & científica_{\text{ADJ}} \Longrightarrow \neg más_{\text{ADV}} \end{array} \tag{3.7}$$

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

REMOVE adj IF (-1 n LINK 2 adj) Linked conditions—identical to the above.

$$científica_{\text{ADJ}} \wedge aproximación_{\text{N}} \quad \Longrightarrow \quad \neg más_{\text{ADJ}} \tag{3.8}$$

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

$$científica_{\text{ADJ}} \vee aproximación_{\text{N}} \quad \Longrightarrow \quad \neg más_{\text{ADJ}} \tag{3.9}$$

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

$$científica_{\text{ADJ}} \wedge \neg científica_{\text{N}} \implies \neg más_{\text{ADJ}} \tag{3.10}$$

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

$$\begin{aligned} sobre_{\text{N}} \vee\ aproximación_{\text{N}} &\implies \neg más_{\text{ADJ}} \\ sobre_{\text{N}} \vee\ aproximación_{\text{N}} &\implies \neg científica_{\text{ADJ}} \end{aligned} \tag{3.11}$$

REMOVE adj IF (-1* n LINK 1 v) Scanning and linked condition. Any noun before the target, followed immediately by a verb, is a valid condition.

$$\begin{aligned} sobre_{\text{N}} \wedge (una_{\text{PRSP3}} \vee una_{\text{PRSP1}} \vee una_{\text{IMPP3}}) &\implies \neg más_{\text{ADJ}} \\ sobre_{\text{N}} \wedge (una_{\text{PRSP3}} \vee una_{\text{PRSP1}} \vee una_{\text{IMPP3}}) &\implies \neg científica_{\text{ADJ}} \end{aligned} \tag{3.12}$$

REMOVE adj IF (-1* n LINK 1* adv) Scanning and linked condition. Any noun before the target, followed anywhere by an adverb, is a valid condition.

$$\begin{aligned} (sobre_{\text{N}} \wedge más_{\text{ADV}}) \vee (\ aproximación_{\text{N}} \wedge más_{\text{ADV}}) &\implies \neg más_{\text{ADJ}} \\ (sobre_{\text{N}} \wedge más_{\text{ADV}}) \vee (\ aproximación_{\text{N}} \wedge más_{\text{ADV}}) &\implies \neg científica_{\text{ADJ}} \end{aligned} \tag{3.13}$$

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: $sobre_{\text{N}}$ is a valid condition only if una_{DET} is false.

$$\begin{aligned} (sobre_{\text{N}} \wedge \neg una_{\text{DET}}) \vee\ aproximación_{\text{N}} &\implies \neg más_{\text{ADJ}} \\ (sobre_{\text{N}} \wedge \neg una_{\text{DET}}) \vee\ aproximación_{\text{N}} &\implies \neg científica_{\text{ADJ}} \end{aligned} \tag{3.14}$$

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 $una_{\text{PRN}} \vee una_{\text{PRSP3}} \vee una_{\text{PRSP1}} \vee una_{\text{IMPP3}}$.

$$\begin{aligned} (sobre_{\text{N}} \wedge una_{\text{Any}}) \vee\ aproximación_{\text{N}} &\implies \neg más_{\text{ADJ}} \\ (sobre_{\text{N}} \wedge una_{\text{Any}}) \vee\ aproximación_{\text{N}} &\implies \neg científica_{\text{ADJ}} \end{aligned} \tag{3.15}$$

REMOVE adj IF (-1C* n) Scanning with careful context. Any unambiguous noun before the target is a valid condition: $sobre_{\text{N}}$ is a valid condition only if it is the only reading in its cohort, i.e. $sobre_{\text{PR}}$ is false.

$$\begin{aligned}(sobre_{\text{N}} \wedge \neg sobre_{\text{PR}}) \vee\ aproximación_{\text{N}} &\implies \neg más_{\text{ADJ}} \\ (sobre_{\text{N}} \wedge \neg sobre_{\text{PR}}) \vee\ aproximación_{\text{N}} &\implies \neg científica_{\text{ADJ}}\end{aligned} \tag{3.16}$$

REMOVE adj IF (-1C* n BARRIER det) Scanning with careful context, up to a barrier. Any unambiguous noun before the target, up to a determiner, is a valid condition: $sobre_{\text{N}}$ is a valid condition only if it is the only reading in its cohort ($sobre_{\text{PR}}$ is false), and there is no determiner between *sobre* and the target (una_{DET} is false).

$$\begin{aligned}(sobre_{\text{N}} \wedge \neg sobre_{\text{PR}} \wedge \neg una_{\text{DET}}) \vee\ aproximación_{\text{N}} &\implies \neg más_{\text{ADJ}} \\ (sobre_{\text{N}} \wedge \neg sobre_{\text{PR}} \wedge \neg una_{\text{DET}}) \vee\ aproximación_{\text{N}} &\implies \neg científica_{\text{ADJ}}\end{aligned} \tag{3.17}$$

REMOVE adj IF (-1C* n CBARRIER det) Scanning with careful context, up to a careful barrier. Like above, but *una* fails to work as a barrier if it is not unambiguously determiner.

$$\begin{aligned}(sobre_{\text{N}} \wedge \neg sobre_{\text{PR}} \wedge una_{\text{Any}}) \vee\ aproximación_{\text{N}} &\implies \neg más_{\text{ADJ}} \\ (sobre_{\text{N}} \wedge \neg sobre_{\text{PR}} \wedge una_{\text{Any}}) \vee\ aproximación_{\text{N}} &\implies \neg científica_{\text{ADJ}}\end{aligned} \tag{3.18}$$

3.B.3 Inverted conditions

In the following, we demonstrate the effect of two inversion operators. The keyword NOT inverts a single contextual test, such 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 allowed, as well as *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 inapplicable, 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. There is no word following *científica*, hence its adjective reading is removed unconditionally.

$$\begin{aligned} \neg científica_{\text{ADJ}} &\implies \neg más_{\text{ADJ}} \\ &\implies \neg científica_{\text{ADJ}} \end{aligned} \tag{3.19}$$

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

$$\begin{aligned} \neg\, aproximación_{\text{N}} \wedge \neg científica_{\text{ADJ}} &\implies \neg más_{\text{ADJ}} \\ &\implies \neg científica_{\text{ADJ}} \end{aligned} \tag{3.20}$$

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

$$\begin{aligned} \neg(sobre_{\text{PR}} \wedge una_{\text{DET}} \wedge\ aproximación_{\text{N}}) &\implies \neg más_{\text{ADJ}} \\ &\implies \neg científica_{\text{ADJ}} \end{aligned} \tag{3.21}$$

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

$$\begin{aligned} \neg(científica_{\text{ADJ}} \wedge \neg científica_{\text{N}}) &\implies \neg más_{\text{ADJ}} \\ &\implies \neg científica_{\text{ADJ}} \end{aligned} \tag{3.22}$$

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

$$\begin{aligned} \neg(sobre_{\text{N}} \vee\ aproximación_{\text{N}}) &\implies \neg más_{\text{ADJ}} \\ \neg(sobre_{\text{N}} \vee\ aproximación_{\text{N}}) &\implies \neg científica_{\text{ADJ}} \end{aligned} \tag{3.23}$$

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.

$$\begin{aligned} \neg((sobre_{\mathrm{N}} \wedge \neg una_{\mathrm{DET}}) \vee\ aproximación_{\mathrm{N}}) \quad &\Longrightarrow \quad \neg más_{\mathrm{ADJ}} \\ \neg((sobre_{\mathrm{N}} \wedge \neg una_{\mathrm{DET}}) \vee\ aproximación_{\mathrm{N}}) \quad &\Longrightarrow \quad \neg científica_{\mathrm{ADJ}} \end{aligned} \tag{3.24}$$

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

$$\begin{aligned} \neg((sobre_{\mathrm{N}} \wedge \neg sobre_{\mathrm{PR}} \wedge una_{\mathrm{Any}}) \vee\ aproximación_{\mathrm{N}}) \quad &\Longrightarrow \quad \neg más_{\mathrm{ADJ}} \\ \neg((sobre_{\mathrm{N}} \wedge \neg sobre_{\mathrm{PR}} \wedge una_{\mathrm{Any}}) \vee\ aproximación_{\mathrm{N}}) \quad &\Longrightarrow \quad \neg científica_{\mathrm{ADJ}} \end{aligned} \tag{3.25}$$

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 n) (NOT 10 n) Conjunction of inverted conditions, one out of scope.
REMOVE adj IF (NEGATE -3 pr LINK 1 punct) Inversion of a conjunction of conditions, some not present in initial analysis.

$$\Longrightarrow \quad \neg más_{\mathrm{ADJ}} \tag{3.26}$$

Chapter 4

Grammar analysis using SAT

In the previous chapter, we presented a tool. In the current chapter, we will solve a problem.

