Towards Natural Language Explanations of Constraint Grammar Rules

Daniel Swanson
Indiana University
Department of Linguistics
Bloomington, Indiana
dangswan@iu.edu

Abstract

This paper presents a general-purpose parser for static analysis of Constraint Grammar rules (that is, examining only the rules, not potential inputs and outputs) and applies it to the task of translating rules into comprehensible explanations of behavior. An interactive interface for exploring how individual components of each rule contribute to these translations is also presented.

1 Introduction

Constraint Grammar (Karlsson et al., 2011; Bick and Didriksen, 2015) is a rule-based procedural text processing paradigm which can be used for a wide range of tasks, from part-of-speech tagging to word sense disambiguation to parsing to translation. The formalism for expressing these rules is quite compact, and can, at times, give rise to extremely complicated and arcane rules. Deciphering what such rules do can often be rather challenging, particularly for beginners.

For example, consider the rule in Figure 1 from the parser in Swanson and Tyers (2022). The grammar operates on a corpus which has been tagged for some syntactic relations but does not have full trees. This particular rule tries to attach a clause root to an immediately preceding clause root that is a verb of speaking and is not as deeply nested in quotations. For example, given the input "He said 'Go.'.", the word "said" would have `<txt:0>`, indicating that it was not a quotation and "go" would have `<txt:1>`, indicating that it was one quote deep and the rule would make "go" a child of "said".

This rule occurs in a collection of roughly 20 others with no comments other than the section heading "Clause Connections" for context. Ideally, any nontrivial rule would be accompanied by an explanatory comment to aid in deciphering it, but this is often not the case, leaving a daunting task for those who might later seek to understand a grammar.

To aid in remedying this problem, we now present an interactive tool which translates Constraint Grammar rules into English sentences and highlights which piece of the rule contributes each piece of the explanation, which can serve as a starting point for documenting existing grammars in addition to providing support to students trying to learn Constraint Grammar.

The paper is organized as follows: Section 2 describes our CG parser, Section 3 describes our translation rules, Section 4 describes the interactive interface for these translations, and Section 5 concludes.

2 Parsing Constraint Grammar

In order to explain a rule, it is first necessary to parse the rule. For this task, we employ the Tree-Sitter library (Brunsfeld et al., 2024). Tree-Sitter provides tooling for writing context-free grammars which compile to efficient parsers with bindings in many major programming languages. The original purpose of the library was as an alternative to regular expressions for syntax highlighting. As a result, it is fast and robust against incomplete or invalid input, which allows our tool to produce useful output even if the user pastes in an invalid rule. It also supports incremental re-parsing after edits, though we do not yet make use of this feature.

An example of a rule in the grammar definition is shown in Figure 2. This rule describes how to parse a tag list, which consists of the `LIST` keyword, a set name, an equals sign, a sequence of tags, and a semicolon. It labels the name and the list of tags, so that processing scripts can more easily retrieve those. These rules are semantically similar to other parser generators, such as

```
SET NonPredAdv = (advb) - (role:P) ;
SET Top = (_) - (conj) - (@discourse) - NonPredAdv ;
WITH Top + (/^<txt:\(\\d+\)>$/r)
  IF (-1* Top + (VSTR:<txt<$1>) BARRIER Top) {
    SETPARENT (*) TO (jC1 (*)) ;
    MAP (@ccomp) (*) IF (jC1 SpeakingVerb) ;
    MAP (@xcomp) (verb infc) IF (jC1 XcompInf) (c (prep)) ;
} ;
```

Figure 1: A compound rule (Swanson et al., 2023) from the parser in Swanson and Tyers (2022) which attaches clause roots to the corresponding speaking verb. The set `SpeakingVerb` is a list of lemmas of verbs which can introduce quotations and the set `XcompInf` is a list of control verbs that take infinitive complements. Tags prefixed with `@` are dependency labels and the rest are part-of-speech tags, apart from `role:P`, which indicates that the source data marks the word as a predicate, `_`, which indicates that the source data marks the word as the root of an independent clause, and `<txt:N>`, which indicates that a segment of text is N layers of quotation deep (so `<txt:0>` is narrative and `<txt:2>` is a quotation within a quotation).

```
    LIST: $ => kwd('LIST'),
    list: $ => seq(
      $.LIST,
      field('name', $.setname),
      choice($.eq, $.pluseq),
      field('value', $.taglist),
      $.semicolon,
    ),
```

Figure 2: A Tree-Sitter rule for parsing `LIST` definitions. A `list` node is defined as consisting of a `LIST` keyword, a name, an operator, a list of tags, and a semicolon.

YACC (Levine et al., 1992), though expressed in JavaScript syntax. The present CG parser consists of 101 such rules and has been tested against every CG file maintained by the Apertium project (Forcada et al., 2011; Khanna et al., 2021).

