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 XeLaTeX
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

Baseline

```
abstract ThankYou = {
  flags startcat = Greeting ;

  cat
    Greeting ; Recipient ;

  fun
    Thanks : Recipient -> Greeting ;
    Koen, Aarne, Colleagues, Friends, Family : Recipient ;
}
```

Random anecdotes

In the beginning of REMU, someone suggested a tagline “SMT meets SMT”. While I’m not quite doing SMT (Satisfiability Modulo Theories) nor SMT (Statistical Machine Translation), the spirit is there.

On the topic

Thanks Eckhard Bick for suggesting CG analysis as a research problem. It has led to many fun discoveries. Other people in the CG community have been very nice and helpful!

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

Constraint Grammar (CG) is a tool for disambiguating text which has various possible analyses for each word. 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":

```
"<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 use syntactic tags and (mostly rudimentary) phrase structure in the analyses. In that case, after disambiguating *bear* by part of speech (noun), it remains to disambiguate whether it is a subject, object, adverbial or any of the possible syntactic roles. An example disambiguation could look like the following:

```
"<the>"
        "the" det def  \textbf{<BEGIN-NP @Det}
"<bear>"
        "bear" noun sg  \textbf{@Subj END-NP>}
"<sleeps>"
        "sleep" verb pres p3 sg  \textbf{@Pred}
```

With the introduction of CG-3, it is also possible to add dependency relations: number the words in the sentence and for each set a parent, such as UNCOMMENT THIS LATER-However, these features are recent and not used in many of the grammars written in the community. In this introduction, I will illustrate my examples with the most basic operations, that is, disambiguating morphological tags.

Properties of Constraint Grammar

All theories and implementations of CG agree that is a *reductionist* system with *shallow* syntax. There are different takes on how *deterministic* the rules are: the state-of-the-art CG parser VISL CG-3 executes the rules strictly based on the order they appear, 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 the following, I will address the three features and relate CG to other formalisms, in terms of the feature in question.

Reductionist vs. licencing

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.

Shallow vs. deep structure

CG is shallow for various reasons.

Locality of context 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. 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.

Little enforcement of grammaticality CG doesn't per se 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. Bottom line: it's not about defining grammaticality, it's providing analyses in any case.

Some relevant heading for this part 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.

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.

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.

Ordering of the rules

The previous properties of Constraint Grammar formalism and rules were specified in [**?**]. However, Karlsson omits some practical details regarding implementation, one being the order in which the rules are executed. 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 and test their performance, using a previously written grammar.

The execution of the rules depends on the following parameters: strict vs. heuristic ordering, and sequential vs. parallel execution. These parameters can 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 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.

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 determine".

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.

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 three transitions from the state *bear* to the state *sleeps* in the automaton representing *the bear sleeps*. 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.

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.

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

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

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

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

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

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.

Conclusion

In the paper 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").

The first criterion can be harder to justify. [**?**] 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. 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: Empirical 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.

Chapter 1

CG as a SAT problem

This used to be a workshop paper. 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.

1.1 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.

1.1.1 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.

1.1.2 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[1]. As the status of *casa* is resolved, the SAT solver can now discard the model where

[1] 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.

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.

1.1.3 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.

1.2 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.

1.2.1 Performance against VISL CG-3

We took a manually tagged corpus[2] containing approximately 22,000 words of Spanish news text, and a small constraint grammar[3], 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[4] of only 19 rules, both SAT-CG and VISL CG-

[2] `https://svn.code.sf.net/p/apertium/svn/branches/apertium-swpost/apertium-en-es/es-tagger-data/es.tagged`

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

[4] `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 1.1: Execution times for 384,155 words.

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.

1.2.2 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 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 2.1 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.

1.3 Applications and future work

Instead of trying to compete with the state of the art, we plan to use SAT-CG for grammar analysis[5]. 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

[5]We thank Eckhard Bick for the idea.

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 2.1.3:

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

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.

1.4 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 2

CG analysis using SAT

This is just random ramblings from my github.

2.0.1 Task

We create an initial symbolic sentence `w` that would make the "last" rule fire. Then we want to make `w` so that none of the earlier rules has effect on it, because

1. conditions are out of scope (trivial, no clauses)
2. conditions in scope, but one or more doesn't hold
3. tag has been removed in target
4. all readings of target have the desired tag (cannot remove)

2.0.2 Examples

Last rule:

```
REMOVE inf  IF (-1 (prn pers))
```

Earlier rules:

```
REMOVE:r_pr_v pr  IF (1 vblex vbmod vaux vbhaver vbser ) (NOT 0 "te" "om te" )
REMOVE:r_adv_n adv  IF (1 n np )
SELECT:s_pr_v pr  + "te" "om te"  IF (1 vblex vbmod vaux vbhaver vbser  + inf )
```

Examined rule creates a model with many solutions, among which the following:

```
"<w1>"
      prn pers
"<w2>"
      <<<
      vblex inf
```

Then we apply the earlier rules to our symbolic string. First two are fine. They don't affect the analyses that are crucial to our examined rule. However, the third SELECT rule has `vblex inf` in its context. When we add this clause to SAT solver, it will try to find another model which satisfies the examined rule, but will make the SELECT rule to have no effect. We find one, which satisfies scenario 2: condition is not met.

```
"<w1>"
      prn pers
"<w2>"
      <<<
      inf
```

An example of scenario 4:
Last rule:

```
SELECT:s_pr_v pr  + "te" "om te"  IF (1 vblex vbmod vaux vbhaver vbser  + inf )
```

Earlier rules:

```
 REMOVE:r_pr_v pr  IF (1 vblex vbmod vaux vbhaver vbser ) (NOT 0 "te" "om te" )
 REMOVE:r_adv_n adv  IF (1 n np )
```

One solution for the initial constraints:

```
"<w1>"
      >>>
      pr "te"
"<w2>"
      vblex inf
```

First REMOVE rule conflicts. Solve by applying scenario 4: only targets are left. Scenario 3 wouldn't work because `(pr "te")` is a requirement. Scenario 2 wouldn't work because ???

Solution after all clauses:

```
"<w1>"
        pr "te"
        pr "om te"
"<w2>"
        vblex inf
```

2.0.3 Interaction of previous rules

Look at the following three rules.

```
r1 = REMOVE V   IF (-1C Det) ;
r2 = REMOVE Det IF ( 1  V)   ;
r3 = REMOVE V   IF (-1  Det) ;
```

Is there an input which can go through the rules and trigger at the last?

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

- r1: Doesn't trigger because condition `(-1C Det)` isn't met.
- r2: Cannot trigger, because r3 requires `w2` to have a `v` analysis.
- Tries to go for "ok what if the target (`w1`) *only* has a `det` analysis!"
- In that case, the input would already trigger `r1`
- Cannot do neither => conflict