Recall the design principles of CG from Section 2.3: by design, the grammars are shallow and low-level. 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", "auxiliary 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. Despite this well-known issue, there has not been a tool that would help grammar writers to detect conflicting rules. Following the idea further, the tool could give feedback that is not restricted to conflicts, but also other features that are helpful in the process of writing grammar. Given the rules in Figure 4.1, a grammar writer may ask the following questions.

- Are all the Portuguese 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?
- Can the Finnish rule on the left be rewritten in the form shown on the right?
- Generate an example sequence that triggers rule(s) R but not rule(s) R'.

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

```
            SELECT V + Prs/Imprt + Act + Neg IF ...
(*-1C Negv LINK NOT *1 Vfin)     (NOT *-1 Niin OR Neg)
(NOT 0 N) (NOT 0 Pron)           (*-1C Negv
(NOT *1 Neg) (NOT *-1 Neg)         LINK NOT 0 Imprt
(NOT 0 Pass) (NOT *-1 Niin)        LINK NOT *1 Vfin OR CLB?)
(*-1C Negv LINK NOT *1 CLB?)     (NOT 0 N OR Pron OR Pass)
(*-1C Negv LINK NOT 0 Imprt) ;   (NOT *1 Neg) ;
```

Figure 4.1: Left: rules to select infinitive in Portuguese. Right: two versions of a condition in Finnish.

The chapter follows with introduction of related work: namely, corpus-based methods to aid grammar writing, and automatic optimisation of a complete, human-written grammar. We continue by presenting our solution, along with a working implementation, and finally, evaluate its performance.

4.1 Related work

There has been previous research on corpus-based methods in manual grammar development [55], as well as optimisation of hand-written CGs [6]. In addition, there is a large body of research on automatically inducing rules, e.g. [45, 19, 33, 47]. However, since our work is aimed to aid the process of hand-crafting rules, we omit those works from our discussion.

Corpus-based methods in manual grammar development Hand-annotated corpora are commonly used in the development of CGs, because they give immediate feedback whether a new rule increases or decreases accuracy. Atro Voutilainen [55] gives a detailed account about best practices of grammar writing and efficient use of corpora to aid the grammar development. For a language with no free or tagset-compatible corpus available, Reynolds and Tyers [44] describe a method where they apply their rules to unannotated Wikipedia texts and pick 100 examples at random for manual check.

CG rules are usually arranged in sections, and run in the following manner. First apply rules from section 1, and repeat until nothing changes in the text. Then apply rules from sections 1–2, then 1–3 and so on, until the set includes all rules. The best strategy is to place the safest and most effective rules in the first sections, so that they make way for the following, more heuristic and less safe rules to act on. A representative corpus is arguably the best way to get concrete numbers—how many times a rule applied and how often it was correct—and to arrange the rules in sections based on that feedback.

Voutilainen [55] states that the around 200 rules are probably enough to resolve 50–75 % of ambiguities in the corpus used in the development. This figure is very much thanks to Zipf's law: we can add rules that target the most frequent *tokens*, thus disambiguating a high number of word forms. However, this method will not notice a missed opportunity or a grammar-internal conflict, nor suggest ways to improve; neither does it guarantee a coherent whole of rules. While the coverage information is easy to obtain from a corpus, there is no tool that would aid grammar writers in including wide coverage of different linguistic phenomena.

Automatic optimisation of hand-written grammars The corpus-based method can tell the effect of each single rule at their place in the rule sequence, and leaves the grammar writer to make changes in the grammar. As a step further, Eckhard Bick [6] modifies the grammar automatically, by trying out different rule orders and altering the contexts of the rules. Bick reports error reduction of 7–15% compared to the original grammars. This is a valuable tool, especially for grammars that are so big that it's hard to keep track manually. A program can try all combinations whereas trying to make sense out of a huge set of rules would be hard for humans. As a downside, the grammar writer will likely not know why exactly does the tuned grammar perform better.

4.2 Analysing CGs

We start by defining a conflict, and present requirements for a solution. Then, we introduce a logical translation of sequential CG, corresponding to [34], and modify it into a SAT-problem about the *original* sentence before applying the rules. We refine our solution by restricting what kind of sentences we can create. The whole method requires only a morphological lexicon, no corpus.

Conflict We define *conflict* as follows: a list of rules R is in conflict with the rule r, if applying R makes it impossible to apply r, regardless of input. Some examples of conflicts follow:

- If two equivalent rules r and r' occur in the grammar, the second occurrence will be disabled by the first
- A list of rules R selects something in a context, and r' removes it

- A list of rules R removes something in a context, and r' selects it
- A list of rules R removes something from the context of a rule r', so r' can never apply
- A rule r has an internal conflict, such as non-existent tag combination, or contradictory requirements for a context word

This definition is very similar to the concept *bleeding order* in generative phonology [29]; however, since we are talking about an explicit list of human-written rules, we include also rule-internal conflicts in our classification. The conflicting (or "bleeding") R can be a single rule or a list 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.

Solution How do we find out if a rule r can act? We could apply the grammar to a large corpus and count the number of times each rule fires; if some rule never fires, we can suspect there is something wrong in the rule itself, or in the interaction with previous rules. But it can also be that r just targets a rare phenomenon, and there was no sentence in the corpus that would trigger it.

Of course, tagged corpora are extremely useful for many questions in CG analysis. A corpus can show which rules are the most used, or which rules give false analyses. Luckily, we have already methods for finding such statistics—the question about conflicts is orthogonal to those, and the solution needs to address only the part about conflicts, not about rules being frequent or accurate. Therefore, we can take a more artificial approach. Remember the definition of conflict: "applying R makes it impossible to apply r *regardless of input*". In a nutshell, we are going to show the absence of a conflict by trying to create a sequence where r can apply after R; conversely, we detect a conflict by showing that such sequence cannot be created.

4.2.1 From disambiguation to generation

SAT-encoding of sequential CG In order to analyse real-life CGs, we need to model the semantics of sequential CG: rule application takes effect immediately, is irreversible[1], and

[1] As pointed out by Eckhard Bick, VISL CG-3 allows reference to deleted readings. In principle, this would not be an obstacle for our implementation, because no variables are removed.

only allows changes in the target. We restrict ourselves to the operations on morphological disambiguation, and only SELECT and REMOVE rules. The following encoding does not support dependency relations, nor rule types which add readings or cohorts.

Last time, we only used ordering as a way to choose the which rules to apply. But there was no state: all clauses operated in parallel, on the same variables. Now, we need a new variable for a reading every time when it is targeted; that is, when its state potentially changes. As before, a sentence is a vector of cohorts, which is, in turn, a vector of variables representing the readings of the current cohort. At each rule application, we create a new variable $word'_{\text{RD}}$ for each targeted reading $word_{\text{RD}}$. The new variable $word'_{\text{RD}}$ is true iff

(a) $word_{\text{RD}}$ was true, and (b) the rule cannot apply: this can be because
- its conditions do not hold, or
- it would remove the last reading.

We keep the same example sequence from Chapter 3, *la casa grande*. In the following, we apply the rule `REMOVE v IF (-1 det)` to it.

$$\text{New variable } casa'_{\text{V}} \iff casa_{\text{V}} \wedge (\overbrace{\neg la_{\text{DET}}}^{\text{invalid condition}} \vee \overbrace{(casa_{\text{V}} \wedge \neg casa_{\text{N}})}^{\text{only target left}})$$

After the application, we replace each $word_{\text{RD}}$ with $word'_{\text{RD}}$ in the sentence vector; those variables that are not touched by the rule, will be carried over to the next round unchanged. For example, if we apply two rules which target verbs and one which targets pronouns, the sentence vector will look like the following.

$$\begin{aligned} la &\rightarrow \{la_{\text{DET}}, la'_{\text{PRN}}\} \\ casa &\rightarrow \{casa''_{\text{V}}, casa_{\text{N}}\} \\ grande &\rightarrow \{grande_{\text{ADJ}}\} \end{aligned}$$

Symbolic sentence We saw how $casa'_{\text{V}}$ was created, based on the status of $casa_{\text{V}}$ and the conditions at the time. Similarly, $casa''_{\text{V}}$ will be created based on $casa'_{\text{V}}$. But what is the status of the variables in the beginning? In fact, that decision makes a crucial difference. If all variables start off as true, then we have, in effect, reimplemented the logical encoding by Lager and Nivre [34]. This option would not make an interesting SAT-problem: there is no search involved, just manipulation of Boolean expressions in a completely deterministic way. All the new variables, created from the rule applications, would get their value immediately: $casa'_{\text{V}} \Leftrightarrow grande_{\text{ADJ}} \wedge \neg(casa_{\text{V}} \wedge \neg casa_{\text{N}})$ just translates into $casa'_{\text{V}} \Leftrightarrow True \wedge \neg(True \wedge \neg True)$.

