4.3: A Parsing Algorithm

In this section, we consider a simple, fairly inefficient parsing algorithm that works for all context-free grammars. In Section 4.6, we consider an efficient parsing method that works for grammars for languages of operators of varying precedences and associativities. Compilers courses cover efficient algorithms that work for various subsets of the context free grammars.

Parsability

Suppose G is a grammar, $w \in \mathbf{Str}$ and $a \in \mathbf{Sym}$. We consider an algorithm for testing whether w is parsable from a using G.

Parsability

Suppose G is a grammar, $w \in \textbf{Str}$ and $a \in \textbf{Sym}$. We consider an algorithm for testing whether w is parsable from a using G.

If $w \notin (Q_G \cup \textbf{alphabet}\, G)^*$ or $a \notin Q_G \cup \textbf{alphabet}\, w$, then the algorithm returns **false**.

Otherwise, it proceeds as follows.

Parsability

Let $A = Q_G \cup \mathbf{alphabet}\, w$ and $B = \{\, x \in \mathbf{Str} \mid x$ is a substring of $w \,\}$.

The algorithm generates the least subset X of $A \times B$ such that:

(1) For all $a \in \mathbf{alphabet}\, w$, $(a, a) \in X$;

Since $A \times B$ is finite, this process terminates.

Parsability

Let $A = Q_G \cup \textbf{alphabet}\, w$ and $B = \{\, x \in \textbf{Str} \mid x \text{ is a substring of } w \,\}$.

The algorithm generates the least subset X of $A \times B$ such that:

(1) For all $a \in \textbf{alphabet}\, w$, $(a, a) \in X$;

(2) For all $q \in Q_G$, if $q \to \% \in P_G$, then $(q, \%) \in X$; and

Since $A \times B$ is finite, this process terminates.

Parsability

Let $A = Q_G \cup \textbf{alphabet}\, w$ and $B = \{\, x \in \textbf{Str} \mid x \text{ is a substring of } w \,\}$.

The algorithm generates the least subset X of $A \times B$ such that:

(1) For all $a \in \textbf{alphabet}\, w$, $(a, a) \in X$;

(2) For all $q \in Q_G$, if $q \rightarrow \% \in P_G$, then $(q, \%) \in X$; and

(3) For all $q \in Q_G$, $n \in \mathbb{N} - \{0\}$, $a_1, \ldots, a_n \in A$ and $x_1, \ldots, x_n \in B$, if

- $q \rightarrow a_1 \cdots a_n \in P_G$,
- for all $i \in [1 : n]$, $(a_i, x_i) \in X$, and
- $x_1 \cdots x_n \in B$,

then $(q, x_1 \cdots x_n) \in X$.

Since $A \times B$ is finite, this process terminates.

Parsability

For example, let G be the grammar

$$\begin{aligned} \mathsf{A} &\to \mathsf{BC} \mid \mathsf{CD}, \\ \mathsf{B} &\to 0 \mid \mathsf{CB}, \\ \mathsf{C} &\to 1 \mid \mathsf{DD}, \\ \mathsf{D} &\to 0 \mid \mathsf{BC}, \end{aligned}$$

and let $w = 0010$ and $a = \mathsf{A} = s_G$.

We have that:

- $(0, 0) \in X$;
- $(1, 1) \in X$;

Parsability

For example, let G be the grammar

$$\begin{aligned} \mathsf{A} &\to \mathsf{BC} \mid \mathsf{CD}, \\ \mathsf{B} &\to 0 \mid \mathsf{CB}, \\ \mathsf{C} &\to 1 \mid \mathsf{DD}, \\ \mathsf{D} &\to 0 \mid \mathsf{BC}, \end{aligned}$$

and let $w = 0010$ and $a = \mathsf{A} = s_G$.

