## Eliminating left-recursion: three steps

Recall: A CFG is left-recursive if it includes a variable $A$ s.t.

$$A \stackrel{+}{\Rightarrow} A\alpha\,.$$

We eliminate left-recursion in three steps.

- eliminate $\epsilon$-productions (impossible to generate $\epsilon$!)
- eliminate cycles ($A \stackrel{+}{\Rightarrow} A$)
- eliminate left-recursion

So we've got some constructions to learn.

Let's try an example of eliminating $\epsilon$-productions before we specify a construction…



Consider the CFG below.

$$\begin{aligned} S &\rightarrow XX \mid Y \\ X &\rightarrow aXb \mid \epsilon \\ Y &\rightarrow aYb \mid Z \\ Z &\rightarrow bZa \mid \epsilon \end{aligned}$$

Notice that

$$S \stackrel{*}{\Rightarrow} \epsilon \qquad X \stackrel{*}{\Rightarrow} \epsilon \qquad Y \stackrel{*}{\Rightarrow} \epsilon \qquad Z \stackrel{*}{\Rightarrow} \epsilon$$

Hence, all the variables in this grammar are what we will call "nullable." So in order to eliminate the $\epsilon$-productions in this grammar, we must alter the grammar to take into account the fact that instances of these variables in a derivation may eventually be replaced by $\epsilon$. So, for instance, we will replace

$$Z \rightarrow bZa \mid \epsilon$$

with

$$Z \rightarrow bZa \mid ba\,.$$

After elimination of $\epsilon$-productions, we obtain

$$\begin{aligned} S &\rightarrow XX \mid X \mid Y \\ X &\rightarrow aXb \mid ab \\ Y &\rightarrow aYb \mid ab \mid Z \\ Z &\rightarrow bZa \mid ba \end{aligned}$$

## Eliminating $\epsilon$-productions

Given a CFG $G = (V, \Sigma, S, P)$, a variable $A \in V$ is *nullable* if

$$A \stackrel{*}{\Rightarrow} \epsilon\,.$$

The main step in the $\epsilon$-production elimination algorithm then is that the set $P$ of productions is replaced with the set $P_\epsilon$ of all productions

$$A \rightarrow \beta$$

s.t. $A \neq \beta$, $\beta \neq \epsilon$, and $P$ includes a production

$$A \rightarrow \alpha$$

s.t. $\beta$ can be obtained from $\alpha$ by deleting zero or more occurrences of nullable variables.

**Example** Applying $\epsilon$-production elimination to the CFG

$$G = (\{S\}, \{a, b\}, S, \{\ S \rightarrow aSb \mid \epsilon\ \})$$

yields the CFG

$$G_\epsilon = (\{S\}, \{a, b\}, S, \{\ S \rightarrow aSb \mid ab\ \})\,.$$



$A$ is *nullable* if $A \stackrel{*}{\Rightarrow} \epsilon$.

$P_\epsilon$ is the set of all productions

$$A \rightarrow \beta$$

s.t. $A \neq \beta$, $\beta \neq \epsilon$, and $P$ includes a production

$$A \rightarrow \alpha$$

s.t. $\beta$ can be obtained from $\alpha$ by deleting zero or more occurrences of nullable variables.

**Example** Let's apply $\epsilon$-production elimination to

$$\begin{array}{rcl} S & \rightarrow & XZ \\ X & \rightarrow & aXb \mid \epsilon \\ Z & \rightarrow & aZ \mid ZX \mid \epsilon \end{array}$$

What are the nullable variables?

What are the new productions?

Eliminating $\epsilon$-productions can greatly increase the size of a grammar.

**Example** Eliminating $\epsilon$-productions from

$$\begin{array}{rcl} S & \rightarrow & A_1 A_2 \cdots A_n \\ A_1 & \rightarrow & \epsilon \\ A_2 & \rightarrow & \epsilon \\ & \vdots & \\ A_n & \rightarrow & \epsilon \end{array}$$

increases the number of productions from $n+1$ to $2^n - 1$.

What about ambiguity?

**Claim** If $G$ is unambiguous, so is $G_\epsilon$.



## Eliminating cycles

A grammar has a *cycle* if there is a variable $A$ s.t.

$$A \stackrel{+}{\Rightarrow} A\,.$$

We'll call such variables *cyclic*.

If a grammar has no $\epsilon$-productions, then all cycles can be eliminated from $G$ without affecting the language generated, using a construction we will specify in a moment.

Let's try an example first. Consider the grammar with productions

$$\begin{array}{rcl} S & \rightarrow & X \mid Xb \mid SS \\ X & \rightarrow & S \mid a \end{array}$$

We have $S \stackrel{+}{\Rightarrow} S$ and $X \stackrel{+}{\Rightarrow} X$.