In order to turn this problem into a SAT-problem, we will have the readings start off unassigned. All the other variables computed along the way depend on the original variables, so the question to the SAT-solver becomes: *"Which readings were originally true?"* For implementing a CG engine, this question does not make much sense: we know which readings were given by the morphological analyser. But our task now is to *generate* the original sentence, which will pass through the rules. Here comes the most important modification: we will apply the rules to something we call *symbolic sentence*. Every cohort, called *symbolic word*, contains every possible reading, and rule applications are responsible for shaping the sentence into a concrete one.

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; we will explain this in more detail in the coming sections. In our experiments, the number of readings has ranged from about 300 to about 9000.

Width of a rule 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 `REMOVE v IF (-1 det)`, the width is 2.

If the context contains a `*` (context word can be anywhere), we need to make an approximation of $w(r)$, which may result in false positives or negatives later on in the analysis; this indeed happened in the Finnish grammar in Section 4.3.3. For example, given the rule `REMOVE v IF (-1* det)`, we tried first a sentence of width 2; if there was a conflict, we tried width 3, and then 4. For a rule with multiple `*`s, we create combinations in the range of $\pm$ 2 symbolic words for each `*`-condition; in addition, we need to specify where the target is in the symbolic sentence. For instance, the rule `REMOVE v IF (-1* det) (1* det)` would be tested with the following combinations, where C means condition and T means target: (C,T,C); (C,C,T,C); (C,T,C,C); (C,C,T,C,C); (C,C,C,T,C,C); (C,C,T,C,C,C) and (C,C,C,T,C,C,C).

Rule application Finally, we define what does it mean for a sentence to "pass through" a rule. Let us enumerate the cases:

1. The sentence is out of scope; target not removed
2. The conditions of the rule do not hold; target not removed
3. The target is the only possible reading left; target not removed

4. The target was never in place

5. The target is removed

Let us illustrate the difference between the last two cases with a rule that targets pronouns, say, `REMOVE prn IF (1 v)`. If the target was never in place, then both the original la_{PRN} and the newly created la'_{PRN} end up as false. If the target is removed, then la_{PRN} will be true and la'_{PRN} false. This means that a rule with the condition `IF (-1 prn)` behaves differently depending on where in the rule sequence it is placed: if it is applied before `REMOVE prn IF (1 v)`, then it will match the original reading la_{PRN}. Any rule applied after that will get the newly created variable la'_{PRN}.

Now we have the definitions in place. In the following, we are going to show two examples; first one a very simple conflict, and the second one with more complicated interaction.

Example 1: conflict Let us look at a conflicting case. For simplicity, we assume that the full set of readings in the language is {*det def, noun sg, noun pl, verb sg, verb pl*}. We ignore all lexical readings for now, and assume dummy words of the form w_i, where i refers to the index of the word. The rules are as follows:

```
r1 = REMOVE verb IF (-1 det) ;
r2 = REMOVE verb IF (-1 det) (1 noun) ;
```

We want to know if the last rule is able to apply, after going through the rules that come before it—in this case, there is only one such rule. The width of the last rule is three: one condition to the left of the target, one to the right, so we create a symbolic sentence of three words. Below, we show the symbolic sentence, consisting of the symbolic words *w1, w2* and *w3*.

```
"<w1>"                "<w2>"                "<w3>"
       "w1" det def          "w2" det def          "w3" det def
       "w1" noun sg          "w2" noun sg          "w3" noun sg
       "w1" noun pl          "w2" noun pl          "w3" noun pl
       "w1" verb sg          "w2" verb sg          "w3" verb sg
       "w1" verb pl          "w2" verb pl          "w3" verb pl
```

After applying the first rule, we have a situation which looks much like the one in the previous chapter, Figure 3.2. If the symbolic word *w1* is a determiner, then *w2* cannot be a verb; the only exception is the case where verb is the only analysis of *w2*. As before, the verb

analysis *w2* can also be false, even if *w1* is not a determiner. The exact same dependencies are created between the symbolic words *w2* and *w3*. The first word *w1* is out of scope of the condition, because it has no preceding word. Below we see the affected readings after the first rule application, with newly created variables:

$$
\begin{aligned}
1 &\rightarrow \{w1_{\text{DETDEF}},\ w1_{\text{NSG}},\ w1_{\text{NPL}},\ w1_{\text{VSG}},\ w1_{\text{VPL}}\} \\
2 &\rightarrow \{w2_{\text{DETDEF}},\ w2_{\text{NSG}},\ w2_{\text{NPL}},\ w2'_{\text{VSG}},\ w2'_{\text{VPL}}\} \\
3 &\rightarrow \{w3_{\text{DETDEF}},\ w3_{\text{NSG}},\ w3_{\text{NPL}},\ w3'_{\text{VSG}},\ w3'_{\text{VPL}}\}
\end{aligned}
$$

There is no "final sentence" yet, simply constraints on the possible combinations of determiners and verbs. If we asked for a solution now, it could be anything; for instance, all three words are determiners. Just asking for a freely chosen sequence is not very interesting; there are so many possibilities, and even after applying more rules, most of the combinations are not explicitly prohibited.

We want to know if there is a sequence that can trigger the last rule. In order to trigger the rule, the sequence must have the following three features:

- Target readings: at least one reading with a *verb* tag in the target position *w2*.
- Ambiguity: at least one reading without a *verb* tag in the target position; if there are only *verb* readings in the target, the rule would not fire.
- Conditions: at least one reading with a *det* tag in the cohort preceding the target, and at least one reading with a *noun* tag in the cohort following the target.

We can see that some of these requirements can be fulfilled: the first rule allows models where *w2* contains *target readings*. If the rule had been `REMOVE verb`, that is, unconditionally remove all verbs, then this requirement could not be fulfilled. But so far we have not run into the wall. The second requirement, for *ambiguity* is no problem either: other readings of *w2* have not been targeted, so we are free to choose anything.

As for *conditions*, the part about `(1 noun)` can be fulfilled; nothing prevents *w3* from being a noun. But the first condition, `(-1 det)`, cannot be true, if the target *w2* has to be a verb—the first rule has prohibited exactly that combination. Therefore, when we try to ask for a solution where all these requirements hold, we will get a conflict. The result shows us that the rule `REMOVE verb IF (-1 det) (1 noun)` cannot ever apply, if the rule `REMOVE verb IF (-1 det)` has been already applied.

Example 2: no conflict The previous was a simple example of a conflict: two rules target the same reading, and the first one had broader conditions. Since we had only one preceding rule, we could not demonstrate why is it important to apply all the rules in order—even with a longer list of rules, the previous example could have also worked unordered, as long as the last rule is separated from rules before it. Now we will show another example, where we illustrate why is it important to create new variables, and how conditions can affect the state of the variables, but in a more limited way. The set of readings is the same, and the rules are as follows.

```
r1 = REMOVE verb IF (-1C det) ;
r2 = SELECT det  IF ( 1 verb) ;
r3 = REMOVE verb IF (-1 det) ;
```

The width of the last rule is 2, thus we create a symbolic sentence with only *w1* and *w2*. The sentence is shown below:

```
"<w1>"                "<w2>"
       "w1" det def          "w2" det def
       "w1" noun sg          "w2" noun sg
       "w1" noun pl          "w2" noun pl
       "w1" verb sg          "w2" verb sg
       "w1" verb pl          "w2" verb pl
```

To begin, let us only look at *r1* and *r3*. Like in the previous example, they target the same reading, but now the order is good: the first rule is the one with a narrower condition, which means that it is possible for a sequence to pass through it, and still have the verb reading in *w2* intact—it just needs to have something else in addition to the determiner in *w1*. The following is an example of such sequence (false readings not shown).

```
"<w1>"                "<w2>"
       "w1" det def          "w2" noun pl
       "w1" noun sg          "w2" verb sg
```

So far, we have passed through *r1* with the case "condition does not hold, target not removed". Now, let us add *r2*: `SELECT det IF (1 verb)`. We must, in fact, select the determiner and remove everything else: the condition of *r2* happens to be the target of *r3*, so it must hold. But we have the notion of state now, so this is no problem. On arriving to *r1*, the symbolic word *w1* had to be ambiguous—that is, one of the variables $w1_{\text{NS}_\text{G}}$, $w1_{\text{NP}_\text{L}}$, $w1_{\text{VS}_\text{G}}$, $w1_{\text{VP}_\text{L}}$

must be true. But after passing through *r2*, the state of *w1* has changed: it contains a set of new variables $w1'_{\text{NSG}}$, $w1'_{\text{NPL}}$, $w1'_{\text{VSG}}$, $w1'_{\text{VPL}}$, and all of them must be false. Since the clauses formed by *r1* and *r3* get access to different variables, there is no conflict.

