Analysing Constraint Grammars with a SAT-solver

Inari Listenmaa, Koen Claessen
University of Gothenburg, Chalmers University of Technology
Gothenburg, Sweden
inari@chalmers.se, koen@chalmers.se

Abstract

We describe a method for analysing Constraint Grammars (CG) that can detect internal conflicts and redundancies in a given grammar, without the need for a corpus. The aim is for grammar writers to be able to automatically diagnose, and then manually improve their grammars. Our method works by translating the given grammar into logical constraints that are analysed by a SAT-solver. We have evaluated our analysis on a number of non-trivial grammars and found inconsistencies.

Keywords: Constraint Grammar, SAT, Grammar Analysis

1. Introduction

Constraint Grammar (CG, Karlsson et al. (1995)) is a formalism used to disambiguate morphologically analysed text. A grammar consists of rules that target specific readings for selection or removal, based on contextual tests. For example, the following rule

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

removes all verb readings from a word which is preceded by a determiner. Given the following text,

```
"<the>"
        "the" det def
"<wish>"
        "wish" noun sg
        "wish" verb pl
        "wish" verb inf
```

the rule will match to the word *wish*, and remove the readings `"wish" verb pl` and `"wish" verb inf`. Note that if the target word has only one remaining reading, then the rule will not apply, even if the condition is met.

CGs are valuable resources for rule-based NLP, especially for lesser resourced languages. They are robust and can be written without large corpora—only a morphological analyser is needed. The formalism is lightweight and language-independent, and resources can be shared between related languages (Bick, 2006; Antonsen et al., 2010). Mature CGs contain some thousands of rules, but even small CGs are shown to be effective (Antonsen and Trosterud, 2011).

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 tool 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.

```
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 1: Rules to select infinitive in Portuguese.

Given the rules in figures 1 and 2, 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`)
- Can two or more rules be merged? (e.g. `SELECT Inf IF (-1C Prep OR Vai OR Vbmod)`)
- Can a messy rule be rewritten in a neater way without changing the meaning?
- What is the best order for the rules?
- Generate a sentence that triggers a given list of rules R but not R'

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

- If two equivalent rules r and r' occur in the grammar, the second occurrence will be disabled by the first
- Rule r selects something in a context, and r' removes it
- A list of tules R removes something from the context of a rule r, so r can never apply
- A rule r has an internal conflict, such as non-existent tag combination, or contradictory requirements for a context word

```
    SELECT V + Prs/Imprt + Act + Neg

 IF (*-1C Negv LINK NOT *1 Vfin/PrsPrc/Inf)
    (NOT 0 N) (NOT 0 Pron)
    (NOT *1 Neg) (NOT *-1 Neg)
    (NOT 0 Pass) (NOT *-1 Niin)
    (*-1C Negv LINK NOT *1 CLB?)
    (*-1C Negv LINK NOT 0 Imprt) ;

