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

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

Tino Didriksen
Institute of Language and Communication
University of Southern Denmark
tinod@sdu.dk

Francis Tyers
Department of Linguistics
Indiana University
ftyers@iu.edu

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. This new operator has been released as part of VISL CG-3 version 1.4.0. Section 2 describes the syntax of this operator, Section 3 provides examples of its application in various domains, Section 4 discusses its performance implications, and Section 5 concludes.

2 Syntax

An example of the `WITH` operator in use 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.

As this example and those in the next section show, the `WITH` operator, while not strictly increasing the expressivity of CG, does allow many

Proceedings of the NoDaLiDa 2023 Workshop on Constraint Grammar - Methods, Tools and Applications, pages 10–14

sets of rules to be written in a much more readable and maintainable manner.

3 Examples

The following subsections give examples of applications of `WITH` to particular domains.

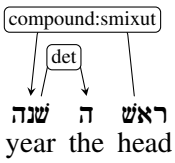
3.1 Dependency Parsing

A set of rules for dealing with numbers and determiners found in the parser from Swanson and Tyers (2022) is presented in Figure 1.

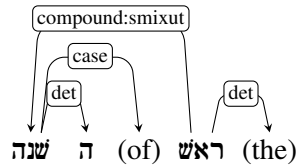
3.2 Translation

Multi-step transformations are also relevant in machine translation. For instance, transforming the dependency tree in (2) for the Hebrew phrase **רֹאשׁ הַשָּׁנָה** to the appropriate tree for the English "the head (or beginning) of the year", given in (3).

(2)



(3)



This can be accomplished using the rules in (4).

(4)

```
WITH (n @compound:smixut)
  IF (p (n))
  (NEGATE c (@compound:smixut))
{
  ADDCOHORT
    ("<the>" "the" det def @det)
    BEFORE (*)
    IF (c (@det)) (jC1A (*)) ;
  ADDCOHORT
    ("<of>" "of" pr @case)
    BEFORE WITHCHILD (*) (*) ;
  UNMAP (@compound:smixut) (*) ;
  MAP (@nmod) (*) ;
} ;
```

Note, particularly, the `UNMAP` followed by the `MAP`, which would otherwise be extremely difficult to do correctly, since the `MAP` would otherwise need some way of finding the cohort it was supposed to replace the tag of, when that cohort no longer has that tag.

3.3 Morphological Disambiguation

Even in tasks that typically do not require composite operations, such as disambiguation, there is often a high degree of duplication in contextual tests which can benefit from the use of `WITH`. For example, in the Apertium morphological disambiguator for Norwegian Nynorsk (Unhammer and Trosterud, 2009) 1504 (38.5%) of the 3903 rules have at least 2 tests and share at least 90% of those tests with another rule in the file. The rules in (5) are given as an example of such overlap.

(5)

```
SELECT:4144 (adj pl) IF
    (NOT 0 fv)
    (NOT 0 subst)
    (NOT 0 det)
    (1 komma/konj)
    (-1C fl-det)
    (NOT 1 subst/adj)
    (NOT 2 adj)
;
SELECT:4145 (adj pl) IF
    (NOT 0 fv)
    (NOT 0 subst)
    (NOT 0 det)
    (NOT 0 pos)
    (-1C fl-det)
    (NOT 1 subst/adj)
;
```

The duplicate tests can be extracted, as in (6).

(6)

```
WITH (adj pl) IF
    (NOT 0 fv)
    (NOT 0 subst)
    (NOT 0 det)
    (-1C fl-det)
    (NOT 1 subst/adj)
{
    SELECT (adj pl) IF
        (1 komma/konj)
        (NOT 2 adj) ;
```

```
# 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 וארבע** "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).

Grammar	Rules	WITH Groups	Runtime	Cohorts	Cohorts/s	Speedup
Hebrew Original	346	0	5.58 s	65K	11,756	0%
Hebrew safe-setparent	346	0	4.80 s	65K	13,690	14%
Hebrew WITH	349	9	4.93 s	65K	13,310	12%
Norwegian Original	3903	0	142.26 s	174K	1,225	0%
Norwegian WITH	3903	180	79.43 s	174K	2,194	44%

Table 1: Performance comparison of the rewrite of the Ancient Hebrew dependency parser from Swanson and Tyers (2022) and an automated refactoring of the Norwegian Nynorsk morphological disambiguator from Unhammer and Trosterud (2009). "Hebrew Original" is the parser presented in the first paper, "Hebrew safe-setparent" is the same parser, but with `safe-setparent` flag enabled, and "Hebrew WITH" is a version that has been partially refactored to use `WITH` groups and also slightly expanded. The parser using `WITH` also uses `safe-setparent`. "Norwegian Original" is the disambiguation grammar distributed by Apertium as of April 2023 and "Norwegian WITH" is an automated transformation of that grammar. In neither language is the grammar using `WITH` perfectly identical to the original in terms of output.

```
    SELECT (adj pl) IF
        (NOT 0 pos) ;
} ;
```

Here the 5 contexts that are shared between the two rules are written only once and each rule need only specify the part that differs, substantially clarifying the purpose of having distinct rules in this instance.

4 Performance

The performance impact of adding a `WITH` group to a grammar is generally small, though measurable. In this section we present the effects on two grammars: the Ancient Hebrew dependency parser from Swanson and Tyers (2022) and the Apertium Norwegian Nynorsk morphological disambiguator (Unhammer and Trosterud, 2009). The results are listed in Table 1.

When `WITH` improves performance, it is generally due to a reduction in the number of contextual tests that need to be evaluated. However, the deduplication of tests is balanced by the fact that `WITH` must evaluate them sequentially in order to populate the `_Cn_` magic sets whereas for most rules the VISL CG-3 processor will internally update the order so as to start with the test that is most likely to fail.

Thus, when refactoring a complex grammar by hand where the total number of `WITH` groups added is likely to be small, the potential speedup is relatively small and is easily overwhelmed by the impact of new rules. In the Hebrew parser, for example, the effect of rearranging the contextual tests of roughly 30 rules (9% of the grammar) into 9 `WITH` groups was negated by adding half a dozen new ones (overall a 2% slowdown), and both of these effects are minor compared to the effect of an unrelated change that removed one test from each of 135 rules (a 14% speedup).

On the other hand, in the Norwegian grammar, the relationships between rules are generally quite simple and we were thus able to write a script that automatically merged adjacent rules which shared the same target and had at least 5 contextual tests in common into a `WITH` group. The results of this conversion are not perfect (just over 5% of the 10K sentences in our test data have different output), but they are good enough for an approximate comparison. The script grouped 1636 rules (42% of the grammar) into 180 `WITH` groups, increasing the speed of the disambiguator by 44%.

5 Conclusion

In this paper we have presented the `WITH` operator, an extension of VISL CG-3 to allow collections of rules to be grouped into composite operations. As shown in the examples, this addition is likely to be useful to grammar authors approaching a wide variety of tasks and can even have a significant impact on grammar performance if deployed on a large scale.

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.

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.

Kevin Unhammer and Trond Trosterud. 2009. Reuse of free resources in machine translation between nynorsk and bokmål. In *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation*, pages 35–42.