We have that:

- $(0, 0) \in X$;
- $(1, 1) \in X$;
- $(\mathsf{B}, 0) \in X$, since $\mathsf{B} \to 0 \in P_G$, $(0, 0) \in X$ and $0 \in B$;
- $(\mathsf{C}, 1) \in X$, since $\mathsf{C} \to 1 \in P_G$, $(1, 1) \in X$ and $1 \in B$;
- $(\mathsf{D}, 0) \in X$, since $\mathsf{D} \to 0 \in P_G$, $(0, 0) \in X$ and $0 \in B$;

Parsability

- $(\mathsf{A}, 01) \in X$, since $\mathsf{A} \to \mathsf{BC} \in P_G$, $(\mathsf{B}, 0) \in X$, $(\mathsf{C}, 1) \in X$ and $01 \in B$;
- $(\mathsf{A}, 10) \in X$, since $\mathsf{A} \to \mathsf{CD} \in P_G$, $(\mathsf{C}, 1) \in X$, $(\mathsf{D}, 0) \in X$ and $10 \in B$;
- $(\mathsf{B}, 10) \in X$, since $\mathsf{B} \to \mathsf{CB} \in P_G$, $(\mathsf{C}, 1) \in X$, $(\mathsf{B}, 0) \in X$ and $10 \in B$;
- $(\mathsf{C}, 00) \in X$, since $\mathsf{C} \to \mathsf{DD} \in P_G$, $(\mathsf{D}, 0) \in X$, $(\mathsf{D}, 0) \in X$ and $00 \in B$;
- $(\mathsf{D}, 01) \in X$, since $\mathsf{D} \to \mathsf{BC} \in P_G$, $(\mathsf{B}, 0) \in X$, $(\mathsf{C}, 1) \in X$ and $01 \in B$;

Parsability

- $(\mathsf{C}, 001) \in X$, since $\mathsf{C} \to \mathsf{DD} \in P_G$, $(\mathsf{D}, 0) \in X$, $(\mathsf{D}, 01) \in X$ and $0(01) \in B$;
- $(\mathsf{C}, 010) \in X$, since $\mathsf{C} \to \mathsf{DD} \in P_G$, $(\mathsf{D}, 01) \in X$, $(\mathsf{D}, 0) \in X$ and $(01)0 \in B$;
- $(\mathsf{A}, 0010) \in X$, since $\mathsf{A} \to \mathsf{BC} \in P_G$, $(\mathsf{B}, 0) \in X$, $(\mathsf{C}, 010) \in X$ and $0(010) \in B$;
- $(\mathsf{B}, 0010) \in X$, since $\mathsf{B} \to \mathsf{CB} \in P_G$, $(\mathsf{C}, 00) \in X$, $(\mathsf{B}, 10) \in X$ and $(00)(10) \in B$;
- $(\mathsf{D}, 0010) \in X$, since $\mathsf{D} \to \mathsf{BC} \in P_G$, $(\mathsf{B}, 0) \in X$, $(\mathsf{C}, 010) \in X$ and $0(010) \in B$;
- Nothing more can be added to X. To verify this, one must check that nothing new can be added to X using rule (3).

Parsability

Lemma 4.3.1
For all $(b, x) \in X$, there is a $pt \in \mathbf{PT}$ such that

- *pt is valid for G,*
- *$\mathbf{rootLabel}\, pt = b$, and*
- *$\mathbf{yield}\, pt = x$.*

Lemma 4.3.2
For all $pt \in \mathbf{PT}$, if

- *pt is valid for G,*
- *$\mathbf{rootLabel}\, pt \in A$, and*
- *$\mathbf{yield}\, pt \in B$,*

then $(\mathbf{rootLabel}\, pt, \mathbf{yield}\, pt) \in X$.

Parsability

Thus, to determine if w is parsable from a, we just have to check whether

Parsability

Thus, to determine if w is parsable from a, we just have to check whether $(a, w) \in X$.

Parsability

Thus, to determine if w is parsable from a, we just have to check whether $(a, w) \in X$.

In the case of our example grammar, we have that $w = 0010$ is parsable from $a = \mathsf{A}$, since $(\mathsf{A}, 0010) \in X$.

Parsability

Thus, to determine if w is parsable from a, we just have to check whether $(a, w) \in X$.

