

# 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 parsers.

## Organization

## 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,  $\Sigma$  is the set of terminal symbols,  $P$  is the set of productions, and  $S$  is the start production [2]. In this paper, a subset of Extended Backus-Naur Form (EBNF) to represent productions [?]. In each production  $\Phi \rightarrow R$ ,  $\Phi$  is a non-terminal symbol and  $R$  is an expression representing what the non-terminal symbol can be expanded into. An expression can be either a non-terminal symbol, a terminal symbol or string, the empty string ( $\epsilon$ ), expressions concatenated together, expressions separated with the  $|$  operator, or an expression surrounded by parenthesis. The  $|$  operator represents the union of sentences that can be expressed by each expression. The concatenation of expressions represents the sentences that can be formed by concatenating the sentences that each expression can form.

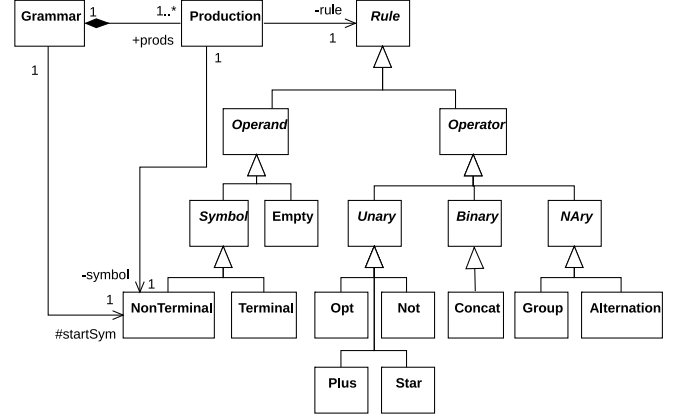


Fig. 1. Grammar metamodel.

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.

## 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 searching for similar rules and productions, 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 non-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.  $\langle \text{Form}_1 \rangle \rightarrow \langle A \rangle a \dots$ , 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.  $\langle \text{Form}_2 \rangle \rightarrow \langle A \rangle \mid a \mid \dots$ , 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

To meet requirement 4, our normalization process is performed using only transformation that are the inverse of transformations we do not want to affect our normalization process. To ensure that the size increase is minimal, we do not attempt to reverse transformations that rely on the distributive property between the concatenation and union  $\mid$  operators. To reverse transformations of this type, it is required to distribute productions. For example,

$$\begin{aligned} \langle A \rangle &\rightarrow a \langle B \rangle \\ \langle B \rangle &\rightarrow b \mid c \end{aligned}$$

would have to be transformed to

$$\langle A \rangle \rightarrow a b \mid a c.$$

Performing transformations of this kind repeatedly would result in an unreasonable increase in the number of productions. In addition, it would be unable to handle the case when a production indirectly references itself.

#### 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.

---

#### Algorithm 1 Normalization Algorithm

---

```

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

```

---

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  $\text{Form}_1$  or  $\text{Form}_2$

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.

##### 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 producible 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  $\varepsilon$ '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.

$$\begin{aligned} \langle a \rangle &\rightarrow \langle b \rangle \\ \langle a \rangle &\rightarrow a \end{aligned}$$

---

**Algorithm 2** Eliminate Unused Productions

---

```
1: function ELIMINATEUNUSEDPRODUCTIONS( $\mathcal{G}$ )
2:    $H \leftarrow (V, E)$ 
    $\triangleright$  Create empty graph
3:   for all  $v \in \mathcal{G}.V$  do
4:      $\mathcal{H}.V \leftarrow \mathcal{H}.V \cup \{v\}$ 
5:      $\text{ADDRULETOGRAPH}(\mathcal{G}, \mathcal{H}, \mathcal{G}.P(v))$ 
6:      $\mathcal{H}.E \leftarrow \mathcal{H}.E \cup \{(v, \mathcal{G}.P(v))\}$ 
7:   end for
8:    $\text{DFS MARK}(\mathcal{G}.S)$ 
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  $\text{ADDRULETOGRAPH}(\mathcal{G}, \mathcal{H}, r)$ 
13:   $\mathcal{H}.V \leftarrow \mathcal{H}.V \cup \{r\}$ 
14:  if  $\text{IS OPERATOR}(r)$  then
15:    for all  $c \in \text{OPERANDS}(r)$  do
16:       $\text{ADDRULETOGRAPH}(\mathcal{G}, \mathcal{H}, c)$ 
17:       $\mathcal{H}.E \leftarrow \mathcal{H}.E \cup \{(r, c)\}$ 
18:    end for
19:  end if
20: end function
```

---

---

**Algorithm 3** Depth First Marking

---

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

---

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:

$$\begin{aligned} \langle a \rangle &\rightarrow \langle b \rangle a b \\ \langle b \rangle &\rightarrow \epsilon \end{aligned}$$

### 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.

### F. Collapsing Compatible Productions

The final step of the normalization process combines productions that are associative with each other. This ensures that

