

## Section 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.

---

Copyright © 2003–4 Alley Stoughton

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

The  $\text{\LaTeX}$  source of these slides, the associated book, and the distribution of the Forlan toolset are available on the WWW at <http://people.cis.ksu.edu/~stough/forlan/>.

## (4.4) Introduction (Cont.)

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:

## (4.4) Introduction (Cont.)

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, there isn't a valid parse tree for  $G$  that starts at  $D$  and whose yield is in  $\text{alphabet}(G)^* = \{0, 1, 2\}^*$ .

## (4.4) Introduction (Cont.)

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, there isn't a valid parse tree for  $G$  that starts at  $D$  and whose yield is in  $\text{alphabet}(G)^* = \{0, 1, 2\}^*$ .

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

As a result, we will say that both  $C$  and  $D$  are “useless”.

## (4.4) Reachable, Generating and Useful Variables

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

- *reachable* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = s_G$  and  $q$  is one of the leaves of  $pt$ ;

## (4.4) Reachable, Generating and Useful Variables

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

- *reachable* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = s_G$  and  $q$  is one of the leaves of  $pt$ ;
- *generating* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = q$  and  $\mathbf{yield}(pt) \in \mathbf{alphabet}(G)^*$ ;

## (4.4) Reachable, Generating and Useful Variables

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

- *reachable* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = s_G$  and  $q$  is one of the leaves of  $pt$ ;
- *generating* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = q$  and  $\mathbf{yield}(pt) \in \mathbf{alphabet}(G)^*$ ;
- *useful* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = s_G$ ,  $\mathbf{yield}(pt) \in \mathbf{alphabet}(G)^*$ , and  $q$  appears in  $pt$ .

## (4.4) Reachable, Generating and Useful Variables

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

- *reachable* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = s_G$  and  $q$  is one of the leaves of  $pt$ ;
- *generating* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = q$  and  $\mathbf{yield}(pt) \in \mathbf{alphabet}(G)^*$ ;
- *useful* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = s_G$ ,  $\mathbf{yield}(pt) \in \mathbf{alphabet}(G)^*$ , and  $q$  appears in  $pt$ .

Thus every useful variable is both reachable and generating, but the converse is false. For example, the variable  $\epsilon$  of our example grammar is reachable and generating, but isn't useful.

## (4.4) Reachable, Generating and Useful Variables

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

- *reachable* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = s_G$  and  $q$  is one of the leaves of  $pt$ ;
- *generating* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = q$  and  $\mathbf{yield}(pt) \in \mathbf{alphabet}(G)^*$ ;
- *useful* iff there is a parse tree  $pt$  such that  $pt$  is valid for  $G$ ,  $\mathbf{rootLabel}(pt) = s_G$ ,  $\mathbf{yield}(pt) \in \mathbf{alphabet}(G)^*$ , and  $q$  appears in  $pt$ .

Thus every useful variable is both reachable and generating, but the converse is false. For example, the variable  $C$  of our example grammar is reachable and generating, but isn't useful.

## (4.4) Simplified Grammars

A grammar  $G$  is *simplified* iff either

- all of  $G$ 's variables are useful; or
- $G$  has a single variable and no productions.

E.g., the grammar with variable  $A$ , start variable  $A$  and no productions is simplified, even though  $A$  is useless. Of course, this grammar generates the empty language.

## (4.4) A Simplification Algorithm

To simplify a grammar  $G$ , we proceed as follows.

- First, we determine which variables of  $G$  are generating. If  $s_G$  isn't one of these variables, then we return the grammar with variable  $s_G$  and no productions.

## (4.4) A Simplification Algorithm

To simplify a grammar  $G$ , we proceed as follows.

- First, we determine which variables of  $G$  are generating. If  $s_G$  isn't one of these variables, then we return the grammar with variable  $s_G$  and no productions.
- Next, we turn  $G$  into a grammar  $G'$  by deleting all non-generating variables, and deleting all productions involving such variables.

## (4.4) A Simplification Algorithm

To simplify a grammar  $G$ , we proceed as follows.

- First, we determine which variables of  $G$  are generating. If  $s_G$  isn't one of these variables, then we return the grammar with variable  $s_G$  and no productions.
- Next, we turn  $G$  into a grammar  $G'$  by deleting all non-generating variables, and deleting all productions involving such variables.
- Then, we determine which variables of  $G'$  are reachable.

## (4.4) A Simplification Algorithm

To simplify a grammar  $G$ , we proceed as follows.

- First, we determine which variables of  $G$  are generating. If  $s_G$  isn't one of these variables, then we return the grammar with variable  $s_G$  and no productions.
- Next, we turn  $G$  into a grammar  $G'$  by deleting all non-generating variables, and deleting all productions involving such variables.
- Then, we determine which variables of  $G'$  are reachable.
- Finally, we turn  $G'$  into a grammar  $G''$  by deleting all non-reachable variables, and deleting all productions involving such variables.

## (4.4) 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.

## (4.4) 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.)

## (4.4) 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.

## (4.4) 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  $A \rightarrow BB1$ .

## (4.4) 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.

## (4.4) Simplification Example (Cont.)

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

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

## (4.4) Simplification Function

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

### Theorem 4.4.1

For all  $G \in \mathbf{Gram}$ :

- (1)  $\mathbf{simplify}(G)$  is simplified;
- (2)  $\mathbf{simplify}(G) \approx G$ ;
- (3)  $\mathbf{alphabet}(\mathbf{simplify}(G)) \subseteq \mathbf{alphabet}(G)$ .

## (4.4) Simplification in Forlan

The Forlan module `Gram` defines the function

```
val simplify : gram -> gram
```

for simplifying grammars.

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

$$A \rightarrow BB1,$$
$$B \rightarrow 0 \mid A \mid CD,$$
$$C \rightarrow 12,$$
$$D \rightarrow 1D2.$$

## (4.4) Forlan Example

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
```