Exploring the Expressivity of Constraint Grammar

Abstract

We believe that for any formalism which has its roots in linguistics, it is a natural question to ask "how expressive is it?" Therefore, in this paper, we begin to address the question of the expressivity of CG. Aside from the obvious theoretical interest, we envision also practical benefits. For instance, we hope that the `cgexp` tool, described in later sections of this paper, could eventually be developed to generate human-readable CG code from regular expressions or a context-free grammar.

1 Introduction and previous work

CG (Karlsson et al., 1995) was born as a practical, rather than formal, approach to NLP. Since the beginning, its authors do not envision it as a tool for generating strings, only for analysing and disambiguating them. It is for these reasons, we believe, that the question of the expressivity of CG as a generative formalism went unasked and unanswered for so long.

The standard measure of formal languages is the Chomsky hierarchy (Chomsky, 1956), with its four classes of grammars and languages, in order from most expressive to least expressive: recursively enumerable (Type 0), context-sensitive (Type 1), context-free (Type 2), and regular (Type 3). The notion of expressive power, "which constructs can we express in the language", is coupled with parsing complexity, "how much time and memory do we need to parse sentences in the language"; more expressive power corresponds to greater parsing complexity.

Previous work covers the expressivity of single rules, such as `IF (NOT 1* Verb OR Noun)`: just seeing this contextual test hints that we can express a subset of regular languages that contains at least disjunction, complement and Kleene star. Tapanainen (1999) gives an account of the expressivity of a single rule for 4 different constraint formalisms. In addition, parsing complexity can be easily defined for a given variant and implementation of CG; see for instance Nemeskey et al. (2014).

However, the expressivity of the whole grammar is harder to define. A given grammar in CG does not describe a language, but a *relation* between an input language and an output language. In the following section, we introduce our approach to emulating generation, and in the rest of the paper, we present some preliminary results.

2 Expressivity of a grammar

We view a constraint grammar as a formal language $\mathscr{L}$, generated over an alphabet Σ. We generate the strings in our language by passing "maximally ambiguous strings" of every length to the grammar. With maximally ambiguous, we mean those strings where each cohort contains the entire alphabet, which we write as $\langle\Sigma\rangle_n$. A constraint grammar is said to accept a string w of length n if, when we pass $\langle\Sigma\rangle_n$ as an input to the CG, w is one of the possible interpretations of its output.

For example, consider the language $a*$ over $\Sigma = \{a,b\}$, shown in Figure 1. The grammar that accepts this language is simply `SELECT (a)`.

Input	Output
`"<w>"`	`"<w>"`
`"a" a`	`"a" a`
`"b" b`	
`"<w>"`	`"<w>"`
`"a" a`	`"a" a`
`"b" b`	
`"<w>"`	`"<w>"`
`"a" a`	`"a" a`
`"b" b`	

Figure 1: Language $a*$ for input $\langle\Sigma\rangle_3$.

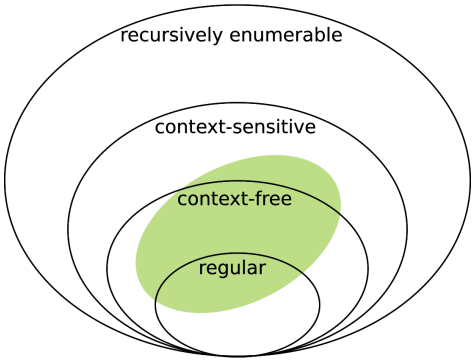

Figure 2: Lower bound on the expressivity of CG.

3 CG is beyond regular

In this section, we demonstrate some non-regular languages that can be expressed in CG.[1] Due to space limitations, we describe only the rough outlines.

3.1 Example grammar: $a^n b^n$

We can express the language $a^n b^n$ as a CG which disambiguates any $\langle\Sigma\rangle_{2n}$ into $\langle a\rangle_n\langle b\rangle_n$. In addition, we allow ourselves a hidden alphabet of helper readings, $\Sigma' = \{\texttt{even}, \texttt{odd}, \texttt{>>>}, \texttt{<<<}, \texttt{opt\_a}, \texttt{opt\_b}\}$. Every cohort starts off with full Σ and Σ'. The process goes as follows:

- Check if the sentence has an even number of cohorts. We know the first cohort is odd, and the rest is handled with rules of the form `REMOVE even IF (NOT -1 odd)`. If the last cohort is odd, then discard the sentence; otherwise continue.
- Disambiguate edges: first one is certainly *a* and last is certainly *b*.
- Grow *a*s and *b*s until they meet in the middle. The disambiguation is done in two passes, first removing the helper reading `opt_a`, and then selecting the main reading *a*.

3.2 Example grammar: $a^n b^n c^n$

We can extend the previous approach to write a grammar that accepts $a^n b^n c^n$. We use three counting symbols, three `opt_X` symbols, and initially we disambiguate the two edges and the middle. The result is the same: any $\langle\Sigma\rangle_{3n}$ gets disambiguated into $\langle a\rangle_n\langle b\rangle_n\langle c\rangle_n$.

