Thesis for the Degree of Licentiate of Philosophy

Analysing Constraint Grammar with SAT

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

Inari Listenmaa

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

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

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

Research groups: Functional Programming / Language Technology
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Sweden
Telephone +46 (0)31-772 1000

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

Abstract

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

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

Acknowledgements

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

The greatest thanks go to my supervisors Koen Claessen and Aarne Ranta. You have guided this work from an afternoon experiment (where I probably procrastinated marking student labs) to an actual thesis. I have learnt a lot about research, **[TODO:** bananas**]** 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. The latter goes for Tino Didriksen, in double. On the other side of the rule-based NLP community, Pepijn Kokke has contributed tremendously to the notion of expressivity of CGs.

Finally, I want to thank all the awesome people I've met at Chalmers and outside! My time at the department has been great–thanks to `[ x | x <- people, inarisFriend 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 example sentences and annotate them manually. Based on these 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 in the sentence.

2. We can write a generative grammar to describe the English language, and analyse this sentence with it. 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 look up all morphological analyses, separately for each word. Given that we know nothing about the context in the lookup phase, we start off with both wishes potentially $wish_N$ and $wish_V$. Now, in the spirit of approach 1, we can look at the context words for help. A learning-based method may encounter plenty of examples where *wish* appearing after *a* is a noun, and learn to assign a high probability to that hypothesis. 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 you find something that can be a verb, but it is preceded by a determiner, throw that guess away". Add some hundreds of these rules, and they expose the structure hidden behind the clutter.

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

In the grand scheme of grammar formalisms, CG is markedly 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, 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. From these two factors, it is natural that errors creep in without anyone noticing. This thesis aims to create methods for analysing and detecting conflicts in CGs, as well as contribute to the theoretical understanding of the formalism and its various implementations.

1.1 Scope of this thesis

Automatic creation and analysis of CG rules has been a research topic since the beginning of the formalism. Already [**?**] envisions ...

This thesis has two main contributions in the field of CG.

Theoretical understanding of CG

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

Throughout its history, CG has been very practically oriented, with little studies on its formal properties, or attempts to relate it to existing frameworks. Notable exeptions are [**?**], who model CG in logic, and [**?**], who evaluate different finite-state based parsers for efficiency.

[TODO: find "Hulden (2011) showed that if the rules are compiled to transducers ..." – what kind of scheme was that? is there an ordered FST-based CG engine? Also read Peltonen 2011.**]**

This brings us to the yet unmentioned part of the title. We model CG in the framework of *Boolean satisfiability*, abbreviated *SAT*. The analyses of the word forms are translated into Boolean variables, and the constraint rules into logical formulas operating on those variables. Applying a rule becomes a task of assigning truth values to different analyses, such that ultimately each word should have one true value.

This simple translation gives us properties that standard CG engines do not have. Most importantly, the rules lose their independence: any conflict between two rules renders the formula unsatisfiable. While this is not necessarily a desirable property in a CG engine—it is hard to justify the loss of robustness—it is nevertheless interesting for other purposes. Encoding the rules in logic enables us to track the effect of each rule: each rule which triggers at the instance of *wish* creates a clause, and in case of a conflict, we can find out which clauses cause it.

Furthermore, we are not restricted analysing only existing texts—in fact, we can analyse *any possible text*. We can feed our rules with a *maximally ambiguous sentence*: every word starts off with every possible analysis. This setup brings us lets us conceive a notions of *generation* and *expressivity* within the framework of CG.

Analysis and quality control of CG

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

The most important contribution is, however, a practical one. With the rules modelled in logic, we can track their effects and find out if they contradict each other. In Chapter 4, we describe a method and a working implementation to analyse existing grammars. In addition, the method which allows us to *generate* sequences, is used for giving the grammar writer feedback of the rules they are writing.

1.2 Structure of this thesis

The core of this thesis is an extension of two articles: [**?**] and **[TODO:** Listenmaa and Claessen (2016)**]**. Some of the content has been updated since the initial publication; in ad-

dition, 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. Chapter 4 presents the method of grammar analysis using our SAT-based implementation, along with evaluation on three different grammars. Chapter 5 concludes the thesis.

Chapter 2

Background

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

2.1 Introduction to CG

This used to be my course paper for Grammar Formalisms. (TODO: should I write here something in depth about e.g. set operations in target/conditions?)