4.2.2 Towards realistic language

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 possible readings. In order to find the set of readings, we expand a morphological lexicon[2], 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 duplicate 1572 entries for each verb lemma. Even ignoring the clitics, each verb lemma still adds 213 new readings.

The readings in a grammar can be underspecified: for example, the rule `REMOVE (verb sg) IF (-1 det)` gives us "verb sg" and "det". In contrast, the lexicon only gives us fully specified readings, such as "verb pres p2 sg". We implemented a version where we took the tag combinations specified in the grammar directly as our readings, and we could insert

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

them into the symbolic sentences as well. The shortcut works most of the time, but if we only take the readings from the grammar and ignore the lexicon, it is possible to miss some cases: e.g. the rule `SELECT (pron rel) IF (0 nom)` may require "pron rel nom" in one reading, but this method only gives "pron rel" and "nom" separately.

In addition, we found that the tag lists in the grammars sometimes contain errors, such as using a nonexistent tag or using a wrong level in a subreading. If we accept those lists as readings, we will generate symbolic sentences that are impossible, and not discover the bug in the grammar. However, if we are primarily interested in rule interaction, then using the underspecified readings from the grammar may be an adequate solution.

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 which 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 do not 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 readings that can be ambiguous with each other. We represent the expanded morphological lexicon as a matrix, as seen above: word forms on the rows and analyses on the columns. Each distinct row forms an ambiguity class. For example, one class may contain words that are ambiguous between plural feminine nouns and separable verb infinitives; another 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 [16], 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 that as a parameter in their HMM.

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 contradict 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.

4.2.3 Use cases

In the beginning of this chapter, we gave a list of questions, regarding the rules in Figure 4.1. After describing our implementation, we return to these questions, and explain how a SAT-solver can answer them. Section 4.3 shows the evaluation of the conflict detection, which we tried out for three grammars of different sizes. The use cases presented in this section have not been tried in practice; however, our setup makes them fairly straight-forward to implement.

Are all the rules distinct? We gave the example of two rules in the Portuguese grammar, where one had the condition `(-1 Prep)` and the other `(-1 Para OR De)`. A grammarian who has some knowledge of Portuguese could tell that *para* and *de* are both prepositions, so these two rules seem to do almost the same thing.

How can we verify this? To start, we apply all candidate rules to the same initial symbolic sentence. For the resulting symbolic sentence, we can ask for solutions with certain requirements. For instance, "give me a model where a reading $a \notin$ `Inf` is true". There are less such models allowed after `SELECT Inf IF (-1 Prep)`—if a non-infinitive reading is in place, it means that the previous word cannot be any determiner—and they are all in the set

of models allowed `SELECT Inf IF (-1 Para OR De)`. Thus, we can show that the first rule implies the second. It is in the hands of the grammar writer to decide whether to keep the stronger or the weaker rule, or if they should be placed in different sections.

Could two or more rules be merged? This example concerns two or more rules with the same target but different conditions in the same position. All the six rules in Figure 4.1 have conditions in position -1; four of them also in 0. Some of them have an additional careful context, and some do not. A grammar writer might think that `IF (-1C Vbmod) (0C V)` and `IF (-1C Vbmod) (0 Ser)` are very similar, and could be merged into `SELECT Inf IF (-1C Vbmod) (0C V OR Ser)`. But could there be some unintended side effects in merging these two rules? Why is there a C in the first rule, but not in the second rule?

The procedure is similar to the previous one. We initialise two symbolic sentences; on one we run the original rules in a sequence, and on the other the merged rule. Then, we can perform tests on the resulting symbolic sentences. Assuming that the ambiguity class constraints are in place, the result may show that C is not needed for the condition about *ser*, simply because *ser* is not ambiguous with anything else in the lexicon; hence the merged rule can safely have the condition `(0C V OR Ser)`. However, if *ser* can be ambiguous with something else, then this merged rule is stricter than the two original rules. If the grammar writer is still unsure whether there would be any meaningful sequences that would be missed, they can ask the SAT-solver to generate all those cases—this requires that the ambiguity classes are implemented, otherwise the number of solutions would blow up due to all the readings that are irrelevant to the rules in question.

Generate a sequence that triggers rule(s) R but not rule(s) R'. For any given rule, we can extract three requirements that would make it trigger. We illustrate them for the rule `REMOVE v IF (-1 det)`. In order to trigger the rule, the sequence must have

- target readings: at least one reading with the v tag in the target position
- ambiguity: at least one reading without the v tag in the target position; if there are only v-readings in the target, the would not fire.
- conditions: at least one reading with the *det* tag in the cohort preceding the target.

We can extract these requirements from both the rules that must be triggered, and the rules that must be not triggered. Then, we can request a solution from the SAT-solver.

What is the best order for the rules? We can generate variants of the rule order, and run them to a set of symbolic sentences. Then, for the resulting symbolic sentences, we can query which one is the most restricted, that is, allows the least models. As a variant, we may be interested in the placement of just one rule in the sequence.

Does a rewritten rule correspond to the original? We can initialise two symbolic sentences, then run the original rule and the rewritten rule, and check if they allow the same models.

4.3 Evaluation

We tested three grammars to find conflicting rules: Dutch[3], with 59 rules; Spanish[4], with 279 rules; and Finnish[5], with 1185 rules. We left out MAP, SUBSTITUTE and other rule types introduced in CG-3, and only tested REMOVE and SELECT rules. The results for Dutch and Spanish are shown in Table 4.1, and the results for Finnish in Table 4.2.

The experiments revealed problems in all grammars. For the smaller grammars, we were able to verify manually that nearly all detected conflicts were true positives—we found one false positive and one false negative, when using a shortcut for the Spanish grammar. We did not systematically check for false negatives in any of the grammars, but we kept track of a number of known tricky cases; mostly rules with negations and complex set operations. As the tool matures and we add new features, a more in-depth analysis will be needed. Another natural follow-up evaluation will be to compare the accuracy of the grammar in the original state, and after removing the conflicts found by our tool.

4.3.1 Dutch

The Dutch grammar had two kinds of errors: rule-internal and rule interaction. As for rule-internal conflicts, one was due to a misspelling in the list definition for personal pronouns, which rendered 5 rules ineffective. The other was about subreadings: the genitive *s* is analysed as a subreading in the Apertium morphological lexicon, but it appeared in one rule as the main reading. There was one genuine conflict with rule interaction, shown below:

```
D1. REMOVE Adv IF (1 N) ;
    REMOVE Adv IF (-1 Det) (0 Adj) (1 N) ;
```

[3] https://svn.code.sf.net/p/apertium/svn/languages/apertium-nld/apertium-nld.nld.rlx, revision number r65111
[4] https://svn.code.sf.net/p/apertium/svn/languages/apertium-spa/apertium-spa.spa.rlx, revision number r66938
[5] https://github.com/flammie/apertium-fin/raw/master/apertium-fin.fin.rlx

	NLD	SPA	SPA $^{\text{sep. lem.}}$
# rules	59	279	279
# readings	336	3905	1735
# true positives $^{\text{AC}}$ (internal + interaction)	7 (6 + 1)	45 (21 + 24)	44 (20 + 24)
# true positives $^{\text{no AC}}$ (internal + interaction)	7 (6 + 1)	43 (18 + 25)	42 (17 + 25)
# false positives	0	0	1
🕒 with amb. classes	7 s	1h 46m	23 min
🕒 no amb. classes	3 s	44 min	16 min

Table 4.1: Results for Dutch and Spanish grammars.

These two rules share a target: both remove an adverb. The problem is that the first rule has a broader condition than the second, hence the second will not have any chance to act. If the rules were in the opposite order, then there would be no problem.

We also tested rules individually, in a way that a grammar writer might use our tool when writing new rules. The following rule was one of them:

```
D2. SELECT DetPos IF (-1 (vbser pres p3 sg)) (0 "zijn") (1 Noun);
```

As per VISL CG-3, the condition `(0 "zijn")` does not require *zijn* to be in the same reading with the target `DetPos`. It just means that at index 0, there is a reading with any possessive determiner, and a reading with any *zijn*. However, the intended action is to select a "det pos *zijn*" all in one reading; this is expressed as `SELECT DetPos + "zijn"`. In contrast, the 0-condition in example D_1 is used correctly: the adjective and the adverb are supposed to be in different readings.

