

4.4: Simplification of Grammars

In this section, we say what it means for a grammar to be simplified, give a simplification algorithm for grammars, and see how to use this algorithm in Forlan.

Motivating Example

Suppose G is the grammar

$$A \rightarrow BB1,$$

$$B \rightarrow 0 \mid A \mid CD,$$

$$C \rightarrow 12,$$

$$D \rightarrow 1D2.$$

Question: what is odd about this grammar?

Answer: First, D doesn't generate anything.

Second, there is no valid parse tree that starts at G 's start variable, A , has a yield that is in $\{0, 1, 2\}^* = \mathbf{alphabet\ } G$, and makes use of C .

Reachable, Generating and Useful Variables

Suppose G is a grammar. We say that a variable q of G is:

- *reachable in G* iff there is a $w \in \mathbf{Str}$ such that w is parsable from s_G using G , and $q \in \mathbf{alphabet } w$;
- *generating in G* iff there is a $w \in \mathbf{Str}$ such that q generates w using G , i.e., w is parsable from q using G , and $w \in (\mathbf{alphabet } G)^*$;
- *useful in G* iff q is both reachable and generating in G .

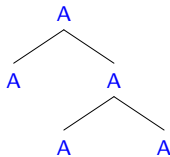
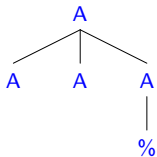
Redundant Productions

Now, suppose H is the grammar

$$A \rightarrow \% \mid 0 \mid AA \mid AAA.$$

What is odd about this grammar?

Here, the productions $A \rightarrow AA$ and $A \rightarrow AAA$ are redundant, although only one of them can be removed:



Redundant Productions

Given a grammar G and a finite subset U of $\{(q, x) \mid q \in Q_G \text{ and } x \in \mathbf{Str}\}$, we write G/U for the grammar that is identical to G except that its set of productions is U .

If G is a grammar and $(q, x) \in P_G$, we say that:

- (q, x) is *redundant* in G iff x is parsable from q using H , where $H = G/(P_G - \{(q, x)\})$; and
- (q, x) is *irredundant* in G iff (q, x) is not redundant in G .

Simplified Grammars

A grammar G is *simplified* iff either

- every variable of G is useful, and every production of G is irredundant; or
- $|Q_G| = 1$ and $P_G = \emptyset$.

Proposition 4.4.1

If G is a simplified grammar, then **alphabet** $G = \mathbf{alphabet}(L(G))$.

Simplified Grammars

Proof. Suppose $a \in \mathbf{alphabet} G$. We must show that $a \in \mathbf{alphabet} w$ for some $w \in L(G)$.

We have that every variable of G is useful, and there are $q \in Q_G$ and $x \in \mathbf{Str}$ such that $(q, x) \in P_G$ and $a \in \mathbf{alphabet} x$.

Thus x is parsable from q . Since every variable occurring in x is generating, we have that q generates a string x' containing a .

Since q is reachable, there is a string y such that y is parsable from s_G , and $q \in \mathbf{alphabet} y$. Since every variable occurring in y is generating, there is a string y' such that y' is parsable from s_G , and q is the only variable of $\mathbf{alphabet} y'$.

Putting these facts together, we have that s_G generates a string w such that $a \in \mathbf{alphabet} w$, i.e., $a \in \mathbf{alphabet} w$ for some $w \in L(G)$. \square

Algorithm for Removing Redundant Productions

Given a grammar G , $q \in Q_G$ and $x \in \mathbf{Str}$, we say that (q, x) is *implicit in G* iff x is parsable from q using G .

Given a grammar G , we define a function $\mathbf{remRedun}_G \in \mathcal{P} P_G \times \mathcal{P} P_G \rightarrow \mathcal{P} P_G$ by well-founded recursion on the size of its second argument.

For $U, V \subseteq P_G$, $\mathbf{remRedun}(U, V)$ proceeds as follows:

- If $V = \emptyset$, then it returns U .
- Otherwise, let v be the greatest element of $\{(q, x) \in V \mid \text{there are no } p \in \mathbf{Sym} \text{ and } y \in \mathbf{Str} \text{ such that } (p, y) \in V \text{ and } |y| > |x|\}$, and $V' = V - \{v\}$. If v is implicit in $G/(U \cup V')$, then $\mathbf{remRedun}$ returns the result of evaluating $\mathbf{remRedun}(U, V')$. Otherwise, it returns the result of evaluating $\mathbf{remRedun}(U \cup \{v\}, V')$.

Our algorithm for removing redundant productions of a grammar G returns $G/(\mathbf{remRedun}_G(\emptyset, P_G))$.

Algorithm for Removing Redundant Productions

For example, if we run our algorithm for removing redundant productions on

$$A \rightarrow \% \mid 0 \mid AA \mid AAA,$$

we obtain

$$A \rightarrow \% \mid 0 \mid AA.$$

Simplification Algorithm

Our simplification algorithm for grammars proceeds as follows, given a grammar G .

- First, it determines which variables of G are generating. If s_G isn't one of these variables, then it returns the grammar with variable s_G and no productions.
- Next, it turns G into a grammar G' by deleting all non-generating variables, and deleting all productions involving such variables.
- Then, it determines which variables of G' are reachable.
- Next, it turns G' into a grammar G'' by deleting all non-reachable variables, and deleting all productions involving such variables.
- Finally, it removes redundant productions from G'' .

