Automatic Synthesis of Constraint Grammar Rules using a Greedy Approach and a SAT-solver
-- a work in progress report

Koen Claessen
Chalmers University of Technology,
Sweden
`koen@chalmers.se`

Abstract

We present a method for automatic synthesis of Constraint Grammar rules from a corpus. The method is designed to aid grammar writers interactively in coming up with new rules or improving existing ones, but also to synthesize a whole set of rules autonomously from scratch. A SAT-solver is used to compute the "best" rule at each stage, according to some measure. Initial experimental results on two corpora look promising: suggesting one rule to improve an existing set of rules typically takes seconds; synthesizing a whole set of rules takes minutes to hours.

1 Introduction

Imagine a CG grammar writer who finds herself in the following situation: There exists a gold standard development corpus, and already a set of rules. Now, the CG writer would like to add a new rule, or improve an existing one. Where to start?

This work describes a tool that can automatically find the next "best" rule to add to an existing set of rules. It deploys a SAT-solver that computes the (1) simplest, most general rule that (2) does not remove any correct reading from the corpus, and (3) maximizes the number of removed incorrect readings from the corpus.

For example, given a spanish corpus and an empty set of rules, the tool may compute the following (rather reasonable) rule:

```
SELECT (det) IF (1 (n));
```

The reason is because this is the rule that removes most ambiguities without removing any correct readings from the corpus. A human grammar writer may also have included this rule high up in the grammar, which is why we call this rule reasonable.

Asking the tool to generate one more rule and then yet another, the following two rules are generated:

```
REMOVE (vblex) IF (-1 (det));
REMOVE (sg) IF (0 (pr));
```

The first one looks again reasonable. The second rule does not. Obviously, there are sentences in which the second rule would remove the correct reading, but none of those sentences appears in our corpus.

2 Three use cases

We envision the use of our tool for grammar development in three different use cases.

A. Interactive mode - in this mode, the grammar writer can ask questions such as: “What is the most general rule I can add to my existing set of rules?” and “Is there a rule that looks like <this> that does not remove any correct readings from the corpus?” and get answers. The grammar writer is in complete control over the grammar and can add any versions of these generated rules anywhere in the grammar.

B. Meta-mode - The tool suggests rules to add to an existing grammar, but the grammar writer inspects the rules, and when the rules do not look good, *adds new sentences to the corpus*. For example, in the case of the third rule from the previous section, the corpus could be augmented with a sentence where that rule would remove the correct reading. Running the tool again would then suggest a different rule.

C. Autonomous mode - The tool starts from an empty set of rules, and greedily computes a set of rules from scratch. Typically, the resulting set of rules is perfect (or almost perfect) on the given corpus, meaning a precision and recall of 100% (or close to). The hope is then that these rules can be generalized to other corpora. Our very preliminary experiments suggest this may be the case.

3 Implementation

Our method only generates rules that satisfy the following:

- Either a SELECT or REMOVE rule
- The head is a set of tags
- The condition is a conjunction of zero or more conditions of the form (i tag_1 .. tag_n) or (iC tag_1 .. tag_n), for i chosen from a window (typically {-2,-1,0,1,2} or {-1,0,1}) and the tag_j can be any tag.

Some of these constraints can be relaxed a bit (for example adding more kinds of rules, or adding BARRIER conditions, but we have not done this yet).

We use a SAT-solver [4] to compute the “best” rule in the following way. First, we run any existing rules on the given corpus. Then, we create a SAT-problem with the following variables:

- A SAT-variable sel, indicating which kind of rule we have,
- For each possible tag t, a SAT-variable h_t, representing the head of the rule,
- For each position i in the window and tag t, a SAT-variable $c_{i,t}$, representing whether or not that tag appears in the condition with that position,
- For each position i in the window, a variable C_i, representing whether the condition for position i is careful or not,
- For each ambiguous cohort w from the corpus, a SAT-variable a_w, representing if the rule removes any reading from that cohort.

Then, we add the following constraints:

- There must be at least one head,
- The rule should never remove a correct reading,
- There must be at least one cohort for which we remove a reading.