[1] More specifically, a subset that consists of the operations `REMOVE, SECTION, IF, OR, NOT`.

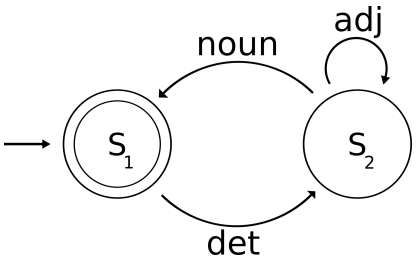

Figure 3: A finite-state automaton describing the regular language `det (adj)* noun`.

4 Are all regular languages in CG?

Is CG in any particular category? It clearly covers some regular, context-free and context-sensitive languages, but it may just have a common subset with those classes, as in Figure 2. Rather than enumerating individual grammars for a given class, we need a method that can transform all languages in that class into CG.

We present a method to transform arbitrary finite-state automata into CG.[2] Figure 3 presents an example automaton, with $\Sigma = \{det, adj, noun\}$, for which we implement a corresponding CG as follows.

As a modification to $\langle\Sigma\rangle_n$, we use $\langle\Sigma, S\rangle_n$: Σ stands for maximally ambiguous *word cohorts*, and S for maximally ambiguous *state cohorts*, which are inserted between each $\langle\Sigma\rangle$. For example, a sequence with two transitions would be modelled with the following sentence $\langle\Sigma, S\rangle_2$, with two word cohorts:

```
"<s>"   "<w>"   "<s>"   "<w>"   "<s>"
  s1      det     s1      det     s1
  s2      adj     s2      adj     s2
          noun            noun
```

The rules of the grammar disambiguate both word cohorts and state cohorts. Thus the desired result shows both the accepted string and the path in the automaton.

```
"<s>"   "<w>"   "<s>"   "<w>"   "<s>"
  s1      det     s2      noun    s1
```

4.1 Disambiguation process

Given that every transition happens between two states, and every state has an incoming and outgoing transition, every rule needs no more than

[2] A subset which includes `LINK`, `NEGATE` and `TEMPLATE`, in addition to the previous.

positions -1 and 1 in their contextual tests. The semantics of the rules are "remove a POS tag, if it is *not* surrounded by allowed states", and "remove a state, if it is *not* surrounded by allowed transitions". For the example automaton, the POS-rules are as follows.

```
REMOVE...
   Det  IF (NEGATE -1 S1 LINK 2 S2) ;
   Adj  IF (NEGATE -1 S2 LINK 2 S2) ;
   Noun IF (NEGATE -1 S2 LINK 2 S1) ;
```

The start and end states naturally correspond to the first and last state cohort in the $\langle \Sigma, S \rangle_n$, and can be trivially disambiguated, in this case both into `s1`. Once we remove a reading from either side of a cohort, some more rules can take action—the context "`s2` on the left side and `s1` on the right side" may be broken by removing either `s2` or `s1`. As a chain reaction, the whole sentence gets eventually disambiguated.

4.2 Disjunction

Consider the regular language with two strings $\{ab, ba\}$. Given $\langle \Sigma \rangle_2$ for any Σ, this is the closest we could get in CG output:

```
"<w>"   "<w>"
     a       a
     b       b
```

As Lager and Nivre (2001) point out, CG has no way of expressing disjunction. Unlike its close cousin FSIG (Koskenniemi, 1990), which would represent this language faithfully, CG substitutes uncertainty on the sentence level ("either *ab* or *ba*") with uncertainty in the cohorts: "the first character may be either *a* or *b*, and the second character may be either *a* or *b*".

An alternative would be to model languages with disjunction as a set of CGs: one that disambiguates any input into *ab* and other that disambiguates into *ba*. But this is just speculation, we have not investigated the idea further.

5 Applications

Does such an idea provide any foreseeable practical benefits? The grammars derived from FSAs look clumsy, and involve extra symbols. But if it turned out that CGs can express some interesting subset of context-free or context-sensitive grammars, then we could derive CGs from already existing formalisms. This could be an alternative for learning CGs from corpus.

Compared to high-level grammar formalisms, such as (insert your favourite), CG processing is generally faster. So a CG derived in this manner could act as a preprocessing step for some more expensive parser—then the rules need not be human-legible. The rules would be derived from the grammar itself, with the sole purpose of making the parsing *faster*, not more accurate.

References

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

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

Kimmo Koskenniemi. 1990. Finite-state parsing and disambiguation. In *Proceedings of 13th International Conference on Computational Linguistics (COLING 1990)*, volume 2, pages 229–232, Stroudsburg, PA, USA. Association for Computational Linguistics.

Torbjörn Lager and Joakim Nivre. 2001. Part of speech tagging from a logical point of view. In *Logical Aspects of Computational Linguistics, 4th International Conference (LACL 2001)*, pages 212–227.

Dávid Márk Nemeskey, Francis Tyers, and Mans Hulden. 2014. Why implementation matters: Evaluation of an open-source constraint grammar parser. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING 2014)*, pages 772–780, Dublin, Ireland, August.

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