Constraint Grammar (CG) is a formalism for disambiguating morphologically analysed text. It was first introduced by [**?**], and has been used for many tasks in computational linguistics, such as POS tagging, surface syntax and machine translation [**?**]. It disambiguates output by morphological analyser by using constraint rules which can select or remove a potential analysis (called 'reading') for a target word, depending on the context words around it. Together these rules disambiguate the whole text.

In the example below, I show possible analyses for the sentence "the bear sleeps". To introduce the terminology: each *word form* appears surrounded by "< >". The analyses for each word form contain one lemma, and a list of morphological tags. A list of lemma and tags is called *reading*, and a word form together with its readings is called a *cohort*. A cohort is ambiguous, if it contains more than one reading.

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

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

We can disambiguate this sentence with two rules:

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

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

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

It is also possible to add syntactic tags and dependency structure within CG. After disambiguating *bear* by part of speech (noun), it remains to disambiguate whether it is a subject, object, adverbial or any other possible syntactic role. This 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). The rules to add work similarly to the rules to remove and select: if a target matches the context, then the new tag is added. These rules are usually run after the morphological disambiguation is finished. The following rules could be applied to the example sentence. The letter 'C' after the position means *careful* context: e.g. `(-1C noun)` requires the previous word to be unambiguously noun. Such rule would not fire if the previous word has any other reading in addition to the noun reading.

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

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

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

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

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

2.2 Properties of Constraint Grammar

[**?**] lists 24 design principles and describes related work at the time of writing. Here we

summarise a set of main features, and relate CG to the developments in grammar formalism since the initial CG description.

All theories and implementations of CG a *reductionist* system, designed primarily for analysis, not generation. Its task is to parse sentences that are given, 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, but rather bits and pieces with concrete conditions. The rules are self-contained and mutually independent—this makes it easy to add exceptions, and exceptions to exceptions, without changing the more general rules.

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

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

2.2.1 Reductionist vs. licencing (or Analysis vs. Generation???)

CG is a reductionist system: it starts from a set of alternative analyses, 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 can be contrasted with a licencing system, where 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.

From Fred's 1995 book

> Notice, in passing, that the overall reductionistic set-up gives an interesting psycholinguistic prediction. In terms of processing load, less processing is required under Constraint Grammar to turn out an ambiguous and therefore unclear analysis than to turn out a disambiguated and therefore structurally clear analysis. In our opinion, this seems to be a reasonable psycholinguistic hypothesis a priori. Mental effort is needed for achieving clarity, precision, and maximal information. Less efforts imply (retention of) unclarity and ambiguity, i.e. information decrease. In several types of parsers, rule applications create rather than

discard ambiguities: the more processing, the less unambiguous information.

According to the initial specification, it is a grammar for analysis: [**?**] does not see it fit for generation:

> It would be optimal if the formalism were general and abstract enough for it, or rather for grammars written in the framework of it, to be used both for sentence analysis and sentence generation. Constraint Grammar takes no explicit stand on this issue. 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, [**?**] describe CG as "a declarative whole of contextual possibilities and impossibilities for a language or genre", which is nevertheless implemented in a low-level way: selecting and removing readings from individual words, without explicit connection between the rules. Bick and Didriksen argue that as a side effect, the rules actually manage to describe language.

We return to the questions of generation and declarativity in section 2.2.4. In chapter 4, we revisit these questions in the context of CG analysis.

No enforcement of grammaticality [**?**] states explicitly:

> The foremost task of a parsing-oriented grammar is rather to aid in parsing every input than to define the notion "grammatically correct sentence", or to describe "all and only the grammatical sentences".

CG does not define sequences as *grammatically correct*. Beside the lack of long-distance dependencies, there is no mandatory enforcement of even local well-formedness, such as gender agreement. However, this is often helpful for practical purposes, because it adds robustness. 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 a noun ('juniper'). The second word, *bord*, is a neuter noun ('table')—the writer has most likely meant to write "a table" instead of "juniper table", but has mistaken the gender of 'table'. The correct form, which also would not be ambiguous, is "ett bord".

CG allows the rules to be as fine-grained or broad as we want: `SELECT det IF (1 noun) (0 noun)` would match any determiner and any noun, and succesfully disambiguate *en* in this case. We can also enforce grammaticality by writing `SELECT det utr sg IF (1 noun utr sg)`. In that case, nothing in the example 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. If we define *grammatical sequences* to be sequences of POS tags, then the question is somewhat easier to answer: `<det><utr> <noun><neutr>` is ungrammatical, and CG doesn't raise an alarm that there is something wrong. However, the alternative `<noun><utr> <noun><neutr>` might be in that particular context even "more wrong"; same goes for the option with no disambiguation at all. As

2.2.2 Shallow syntax

CG can be described as shallow for various reasons. The rules usually operate on a local context of a few words.

