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

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

Question: what is odd about this grammar?

Answer:

Motivating Example

Suppose G is the grammar

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

Question: what is odd about this grammar?

Answer: First, D doesn't generate anything.

Motivating Example

Suppose G is the grammar

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

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 $\{0, 1, 2\}^* = \textbf{alphabet}\, G$, 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 $q \in \mathbf{alphabet}\, w$;

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

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

Redundant Productions

Now, suppose H is the grammar

$$\mathsf{A} \to \% \mid 0 \mid \mathsf{AA} \mid \mathsf{AAA}.$$

What is odd about this grammar?

Redundant Productions

Now, suppose H is the grammar

$$\mathsf{A} \to \% \mid 0 \mid \mathsf{AA} \mid \mathsf{AAA}.$$

What is odd about this grammar?

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

Redundant Productions

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

Redundant Productions

Given a grammar G and a finite subset U of $\{\, (q, x) \mid q \in Q_G$ 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

Redundant Productions

Given a grammar G and a finite subset U of $\{\, (q, x) \mid q \in Q_G$ and $x \in \textbf{Str}\,\}$, 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 $P_G = \emptyset$.

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 4.4.2

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

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

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

□

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

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

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

□

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

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

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 $q \in \textbf{alphabet}\, y$. 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 $\textbf{alphabet}\, y'$.

□

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

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

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 $q \in \textbf{alphabet}\, y$. 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 $\textbf{alphabet}\, y'$.

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

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.

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 $\mathbf{remRedun}_G \in \mathcal{P}\, P_G \times \mathcal{P}\, P_G \to \mathcal{P}\, P_G$ by well-founded recursion on the size of its second argument.

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

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 $\mathbf{remRedun}_G \in \mathcal{P}\,P_G \times \mathcal{P}\,P_G \to \mathcal{P}\,P_G$ 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.

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 $\mathbf{remRedun}_G \in \mathcal{P}\, P_G \times \mathcal{P}\, P_G \to \mathcal{P}\, P_G$ 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 $\{\, (q, x) \in V \mid$ there are no $p \in \mathbf{Sym}$ and $y \in \mathbf{Str}$ such that $(p, y) \in V$ and $|y| > |x| \,\}$, and $V' = V - \{v\}$.

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 $\mathbf{remRedun}_G \in \mathcal{P}\, P_G \times \mathcal{P}\, P_G \to \mathcal{P}\, P_G$ 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 $\{\, (q, x) \in V \mid$ there are no $p \in \mathbf{Sym}$ and $y \in \mathbf{Str}$ such that $(p, y) \in V$ and $|y| > |x| \,\}$, and $V' = V - \{v\}$. If v is implicit in $G/(U \cup V')$, then **remRedun** returns the result of evaluating $\mathbf{remRedun}(U, V')$.

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 $\mathbf{remRedun}_G \in \mathcal{P}\, P_G \times \mathcal{P}\, P_G \to \mathcal{P}\, P_G$ 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 $\{\, (q, x) \in V \mid$ there are no $p \in \mathbf{Sym}$ and $y \in \mathbf{Str}$ such that $(p, y) \in V$ and $|y| > |x| \,\}$, and $V' = V - \{v\}$. If v is implicit in $G/(U \cup V')$, then **remRedun** returns the result of evaluating $\mathbf{remRedun}(U, V')$. Otherwise, it returns the result of evaluating $\mathbf{remRedun}(U \cup \{v\}, V')$.

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 $\mathbf{remRedun}_G \in \mathcal{P}\, P_G \times \mathcal{P}\, P_G \to \mathcal{P}\, P_G$ 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 $\{\, (q, x) \in V \mid$ there are no $p \in \mathbf{Sym}$ and $y \in \mathbf{Str}$ such that $(p, y) \in V$ and $|y| > |x|\,\}$, and $V' = V - \{v\}$. If v is implicit in $G/(U \cup V')$, then **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

$$\mathsf{A} \to \% \mid 0 \mid \mathsf{AA} \mid \mathsf{AAA},$$

we obtain

$$\mathsf{A} \to \% \mid 0 \mid \mathsf{AA}.$$

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.

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.

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.

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.

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

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

Here is what happens if we apply our simplification algorithm to G.

First, we determine which variables are generating.

Simplification Example

Suppose G, once again, is the grammar

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

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 $A \to BB1$. (If this production had been $A \to BD1$, 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

$$\begin{aligned} \mathsf{A} &\to \mathsf{BB1}, \\ \mathsf{B} &\to \mathsf{0} \mid \mathsf{A}, \\ \mathsf{C} &\to \mathsf{12}. \end{aligned}$$

Next, we determine which variables of G' are reachable.

Simplification Example (Cont.)

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

$$\begin{aligned} \mathsf{A} &\to \mathsf{BB1}, \\ \mathsf{B} &\to \mathsf{0} \mid \mathsf{A}, \\ \mathsf{C} &\to \mathsf{12}. \end{aligned}$$

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

Simplification Example (Cont.)

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

$$\begin{aligned} \mathsf{A} &\rightarrow \mathsf{BB1}, \\ \mathsf{B} &\rightarrow \mathsf{0} \mid \mathsf{A}, \\ \mathsf{C} &\rightarrow \mathsf{12}. \end{aligned}$$

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

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

$$A \to BB1,$$
$$B \to 0 \mid A.$$

Simplification Example (Cont.)

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

$$\mathrm{A} \rightarrow \mathrm{BB1},$$
$$\mathrm{B} \rightarrow 0 \mid \mathrm{A}.$$

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** $\to$ **Gram** by: for all $G \in$ **Gram**, **simplify** G is the result of running the above algorithm on G.

Theorem 4.4.3

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.

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

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

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

$$\mathsf{A} \rightarrow \% \mid 0 \mid \mathsf{AA} \mid \mathsf{AAA} \mid \mathsf{AAAA}.$$

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 -> % | 0 | 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 that appears on the left side of exactly one production: $q \rightarrow x$. Because G is simplified, it follows that q does not occur in x (or q would not be generating).

Then we can form an equivalent grammar G' by deleting q and $q \rightarrow x$ from G, and transforming each remaining production $p \rightarrow y$ into $p \rightarrow z$, where z is the result of replacing each occurrence of q in y by x.

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 $s_G \to q$, for some variable q of G.

Then we can form an equivalent grammar G' by deleting s_G and $s_G \to q$ 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

$$\mathsf{A} \rightarrow \mathsf{B},$$
$$\mathsf{B} \rightarrow \mathsf{0} \mid \mathsf{C1C},$$
$$\mathsf{C} \rightarrow \mathsf{1B2}.$$

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 | 1B211B2
val it = () : unit
- val gram'' = Gram.restart gram;
val gram'' = - : gram
- Gram.output("", gram'');
{variables} B, C {start variable} B
{productions} B -> 0 | C1C; C -> 1B2
val it = () : unit
```