WITH Context: Adding Rule-Grouping to VISL CG-3

Anonymous Author
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

Anonymouser Author
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

Anonymousest Author
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

Abstract

This paper presents an extension to the VISL CG-3 compiler and processor which enables complex contexts to be shared between rules. This sharing substantially improves the readability and maintainability of sets of rules performing multi-step operations.

1 Introduction

When writing constraint grammars for more complex tasks, such as parsing or translation, situations often arise in which a particular context triggers multiple operations. For example, when writing a dependency parser, the head of a word and its grammatical function label are often determined jointly. Similarly, for tasks such as translation that involve modifying either the syntactic structure or the linear order of the words, a change in one word will typically necessitate changes to its dependents as well.

One way to handle such cases in CG is to have each operation repeat the entire set of contextual tests, which is tedious to write, difficult to read, and error-prone to maintain. Another way is to add an initial rule which checks the conditions and adds a label to the target word and then have each other rule simply check for the appropriate label. This, however, leads to a proliferation of single-use tags in the grammar (which may need to be documented), and does not solve the problem that rules which operate on relationships between words, such as `SETPARENT` or `ADDRELATION` still need to duplicate contextual tests in order to locate the second cohort.

To address these difficulties, we extend the VISL CG-3 processor (Bick and Didriksen, 2015) with the operator `WITH`, which matches a context and then runs multiple rules, all with that same context. An example is given in (1).

(1)

```
WITH (n) IF (-1* (det)) {
  SETCHILD (*) TO (jC1 (*)) ;
  SETCHILD REPEAT (*) TO
    (-1*A (adj) LINK -1* _C1_) ;
} ;
```

Here the context being matched is a noun preceded at any distance by a determiner. The subsequent rules are then run with the noun as their target, so the target can be the any set (if a rule specifies a target set, then it will only be run if that set matches the target of the `WITH`). The rules can refer to the cohorts matched by the contextual tests of the `WITH` using either the position specifiers `jC1`, `jC2`, ... `jC9` for the first through ninth tests, respectively, or using the magic sets `_C1_`, `_C2_`, ... `_C9_`.

Thus the first `SETCHILD` attaches the determiner (here matched with `jC1 (*)`) to the noun and the second one finds any adjectives which are between the noun and the determiner (here matched with `-1* _C1_`) and attaches them to the noun. By default, rules inside a `WITH` are run once when the `WITH`, but `REPEAT` has the usual effect of causing the rule to be repeated until it has no effect.

A more extensive example, taken from an in-progress rewrite of an existing parser, is presented in Figure 1.

As these examples show, the `WITH` operator, while not strictly increasing the expressivity of CG, does allow many sets of rules to be written in a much more readable and maintainable manner.

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.

```
# Original rules

MAP @flat BigNumber + Number IF (-1 Number) ;
SETPARENT @flat + Number (NOT p (*)) TO (-1 Number) ;

MAP @conj Number
    IF (-1 @cc LINK -1* Number BARRIER (*) - @flat) ;
SETPARENT @cc (NOT p (*)) TO (1 Number + @conj) ;
SETPARENT Number + @conj (NOT p (*))
    TO (-1* Number - @flat BARRIER (*) - @cc - @flat) ;
REMCOHORT IGNORED WITHCHILD (*)
    Number + @conj OR Number + @flat
    IF (p Number) ;


# Rules rewritten using WITH

WITH BigNumber + Number (-1 Number) (NOT p (*)) {
  MAP @flat (*) ;
  SETPARENT (*) TO (jC1 (*)) ;
  REMCOHORT IGNORED (*) ;
} ;

WITH Number (-1 @cc) (-2 Number) (NOT p (*)) {
  MAP @conj (*) ;
  SETCHILD (*) TO (jC1 (*)) ;
  SETPARENT (*) TO (jC2 (*)) ;
  REMCOHORT IGNORED WITHCHILD (*) (*) ;
} ;
```

Figure 1: A set of rules for parsing Hebrew number phrases according to Universal Dependencies (Nivre et al., 2020), with and without the `WITH` operator. The original set of rules is taken from the parser described in Swanson and Tyers (2022). In each set, the first group of rules matches a phrase such as שלוש מאה "three hundreds" and makes the second word dependent on the first with the label `flat`. Then the second group matches a phrase like תשע וערבע "nine and four" and attaches the conjunction to the second number and the second number to the first, giving the second number the label `conj`. Finally the dependent words are ignored (treated as deleted for the remainder of parsing, but included in the output).

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. Universal Dependencies v2: An evergrowing multilingual treebank collection. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4034–4043, Marseille, France. European Language Resources Association.

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.