Right-Linear Grammars

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Objectives

You should be able to ...

- Convert between a regular expression and a right-linear grammar.

Right-Linear Grammars

A right-linear grammar is one in which every production has the form

$$A \to x$$

or

$$A \to xB$$

or

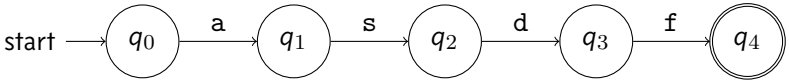
$$A \to B$$

where A and B are arbitrary (possibly identical) nonterminal symbols, and x is an arbitrary terminal symbol.

- "At most one nonterminal symbol in the right-hand side."
- It turns out these are equivalent to NFAs!
- Have one nonterminal symbol for each state, one terminal symbol for each production.

Example 1

- Regular expression: `asdf`
- State machine:



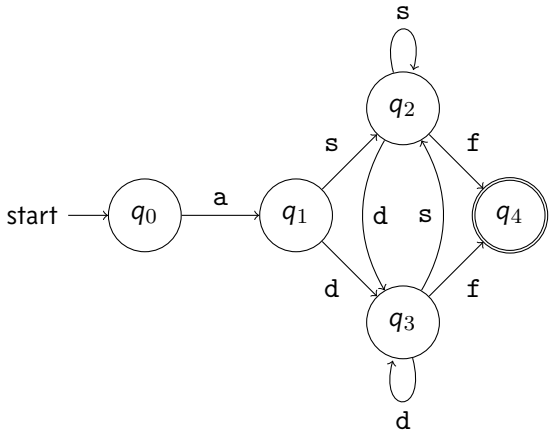
- Grammar:

$$
\begin{aligned}
S_0 &\rightarrow \texttt{a}S_1 \\
S_1 &\rightarrow \texttt{s}S_2 \\
S_2 &\rightarrow \texttt{d}S_3 \\
S_3 &\rightarrow \texttt{f}S_4 \\
S_4 &\rightarrow \epsilon
\end{aligned}
$$

Example 2

- Regular expression: `a(s|d)+f`

$$
\begin{aligned}
S_0 &\to \texttt{a}S_1 \\
S_1 &\to \texttt{s}S_2 \\
&\mid \texttt{d}S_3 \\
S_2 &\to \texttt{s}S_2 \\
&\mid \texttt{d}S_3 \\
&\mid \texttt{f}S_4 \\
S_3 &\to \texttt{s}S_2 \\
&\mid \texttt{d}S_3 \\
&\mid \texttt{f}S_4
\end{aligned}
$$



Going from Regular Expression to Right-Linear Grammar

- One way: regular expression $\rightarrow$ NFA $\rightarrow$ DFA $\rightarrow$ RLG
- Aonther way: direct conversion. We'll use a "bottom up" strategy.

Characters To convert a single character `a`, we make a simple production.
$S \rightarrow$ `a`
where S is the start symbol.

Concatenation To concatenate two regular expressions, add the second start symbol to the end of any "accepting" states from the first grammar.

Regexp: `a`
$S_1 \rightarrow$ `a`

Regexp: `b`
$S_2 \rightarrow$ `b`

Regexp: `ab`
$S_1 \rightarrow$ `a`S_2
$S_2 \rightarrow$ `b`

Choice and Repetition

Choice To choose between two regular expressions, add a new start symbol that "picks" one of the choices.

Regexp: `a`

$S_1 \rightarrow$ `a`

Regexp: `b`

$S_2 \rightarrow$ `b`

Regexp: `a|b`

$S \rightarrow S_1 | S_2$

$S_1 \rightarrow$ `a`

$S_2 \rightarrow$ `b`

Kleene Plus If S is the start symbol, then for every rule of the form $A \rightarrow \texttt{x}$ ("accepting states") add another rule of the form $A \rightarrow \texttt{x}S$. You may have to remove ϵ productions first.

Regexp: `a|b`

$S \rightarrow S_1 | S_2$

$S_1 \rightarrow$ `a`

$S_2 \rightarrow$ `b`

Regexp: `(a|b)+`

$S \rightarrow S_1 | S_2$

$S_1 \rightarrow$ `a`$|$`a`S

$S_2 \rightarrow$ `b`$|$`b`S

Choice and Repetition

Kleene Star If S is the start symbol, then for every rule of the form $A \to \mathtt{x}$ ("accepting states") add another rule of the form $A \to \mathtt{x}S$. Also add an ϵ rule.

Regexp: `a|b`

$$\begin{aligned} S &\to S_1 | S_2 \\ S_1 &\to \mathtt{a} \\ S_2 &\to \mathtt{b} \end{aligned}$$

Regexp: `(a|b)*`

$$\begin{aligned} S &\to S_1 | S_2 | \epsilon \\ S_1 &\to \mathtt{a} | \mathtt{a}S \\ S_2 &\to \mathtt{b} | \mathtt{b}S \end{aligned}$$

Credits

The algorithm for converting a regular expression to a right-linear grammar is based partly on the discussion here:
http://vasy.inria.fr/people/Gordon.Pace/Research/Software/Relic/Transformations/RE/toRG.html