 IF (NOT *-1 Niin OR Neg)
    (*-1C Negv
      LINK NOT 0 Imprt
      LINK NOT *1 Vfin/PrsPrc/Inf OR CLB?)
    (NOT 0 N OR Pron OR Pass)
    (NOT *1 Neg) ;
```

Figure 2: Two versions of a condition in Finnish.

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

While rule-internal conflicts can be detected by simpler means, taking care of rule interaction requires a *semantic* rather than a *syntactic* analysis. We must keep track of all the possible sentences after applying each rule. At each step, we have two options: either the rule fires, or it does not fire. In case the rule does not fire, we have two reasons why not: one or more of its conditions does not hold, or its target is the only remaining analysis. The result is a complex table of interdependent decisions—some determined by being the target of a rule, others by being a condition.

We express these constraints as a *Boolean satisfiability problem* (SAT). A SAT-problem consists of two components: a set of Boolean variables, and a set of clauses on those variables. For instance, let the set of variables be $\{a, b\}$ and the formulas $\{a \vee b, \neg a\}$. A program called a *SAT-solver* will try to find a solution, where all the variables are replaced by a Boolean value. For this particular problem, the unique solution is $\{a = False, b = True\}$, but it is also possible for a SAT-problem to have no solution or multiple solutions. In the case of CG analysis, the solution we build (called the *model*) represents a *sentence* that starts as potentially anything, and is being shaped by all the rules along the way. This is how we can generate possible inputs, as well as check if the grammar is internally consistent.

The paper is structured as follows. Section 2. relates our work to previous work. Section 3. discusses the implementation, and Section 4. presents preliminary results. Section 5. discusses future work and concludes the paper.

2. Related work

We combine elements from the following aspects of CG research:

- Corpus-based methods in manual grammar development (Voutilainen, 2004)
- Optimising hand-written CGs (Bick, 2013)
- Encoding CG in logic (Lager, 1998; Lager and Nivre, 2001; Listenmaa and Claessen, 2015)

In addition, there is a large body of research on automatically inducing rules, e.g. Samuelsson et al. (1996), Eineborg and Lindberg (1998), Lager (2001) and Sfrent (2014). However, since our work is aimed to aid the process of hand-crafting rules, we omit those works from our discussion.

Corpus-based methods in manual grammar development Hand-annotated corpora are commonly used in the development of CGs, because they give immediate feedback whether a new rule increases or decreases accuracy (Voutilainen, 2004). This helps the grammar writer to arrange the rules in appropriate sections, with safest and most effective rules coming first. However, this method will not notice a missed opportunity or a grammar-internal conflict, nor suggest ways to improve.

Automatic optimisation of hand-written grammars Bick (2013) modifies the grammar automatically, by trying out different rule orders and altering the contexts of the rules. Bick reports error reduction of 7–15% compared to the original grammars. As a downside, the grammar writer will likely not know why exactly does the tuned grammar perform better.

CG encoded in logic Lager (1998) and Lager and Nivre (2001) reconstruct the CG formalism in first-order predicate logic. Grammar analysis is a natural use case, due to some key features of the logical reconstruction. The traditional CG compiler cannot capture any dependencies between rules. In contrast, a logic-based CG compiler does that by default. The rules are modelled as implications and composed in the order of the rule sequence, such that the consequent from the i^{th} rule becomes the antecedent of the $i+1^{th}$ rule. Given this design, we added on top a way to ask for solutions with certain properties.

3. Implementation

In this section, we describe the implementation of the tool. The SAT-encoding we use is similar to the one introduced in Listenmaa and Claessen (2015), 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.

3.1. Preliminaries

Our analysis operates on a rule r, which is preceded by a list of rules 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 9000.

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 $\star$ (context word can be anywhere), we may need to make an approximation of $w(r)$, which may result in false positives or negatives later on in the analysis.

3.2. 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:

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

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.

3.3. Applying a rule

Next, we need to be able to apply any 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 must also add the following constraint for w'_4:

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

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:

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

3.4. Putting it all together

Once we know how to apply any rule r' to a symbolic sentence, resulting in a new 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:

$$w'_4 \lor w'_5$$

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

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

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

3.5. Creating realistic readings

Earlier we have shown an example with 5 readings ("det def", "noun sg", ...). In a realistic case, we operate between hundreds and thousands of possible readings. In order to find the set of readings, we expand a morphological lexicon[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) ;
```

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

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

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

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

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

The readings in a grammar can be underspecified: for example, the rule `REMOVE (verb sg) IF (-1 det)` gives us “verb sg” and “det”. In contrast, the lexicon only gives us fully specified readings, such as “verb pres p2 sg”. We implemented a version where we took the tag combinations specified in the grammar directly as our readings, and we could insert them into the symbolic sentences as well. The shortcut works most of the time, but if we only take the readings from the grammar and ignore the lexicon, it is possible to miss some cases: e.g. the rule `SELECT Pron + Rel IF (0 Nom)` may require “pron rel nom” in one reading, but this method only gives “pron rel” and “nom” separately.