Finally, we look for solutions to these constraints, applying an optimization strategy that optimizes, in this order:

- Maximize the number of cohorts where the rule removes a reading,
- Minimize the number of tags appearing as conditions,
- Minimize the number of C-conditions,
- Minimize the number of tags in the head,
- Prefer SELECT over REMOVE.

The solution to the SAT-problem is a model that assigns true or false to all of the variables. The actual rule can easily be constructed from that model.

4 Preliminary Results

Our preliminary results contain experiments on 3 corpora: One Spanish corpus with ~21.000 words, taken from the Apertium repositories. It contains news texts that have been hand-tagged by students. One Basque corpus with ~61.000

words, called EPEC. And one English corpus with ~29.000 words, taken from the minutes of the EU parliament meetings.

	#tags	avg. #readings
Spanish	116	1.39
Basque	203	2.84
English	78	1.48

The number of tags and average number of readings per cohort are given in the table above.

We ran the interactive mode on both corpora, and most questions could be answered within seconds to sometimes minutes.

We also ran the autonomous mode on the first 3000 words of the Spanish corpus, which took ~10 minutes (the resulting 88 rules are shown in the appendix). Although this is only 1/7th of the total corpus, the evaluation of this grammar on the rest (6/7th) of the corpus, yielded 96% correct readings after full disambiguation[1]! This fraction of correctness is on par or better than existing hand-written grammars we had for this corpus. Increasing the size of the training data to 16.800 words (80% of the corpus) yielded a grammar with 97% correct readings for the other 20%.

We also ran the autonomous mode on the first 2500 words of the Basque corpus (4%), which took a bit over 1 hour. The evaluation of the resulting grammar on the rest of the corpus yielded 79% correct readings after full disambiguation. A result that may be disappointing, but it is better than the CG we had at hand, written by humans. Basque turns out to be more computationally expensive than Spanish, mostly because it has more tags and more ambiguity in its readings. Also, word order in Basque is less strict than Spanish.

We were able to increase the size of the training corpus for Basque to 30.000 words (almost 50% of the corpus) by using non-exact optimization methods for picking the best rule. This yielded a grammar with 84% correct readings on the other half of the corpus.

For English, we reached close to 97% with 3000 words. Adding more words did not significantly improve the quality of the generated grammar.

In all our experiments, we used a window of {-1,0,1}, and no limit on the size of the conditions. For Spanish and English, we also tried a window {-2,-1,0,1,2}, which took ~3 times longer time, but did not produce a significantly better grammar. Only a handful of rules actually made use of distance -2 or 2. For Basque, it took too long time to run with a larger window.

5 Discussion and Conclusion

Our experiments are promising but far too few to draw any significant conclusion. We believe that we have enough evidence to suggest that our work may be a useful tool. One important factor that is limiting our experiments is the lack of good quality corpora to use for our experiments!

We have not studied the usefulness of the "interactive mode" or "meta mode" that would help a CG writer who is looking for suggestions on what to do next. This is left as future work.

This work is not the first to propose automatic generation of disambiguation rules from a corpus; in fact there has been work on this since the 1990s [1,2,3]. Our work differs from the ones that generate CG rules in two significant ways: (1) our greedy approach computes rules that remove as much ambiguity without also removing correct readings, as opposed to earlier work that is based on statistics, and (2) we use a SAT-solver to compute rules which avoids enumeration of rules and allows for a

[1] This precentage is both the precision and the recall after making a random choice for all ambiguities that are still left after running the rules.

constraint-based specification of what kind of rules we are looking for, which is more flexible.

We argue that an approach that avoids rules that remove correct readings is the reasonable choice in a setting where rules are generated greedily one-by-one. Allowing such rules to remove correct readings would be non-compositional; a later rule can never correct such a mistake made by an earlier rule.

6 References

[1] Samuelsson, Tapanainen, Voutilainen. Inducing Constraint Grammars. International Colloquium on Grammatical Inference. 1996.

[2] Lindberg, Eineborg. Learning Constraint-grammar style disambiguation rules using Inductive Logic Programming. COLING. 1998.

