SIGMA - Normalization

Isaac Griffith and Rosetta Roberts
Empirical Software Engineering Laboratory
Informatics and Computer Science
Idaho State University
Pocatello, Idaho, 83208
Email: {grifisaa@isu.edu, roberose@isu.edu}

***Abstract*—Introduction:**
Objective:
Methods:
Results: What are the main findings? Practical implications?
Limitations: What are the weaknesses of this research?
Conclusions: What is the conclusion?

***Index Terms*—Island Grammars, Automated Grammar Formation, Software Language Engineering**

I. Introduction

Multilingual parsing is an open problem that is currently being worked on. One method of multilingual parsing that has been developed is by creating island grammars for the combined grammars [1]. An automated method for doing this is currently being developed [SIGMA REFERENCE HERE]. One challenge to automating the creation of island grammar based multilingual parsers is detecting similar parts of grammars and combining them. This process becomes easier with an appropriate normalization process and normal form.

In this project, we propose to design and verify a procedure for appropriate normalization.

RG Design a normal form and normalization process for grammars that is conducive to identifying and merging similar parts across grammars to be used for merging grammars for the automated creation of multilingual parserss.

Organization

The remainder of this paper is organized as follows. Sec. II discusses the theoretical foundations of this work while also discussing other related studies. Sec. V details the design of experiments which evaluate the normalization steps of SIGMA. Sec. VI details the threats to the validity of this study. Finally, this paper is concluded in Sec. VII.

II. Background and Related Work

A. Theoretical Foundations

Context-free grammars are defined as $G = (V, \Sigma, P, S)$, where V is the set of non-terminal symbols, Σ is the set of terminal symbols, P is the set of productions, and S is the start production [2]. Individual productions can be written as $\Phi \to R$, where Φ is any symbol and R is a rule that is either a symbol, ϵ, rules concatenated together, or rules unioned together with the | operator. Rather than using multiple productions when a symbol can produce multiple rules, the rules are combined with the | operator.

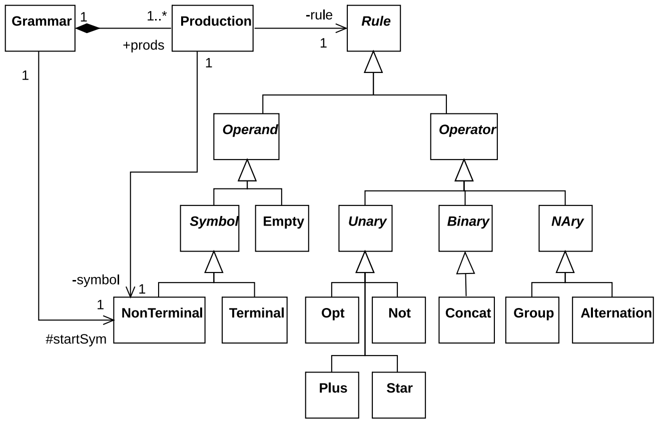

Fig. 1. Grammar metamodel.

Various normal forms of grammars have been proposed by people. These normal forms have mainly been introduced to make parsing and manipulating grammars easier. The most used normal form is the Chomsky Normal Form (CNF) [3]. Grammars are transformed into CNF and other normal forms through simple transformations which preserve the language of the grammar. Normal forms for finding and merging similar parts across grammars have not been designed.

B. Related Studies

C. Research Contributions

III. Approach

A. Domain Object Model

B. Design

To design our normal form, we decided that it should have the following properties.

1. Our domain object model represents each rule as a tree of operators and operands. To simplify different processes, we want the tree for each rule to be flat. We can ensure this by requiring that each rule has a single operator at most.
2. We desire that the size increase induced by normalization is minimal.

3. We desire that the normalized grammar is unambiguous given a grammar. I.e. there is exactly one normalized grammar for each grammar.
4. We desire that certain transformations on the input grammar before normalization do not change the normalization result. These transformations are
    1. Refactoring common terms of rules into a separate rule.
    2. Duplicating a rule.
    3. Introducing an unused non-terminal symbol and its production.
    4. Replacing a non-terminal symbol with a non-terminal symbol that produces that no-terminal symbol.
    5. Replacing all usages of a non-terminal symbol with its rule.