More importantly, CG analysis is not aimed to produce trees. (IS THIS TOO MUCH OF REPETITION OF PREVIOUS?)

Whereas a generative grammar could fail to produce any analysis, if even one of the words is unknown or misspelt, CG would, at worst, just leave that part ambiguous, and do as good a job it can elsewhere in the sentence.

Flat structure The rules do not produce any hierarchical structure.

Syntactic labels may be added, but no invisible structure is postulated. The syntax of CG is dependency syntax.

Syntactic labels are functionally same as morphological: `@SUBJ` is just one more tag in a reading.

The rules do not usually handle long-distance dependencies; they operate on a unit of

a few words, and don't abstract away from surface features such as word order. You need to define rules separately for stuff like "select noun if -1 determiner" and "select noun if -2 determiner -1 adjective".

More importantly, CG doesn't introduce a full tree structure to the whole sentence. Some local subtrees can be detected, but mostly for the purposes of aiding the disambiguation. (TODO: should I talk about dependency trees in CG-3?)

Description by elimination [?]

> For example, ordinary phrase structure rules for English NPs would state that a determiner is followed by certain optional premodifiers plus an obligatory noun. A consequence of this is that a determiner cannot be followed by a finite or infinitive verb form, without an intervening noun. In Constraint Grammar, a constraint could be postulated to this effect, discarding the verbal readings of the word-form bank in expressions like the bank, the old bank.

CG has a more lightweight task than e.g. a phrase structure grammar: a successful disambiguation can depend on just a local context, in a window of 1-3 words left or right. To demonstrate the the difference, let us take the same the example sentence *the bear sleeps*, and a context-free grammar with the following productions:

```
S   -> NP VP
VP  -> V NP
NP  -> Det N
Det -> "the"
N   -> "bear" | "sleeps"
V   -> "bear" | "sleeps"
```

There is no way to reach `S` if *bear* is analysed as `V` and *sleeps* as `N`; thus we get the individual words disambiguated thanks to the parser, which introduces a structure to the whole sentence. If one of the words was out of vocabulary, say *the bear warbles*, there would be no analysis for *the* or *bear* either. In CG, an unrecognised verb would be no problem at all in disambiguating the previous words.

Low abstraction Contrasting with phrase-structure or dependency grammars, the rules in CG are often on a lower level of abstraction. The rule that tells to remove verb reading after a determiner doesn't tell us about the structure of noun phrase; we need a second rule to

remove a verb after a determiner and an adjective (“the happy bear sleeps”), and a third rule to remove verb after a possessive pronoun (“my bear sleeps”). A phenomenon that is expressed with one big rule in a deep formalism such as HPSG or categorial grammar, is split into many low-level rules.

Mutual independence On the other hand, these features can provide flexibility that is hard to mimic by a deeper formalism. For instance, 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 doesn’t create a complicated inheritance hierarchy of different verb types. We can introduce a similar granularity in other systems, for instance, a grammar in Grammatical Framework, which would include the functions `GenVP` for whichever verbs and `CogVP` for cognitive verbs:

```
GenVP : V2    -> NP -> VP ;
CogVP : CogV2 -> NP -> VP ;
```

In order to parse as much text as without the subcategorisation, the grammar writer would have to add `CogV*` counterparts all functions that exist for `V*`, which duplicates code; or add coercion functions from `CogV*` to `V*`, which increases ambiguity. In case of multiple parses, likely the one with more information would be preferred, but the CG solution gives more flexibility. If a rule which matches `CogV` is introduced before (more on ordering in the following section) a rule which matches `V`, it is applied before, and vice versa. This allows for very fine control, for example combining the semantic properties of the verb and the animacy of the arguments.

Theoretical principles after all these practical examples This lack of deep structure is intentional in the design of CG. [**?**] 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 a CG, one has access to all levels at the same time, ranging from morphology to syntax or dependency labels, or even semantics, if one wants to introduce semantic roles in the tagset. One can have a rule such as `REMOVE ... IF (+1 "bear" <noun> @Object §Patient)`, followed by a second rule which only considers the POS of the context word.