[3] Sfrent. Machine Learning of Rules for Part of Speech Tagging. MSc. thesis. Imperial College London. 2014.

[4] Een, Sörensson. An Extensible SAT-solver. SAT conference. 2003.

Appendix - Generated CG for Spanish

```
SELECT (det) IF (1 (n));
REMOVE (vblex) IF (-1 (det));
REMOVE (sg) IF (0 (pr));
REMOVE (n) IF (-1 (n)) (0 (adj));
REMOVE (imp) IF (-1 (sg));
REMOVE (p3) IF (-1 (pr));
SELECT (n) IF (-1 (det)) (-1 (m));
REMOVE (p1 prs) ;
REMOVE (p3) IF (0 (p1));
REMOVE (p2) IF (0 (n));
REMOVE (vblex) IF (1 (vblex));
REMOVE (cnjsub) IF (-1C (n));
SELECT (al) IF (0 (ant));
REMOVE (p1) IF (0 (m));
REMOVE (rel) IF (-1 (vblex));
REMOVE (sg) IF (0 (vbser));
SELECT (pri) IF (-1 (mf));
SELECT (np) IF (-1 (np));
REMOVE (adj) IF (-1 (pr)) (0 (n));
SELECT (det) IF (1 (adj));
SELECT (mf) IF (1 (p3));
REMOVE (mf) IF (0 (n));
SELECT (mf) IF (0 (vblex));
REMOVE (sp) IF (1C (adj));
SELECT (f) IF (1 (pr));
SELECT (vblex) IF (-1C (n));
SELECT (pr) IF (1 (m));
SELECT (np) IF (-1 (pr));
REMOVE (n sg) IF (1 (cm));
REMOVE (p3 sg) IF (0 (m));
SELECT (n sg) IF (-1 (f));
SELECT (np) ;
REMOVE (imp p3) ;
REMOVE (adj pl) IF (-1 (mf));
REMOVE (pl prn) IF (-1C (pr));
SELECT (ind sp) ;
SELECT (ind) IF (0C (f));
SELECT (pri) IF (1C (m));
REMOVE (sp) IF (0 (sg));
SELECT (n) IF (-1 (cnjcoo));
SELECT (adj) IF (1 (sent));
REMOVE (adj m sg) IF (0 (n));
REMOVE (m n) IF (-1C (sg));
SELECT (m) ;
REMOVE (p3) IF (-1C (sent));
SELECT (pri) IF (0C (vblex));
SELECT (pp) IF (-1C (vbser));
REMOVE (pp) IF (1 (pr));
REMOVE (mf n) ;
REMOVE (adj m) ;
REMOVE (det ind sg) ;
REMOVE (mf) IF (-1 (adv));
REMOVE (prn) IF (1 (m));
SELECT (adv) IF (1 (pr));
REMOVE (vblex) IF (-1 (pr));
SELECT (pp) ;
REMOVE (adv) IF (1C (sg));
REMOVE (vblex) IF (1C (sg));
SELECT (pri) ;
SELECT (adv) IF (1C (vblex));
SELECT (mf) IF (0 (adj));
SELECT (mf) IF (-1 (cm));
REMOVE (detnt) ;
REMOVE (al) ;
SELECT (cnjsub) IF (1 (f));
SELECT (pr) IF (1 (det));
SELECT (rel) IF (1 (pii));
SELECT (adv) IF (1 (pl));
SELECT (sp) IF (-1C (mf));
REMOVE (n) IF (-1 (sg));
REMOVE (adj) ;
REMOVE (cnjadv) IF (1 (np));
REMOVE (cnjcoo) ;
REMOVE (sent) ;
REMOVE (pr) ;
SELECT (vblex) ;
REMOVE (n) ;
SELECT (ind sg) ;
REMOVE (det) IF (1C (pri));
REMOVE (prn) IF (1 (sg));
SELECT (cnjsub) IF (1 (dem));
SELECT (an) IF (1 (det));
REMOVE (rel) IF (-1 (sg));
SELECT (an) ;
REMOVE (loc) IF (1 (rpar));
REMOVE (ant) ;
SELECT (prn) IF (-1 (p3));
SELECT (ind) ;
```