To meet these requirements, we decided that the normal form would have each production as one of the following forms.

1. $\langle\text{Form}_1\rangle \to \langle\text{A}\rangle\ a\ \ldots$, where each term is a terminal symbol or a non-terminal symbol with an F_2 production and there are at least two terms in the rule.
2. $\langle\text{Form}_2\rangle \to \langle\text{A}\rangle\ \mid\ a\ \mid\ \ldots$, where each term is a terminal symbol, the empty string, or a non-terminal symbol with an F_1 production and there are at least two terms in the rule except for the special case when there is only production.

The reason we chose these two forms is that because rules of these forms are relatively easy to compare. This allows rules to easily be compared and merged. The restriction that non-terminal symbols referenced in each rule must have productions of the opposite form is so that examples

IV. Normalization Algorithm

The following algorithm defines the approach for normalizing a given grammar. The normalization process defined here facilitates the ability to merge productions, in pursuit of the overarching goal of automated generation of Island [X], Tolerant [X], Bridge [X], and Bounded Seas [X] grammars.

This algorithm assumes that the source grammar, G, was initially in some defined formalism such as Antlr [X], EBNF [X], BNF [X], SDF [X], TXL [X], etc. The grammar was then read in and processed to conform to the metamodel depicted in Figure 1. Assuming that the grammar meets this condition, the goal of this algorithm is then to reformat the grammar such that each production is of one of Form_1 or Form_2

The normalization process, as defined in Algorithm 1, repeatedly executes six processes until the grammar stabilizes. These six processes are: i) eliminating unused rules, ii) simplifying productions, iii) merging equivalent rules, iv) eliminating unit rules, v) expanding productions, and vi) collapsing compatible productions.

Algorithm 1 Normalization Algorithm

```
1: procedure NORMALIZE($\mathcal{G}$)
2:     repeat
3:         $\mathcal{G} \leftarrow$ ELIMINATEUNUSEDPRODUCTIONS($\mathcal{G}$)
4:         $\mathcal{G} \leftarrow$ SIMPLIFYPRODUCTIONS($\mathcal{G}$)
5:         $\mathcal{G} \leftarrow$ MERGEEQUIVPRODUCTIONS($\mathcal{G}$)
6:         $\mathcal{G} \leftarrow$ ELIMINATEUNITPRODUCTIONS($\mathcal{G}$)
7:         $\mathcal{G} \leftarrow$ EXPANDPRODUCTIONS($\mathcal{G}$)
8:         $\mathcal{G} \leftarrow$ COLLAPSEPRODUCTIONS($\mathcal{G}$)
9:     until UNCHANGED($\mathcal{G}$)
10: end procedure
```

Algorithm 2 Eliminate Unused Productions

```
1: function ELIMINATEUNUSEDPRODUCTIONS($\mathcal{G}$)
2:     $H \leftarrow (V, E)$
       ▷ Create empty graph
3:     for all $v \in \mathcal{G}.V$ do
4:         $\mathcal{H}.V \leftarrow \mathcal{H}.V \cup \{v\}$
5:         ADDRULETOGRAPH($\mathcal{G}, \mathcal{H}, \mathcal{G}.P(v)$)
6:         $\mathcal{H}.E \leftarrow \mathcal{H}.E \cup \{(v, \mathcal{G}.P(v))\}$
7:     end for
8:     DFSMARK($\mathcal{G}.S$)
9:     $\mathcal{G}.V \leftarrow \{\, v \in \mathcal{G}.V \mid \text{MARKED}(v) \,\}$
10:    $\mathcal{G}.P \leftarrow \{\, (v, \mathcal{G}.P(v)) \mid v \in \mathcal{G}.V \,\}$
11: end function
12: function ADDRULETOGRAPH($\mathcal{G}, \mathcal{H}, r$)
13:     $\mathcal{H}.V \leftarrow \mathcal{H}.V \cup \{r\}$
14:     if ISOPERATOR($r$) then
15:         for all $c \in$ OPERANDS($r$) do
16:             ADDRULETOGRAPH($\mathcal{G}, \mathcal{H}, c$)
17:             $\mathcal{H}.E \leftarrow \mathcal{H}.E \cup \{(r, c)\}$
18:         end for
19:     end if
20: end function
```

A. *Eliminating Unused Rules*

This process removes all productions that are not produced, directly or indirectly, from the start production. This is accomplished by enumerating all symbols producuable from the start symbol via a depth first search (see Algorithm 3) and then creating a new grammar using only the enumerated symbols, as shown in Algorithm 2.