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

3.6. Creating realistic ambiguities

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

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

We solve the problem by creating *ambiguity classes*: groups of readings that can be ambiguous with each other.

We represent the expanded morphological lexicon as a matrix, as seen above: word forms on the rows and analyses on the columns. Each distinct row forms an ambiguity class. For example, one class may contain words that are ambiguous between plural feminine nouns and separable verb infinitives; another contains masculine plural adjectives and masculine plural past participles. Then we form SAT-clauses that allow or prohibit certain combinations. These clauses will interact with the constraints created from the rules, and the end result will be closer to real-life sentences.

Our approach is similar to Cutting et al. (1992), who use ambiguity classes instead of distinct word forms, in order to reduce the number of parameters in a Hidden Markov Model. They take advantage of the fact that they don’t have to model “bear” and “wish” as separate entries, but they can just reduce it to “word that can be ambiguous between noun and verb”, and use that as a parameter in their HMM.

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

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

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

4. Evaluation

We tested three grammars to find conflicting rules: Dutch[2], with 59 rules; Spanish[3], with 279 rules; and Finnish[4], with 1185 rules. We left out ADD, MAP and other rule types introduced in CG-3, and only tested REMOVE and SELECT rules. The results for Dutch and Spanish are shown in Table 1, and the results for Finnish in Table 2.

A natural follow-up evaluation would be to compare the performance of the grammar in the original state, and after removing the conflicts found by our tool. Unfortunately, we did not have time to perform such evaluation, and in addition, we only have gold standard corpus (20 000 words) for Spanish.

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

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

[4] https://github.com/flammie/apertium-fin/raw/master/apertium-fin.fin.rlx

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

Table 1: Results for Dutch and Spanish grammars.

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

4.1. Dutch

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

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

These two rules share a target: both remove an adverb. The problem is that the first rule has a broader condition than the second, hence the second will not have any chance to act. If the rules were in the opposite order, then there would be no problem.
We also tested rules individually, in a way that a grammar writer might use our tool when writing new rules. The following rule was one of them:

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

As per VISL CG-3, the condition `(0 "zijn")` does not require *zijn* to be in the same reading with the target *DetPos*. It just means that at index 0, there is a reading with any possessive determiner, and a reading with any *zijn*. However, the intended action is to select a "det pos *zijn*" all in one reading; this is expressed as `SELECT DetPos + "zijn"`. In contrast, the 0-condition in example D$_1$ is used correctly: the adjective and the adverb are supposed to be in different readings.

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

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

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

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

4.2. Spanish

The Spanish grammar had proportionately the highest number of errors. The grammar we ran is like the one found in the Apertium repository (linked on the previous page), apart from two changes: we fixed some typos (capital O for 0) in order to make it compile, and commented out two rules that used regular expressions, because we did not implement the support for them yet. For a full list of found conflicts, see the annotated log of running our program in https://github.com/inariksit/cgsat/blob/master/data/spa/conflicts.log.
We include two versions of the Spanish grammar in Table 1: in column SPA, we added the lemmas and word forms as described in Section 3.5., and in column SPA$^{\text{sep. lem.}}$, we just added each word form and lemma as individual readings, allowed to combine with any other reading. This latter version ran much faster, but failed to detect an internal conflict for one rule, and reported a false positive for another.
When we added ambiguity class constraints, we found three more internal conflicts. Interestingly, the version with ambiguity classes fails to detect an interaction conflict, which the simpler version reports, because one of the rules is first detected as an internal conflict. We think that neither of these versions is a false positive or negative; it is just a matter of priority. Sometimes we prefer to know that the rule cannot apply, given the current lexicon. However, we may know that the lexicon is about to be updated, and would rather learn about all potential interaction conflicts.
As an example of internal conflict, there are two rules that use `SET Cog = (np cog)`: the problem is that the tag "cog" does not exist in the lexicon. As another example, four rules require a context word tagged as NP with explicit number, but the lexicon does not indicate any number with NPs. It is likely that this grammar has been written for an earlier version, where such tags have been in place. One of the conflicts that was only caught by the ambiguity class constraints had the condition `IF (1 Comma) (..) (1 CnjCoo)`. The additional constraints correctly prevent commas from being ambiguous with anything else.