Simplification Example

Suppose G , once again, is the grammar

$$A \rightarrow BB1,$$

$$B \rightarrow 0 \mid A \mid CD,$$

$$C \rightarrow 12,$$

$$D \rightarrow 1D2.$$

Here is what happens if we apply our simplification algorithm to G . First, we determine which variables are generating. Clearly B and C are. And, since B is, it follows that A is, because of the production $A \rightarrow BB1$. (If this production had been $A \rightarrow BD1$, we wouldn't have added A to our set.)

Simplification Example (Cont.)

Thus, we form G' from G by deleting the variable D , yielding the grammar

$$A \rightarrow BB1,$$

$$B \rightarrow 0 \mid A,$$

$$C \rightarrow 12.$$

Next, we determine which variables of G' are reachable. Clearly A is, and thus B is, because of the production $A \rightarrow BB1$.

Note that, if we carried out the two stages of our simplification algorithm in the other order, then C and its production would never be deleted.

Simplification Example (Cont.)

Next, we form G'' from G' by deleting the variable C , yielding the grammar

$$\begin{aligned}A &\rightarrow BB1, \\ B &\rightarrow 0 \mid A.\end{aligned}$$

Finally, we would remove redundant productions from G'' . But G'' has no redundant productions, and so we are done.

Simplification Function

We define a function **simplify** \in **Gram** \rightarrow **Gram** by: for all $G \in$ **Gram**, **simplify** G is the result of running the above algorithm on G .

Theorem 4.4.2

For all $G \in$ **Gram**:

- (1) **simplify** G is simplified;
- (2) **simplify** $G \approx G$; and
- (3) **alphabet**(**simplify** G) = **alphabet**($L(G)$) \subseteq **alphabet** G .

Testing Whether $L(G) = \emptyset$

Our simplification algorithm gives us an algorithm for testing whether the language generated by a grammar G is empty. We first simplify G , calling the result H . We then test whether $P_H = \emptyset$. If the answer is “yes”, clearly $L(G) = L(H) = \emptyset$. And if the answer is “no”, then s_H is useful, and so H (and thus G) generates at least one string.

Simplification in Forlan

The Forlan module `Gram` defines the functions

```
val simplify    : gram -> gram  
val simplified  : gram -> bool
```


Forlan Examples

Suppose `gram` of type `gram` is bound to the grammar

$$\begin{aligned}A &\rightarrow BB1, \\ B &\rightarrow 0 \mid A \mid CD, \\ C &\rightarrow 12, \\ D &\rightarrow 1D2.\end{aligned}$$

We can simplify our grammar as follows:

```
- val gram' = Gram.simplify gram;
val gram' = - : gram
- Gram.output("", gram');
{variables} A, B {start variable} A
{productions} A -> BB1; B -> 0 | A
val it = () : unit
```

Forlan Examples

Suppose `gram''` of type `gram` is bound to the grammar

$$A \rightarrow \% \mid 0 \mid AA \mid AAA \mid AAAA.$$

We can simplify our grammar as follows:

```
- val gram''' = Gram.simplify gram'';  
val gram''' = - : gram  
- Gram.output("", gram''');  
{variables} A {start variable} A  
{productions} A -> \% \mid 0 \mid AA  
val it = () : unit
```

Hand-simplification Operations

Given a simplified grammar G , there are often ways we can hand-simplify the grammar further. Below are two examples.

Suppose G has a variable q that is not s_G , and where no production having q as its left-hand side is *self-recursive*, i.e., has q as one of the symbols of its right-hand side. Let x_1, \dots, x_n be the right-hand sides of all of q 's productions.

Then we can form an equivalent grammar G' by deleting q and its productions from G , and transforming each remaining production $p \rightarrow y$ of G into all the productions from p that can be formed by substituting for each occurrence of q in y some choice of x_i .

We refer to this operation as *eliminating q from G* .

Hand-simplification Operations

Suppose there is exactly one production of G involving s_G , where that production has the form $s_G \rightarrow q$, for some variable q of G .

Then we can form an equivalent grammar G' by deleting s_G and $s_G \rightarrow q$ from G , and making q be the start variable of G' .

We refer to this operation as *restarting* G .

Hand-simplification Operations

The Forlan module `Gram` has functions corresponding to these two operations:

```
val eliminateVariable : gram * sym -> gram
val restart           : gram -> gram
```

Both begin by simplifying the supplied grammar.

For instance, suppose `gram` is the grammar

$$A \rightarrow B,$$
$$B \rightarrow 0 \mid C3C,$$
$$C \rightarrow 1B2 \mid 2B1.$$

Hand-simplification Operations

Then we can proceed as follows:

```
- val gram' =  
=      Gram.eliminateVariable  
=      (gram, Sym.fromString "C");  
val gram' = - : gram  
- Gram.output("", gram');  
{variables} A, B {start variable} A  
{productions}  
A -> B; B -> 0 | 1B231B2 | 1B232B1 | 2B131B2 | 2B132B1  
val it = () : unit  
- val gram'' = Gram.restart gram;  
val gram'' = - : gram  
- Gram.output("", gram'');  
{variables} B, C {start variable} B  
{productions} B -> 0 | C3C; C -> 1B2 | 2B1  
val it = () : unit
```