B. *Simplifying Productions*

This process aims to simplify productions. This is achieved by removing unnecessary ε's concatenated with other rules and replacing operators with only one operand with their operator. This process is embodied in Algorithm 4.

C. *Merging Equivalent Productions*

Productions that have identical rules are replaced by a single production. This new production is given a name derived from the productions that were merged to create it. The algorithm for this is shown in Alg. 5.

D. *Eliminating Unit Productions*

All non-terminals with productions of one of the following two forms will have their non-terminal symbols replaced by their rules, and their productions eliminated.

Algorithm 3 Depth First Marking

```
1:  function DFSMARK($start$)
2:      $\mathcal{S} \leftarrow [start]$
3:      while $\mathcal{S} \neq \varnothing$ do
4:          $p \leftarrow$ POP($\mathcal{S}$)
5:          MARK($p$)
6:          for all $s \in$SUCC($p$) do
7:              if !ISMARKED($s$) then
8:                  PUSH($\mathcal{S}, s$)
9:              end if
10:         end for
11:     end while
12: end function
```

Algorithm 4 Simplify Productions

```
1:  function SIMPLIFYPRODUCTIONS($\mathcal{G}$)
2:      for all $v \in \mathcal{G}.V$ do
3:          $\mathcal{G}.P(v) \leftarrow$ SIMPLIFYRULE($\mathcal{G}.P(v)$)
4:      end for
5:  end function
6:  function SIMPLIFYRULE($r$)
        ▷ Replace empty terminal string with $\epsilon$
7:      if ISTERMINAL($r$) $\wedge$ ISEMPTY($r$) then
8:          return $\epsilon$
9:      end if
10:     if ISOPERATOR($r$) then
11:         let $C$ be CHILDREN($r$)
12:         $C \leftarrow \{$ SIMPLIFYRULE($c$) $\mid c \in C \}$
13:         if ISCONCATENATE($r$) then
14:             $C \leftarrow \{ c \in C \mid c \neq \epsilon \}$
15:             if $|C| = 0$ then
16:                 return $\epsilon$
17:             end if
18:         end if
19:         if ISCONCATENATE($r$) $\vee$ ISUNION($r$) then
        ▷ Replace operators with single operand with operand
20:             if $|C| = 1$ then
21:                 let $\{c\}$ be $C$
22:                 return $c$
23:             else
24:                 return $r$
25:             end if
26:         end if
27:     end if
28: end function
```

$$\langle \text{a} \rangle \rightarrow \langle \text{b} \rangle$$
$$\langle \text{a} \rangle \rightarrow \texttt{a}$$

Elimination of productions of the first form, is derived from Chomsky Normal Form (CNF) [X]. Eliminations of productions of the second form, a derivation from CNF, allows the simplification process to simplify rules of the following form:

Algorithm 5 Merge Equivalent Productions

```
1:  function MERGEEQUIVPRODUCTIONS($\mathcal{G}$)
2:      $pairs \leftarrow \varnothing$
3:      for $i \in [0, |\mathcal{G}.\Sigma|)$ do
4:          for $j \in (i, |\mathcal{G}.\Sigma|)$ do
5:              if $i \neq j$ then
6:                  $pairs \leftarrow pairs \cup (\mathcal{G}.\Sigma[i], \mathcal{G}.\Sigma[j])$
7:              end if
8:          end for
9:      end for
10:     for all $p \in pairs$ do
11:         if $p.left.rule = p.right.rule$ then
12:             COMBINEANDREPLACE($p.left, p.right$)
13:         end if
14:     end for
15: end function
```

Algorithm 6 Eliminate Unit Productions

```
1:  function ELIMINATEUNITPRODUCTIONS($\mathcal{G}$)
2:      for all $p \in \mathcal{G}.\Sigma$ do
3:          if $|p.rule| = 1$ then
4:              REPLACE($uses(p), p.rule$)
5:          end if
6:      end for
7:  end function
```

$$\langle \text{a} \rangle \rightarrow \langle \text{b} \rangle \; \texttt{a b}$$
$$\langle \text{b} \rangle \rightarrow \epsilon$$

When this step is applied to grammar G_6, it is transformed into grammar G_7, as depicted in Figure **??**.