An example of the output of this rule is shown in Figure 3. Given the list of gender tags shown, the overall parser will produce the tree. This is the default string representation of the trees, with nodes denoted by S-expressions and field names indicated by colons. For example, `(tag (ntag))` indicates an `ntag` node[1] which is a child of a `tag` node.

The parser is usable for static analysis and syntax highlighting and is available from GitHub[2], NPM[3], and PyPI[4].

Input:

```
LIST Gender = m f nt ;
```

Output:

```
(source_file
  (list
    (LIST)
    name: (setname)
    (eq)
    value: (taglist
      (tag (ntag))
      (tag (ntag))
      (tag (ntag)))
    (semicolon)))
```

Figure 3: The parse tree of a CG declaration as produced by the Tree-Sitter grammar.

3 Translation Rules

In order to generate the English translation of a given tree, we apply a set of rules that extract a node and its descendants using Tree-Sitter's builtin query system and map them to string templates which those descendants are inserted into. For example, the template for a `setname` node is `the set {name}`, where `name` is the name of the set as it appears in the rule. Rules are applied

[1] `ntag` is a non-quoted tag, contrasted with `qtag`, which is a quoted tag.

[2] `https://github.com/apertium/tree-sitter-apertium`

[3] `https://www.npmjs.com/package/tree-sitter-cg`

[4] `https://pypi.org/project/tree-sitter-cg/`

Define the set NonPredAdv as any word which has the tag advb and does not have the tag role:P.

Define the set Top as any word which has the tag _ and does not have the tag conj and does not have the tag @discourse and does not match the set NonPredAdv.

Find a word which matches the set Top and has the tag /^<txt:\(\d+\)>$/r in context some reading in the cohort 1 positions to the left or further in that direction (stop looking if you reach one which matches the set Top) is one which matches the set Top and has the tag VSTR:<txt<$1> and run the following rules:

> *Set the parent of any word which is the target of the containing WITH rule to some reading in the cohort found by context test 1 in the containing WITH rule which must be one which can be any word.*
>
> *If some reading in the cohort found by context test 1 in the containing WITH rule is one which matches the set SpeakingVerb then add the tag @ccomp to a word which is the target of the containing WITH rule and prevent other rules from adding tags.*
>
> *If some reading in the cohort found by context test 1 in the containing WITH rule is one which matches the set XcompInf and some reading in a child cohort is one which has the tag prep then add the tag @xcomp to a word the tags verb, infc and prevent other rules from adding tags.*

Figure 4: The output of the translator for the rule shown in Figure 1.

in order and from top to bottom of the tree, with the first to match taking precedence.

Most of the rules are written to capture a single node and translate it without knowing what its parent or child is. Thus an `inlineset` node, which can be a setname or a list of tags in parentheses or various combinations of the two, is translated using `which has ...` or `which matches ...` and most other nodes which might contain `inlinesets` are written to ensure that that makes grammatical sense. Occasionally, this is not feasible and in those cases, a rule can be written which captures its grandchildren or is conditioned on its parent. An example of the former is a special case of the `inlineset` rule which checks for an inline set that only contains the tag `*` and translates it to `any word`. An example of the latter is a special case of that which further checks if the set is the target of a rule inside a `WITH` block, in which case the translation is `the target of the containing WITH rule`. An example of the output of the translator is shown in Figure 4.

We have implemented processors for these rules in both Python and JavaScript, enabling usage in both offline scripts and the browser.

4 Interactive Interface

We have created an online front-end for the translation rules[5]. The interface presents two panes. The user can type or paste rules in the left pane and the right pane will show a live-updating translation of the rules. A screenshot of the interface is shown in Figure 5.

We use the Tree-Sitter parse tree to provide syntax highlighting to the rules in the left pane. In addition, we tag each node of the tree in both the source rules and the translation, so that if the user hovers over either side, the corresponding text in the other pane is highlighted.

5 Conclusion and Future Work

In this paper, we have presented a general-purpose parser for static analysis of Constraint Grammar rules, as well as a system on top of that for translating those rules into English sentences.