We can eliminate occurrences of cyclic variables as rhs's of productions, thus eliminating cycles.

$$\begin{array}{rcl} S & \rightarrow & a \mid Xb \mid SS \\ X & \rightarrow & Xb \mid SS \mid a \end{array}$$

Given a CFG $G = (V, \Sigma, S, P)$, the main step in the cycle elimination algorithm is to replace the set $P$ of productions with the set $P_c$ of productions obtained from $P$ by replacing

- each production $A \rightarrow B$ where $B$ is cyclic
- with new productions $A \rightarrow \alpha$ s.t. $\alpha$ is not a cyclic variable and there is a production $C \rightarrow \alpha$ s.t. $B \overset{*}{\Rightarrow} C$.

The resulting CFG is $G_c = (V, \Sigma, S, P_c)$.

Note: You can probably convince yourself that $P_c$ has no cycles, and that $G_c$ generates the same languages as $G$.

Let's try the construction...

$$\begin{aligned} S &\rightarrow X \mid Xb \mid Ya \\ X &\rightarrow Y \mid b \\ Y &\rightarrow X \mid a \end{aligned}$$

Which variables are cyclic?

Which productions will be replaced?

With what?



Replace the set $P$ of productions with the set $P_c$ of productions obtained from $P$ by replacing

- each production $A \rightarrow B$ where $B$ is cyclic
- with new productions $A \rightarrow \alpha$ s.t. $\alpha$ is not a cyclic variable and there is a production $C \rightarrow \alpha$ s.t. $B \overset{*}{\Rightarrow} C$.

---

Let's consider an example illustrating the importance of requiring that there be no $\epsilon$-productions.

$$S \rightarrow a \mid SS \mid \epsilon$$

Notice that $S$ is a cyclic variable, since

$$S \Rightarrow SS \Rightarrow S\,.$$

But $S$ never appears as the rhs of a production, so the construction does nothing.

(BTW What does the $\epsilon$-elimination algorithm do to this grammar?)

## Eliminating "immediate" left recursion

Let's begin with an easy example, already considered:

$$A \to Ab \mid b$$

This grammar is left-recursive, since

$$A \overset{+}{\Rightarrow} Ab\,.$$

In this case, we can eliminate left recursion as follows:

$$\begin{aligned} A &\to bA' \\ A' &\to bA' \mid \epsilon \end{aligned}$$

More generally, we can eliminate "immediate" left recursion as follows. If

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

represents all the $A$-productions of the grammar, and no $\beta_i$ begins with $A$, then we can replace these $A$-productions by

$$\begin{aligned} A &\to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' &\to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{aligned}$$



If

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

represents all the $A$-productions of the grammar, and no $\beta_i$ begins with $A$, then we can replace these $A$-productions by

$$\begin{aligned} A &\to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' &\to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{aligned}$$

---

If our grammar has $S$-productions

$$S \to SX \mid SSb \mid XS \mid a$$

we can replace them with

$$\begin{aligned} S &\to XSS' \mid aS' \\ S' &\to XS' \mid SbS' \mid \epsilon \end{aligned}$$

Notice that this construction can fail to eliminate left-recursion if we have the production

$$A \to A\,!$$

For instance,

$$A \to A \mid Ab \mid b$$

becomes

$$\begin{aligned} A &\to bA' \\ A' &\to A' \mid bA' \mid \epsilon \end{aligned}$$

If

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

represents all the $A$-productions of the grammar, and no $\beta_i$ begins with $A$, then we can replace these $A$-productions by

$$\begin{aligned} A &\to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' &\to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{aligned}$$

---

Another interesting special case. What if there are no $\beta_i$'s?

For example,

$$A \to AA \mid Ab\,.$$

Then everything that can be derived from $A$ has a variable in it, so $A$ cannot appear in a derivation of a sentence.

And the construction handles this in an interesting way, yielding

$$A' \to AA' \mid bA' \mid \epsilon$$

but no $A$-productions.



If

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

represents all the $A$-productions of the grammar, and no $\beta_i$ begins with $A$, then we can replace these $A$-productions by

$$\begin{aligned} A &\to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' &\to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{aligned}$$

---

Notice also that this construction works only "locally":

That is, indirect recursion is not eliminated.

For example, if we apply this construction to both variables in

$$\begin{aligned} S &\to SX \mid SSb \mid XS \mid a \\ X &\to Sa \mid Xb \end{aligned}$$

we obtain

$$\begin{aligned} S &\to XSS' \mid aS' \\ S' &\to XS' \mid SbS' \mid \epsilon \\ X &\to SaX' \\ X' &\to bX' \mid \epsilon \end{aligned}$$

and so have $S \overset{+}{\Rightarrow} SaX'SS'$, for instance.

Here's an algorithm that eliminates all left-recursion for any CFG without $\epsilon$-productions and without cycles.