E. Expanding Productions

Productions that have nested rules have all nested content replaced by with a non-terminal. The new non-terminal defines a production pointing to their content. When this step is applied to grammar G_7, it is transformed into grammar G_8, as depicted in Figure **??**.

F. Collapsing Compatible Productions

The final step of the normalization process combines productions that are compatible with each other. This ensures that any non-terminal symbols referenced in a rule will not define a duplicate production. The following provides an example:

$$\langle \text{A} \rangle \rightarrow \texttt{a} \; \langle \text{B} \rangle$$
$$\langle \text{B} \rangle \rightarrow \texttt{b c}$$
$$\langle \text{C} \rangle \rightarrow \texttt{c} \mid \langle \text{D} \rangle$$
$$\langle \text{D} \rangle \rightarrow \texttt{d} \mid \texttt{e}$$

would then collapse to form:

Algorithm 7 Expand Productions

```
1: function ExpandProductions($\mathcal{G}$)
2:     repeat
3:         $changed \leftarrow \bot$
4:         for all $p \in \mathcal{G}.\Sigma$ do
5:             if isConcat($p.rule$) then
6:                 for all $g \in p.rule$ do
7:                     if isGroup($g$) then
8:                         createAndReplaceWithProd($g$)
9:                         $changed \leftarrow \top$
10:                    end if
11:                end for
12:            else if isAlt($p.rule$) then
13:                for all $a \in p.rule$ do
14:                    createAndReplaceWithProd($a$)
15:                    $changed \leftarrow \top$
16:                end for
17:            end if
18:        end for
19:    until $changed = \bot$
20: end function
```

Algorithm 8 Collapse Productions

```
1: function CollapseProductions($\mathcal{G}$)
       ▷ Split productions into form1 and form2
2:     $f_1 \leftarrow$ Collect("form1")
3:     $f_2 \leftarrow$ Collect("form2")
4:     for all $p \in f_1$ do
5:         if OnlyTerminals($p.rule$) then
6:             ReplaceF1UsesWithRule($p$)
7:         end if
8:     end for
9:     for all $p \in f_2$ do
10:        if OnlyTerminals($p.rule$) then
11:            ReplaceF2UseWithRule($p$)
12:        end if
13:    end for
14: end function
```

$$\langle \text{A} \rangle \rightarrow \texttt{a b c}$$
$$\langle \text{C} \rangle \rightarrow \texttt{c} \mid \texttt{d} \mid \texttt{e}$$

When this step is applied to grammar G_8, it is transformed into grammar G_9, as depicted in Figure **??**.

V. Experimental Design

A. Goals, Hypotheses, and Variables

Goals: - Evaluate each step of normalization

Hypotheses: - Null hypothesis: each step has no effect on the halstead effort for MCC.

Variables: - Size of grammar as chosen in SIGMA - Blocking, - Halstead/MCC - The steps that the normalization goes through. (2ˆ#numSteps)

B. Design

Blocked factorial design

Reasons - Effect from size found in SIGMA - Interactions from size also seen in SIGMA

C. Experimental Units

D. Data Collection Procedures

E. Analysis Procedures

F. Evaluation of Validity

Conclusion Validity:

Internal Validity:

Construct Validity:

External Validity:

VI. Threats to Validity

VII. Conclusions and Future Work

The timeline to complete this study is as follows:

We intend to publish these results at one of the following conferences:

Acknowledgements

References

[1] N. Synytskyy, J. R. Cordy, and T. R. Dean, "Robust multilingual parsing using island grammars," in *Proceedings of the 2003 Conference of the Centre for Advanced Studies on Collaborative Research*. IBM Press, 2003, pp. 266–278.

[2] M. Haoxiang, *Languages and Machines: An Introduction to the Theory of Computer Science*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co. Inc., 1988.

[3] N. Chomsky, "On certain formal properties of grammars," *Information and Control*, vol. 2, no. 2, pp. 137–167, Jun. 1959.