Can we catch this imprecise formulation with our tool? The SAT-solver will not mark it as a conflict (which is the correct behaviour). But if we ask it to generate an example sequence, the target word may be either of the following options. Seeing interpretation a) could then direct the grammar writer to modify the rule.

a)
```
"<w2>"
    "w2" det pos f sg
    "zijn" vbser inf
```

b)
```
"<w2>"
    "zijn" det pos mfn pl
```

We found the same kind of definition in many other rules and grammars. To catch them more systematically, we could add a feature that alerts in all cases where a condition with 0 is used. As a possible extension, we could automatically merge the 0-condition into the target reading, then show the user this new version, along with the original, and ask which one was intended.

4.3.2 Spanish

The Spanish grammar had proportionately the highest number of errors. The grammar we ran is like the one found in the Apertium repository (linked on the previous page), apart from two changes: we fixed some typos (capital O for 0) in order to make it compile, and commented out two rules that used regular expressions, because we did not implement the support for them yet. For a full list of found conflicts, see the annotated log of running our program in https://github.com/inariksit/cgsat/blob/master/data/spa/conflicts.log.

We include two versions of the Spanish grammar in Table 4.1: in column SPA, we added the lemmas and word forms as described in Section 4.2.2, and in column $\text{SPA}^{\text{sep. lem.}}$, we just added each word form and lemma as individual readings, allowed to combine with any other reading. This latter version ran much faster, but failed to detect an internal conflict for one rule, and reported a false positive for another.

When we added ambiguity class constraints, we found three more internal conflicts. Interestingly, the version with ambiguity classes fails to detect an interaction conflict, which the simpler version reports, because one of the rules is first detected as an internal conflict. We think that neither of these versions is a false positive or negative; it is just a matter of priority. Sometimes we prefer to know that the rule cannot apply, given the current lexicon. However, we may know that the lexicon is about to be updated, and would rather learn about all potential interaction conflicts.

As an example of internal conflict, there are two rules that use `SET Cog = (np cog)`: the problem is that the tag "cog" does not exist in the lexicon. As another example, four rules require a context word tagged as NP with explicit number, but the lexicon does not indicate any number with NPs. It is likely that this grammar has been written for an earlier version, where such tags have been in place. One of the conflicts that was only caught by the ambiguity class constraints had the condition `IF (1 Comma) (..) (1 CnjCoo)`. The additional constraints correctly prevent commas from being ambiguous with anything else.

As for the 25 interaction conflicts, there were only 9 distinct rules that rendered 25 other rules ineffective. In fact, we can reduce these 9 rules further into 3 different groups: 4 + 4 + 1,

where the groups of 4 rules are variants of otherwise identical rule, each with different gender and number. An example of such conflict is below (gender and number omitted for readability):

```
s1. # NOM ADJ ADJ
    SELECT A OR PP IF (-2 N) (-1 Adj_PP) (0 Adj_PP) (NOT 0 Det);

    # NOM ADJ ADJ ADJ
    SELECT A OR PP IF (-3 N) (-2 N) (-1 Adj_PP) (0 Adj_PP) (NOT 0 Det);
```

In addition, the grammar contains a number of set definitions that were never used. Since VISL CG-3 already points out unused sets, we did not add such feature in our tool. However, we noticed an unexpected benefit when we tried to use the set definitions from the grammar directly as our readings: this way, we can discover inconsistencies even in set definitions that are not used in any rule. For instance, the following definition requires the word to be all of the listed parts of speech at the same time—most likely, the grammar writer meant OR instead of +:

```
s2. SET NP_Member = N + A + Det + PreAdv + Adv + Pron ;
```

If it was used in any rule, that rule would have been marked as conflicting. We noticed the error by accident, when the program offered the reading "*w2* n adj det preadv adv prn" in an example sequence meant for another rule.

As with the Dutch grammar, we ran the tool on individual rules and examined the sequences that were generated. None of the following was marked as a conflict, but looking at the output indicated that there are multiple interpretations, such as whether two analyses for a context word should be in the same reading or different readings. We observed also cases where the grammar writer has specified desired behaviour in comments, but the rule does not do what the grammar writer intended.

```
s3. REMOVE Sentar IF (0 Sentar) (..) ;

    SELECT PP IF (0 "estado") (..) ;
```

The comments make it clear that the first rule is meant to disambiguate between *sentar* and *sentir*, but the rule does not mention anything about *sentir*. Even with the ambiguity class constraints, the SAT-solver only created an ambiguity where *sentar* in 1st person plural

is ambiguous with an anonymous 1st person plural reading. This does not reflect the reality, where the target is only ambiguous with certain verbs, and in certain conjugated forms.

The second case is potentially more dangerous. The word form $estado_W$ can be either a noun ($estado_L$, 'state'), or the past participle of the verb $estar_L$. The condition, however, addresses the lemma of the noun, $estado_L$, whereas the lemma of the PP is $estar_L$. This means that, in theory, there can be a case where the condition to select the PP is already removed. As for now, the lexicon does not contain other ambiguities with the word form $estado_W$, but we could conceive of a scenario where someone adds e.g. a proper noun $Estado_L$ to the lexicon. Then, if some rule removes the lemma $estado_L$, the rule to select PP will not be able to trigger.

Another question is whether this level of detail is necessary. After all, the grammar will be used to disambiguate real life texts, where neither *sentar* nor *estado* are likely to have any other ambiguities. In fact, we are planning to change how we handle the lexical forms; with those changes, it will become clear whether the imprecision will result in potential errors, given the current lexicon.

4.3.3 Finnish

The results for the Finnish grammar are shown separately, in Table 4.2. We encountered a number of difficulties and used a few shortcuts, which we did not need for the other grammars—most importantly, not using the ambiguity class constraints. Due to these complications, the results are not directly comparable, but we include Finnish in any case, to give an idea how our method scales up: both to more rules, and more complex rules.

Challenges with Finnish The first challenge is the morphological complexity of Finnish. There are more than 20,000 readings, when all possible clitic combinations are included. After weeding out the most uncommon combinations, we ended up with sets of 4000–8000 readings.

The second challenge comes from the larger size of the grammar. Whereas the Spanish and Dutch had only tens of word forms or lemmas, the Finnish grammar specifies around 900 of them. Due to both of these factors, the procedure described in Section 4.2.2 would have exploded the number of readings, so we simply took the lemmas and word forms, and added them as single readings. In cases where they were combined with another tag in the grammar, we took that combination directly and made it into an underspecified reading: for

	1 CL + LEM	2 CL + LEM	1 CL + RDS	ONLY RDS
# readings	5851	9494	6657	2394
lexicon + grammar	4263 + 1588	7906 + 1588	4263 + 2394	0 + 2394
# conflicts	214	214	22	22
internal + interaction	211 + 3	211 + 3	19 + 3	19 + 3
🕒 all rules (approx.)	~4h 30min	~9h 30min	~7h 45min	~2h 30min

Table 4.2: Results for Finnish (1185 rules). (1 CL + LEM) 1 clitic + lemmas from the grammar; (2 CL + LEM) 2 clitics + lemmas from the grammar; (1 CL + RDS) 1 clitic + all readings from the grammar; (ONLY RDS) all readings from the grammar.

instance, we included both *aika* and "*aika* n" from the rule `SELECT "aika" + N`, but nothing from the rule `SELECT Pron + Sg`. This method gave us 1588 additional readings.

Finally, we were not able to create ambiguity class constraints—expanding the Finnish morphological lexicon results in 100s of gigabytes of word forms, which is simply too big for our method to work. For future development, we will see if it is possible to manipulate the finite automata directly to get hold of the ambiguities, instead of relying on the output in text.

Results The results are shown in Table 4.2. In the first column, we included only possessive suffixes. In the second column, we included question clitics as well. Both of these readings include the 1588 lemmas and word forms from the grammar. In the third column, we included all the tag combinations specified in the grammar, and in the fourth, we took only those, ignoring the morphological lexicon.

The first two variants reported a high number of internal conflicts. These are almost all due to nonexisting tags. The grammar was written in 1995, and updated by [43]; such a high number of internal conflicts indicates that possibly something has gone wrong in the conversion, or in our expansion of the morphological lexicon. As for accuracy, adding the question clitics did not change anything: they were already included in some of the 1588 sets with word forms or lemmas, and that was enough for the SAT-solver to find models with question clitics. We left the result in the table just to demonstrate the change in the running time.