As for the 25 interaction conflicts, there were only 9 distinct rules that rendered 25 other rules ineffective. In fact, we can reduce these 9 rules further into 3 different groups: 4 + 4 + 1, where the groups of 4 rules are variants of otherwise identical rule, each with different gender and number. An example of such conflict is below (gender and number omitted for readability):

```
S1. # NOM ADJ ADJ
    SELECT A OR PP IF (-2 N)
     (-1 Adj_PP) (0 Adj_PP) (NOT 0 Det);

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

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

```
S2. SET NP_Member = N + A + Det + PreAdv
    + Adv + Pron ;
```

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

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

```
S3. REMOVE Sentar IF (0 Sentar) (..) ;

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

The comments make it clear that the first rule is meant to disambiguate between *sentar* and *sentir*, but the rule does not mention anything about *sentir*. Even with the ambiguity class constraints, the SAT-solver only created an ambiguity where *sentar* in 1st person plural is ambiguous with an anonymous 1st person plural reading. This does not reflect the reality, where the target is only ambiguous with certain verbs, and in certain conjugated forms.

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

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

4.3. Finnish

The results for the Finnish grammar are shown separately, in Table 2. We encountered a number of difficulties and used a few shortcuts, which we did not need for the other grammars—most importantly, not using the ambiguity class constraints. Due to these complications, the results are not directly comparable, but we include Finnish in any case, to give an idea how our method scales up: both to more rules, and more complex rules.

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

The second challenge comes from the larger size of the grammar. Whereas the Spanish and Dutch had only tens of word forms or lemmas, the Finnish grammar specifies around 900 of them. Due to both of these factors, the procedure described in Section 3.5. would have exploded the number of readings, so we simply took the lemmas and word forms, and added them as single readings. In cases where they were combined with another tag in the grammar, we took that reading directly: for instance, we included both *aika* and "*aika* n" from the rule `SELECT "aika" + N`, but nothing from the rule `SELECT Pron + Sg`. This method gave us 1588 additional readings.

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

Results The results are shown in Table 2. In the first column, we included only possessive suffixes. In the second column, we included question clitics as well. Both of these readings include the 1588 lemmas and word forms from the grammar. In the third column, we included all the tag combinations specified in the grammar, and in the fourth, we took only those, ignoring the morphological lexicon.

The first two variants reported a high number of internal conflicts. These are almost all due to nonexisting tags. The grammar was written in 1995, and updated by Pirinen (2015); such a high number of internal conflicts indicates that possibly something has gone wrong in the conversion,

	1 clitic + lemmas from grammar	2 clitics + lemmas from grammar	1 clitic + all readings from grammar	all readings from grammar
# readings	5851 (4263 + 1588)	9494 (7906 + 1588)	6657 (4263 + 2394)	2394
# conflicts (internal + interaction)	214 (211 + 3)	214 (211 + 3)	22 (19 + 3)	22 (19 + 3)
◷ all rules (approx.)	˜4h 30min	˜9h 30min	˜7h 45min	˜2h 30min

Table 2: Results for Finnish (1185 rules).

or in our expansion of the morphological lexicon. As for accuracy, adding the question clitics did not change anything: they were already included in some of the 1588 sets with word forms or lemmas, and that was enough for the SAT-solver to find models with question clitics. We left the result in the table just to demonstrate the change in the running time.

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

Some examples of true positives are shown in the following.

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

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

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

The following two examples are interaction conflicts:

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

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

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

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

4.4. Performance

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

However, many of the use cases do not require running the whole grammar. Testing the interaction between 5–10 rules takes just seconds in all languages, if the ambiguity class constraints are not included. A downside in the ambiguity classes is that generating them takes a long time, and while the overhead may be acceptable when checking the full grammar, it is hardly so when analysing just a handful of rules. We are working on an option to store and reuse the ambiguity class constraints.