The biggest limitation of the current system is that the parser does not distinguish between different types of tags, so simple tags like `n`, regular expressions like `/n\(\\d\)/r`, string substitutions like `VSTR:$1`, and numeric comparisons like `<P>3>` are all currently parsed as simply unquoted tags, making it difficult to fully translate them based solely on the parse tree. Thus in the translation in Figure 4, there is no indication of the fact that the `WITH` condition is looking for a pair of words where one has a numeric `<txt>` tag with a lower value than the other. Unfortunately, it is probably impossible to solve this problem completely, since string substitution can generate special tags, and thus some thing cannot be identi-

[5] Available at `https://mr-martian.github.io/rule-explainer/`

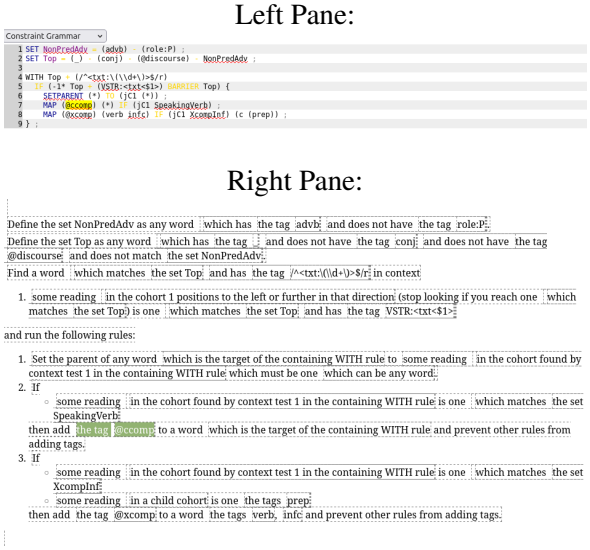

Figure 5: The interactive interface for the rule-explainer. The user is hovering over the words *the tag @ccomp* in the right-hand pane and so the `@ccomp` tag on line 7 is highlighted in the left-hand pane. (In a webbrowser, these sections will appear next to each other, but here they are shown vertically for the sake of legibility.)

fied without executing the rule, but some amount of greater specificity is probably possible.

Another avenue for expansion is to translate into other natural languages or into multiple levels of detail (that is, to collapse some parts of the explanation if they are not needed). Perhaps there could be a more beginner-friendly set of rules that explain each step at length and a more simple version for users who understand CG rules in general and just happen to be confused by one in particular.

Finally, one could imagine a system that does the reverse of this one and translates English descriptions into actual CG rules, along the lines of what Tyers and Howell (2021) report doing manually. This system could be used to generate an initial parallel corpus for training a reverse system and also to help users double check the output.

Acknowledgments

I would like to thank Tino Didriksen for his assistance in the development and packaging of the Tree-Sitter grammar. I would like to thank Matthew Fort and Meesum Alam for their feedback on the structure of some of the rule descriptions. Finally, I would like to thank Robert Pugh, Nils Hjortnaes, and Francis Tyers for their comments on earlier drafts of this paper.

References

Eckhard Bick and Tino Didriksen. 2015. CG-3 — beyond classical constraint grammar. In *Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*, pages 31–39, Vilnius, Lithuania. Linköping University Electronic Press, Sweden.

Max Brunsfeld, Amaan Qureshi, Andrew Hlynskyi, Patrick Thomson, ObserverOfTime, Josh Vera, dundargoc, Phil Turnbull, Timothy Clem, Douglas Creager, Andrew Helwer, Rob Rix, Daumantas Kavolis, Hendrik van Antwerpen, Michael Davis, Will Lillis, Ika, Amin Yahyaabadi, Tuan-Ahn Nguyen, bfredl, Matt Massicotte, Stafford Brunk, Christian Clason, Niranjan Hasabnis, Mingkai Dong, Samuel Moelius, Steven Kalt, Segev Finer, and Kolja. 2024. tree-sitter/tree-sitter: v0.24.4.

Mikel L Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M Tyers. 2011. Apertium: a free/open-source platform for rule-based machine translation. *Machine translation*, 25:127–144.

Fred Karlsson, Atro Voutilainen, Juha Heikkilae, and Arto Anttila. 2011. *Constraint Grammar: a language-independent system for parsing unrestricted text*, volume 4. Walter de Gruyter.

Tanmai Khanna, Jonathan N Washington, Francis M Tyers, Sevilay Bayatlı, Daniel G Swanson, Tommi A Pirinen, Irene Tang, and Hector Alos i Font. 2021. Recent advances in apertium, a free/open-source rule-based machine translation platform for low-resource languages. *Machine Translation*, 35(4):475–502.

John R Levine, Tony Mason, and Doug Brown. 1992. *Lex & yacc*. O'Reilly Media, Inc.

Daniel Swanson, Tino Didriksen, and Francis M. Tyers. 2023. WITH context: Adding rule-grouping to VISL CG-3. In *Proceedings of the NoDaLiDa 2023 Workshop on Constraint Grammar - Methods, Tools and Applications*, pages 10–14, Tórshavn, Faroe Islands. Association of Computational Linguistics.

Daniel Swanson and Francis Tyers. 2022. A Universal Dependencies treebank of Ancient Hebrew. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 2353–2361, Marseille, France. European Language Resources Association.

Francis M Tyers and Nick Howell. 2021. Morphological analysis and disambiguation for breton. *Language Resources and Evaluation*, 55(2):431–473.