The second two variants are playing with the full set of readings from the grammar. For both of these, the number of reported conflicts was only 22. Given the preliminary nature of the results, we did not do a full analysis of all the 214 reported conflicts. Out of the 22, we found 17 of them as true conflicts, but 5 seemed to be caused by our handling of rules with `*`: all of these 5 rules contain a LINK and multiple `*`s. On a positive note, our naive handling

of the * seems to cover the simplest cases. Some examples of true positives are shown in the following.

```
F1. "oma" SELECT Gen IF (..) (0C Nom) ;

    SELECT Adv IF (NOT 0 PP) (..) ;
```

Both of these are internal conflicts, which may not be trivial to see. The first rule requires the target to be genitive and unambiguously nominative; however, these two tags cannot combine in the same reading. As for the second rule, the definition of `PP` includes `adv` among others—with the sets unfolded, this rule becomes `SELECT adv IF (NOT 0 pp|adv|adp|po|pr) (..)`.

The following two examples are interaction conflicts:

```
F2. REMOVE A (0 Der) ;
    REMOVE N (0 Der) ;
    REMOVE A/N (0 Der) ;
```

This is the same pattern we have already seen before, but with a set of rules as the reason for conflict. The first two rules together remove the target of the third, leaving no way for there to be adjective or noun.

```
F3. SELECT .. IF (-1 Comma/N/Pron/Q) ;
    SELECT .. IF (-2 ..) (-1 Comma) ;
```

The rules above have been simplified to show only the relevant part. The conflict lies in the fact that `Comma` is a subset of `Comma/N/Pron/Q`: there is no way to trigger the second rule without placing a comma in position -1, and thereby triggering the first rule.

4.3.4 Performance

The running time of the grammars ranges from seconds to hours. Note that the times in the Finnish table are not entirely comparable with each other: we were forced to run the tests in smaller batches, and it is possible that there are different overheads, unrelated to the size of the SAT-problem, from testing 50 or 500 rules at a time. Despite the inaccuracies, we can see that increasing the number of readings and adding the ambiguity class constraints slow the program down significantly.

However, many of the use cases do not require running the whole grammar. Testing the interaction between 5–10 rules takes just seconds in all languages, if the ambiguity class constraints are not included. A downside in the ambiguity classes is that generating them takes

a long time, and while the overhead may be acceptable when checking the full grammar, it is hardly so when analysing just a handful of rules. We are working on an option to store and reuse the ambiguity class constraints.

4.4 Conclusions and future work

We set out to design and implement an automatic analysis of constraint grammars that can find problematic rules and rule combinations, without the need for a corpus. Our evaluation indicates that the tool indeed finds non-trivial conflicts and dead rules from actual grammars.

We did not have a volunteer to test the tool in the process of grammar writing, so we cannot conclude whether the constructed examples are useful for getting new insights on the rules. In any case, there are still a number of features to improve and add. Future work can be divided in roughly three categories: (a) general improvement of the tool (b) evaluation with users, and (c) integration as a part of CG development framework.

4.4.1 General improvement

Combining morphological and lexical tags Our solution to hardcode the tag combinations in the readings is feasible for simple morphology, but it can cause problems with more complex morphology. Currently, if we add one new lemma to the set of readings, we need to create as many new variables as there are inflectional forms for that lemma.

We are currently working on adding the concepts of lemmas and word forms directly to the representation of the possible readings. A possible solution would be to make each tag a variable, and ask the question "can this reading be a noun? singular? conditional?" separately for each tag. Then we could lift the restriction of tag combinations into the SAT side: make SAT-clauses that prohibit a comparative to go with a verb, or conditional with a noun. Alternatively, we can still hardcode the set of morphological readings, and only use SAT-clauses to restrict which lexical form can go with which morphological analysis.

Heuristic checks for common issues As mentioned earlier, some grammar design choices are common sources of misinterpretation. Many of these issues concern the case where the conditions include the target cohort—does `SELECT foo IF (0 bar)` mean that "foo" and "bar" should be in the same reading or in different readings? Lemmas and word forms are another

source of confusion, which is easy to check automatically against the lexicon. Ideally, these checks should be included in a special "paranoid mode", to not clutter the analysis[6].

Support for more features of VISL CG-3 As for longer-term goals, we want to handle more of the features in VISL CG-3, such as MAP, APPEND and SUBSTITUTE rules, as well as dependency structure. This also means finding different kinds of conflicts, such as dependency circularity. In order to implement rules that may add new readings, or new tags to existing readings, we need to modify our approach in the SAT-encoding. Even if the lexicon gives all readings that exist in the lexicon, the user might give a nonexistent reading, or in the case of MAP, a syntactic tag, which is (by definition) not in the lexicon. We may need to move to a more scalable solution.

4.4.2 Evaluation with user base

Our next step is to evaluate our tools together with actual grammar writers, in comparison with a corpus-based method or machine learning. Below, we envision some properties that might be interesting; however, we would be interested in getting feedback from actual grammarians and adding features based on what is needed.

Reformatting a rule Another possible feature is to suggest reformattings for a rule. Recall Figure 4.1 from the introduction; in the case on the right, the original rule was written by the original author, and another grammarian thought that the latter form is nicer to read. Doing the reverse operation could also be possible. If a rule with long disjunctions conflicts, it may be useful to split it into smaller conditions, and eliminate one at a time, in order to find the reason(s) for the conflict.

Suggesting alternative orders On a speculative note, it could be interesting to identify pairs for a potential "feeding order" that is missed in the grammar. Say we have the following rule sequence:

```
REMOVE:r1 x IF (-1C y)
SELECT:s2 y IF (...)
```

[6] As pointed out by Eckhard Bick, the program should act upon this only if the 0 relates to an existing ambiguity class.

If $s2$ appears before $r1$, if makes way for $r1$ to act later on the same round. However, if the rules are ordered as shown, and y is not unambiguous from the beginning, then $r1$ has to wait for the next round to be applied.

Lager [33] observed that the rule sequence learned by the μTBL system did not remove further ambiguities after its first run, and concluded that the sequence was "optimal". It would be interesting to recreate the goals in [33] and [6], to see if this semantic analysis of rule dependencies could lead also to better ordering within a grammar.

Of course, it remains to be seen if any of these improvements would make a difference in speed; VISL CG-3 is already very fast, when the grammars are run multiple times.

4.4.3 Integration as a part of CG development environment

In order to attract the attention of the CG community, it would be desirable to incorporate the tools as a part of existing CG software. Currently, the described software consists of just under 3000 lines of Haskell code, including both the CG engine and the grammar analysis tool. The grammar used for parsing the original CG files is written in BNFC [1], and it is missing many constructs in CG-3. Given these factors, the preferred option would be a full reimplementation and integration as a part of VISL CG-3, or any other CG development framework. We believe this would make the tools easier to use, more cohesive and synchronised with the main CG engine, and likely much faster. Of course, it is up to the community and the developers to decide if these tools are of interest.

Chapter 5

Conclusions

This chapter concludes the thesis. Firstly, we summarise the main results of this thesis. For the remainder of this chapter, we discuss insights gained from this work, and possible directions for future research.

5.1 Summary of the thesis

We set out to express CG as a SAT-problem with the observation that CG rules resemble logical formulas; REMOVE and IF can be expressed as negations and implications respectively. As noted by Torbjörn Lager [32], a straight-forward implementation in a logical framework results in a parallel and unordered CG. Our implementation of the sequential CG is very similar to the encoding by Lager and Nivre [34]. However, we modified the setup in a crucial detail: instead of an actual sentence, we applied the rules to a *symbolic sentence*, where every word contains every possible reading. Then, instead of the original question "Which readings are true after all rule applications?", we found a meaningful SAT-problem, with the question "Which readings were originally true?"

We exploited this newly found property for grammar analysis: for each rule r, we tried to create a sentence which has passed through all previous rules R, and could still trigger r. If such sentence could not be found, this means that some of the rules in R made r impossible to apply: for instance, by removing the same target with less conditions, or removing something from the context of r. This method was successful in finding such conflicts, when tested with grammars between 60–1200 rules.

5.2 Gained insights and open questions

Let us refer back to Lager and Nivre [34]. Before starting their logical reconstruction of POS tagging methods, they give a list of motivations why one would want to reconstruct a method in the first place. Below we quote three of them, which we feel that apply to our work.

- It allows us—even forces us—to clarify and make more precise a formulation of a method $[M]$ which is vague and open to interpretation.
- It may put us in a position to find novel implementation techniques for M. In particular, T_M in combination with a theorem prover implementing I_M may turn out to be a perfectly practical implementation of M.
- By factoring out the knowledge component T_M from a method M, we may be able to find interesting novel uses for T_M which M was not designed for in the first place.