In the case of our example grammar, we have that $w = 0010$ is parsable from $a = \mathsf{A}$, since $(\mathsf{A}, 0010) \in X$.

Hence $0010 \in L(G)$.

Parsability

Thus, to determine if w is parsable from a, we just have to check whether $(a, w) \in X$.

In the case of our example grammar, we have that $w = 0010$ is parsable from $a = \mathrm{A}$, since $(\mathrm{A}, 0010) \in X$.

Hence $0010 \in L(G)$.

Note that any production whose right-hand side contains an element of $\textbf{alphabet}\, G - \textbf{alphabet}\, w$ won't affect the generation of X. Thus our algorithm ignores such productions.

Parsability

For efficiency, our parsability algorithm actually generates X in a sequence of stages. At each point, it has subsets U and V of $A \times B$.

- First, it lets $U = \emptyset$ and sets V to be the union of $\{\, (a, a) \mid a \in \textbf{alphabet}\, w \,\}$ and $\{\, (q, \%) \mid (q, \%) \in P_G \,\}$. It then enters its main loop.

Parsability

For efficiency, our parsability algorithm actually generates X in a sequence of stages. At each point, it has subsets U and V of $A \times B$.

- First, it lets $U = \emptyset$ and sets V to be the union of $\{\, (a, a) \mid a \in \textbf{alphabet}\, w \,\}$ and $\{\, (q, \%) \mid (q, \%) \in P_G \,\}$. It then enters its main loop.
- At a stage of the loop's iteration, it lets Y be $U \cup V$, and then lets Z be the set of all $(q, x_1 \cdots x_n)$ such that $n \geq 1$ and there are $a_1, \ldots, a_n \in A$ and $i \in [1 : n]$ such that
    - $q \rightarrow a_1 \cdots a_n \in P_G$,
    - $(a_i, x_i) \in V$,
    - for all $k \in [1 : n] - \{i\}$, $(a_k, x_k) \in Y$,
    - $x_1 \cdots x_n \in B$, and
    - $(q, x_1 \cdots x_n) \notin Y$.

Parsability

For efficiency, our parsability algorithm actually generates X in a sequence of stages. At each point, it has subsets U and V of $A \times B$.

- First, it lets $U = \emptyset$ and sets V to be the union of $\{\,(a, a) \mid a \in \textbf{alphabet}\, w\,\}$ and $\{\,(q, \%) \mid (q, \%) \in P_G\,\}$. It then enters its main loop.
- At a stage of the loop's iteration, it lets Y be $U \cup V$, and then lets Z be the set of all $(q, x_1 \cdots x_n)$ such that $n \geq 1$ and there are $a_1, \ldots, a_n \in A$ and $i \in [1 : n]$ such that
  - $q \to a_1 \cdots a_n \in P_G$,
  - $(a_i, x_i) \in V$,
  - for all $k \in [1 : n] - \{i\}$, $(a_k, x_k) \in Y$,
  - $x_1 \cdots x_n \in B$, and
  - $(q, x_1 \cdots x_n) \notin Y$.

  If $Z \neq \emptyset$, then it sets U to Y, and V to Z, and repeats; Otherwise, the result is Y.

Parsing Algorithm

We say that a parse tree pt is *a minimal parse* of a string w *from* a symbol a *using* a grammar G iff pt is valid for G, **rootLabel** $pt = a$ and **yield** $pt = w$, and there is no strictly smaller $pt' \in \mathbf{PT}$ such that pt' is valid for G, **rootLabel** $pt' = a$ and **yield** $pt' = w$.

Parsing Algorithm

We say that a parse tree pt is *a minimal parse* of a string w *from* a symbol a *using* a grammar G iff pt is valid for G, $\textbf{rootLabel}\, pt = a$ and $\textbf{yield}\, pt = w$, and there is no strictly smaller $pt' \in \textbf{PT}$ such that pt' is valid for G, $\textbf{rootLabel}\, pt' = a$ and $\textbf{yield}\, pt' = w$.