5. Conclusions and Future Work

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

We did not have a volunteer to test the tool in the process of grammar writing, so we cannot conclude whether the constructed examples are useful for getting new insights on the rules. In any case, there are still a number of features to improve and add.

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

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

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

Support for grammar writers As mentioned earlier, we could support additional checks for common issues, such as conditions that concern the target word. Another possible feature is to suggest reformattings for a rule. Recall Figure 2 from the introduction; in that case, the original rule was written by the original author, and another grammarian thought that the latter form is nicer to read. Doing the reverse operation could also be possible. If a rule with long disjunctions conflicts, it may be useful to split it into smaller conditions, and eliminate one at a time, in order to find the reason(s) for the conflict. Our next step is to evaluate our tools together with actual grammar writers, in comparison with a corpus-based method or machine learning.

Other grammar formalisms Finally, we would like to investigate logic-based methods for analysing other grammar formalisms, for example Grammatical Framework (Ranta, 2010).

Acknowledgments

We thank Eckhard Bick for the idea to apply SAT to CG analysis, and Francis Tyers for insightful discussions, as well as our anonymous reviewers for comments and suggestions.

6. Bibliographical References

Antonsen, L. and Trosterud, T. (2011). Next to nothing – a cheap South Saami disambiguator. In *Proceedings of the Constraint Grammar workshop at NODALIDA*.

Antonsen, L., Wiechetek, L., and Trosterud, T. (2010). Reusing grammatical resources for new languages. In *Proceedings of the International conference on Language Resources and Evaluation LREC2010*.

Bick, E. (2006). A Constraint Grammar Parser for Spanish. In *Proceedings of TIL 2006 - 4th Workshop on Information and Human Language Technology*.

Bick, E. (2013). ML-Tuned Constraint Grammars. In *Proceedings of the 27th Pacific Asia Conference on Language, Information and Computation*.

Cutting, D. R., Kupiec, J., Pedersen, J. O., and Sibun, P. (1992). A practical part-of-speech tagger. In *ANLP*, pages 133–140.

Eineborg, M. and Lindberg, N. (1998). Induction of constraint grammar-rules using progol. In David Page, editor, *Inductive Logic Programming*, volume 1446 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (1995). *Constraint Grammar: a language-independent system for parsing unrestricted text*, volume 4. Walter de Gruyter.

Lager, T. and Nivre, J. (2001). Part of speech tagging from a logical point of view. In *Proceedings of LACL, Logical Aspects of Computational Linguistics*.

Lager, T. (1998). Logic for part of speech tagging and shallow parsing. In *Proceedings of the 11th Nordic Conference on Computational Linguistics*.

Lager, T. (2001). Transformation-based learning of rules for constraint grammar tagging. In *Proceedings of the 13th Nordic Conference on Computational Linguistics*.

Listenmaa, I. and Claessen, K. (2015). Constraint Grammar as a SAT problem. In *Proceedings of the Constraint Grammar workshop at NODALIDA*.

Pirinen, T. (2015). Using weighted finite state morphology with VISL CG-3—Some experiments with free open source Finnish resources. In *Proceedings of the Constraint Grammar workshop at NODALIDA*.

Ranta, A. (2010). *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications.

Samuelsson, C., Tapanainen, P., and Voutilainen, A. (1996). Inducing constraint grammars. In Laurent Miclet et al., editors, *Grammatical Interference: Learning Syntax from Sentences*, volume 1147 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

Sfrent, A. (2014). Machine Learning of Rules for Part of Speech Tagging. Master's thesis, Imperial College London, United Kingdom.

Voutilainen, A. (2004). Hand crafted rules. In H. van Halteren, editor, *Syntactic Wordclass Tagging*, pages 217–246. Kluwer Academic.