Clarify and make more precise There has never been a single specification of all the implementation details of CG; specifically, the rule ordering and the execution strategy. The lack of specification in CG formalism can be seen as a feature, rather than a bug—Karlsson states the following in the initial specification of the design goals of CG [28] (page 11):

> More generally, the formalism should be such that individual constraints are unordered and maximally independent of each other. These properties facilitate incremental adding of new constraints to the grammar, especially to a large one.

The spirit is retained in later descriptions, cf. Huldén [26]:

> The formalism specifies no particular rule ordering per se, and different implementations of the CG formalism apply rules in varying orders (Bick, 2000). In this respect, it is up to the grammar writer to design the rules so that they operate correctly no matter in what order they are called upon.

If we were to take these requirements literally, we would run into problems with any real-life grammar; that is, if any two rules target the same cohort, or if a rule r targets the context of another rule r', we step out of the ideal of the maximal independence of rules. In any grammar of interesting coverage, rule interaction must be taken into account.

According to our initial experiments, there are differences of 1-2 percentage points in the F-scores, when running the same grammar with different execution and ordering schemes.

However, the results are not very conclusive, because the tested grammars were very small and poor quality. Running the same grammar with different rule orders have been tried before; Lager [33] reports that randomising the order of the 902 learned rules decreases both precision and recall around 1.3 %, compared to the initial learned rule order. Bick [6] was able to improve the performance of a human-written grammar by automatically modifying a number of variants, including rule order. It would make an interesting addition to those studies, if we repeated our experiments in Chapter 3 with large, high-quality grammars. Then we could better isolate the effects of rule ordering and execution strategy for different grammars. Compared to other parallel formalisms [30, 53, 39, 32], SAT-CG has the benefit that it reads the exact same CG files as the major CG-2 and CG-3 implementations—this means that we can run the same grammars and compare the results directly.

In the same spirit, if we want to be able to implement grammar development and debugging facilities, we must specify an ordering scheme and an execution strategy. Our grammar analysis tool, presented in Chapter 4, is meant for strict and sequential variants of CG, especially VISL CG-3 [17].

The moral of the story is that CG cannot be defined just by a set of rules: rather, it is the combination of rules, a rule ordering scheme (generalisable to a *conflict solving scheme*), and an execution strategy.

Novel implementation techniques Parallel CG is an instance of a more abstract constraint solving problem. As we motivate in Section 2.4, it is a smart idea to formulate such task as a SAT-problem: the SAT-solving community has spent decades on optimising search. By spending one's individual effort in finding a SAT-encoding, one can reap the benefits of fast search, including any advances that will happen in the future.

Despite the argument, our evaluations in Chapter 3 show that the performance of SAT-CG was 1-2 orders of magnitude slower, compared to VISL CG-3. However, we cannot conclude that the fault is at SAT; rather, we attribute some of the differences to pure software engineering aspects, such as the chosen programming languages, the experience of the developers and the time available to dedicate for the development. As an analogy, Peltonen [41] presents an implementation of CG using FSTs, which runs 1,500 times slower than VISL CG-3—this was clearly not the last word on the performance of FST implementations. Only three years later, Nemeskey et al. [38] evaluate different finite-state implementations of CG, and report more competetive figures for performance.

From another point of view, we can argue that executing parallel CG is more complex than executing sequential CG. Hence the comparison between SAT-CG and VISL CG-3 would

be unfair, because the two engines would not be doing the same task. In order to investigate this hypothesis further, it would be interesting to experiment with a SAT-based parser for FSIG.

For the grammar analysis tool, there is no pre-existing counterpart, with or without SAT. In the spirit of "novel implementation techniques", we would like to throw the challenge to the community: can we replicate or improve the results of the SAT-based, semantic grammar analysis?

Novel uses for the knowledge component of CG CG is first and foremost a parsing grammar, meant to reduce ambiguities that appear in real-life texts. Usually, CG rules are written to target only what is needed to disambiguate real texts, not to provide a full description of language; a rule such as `SELECT punct IF (0 ".")` would be utterly redundant. However, these words may still appear frequently in the rules, as contexts.

Let us return back to the claim by Bick and Didriksen [7], of CG being a declarative whole of possibilities and impossibilities of a language or a genre. Applying CG rules to a symbolic sentence, instead of real sentences from a corpus, we can use CG—even sequential CG—in a more declarative way: `REMOVE v IF (-1 det)` does not remove the verb after determiner, it prevents such a sequence from being created in the first place. Thus, we can use CG for a task it was not designed for: creating language. The "knowledge component" of CG would be then all the hidden assumptions about the targets and possible contexts for them.

But do the rules contain enough information to actually create language? Some tendencies would be expected to show: for instance, sentence boundaries usually appear as the left- or rightmost condition in any rule, thus, they would likely appear at the borders in the generated sequences as well. For more general phenomena, our experiments in Chapter 4 did not result in very meaningful sequences; however, as we state in conclusions of the said chapter, there is a lot of room for improvement in our encoding, such as the handling of lexical forms. As another factor, we have only tested the approach with grammars of low coverage—it is possible that a high-quality grammar would contain more information, and generate sequences with more resemblance to real sentences.

The question about expressivity of CG deserves further investigation. We believe that the concepts and methods developed in the present thesis can help; the symbolic sentences we used for grammar analysis in Chapter 4 correspond to the earlier presented notion of maximally ambiguous sentence. If the input language is standardised to the maximally ambiguous sentences, where every cohort contains all the readings in Σ, we may be able to define grammars for interesting languages more easily.

Bibliography

[1] *BNF Converter, vers. 2.7.1.* http://bnfc.digitalgrammars.com.

[2] Bick, Eckhard. 2000. *The parsing system "Palavras". Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework*. Ph.D. thesis, University of Århus.

[3] Bick, Eckhard. 2003. A CG & PSG hybrid approach to automatic corpus annotation. *Pages 1–12 of: Proceedings of the Shallow Processing of Large Corpora Workshop (SProLaC 2003)*.

[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.

[6] Bick, Eckhard. 2013. ML-Tuned Constraint Grammars. *Pages 440–449 of: Proceedings of the 27th Pacific Asia Conference on Language, Information and Computation (PACLIC 2013)*.

[7] Bick, Eckhard, & Didriksen, Tino. 2015. CG-3 – Beyond Classical Constraint Grammar. *In: Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*.

[8] Biere, Armin. 2009. *Handbook of Satisfiability*. Frontiers in artificial intelligence and applications. IOS Press.

[9] Biere, Armin, Cimatti, Alessandro, Clarke, Edmund, & Zhu, Yunshan. 1999. Tools and Algorithms for the Construction and Analysis of Systems: 5th International Conference, TACAS'99 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'99 Amsterdam, The Netherlands, March 22–28, 1999 Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg.

[10] Bradley, Aaron R. 2011. SAT-Based Model Checking without Unrolling. *Pages 70–87 of:* Jhala, Ranjit, & Schmidt, David (eds), *Proceedings of the 12th International Conference on*

Verification, Model Checking, and Abstract Interpretation (VMCAI 2011). Berlin, Heidelberg: Springer Berlin Heidelberg.

[11] Brill, Eric. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, **21**(4), 543–565.

[12] Chomsky, Noam. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, **2**(3), 113–124.

[13] 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.

[14] Claessen, Koen, Fisher, Jasmin, Ishtiaq, Samin, Piterman, Nir, & Wang, Qinsi. 2013. *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. Model-Checking Signal Transduction Networks through Decreasing Reachability Sets, pages 85–100.

[15] Cook, Stephen A. 1971. The Complexity of Theorem-proving Procedures. *Pages 151—158 of: Proceedings of the Third Annual Symposium on Theory of Computing*.

[16] Cutting, Doug, Kupiec, Julian, Pedersen, Jan, & Sibun, Penelope. 1992. A Practical Part-of-speech Tagger. *Pages 133–140 of: Proceedings of 3rd Conference on Applied Natural Language Processing*. Stroudsburg, PA, USA: Association for Computational Linguistics.

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

[18] Eén, Niklas, & Sörensson, Niklas. 2004. An Extensible SAT-solver. *Pages 502–518 of:* Giunchiglia, Enrico, & Tacchella, Armando (eds), *Theory and Applications of Satisfiability Testing*. Lecture Notes in Computer Science, vol. 2919. Springer Berlin Heidelberg.