2.2.3 Ordering of the rules

The previous properties of Constraint Grammar formalism and rules were specified in [**?**], and retained in further implementations. However, the notion of *ordering* has been interpreted more freely. [**?**] states on page 17:

> Each single statement is true when examined in isolation, either absolutely or with some degree of certainty, depending upon how careful the grammar writer has been. Furthermore, disregarding morphosyntactic mappings, the constraints are *unordered*.

After the initial specification, there have been several independent implementations of CG, with different ordering schemes. In the following section, we will present the different parameters of ordering.

The execution of the rules depends on the following parameters: strict vs. heuristic ordering, and sequential vs. parallel execution. These parameters may combine freely: strict and sequential, strict and parallel, heuristic and sequential, or heuristic and parallel. Throughout the section, we will apply the rules to the following ambiguous passage:

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

Strict vs. heuristic This aspect concerns the ordering of the rules in the file. An implementation with *strict order* applies each rule in the order in which they appear in the file. If a grammar contains the rules "select noun after determiner" and "select verb after pronoun" in the given order, the rule that selects noun will be applied first. After it has selected the noun reading for *question*, 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. More careful rules are usually placed first, followed by stronger rules—see the pattern below.

```
SELECT <rare analysis> IF <rare condition> ;
...
REMOVE <rare analysis> ;
```

If a rare condition is met, the rare analysis is selected, and since it will be the only analysis, it will be protected from the remove rule. If the rare condition is not met, the rare analysis is removed.

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

Both methods have their strengths and weaknesses. A strict order is more predictable, but it also means that the grammar writers need to pay more thought to rule interaction. A heuristic order frees the grammar writer from finding an optimal order, but it can give unexpected results, which are harder to debug. As for existing implementations, [**?**] follows the strict scheme, whereas [**?**] is heuristic.

Sequential vs. parallel 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 "select noun after determiner".

In contrast, a *parallel execution* disambiguates all the words at the same time, using their initial, ambiguous context. Both "select noun after determiner" and "select verb after pronoun" will be applied to *question*, because the original input from the morphological analyser contains both determiner and pronoun as the preceding word.

Which execution scheme is better? For most purposes, a sequential execution is preferred. Sequential execution—together with strict or heuristic rule order—makes the rule take action immediately, and updates the context for the following words. If we know the correct part-of-speech for *what*, then it is easier to disambiguate *question*. Parallel execution may have

uses in more marginal cases: for instance, if we have a small rule set and we want to find out if the rules are consistent with each other.

[TODO: See Tino's table in CG3 manual: foreach word: foreach rule etc.**]**

No ordering What does it even mean? Unordered is just FSIG–how can it not break with conflicts? Don't tell me "there are

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

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

Comparison to FSIG Another example of a reductionist and shallow-syntax formalism is Finite-State Intersection Grammar (FSIG) [**?**]. In the formalism, the tagged sequences of words, as well as rules, are modelled as finite state automata: for instance, there are two transitions from the state *what* to the state *question* in the automaton representing *what question is this*. This automaton is intersected with a rule automaton, and each path through the resulting automaton represents a possible analysis for the whole sequence.

A parallel and unordered rule set in CG corresponds to FSIG. [**?**] explicitly says that an FSIG grammar must be consistent:

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

In FSIG, the path that consists of choosing the `det` arc followed by `verb` arc is ruled out, but other interpretations are still valid: `det+noun`, `pron+noun` and `pron+verb`. In contrast, applying a

rule in CG removes an analysis permanently; if the rule removes the verb reading in *question*, then we cannot retrieve the `pron+verb` possibility. Therefore, it is important to consider the order of the rules. It is possible to operate with *cautious context*, where the rule only takes action if the context is unambiguous. For this case, the "remove `verb` after `det`" rule would not fire before the word *what* is fully disambiguated as a determiner.

2.2.4 Expressivity

As we have learnt, CG is not a generative grammar—it needs a list of alternatives to start removing some of them. But can we emulate some kind of behaviour that would put it into the Chomsky hierarchy?

We can assume that an input sentence is a finite sequence, where every cohort is maximally ambiguous, ie. it contains all possible readings in Σ.

2.3 Summary

In this chapter, I've presented an introduction to Constraint Grammar, highlighting some key properties and contrasting it to other grammar formalisms. CG is relatively young, popularised 20 years ago, and throughout its history, it has been in favour of linguists more than computer scientists, which might explain why its formal background has not been studied extensively—notable exceptions being [**?**] and [**?**].