---

**Algorithm 4** Simplify Productions

---

```
1: function  $\text{SIMPLIFYPRODUCTIONS}(\mathcal{G})$ 
2:   for all  $v \in \mathcal{G}.V$  do
3:      $\mathcal{G}.P(v) \leftarrow \text{SIMPLIFYRULE}(\mathcal{G}.P(v))$ 
4:   end for
5: end function
6: function  $\text{SIMPLIFYRULE}(r)$ 
    $\triangleright$  Replace empty terminal string with  $\epsilon$ 
7:   if  $\text{IS TERMINAL}(r) \wedge \text{IS EMPTY}(r)$  then
8:     return  $\epsilon$ 
9:   end if
10:  if  $\text{IS OPERATOR}(r)$  then
11:    let  $C$  be  $\text{CHILDREN}(r)$ 
12:     $C \leftarrow \{\text{SIMPLIFYRULE}(c) \mid c \in C\}$ 
13:    if  $\text{IS CONCATENATE}(r)$  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  $\text{IS CONCATENATE}(r) \vee \text{IS UNION}(r)$  then
    $\triangleright$  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
```

---

---

**Algorithm 5** Merge Equivalent Productions

---

```
1: function  $\text{MERGEEQUIVPRODUCTIONS}(\mathcal{G})$ 
2:    $pairs \leftarrow \emptyset$ 
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:       $\text{COMBINEANDREPLACE}(p.left, p.right)$ 
13:    end if
14:  end for
15: end function
```

---

any non-terminal symbols referenced by a rule will not define a duplicate production. The following provides an example:

---

**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

```

---



---

**Algorithm 7** Expand Productions

---

```

1: function EXPANDPRODUCTIONS( $\mathcal{G}$ )
2:   repeat
3:      $changed \leftarrow \perp$ 
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:             CREATEANDREPLACWITH-
9:             PROD( $g$ )
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:          CREATEANDREPLACWITHPROD( $a$ )
15:           $changed \leftarrow \top$ 
16:        end for
17:      end if
18:    end for
19:  until  $changed = \perp$ 
20: end function

```

---


$$\begin{aligned}
\langle A \rangle &\rightarrow a \langle B \rangle \\
\langle B \rangle &\rightarrow b \ c \\
\langle C \rangle &\rightarrow c \mid \langle D \rangle \\
\langle D \rangle &\rightarrow d \mid e
\end{aligned}$$

would then collapse to form:

$$\begin{aligned}
\langle A \rangle &\rightarrow a \ b \ c \\
\langle C \rangle &\rightarrow c \mid d \mid e
\end{aligned}$$

## V. EXPERIMENTAL DESIGN

### A. Pilot Study

To evaluate the above approach, we performed a small pilot study on three grammars. We selected these grammars from the ANTLR grammar repository. To select these three grammars, we looked for three grammars of varying sizes and applications. The three grammars chosen were the Brainfuck, XML, Java™ grammars.

Brainfuck is an esoteric language notable for its extreme simplicity [CITE]. It is Turing-complete despite only having

---

**Algorithm 8** Collapse Productions

---

```

1: function COLLAPSEPRODUCTIONS( $\mathcal{G}$ )
2:    $\triangleright$  Split productions into form1 and form2
3:    $f_1 \leftarrow \text{COLLECT}(\text{"form1"})$ 
4:    $f_2 \leftarrow \text{COLLECT}(\text{"form2"})$ 
5:   for all  $p \in f_1$  do
6:     if ONLYTERMINALS( $p.rule$ ) then
7:       REPLACEF1USESWITHRULE( $p$ )
8:     end if
9:   end for
10:  for all  $p \in f_2$  do
11:    if ONLYTERMINALS( $p.rule$ ) then
12:      REPLACEF2USEWITHRULE( $p$ )
13:    end if
14:  end for
15: end function

```

---

8 commands. We chose this language because its grammar is extremely small which allows it to be easily inspected. XML was also chosen because of its grammar's small size. However, it is still significantly more complex than Brainfuck. XML is commonly used for sending information between applications [CITE something about SOAP] and configuration files [CITE]. Java™ is a general purpose programming language used all over the world [CITE]. Its grammar is significantly more complicated than either of the two previously mentioned grammars. We chose to include this grammar because applications with a need for multilingual parsing would likely include Java as one of their languages [CITE something about JSP, something about embedded languages in strings, something about mixed language systems (e.g. migration to SCALA or Kotlin)].

To evaluate each grammar, we normalize each grammar. Before and after normalization, we measure the number of productions. After normalization, each grammar is checked manually. In this checking process, each rule is verified to be of either Form 1 or Form 2. In addition, we examine each normalized grammar for unexpected rules.

## VI. RESULTS

### A. Data Set Reduction

How we selected grammars - brainfuck - small, simple - java - large, complex, actual application - xml or something else - slightly complex, actual application

Selected parts of grammars

Before/after - table of before and after of sizes

Effects - size differences - rule sizes/styles

## VII. INTERPRETATION

- A. *Evaluation of Results*
- B. *Limits of the Study*
- C. *Inferences*
- D. *Lessons Learned*

## VIII. THREATS TO VALIDITY

## IX. CONCLUSIONS AND FUTURE WORK

- A. *Summary of Findings*
- B. *Relation to Existing Evidence*
- C. *Impact*
- D. *Limitations*

## 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.