We can convert our parsability algorithm into a parsing algorithm as follows. Given $w \in (Q_G \cup \textbf{alphabet}\, G)^*$ and $a \in (Q_G \cup \textbf{alphabet}\, w)$, we generate our set X as before, but we annotate each element (b, x) of X with a parse tree pt such that

- pt is valid for G,
- $\textbf{rootLabel}\, pt = b$, and
- $\textbf{yield}\, pt = x$,

Thus we can return the parse tree labeling (a, w), if this pair is in X, and indicate failure otherwise.

Parsing Algorithm

With a little more work, we can arrange that the parse trees returned by our parsing algorithm are minimally-sized, and this is what the official version of our parsing algorithm guarantees.

This goal is a little tricky to achieve, since some pairs will first be labeled by parse trees that aren't minimally sized.

But we keep going as long as either new pairs are found, or smaller parse trees are found for existing pairs.

Parsing in Forlan

The Forlan module `Gram` defines the functions

```
val parsable                 : gram -> sym * str -> bool
val generatedFromVariable : gram -> sym * str -> bool
val generated                : gram -> str -> bool
```

The function `parsable` tests whether a string w is parsable from a symbol a using a grammar G. The function `generatedFromVariable` tests whether a string w is generated from a variable q using a grammar G; it issues an error message if q isn't a variable of G. And the function `generated` tests whether a string w is generated by a grammar G.

Parsing in Forlan

`Gram` also includes:

```
val parse                        : gram -> sym * str -> pt
val parseAlphabetFromVariable : gram -> sym * str -> pt
val parseAlphabet                : gram -> str -> pt
```

The function `parse` tries to find a minimal parse of a string w from a symbol a using a grammar G; it issues an error message if $w \notin (Q_G \cup \textbf{alphabet}\, G)^*$, or $a \notin Q_G \cup \textbf{alphabet}\, w$, or such a parse doesn't exist. The function `parseAlphabetFromVariable` tries to find a minimal parse of a string $w \in (\textbf{alphabet}\, G)^*$ from a variable q using a grammar G; it issues an error message if $q \notin Q_G$, or $w \notin (\textbf{alphabet}\, G)^*$, or such a parse doesn't exist. And the function `parseAlphabet` tries to find a minimal parse of a string $w \in (\textbf{alphabet}\, G)^*$ from s_G using a grammar G; it issues an error message if $w \notin (\textbf{alphabet}\, G)^*$, or such a parse doesn't exist.

Parsing in Forlan

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

$$
\begin{aligned}
\mathsf{A} &\to \mathsf{BC} \mid \mathsf{CD}, \\
\mathsf{B} &\to 0 \mid \mathsf{CB}, \\
\mathsf{C} &\to 1 \mid \mathsf{DD}, \\
\mathsf{D} &\to 0 \mid \mathsf{BC}.
\end{aligned}
$$

We can check whether some strings are generated by this grammar as follows:

```
- Gram.generated gram (Str.fromString "0010");
val it = true : bool
- Gram.generated gram (Str.fromString "0100");
val it = true : bool
- Gram.generated gram (Str.fromString "0101");
val it = false : bool
```

Forlan Parsing Examples

And we can try to find parses of some strings as follows:

```
- fun test s =
=       PT.output
=       ("",
=        Gram.parseAlphabet gram (Str.fromString s));
val test = fn : string -> unit
- test "0010";
A(C(D(0), D(B(0), C(1))), D(0))
val it = () : unit
- test "0100";
A(C(D(B(0), C(1)), D(0)), D(0))
val it = () : unit
- test "0101";
no such parse exists

uncaught exception Error
```

Forlan Parsing Examples

But we can also check parsability of strings containing variables, as well as try to find parses of such strings:

```
- Gram.parsable gram
= (Sym.fromString "A", Str.fromString "0D0C");
val it = true : bool
- PT.output
= ("",
=  Gram.parse gram
=  (Sym.fromString "A", Str.fromString "0D0C"));
A(C(D(0), D), D(B(0), C))
val it = () : unit
```