[19] Eineborg, Martin, & Lindberg, Nikolaj. 1998. Induction of Constraint Grammar-rules using Progol. *Pages 116–124 of:* Page, David (ed), *Inductive Logic Programming*. Lecture Notes in Computer Science, vol. 1446. Springer Berlin Heidelberg.

[20] Eén, Niklas, & Sörensson, Niklas. 2006. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, **2**, 1–26.

[21] Graña, Jorge, Andrade, Gloria, & Vilares, Jesús. 2002. Compilation of constraint-based contextual rules for part-of-speech tagging into finite state transducers. *Pages 128–137 of:* Champarnaud, Jean-Marc, & Maurel, Denis (eds), *Implementation and Application of Automata*. Lecture Notes in Computer Science, vol. 2608. Springer.

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

[23] Gross, Maurice. 1997. The construction of local grammars. *Pages 329—354 of:* Roche, Emmanuel, & Schabes, Yves (eds), *Finite-state language processing*.

[24] Herz, Jacky, & Rimon, Mori. 1991. Local syntactic constraints. *Pages 200–209 of: Proceedings of the Second International Workshop on Parsing Technologies*.

[25] Hindle, D. 1989. Acquiring Disambiguation Rules from Text. *Pages 118–125 of: Proceedings of the 27th Annual Meeting of the ACL*.

[26] Hulden, Mans. 2011. Constraint Grammar parsing with left and right sequential finite transducers. *Pages 39–47 of: Proceedings of 9th International Workshop on Finite State Methods and Natural Language Processing*. Association for Computational Linguistics.

[27] Karlsson, Fred. 1990. Constraint Grammar as a Framework for Parsing Running Text. *Pages 168–173 of: Proceedings of 13th International Conference on Computational Linguistics (COLING 1990)*, vol. 3. Stroudsburg, PA, USA: Association for Computational Linguistics.

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

[29] Kiparsky, Paul. 1968. Linguistic Universals and Linguistic Change. *Pages 170–202 of:* Bach, Emmon, & Harms, R. (eds), *Universals in Linguistic Theory*. Holt, Rinehart, and Winston.

[30] Koskenniemi, Kimmo. 1990. Finite-state Parsing and Disambiguation. *Pages 229–232 of: Proceedings of 13th International Conference on Computational Linguistics (COLING 1990)*, vol. 2. Stroudsburg, PA, USA: Association for Computational Linguistics.

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

[32] Lager, Torbjörn. 1998. Logic for Part of Speech Tagging and Shallow Parsing. *In: Proceedings of the 11th Nordic Conference on Computational Linguistics (NODALIDA 1998)*.

[33] Lager, Torbjörn. 2001. Transformation-based learning of rules for constraint grammar tagging. *In: Proceedings of the 13th Nordic Conference on Computational Linguistics (NODALIDA 2001)*.

[34] Lager, Torbjörn, & Nivre, Joakim. 2001. Part of speech tagging from a logical point of view. *Pages 212–227 of: Logical Aspects of Computational Linguistics, 4th International Conference (LACL 2001)*.

[35] Listenmaa, Inari, & Claessen, Koen. 2015. Constraint Grammar as a SAT problem. *In: Constraint Grammar workshop at the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*.

[36] Listenmaa, Inari, & Claessen, Koen. 2016. Analysing Constraint Grammars with a SAT-solver. *In: Proceedings of the 10th edition of the Language Resources and Evaluation Conference (LREC 2016)*.

[37] 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.

[38] Nemeskey, Dávid Márk, Tyers, Francis, & Hulden, Mans. 2014 (August). Why Implementation Matters: Evaluation of an Open-source Constraint Grammar Parser. *Pages 772–780 of: Proceedings of the 25th International Conference on Computational Linguistics (COLING 2014)*.

[39] Oflazer, Kemal, & Tür, Gökhan. 1997. Morphological disambiguation by voting constraints. *Pages 222–229 of: Proceedings of the 8th Conference of the European Chapter of the Association for Computational Linguistics (EACL 1997)*.

[40] Padró, Lluís. 1996. A constraint satisfaction alternative for POS tagging. *In: Proceedings of NLP+IA/TAL+AI*.

[41] Peltonen, Janne. 2011. *Rajoitekielioppien toteutuksesta äärellistilaisin menetelmin*. M.Phil. thesis, University of Helsinki.

[42] Piitulainen, Jussi. 1995. Locally tree-shaped sentence automata and resolution of ambiguity. *Pages 50–58 of: Proceedings of the 10th Nordic Conference of Computational Linguistics*.

[43] Pirinen, Tommi. 2015. Using weighted finite state morphology with VISL CG-3—Some experiments with free open source Finnish resources. *In: Constraint Grammar workshop at the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*.

[44] Reynolds, Robert, & Tyers, Francis. 2015. A preliminary constraint grammar for Russian. *In: Constraint Grammar workshop at the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*.

[45] Samuelsson, Christer, Tapanainen, Pasi, & Voutilainen, Atro. 1996. Inducing constraint grammars. *Pages 146–155 of:* Miclet, Laurent, & de la Higuera, Colin (eds), *Grammatical Interference: Learning Syntax from Sentences*. Lecture Notes in Computer Science, vol. 1147. Springer Berlin Heidelberg.

[46] Selman, B., & Kautz, H. 1992. Planning as satisfiability. *Pages 359—363 of: European Conference on Artificial Intelligence*.

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

[48] Sheeran, Mary, & Stålmarck, Gunnar. 1998. *Formal Methods in Computer-Aided Design: Second International Conference, FMCAD' 98 Palo Alto, CA, USA, November 4–6, 1998 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. A Tutorial on Stålmarck's Proof Procedure for Propositional Logic, pages 82–99.

[49] Silfverberg, Miikka, & Lindén, Krister. 2009. Conflict resolution using weighted rules in HFST-TWOLC. *Pages 174–181 of: Proceedings of 17th Nordic Conference of Computational Linguistics (NODALIDA 2009)*.

[50] Tapanainen, Pasi. 1996. *The Constraint Grammar Parser CG-2*. Publications of the Department of General Linguistics, University of Helsinki, vol. 27. Yliopistopaino, Helsinki.

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

[52] Tapanainen, Pasi, & Järvinen, Timo. 1997. A non-projective dependency parser. *Pages 64—71 of: Proceedings of 5th Conference on Applied Natural Language Processing*. Stroudsburg, PA, USA: Association for Computational Linguistics.

[53] Voutilainen, Atro. 1994. *Designing a parsing grammar*. Publications of the Department of General Linguistics, University of Helsinki, vol. 22. Yliopistopaino.

[54] Voutilainen, Atro. 1998. Does tagging help parsing? A case study on finite state parsing. *Pages 25–36 of: Proceedings of the International Workshop on Finite State Methods in Natural Language Processing (FSMNLP 1998)*. Association for Computational Linguistics.

[55] Voutilainen, Atro. 2004. Hand Crafted Rules. *Pages 217–246 of:* van Halteren, H. (ed), *Syntactic Wordclass Tagging*. Kluwer Academic.

[56] Voutilainen, Atro. 2011. FinnTreeBank: Creating a research resource and service for language researchers with Constraint Grammar. *Pages 41–49 of: Proceedings of the Constraint Grammar workshop at the 18th Nordic Conference of Computational Linguistics (NODALIDA 2011)*.

[57] Yli-Jyrä, Anssi. 2003a. Describing syntax with star-free regular expressions. *Pages 379–386 of: Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics*, vol. 1. Association for Computational Linguistics.

[58] Yli-Jyrä, Anssi. 2003b. Regular approximations through labeled bracketing. *Pages 189—201 of:* G. Jäger, P. Monachesi, G. Penn, & Wintner, S. (eds), *Proceedings of the 8th conference on Formal Grammar (FGVienna 2003)*.

[59] Yli-Jyrä, Anssi. 2004. Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyse crossing bracketings. *Pages 33–40 of:* Kruijff, G.-J. M., & Duchier, D. (eds), *Workshop of Recent Advances in Dependency Grammar*.

[60] Yli-Jyrä, Anssi. 2005. *Contributions to the theory of finite-state based grammars*. Ph.D. thesis, University of Helsinki.

[61] Yli-Jyrä, Anssi. 2011. An efficient constraint grammar parser based on inward deterministic automata. *Pages 50–60 of: Constraint Grammar workshop at the 18th Nordic Conference of Computational Linguistics (NODALIDA 2011)*.

[62] Yli-Jyrä, Anssi. 2001. Structural Correspondence between Finite-State Intersection Grammar and Constraint Satisfaction Problem. *Pages 1–4 of: Finite State Methods in Natural Language Processing 2001 (FSMNLP 2001), ESSLLI Workshop*.