Is CG a formalism? In order to be a grammar formalism, a candidate framework should discriminate between grammatical and ungrammatical sequences, and provide a structure to the grammatical sequences. Let us start from the latter criterion. Clearly a successful run of a high-quality rule set leaves us with more structure than we started with. In the beginning, we know that *bear* could be a verb (infinitive or present tense, any number or person except 3rd singular) or a noun (subject, object, predicative, ...), and in the end, we will know the right analysis for it, given that the rule set is good. The requirement of a good rule set doesn't affect CG's status as a formalism or not; one could as well write a terrible context-free grammar that doesn't correspond to natural language at all. If POS tags are not enough to consider as a structure, CG can also be used to manipulate syntactic analyses and even to add dependency relations, using exactly the same mechanisms ("set parent to +1 if the word at 0 is *bear* and -1 is a determiner").

CG seems to even have some interest as a tool for psycholinguistics research: there is a Russian being developed within the research group CLEAR (Cognitive Linguistics: Em-

pirical Approaches to Russian); accessing all levels from morphology to semantics at the same time is particularly attractive to cognitive linguists (personal communication). As for its descriptive power, we can again compare CG to CFG. If badly constructed, both CG and CFG can analyse almost everything, making it useless for language description. If we accept that a possibility to construct a bad grammar doesn't revoke CFG's status as a formalism, it shouldn't do so for CG either. One can always compare the relative ease of constructing a good grammar between formalisms, but that is another topic already.

As a bottom line, CG provides a lightweight framework where already a moderate amount of rules can be useful (e.g. [**?**] with 115 rules), although high quality requires in general thousands of rules. Like any formalism, CG itself leaves the users with much freedom; a single grammar can be anything from completely ungrammatical rules to near-perfect description of language.

2.4 Boolean satisfiability (SAT)

Example

What it is and what it can do

What else it has been used for

WHY? What does it help you to formulate something as a SAT-problem?

Chapter 3

CG as a SAT problem

??

3.1 Introduction

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

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

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

3.1.1 Previous work: Encoding in logic

[**?**] represents a CG-like, shallow and reductionist system in logic. [**?**] builds on that in a study which reconstructs four formalisms in logic. CG is contrasted with Finite-State

Intersection Grammar (FSIG) and Brill tagging; all three work on a set of constraint rules which modify the initially ambiguous input, but with some crucial differences.

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

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

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

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

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

To show a concrete example, the following shows the two rules in the specified order: 1. `REMOVE verb IF (-1 det)`; and 2. `REMOVE noun IF (-1 noun)`.

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

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

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

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

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

3.2 CG as a SAT problem

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

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

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

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

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

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

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

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

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

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

3.2.1 Rules disambiguate more

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

This behaviour is explained by simple properties of logical formulae. When the rules are applied to the text, they are translated into implications: `REMOVE prn IF (1 n)` becomes

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

3.2.2 Cautious context is irrelevant

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

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

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

la casa grande 'the big house'

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

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

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

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

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

3.2.3 Rules can be unordered

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

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

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

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

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

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

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

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

3.3 Evaluation

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

3.3.1 Performance against VISL CG-3

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

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

3.3.2 Execution time

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

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

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

Table 3.1: Execution times for 384,155 words.

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

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

3.4 Applications and future work

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

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

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

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

[6]We thank Eckhard Bick for the idea.

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

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

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

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

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

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

3.5 Conclusions

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

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

Chapter 4

Grammar analysis using SAT

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

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

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

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

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

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

Figure 4.1: Rules to select infinitive in Portuguese.

of such conflicts and describe how to detect them.

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

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

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

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

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

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

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

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

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

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

4.1 Implementation

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

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

Before we can do any analysis any of the rules, we need to find out what the set of all possible readings of a word is. We can do this by extracting this information from a lexicon,

but there are other ways too. In our experiments, the number of readings has ranged from about 300 to about 6000.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$v_1 \wedge (v_2 \vee v_3 \vee v_4 \vee v_5)$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Our approach is similar to [**?**], who use ambiguity classes instead of distinct word forms, in order to reduce the number of parameters in a Hidden Markov Model. They take advantage of the fact that they don't have to model "bear" and "wish" as separate entries, but they

can just reduce it to "word that can be ambiguous between noun and verb", and use it as a parameter in their HMM. We can do a similar thing by saving a list of words with each ambiguity class. For example, we map the ambiguity class "feminine plural noun or a separable verb infinitive" to the list of word forms {"uitgaven", "toespraken"}, and then, if we generate such reading for our symbolic word, we can give one of these words as an example word.

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

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

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