Arrange the variables in some order $A_1, A_2, \ldots, A_n$.

**for** $i := 1$ to $n$ **do begin**
  **for** $j := 1$ to $i-1$ **do begin**
    replace each production of the form $A_i \to A_j\gamma$
      by the productions $A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$
      where $A_j \to \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$ are all the current $A_j$-productions;
  **end**
  eliminate the immediate left recursion among the $A_i$-productions;
**end**

Consider the grammar

$$\begin{aligned} S &\to SX \mid SSb \mid XS \mid a \\ X &\to Xb \mid Sa \mid b \end{aligned}$$

Let's order the variables $S, X$:

The first time through we simply eliminate immediate left recursion in $S$-productions, yielding

$$\begin{aligned} S &\to XSS' \mid aS' \\ S' &\to XS' \mid SbS' \mid \epsilon \\ X &\to Xb \mid Sa \mid b \end{aligned}$$



Arrange the variables in some order $A_1, A_2, \ldots, A_n$.
**for** $i := 1$ to $n$ **do begin**
  **for** $j := 1$ to $i-1$ **do begin**
    replace each production of the form $A_i \to A_j\gamma$
      by the productions $A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$
      where $A_j \to \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$ are all the current $A_j$-productions;
  **end**
  eliminate the immediate left recursion among the $A_i$-productions;
**end**

---

So at this point we have grammar

$$\begin{aligned} S &\to XSS' \mid aS' \\ S' &\to XS' \mid SbS' \mid \epsilon \\ X &\to Xb \mid Sa \mid b \end{aligned}$$

and the next obligation is to replace the production

$$X \to Sa$$

with the productions

$$X \to XSS'a \mid aS'a\,.$$

We then eliminate immediate left recursion among

$$X \to XSS'a \mid aS'a \mid Xb \mid b\,.$$

Eliminating immediate left recursion among

$$X \to XSS'a \mid Xb \mid b \mid aS'a$$

yields

$$\begin{aligned} X &\to bX' \mid aS'aX' \\ X' &\to SS'aX' \mid bX' \mid \epsilon \end{aligned}$$

So the final result is

$$\begin{aligned} S &\to XSS' \mid aS' \\ S' &\to XS' \mid SbS' \mid \epsilon \\ X &\to bX' \mid aS'aX' \\ X' &\to SS'aX' \mid bX' \mid \epsilon \end{aligned}$$



Let's look at examples showing that this algorithm can fail if the grammar has $\epsilon$-productions or cycles.

In the simplest case, when there is only one variable, call it $X$, the presence of a cycle implies that the grammar includes the production

$$X \to X\,.$$

Moreover, the whole left recursion elimination algorithm reduces to elimination of immediate left recursion among $X$-productions.

And we have previously observed that our construction for immediate left recursion elimination is no good in the presence of $X \to X$.

For example, if the grammar is

$$X \to X \mid a$$

the construction for eliminating immediate left recursion yields

$$\begin{aligned} X &\to aX' \\ X' &\to X' \end{aligned}$$

What about more complex cycles?

$$
\begin{aligned}
S &\rightarrow X \mid b \\
X &\rightarrow S \mid a
\end{aligned}
$$

Try ordering $S, X$.

First step: eliminate immediate left recursion in $S$-productions.

There is none.

Next: replace production

$$X \rightarrow S$$

with productions

$$X \rightarrow X \mid b\,.$$

It remains only to eliminate immediate left recursion in the current $X$-productions, which are

$$X \rightarrow X \mid b \mid a\,.$$

As before, the presence of production $X \rightarrow X$ breaks our construction — which yields

$$
\begin{aligned}
X &\rightarrow bX' \mid aX' \\
X' &\rightarrow X'
\end{aligned}
$$



Here's an example with an $\epsilon$-production and no cycles:

$$
\begin{aligned}
S &\rightarrow XSa \mid b \\
X &\rightarrow \epsilon
\end{aligned}
$$

Try order $S, X$.

First step: eliminate immediate left recursion in $S$-productions.

There is none.

Next: replace any $X$-productions whose rhs begins with $S$.

There are none.

Last: eliminate immediate left recursion in the current $X$-productions.

There is none.

So our left-recursion elimination algorithm leaves the grammar unchanged.

Yet $S \stackrel{+}{\Rightarrow} Sa$, so the grammar is left-recursive.

So now we can take any grammar and eliminate left-recursion (in three steps), making it suitable for top-down parsing (with backtracking!).

Notice that this works even for ambiguous grammars.

Next time we'll define the main component of a top-down parser — the **parsing table**.

But practically speaking, we would also like to avoid backtracking.

Next time we'll see how this can be done for top-down parsing.

We'll define the class of LL(1) grammars, suitable for predictive parsing.



## For next time

Read 4.4.