Specialized Grammars

Linear Grammars and Chomsky Normal Form

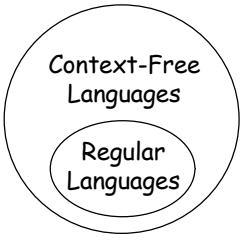
Monday, November 5, 2007
Reading: Stoughton 4.4, 4.8-4.9; Sipser 2.1; Kozen 21

CS235 Languages and Automata

Department of Computer Science
Wellesley College

Overview

1. Introduce **right-linear** and **left-linear** grammars and use these to prove that regular languages are a proper subset of context-free languages.



2. Introduce **Chomsky Normal Form**, and show how to convert any context-free grammar to Chomsky Normal form.

Linear Grammars

A CFG is **linear** iff every production has at most one variable in its RHS.

A CFG is **right-linear** iff every production has one of these two forms, where V and W are any variables and x and y are any string of terminals.

$V \rightarrow x$
$V \rightarrow yW$

A CFG is **left-linear** iff every production has one of these two forms, where V and W are any variables and x and y are any string of terminals.

$V \rightarrow x$
$V \rightarrow Wy$

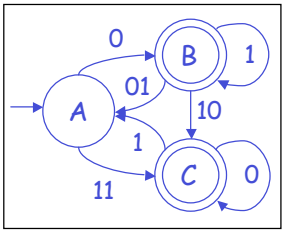
Right-Linear CFGs Generate All Regular Languages

(1) We'll show that every right-linear grammar can be converted to an FA, and so generates a regular language.

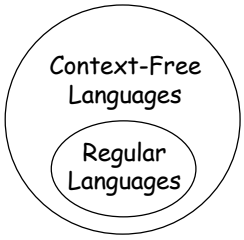
$S \rightarrow \% \mid 0S \mid 00P \mid 11Q$
$P \rightarrow 01P \mid 0Q \mid 1R$
$Q \rightarrow \% \mid 0S \mid 1Q \mid 00R$
$R \rightarrow \% \mid 10R$

FA

(2) We'll show that every FA (and thus every regular language) can be converted to a right-linear grammar.



right-linear grammar

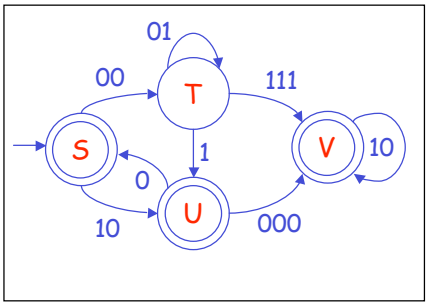


Together, (1) and (2) imply that regular languages are a subset of the context-free languages.

Non-regular CFLs (like 0^n1^n) show that the subset relation is proper.

Converting Right-Linear Grammars to FAs

S → % | 00T | 10U
T → 01T | 1U | 111V
U → % | 0S | 000V
V → % | 11V



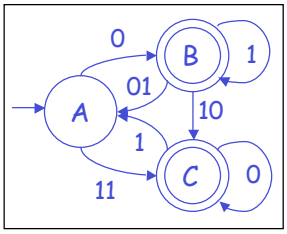
(1) The **states** of the FA are named by the variables of the CFG.

(2) The **start state** is the state named by the start variable.

(3) A state Q is an **accepting state** iff there is a production Q → %.

(4) There is a **transition** (P, x, Q) for each production P → xQ.

Converting FAs to Right-Linear Grammars



A → 0B | 11C
B → % | 01A | 1B | 10C
C → % | 1A | 0C

(1) The **variables** of the CFG are the states of the CFG.

(2) The **nonterminals** of the CFG are the transition symbols in the FA.

(3) The **start variable** of the CFG is the FA's start state.

(4a) There is a **production** Q → % for each accepting state Q.

(4b) There is a **production** P → xQ for each transition (P, x, Q).

What About Left-Linear Grammars?

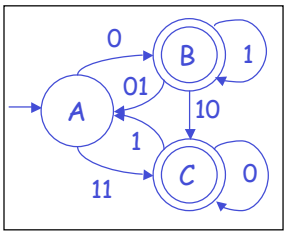
We have seen that a language is regular iff it can be expressed via a right-linear grammar.

It turns out that the same holds for left-linear grammars:

A language is regular iff it can be expressed via a left-linear grammar.

(You will work out the details on PS5.)

$A \rightarrow A10 \mid B1 \mid C0$
$B \rightarrow \% \mid A11 \mid B01 \mid C101$
$C \rightarrow \% \mid B010 \mid C00$



left-linear grammar

FA

Careful! Regularity is not guaranteed if right-linear and left-linear productions are mixed! E.g.:

$A \rightarrow \% \mid 0B$
$B \rightarrow A1$

Chomsky Normal Form

Sometimes it's helpful to require a CFG to be in a standard form. E.g., we'll see this soon in regards to *a pumping lemma for CFLs*.

One such form is **Chomsky Normal Form (CNF)**, in which all productions must have one of the following two forms:

$V \rightarrow UW$ Variable rewrites to two variables; U,V can't be start variable
$V \rightarrow t$ Variable rewrites to a single terminal

Chomsky Normal Form can generate any CFL not containing %. In order to allow languages with %, we also allow the production:

$S \rightarrow \%$ % production allowed only for start variable S

Intuitions:

- Parse trees for CNF have variables arranged in binary trees.
- Every step in a derivation from a CNF grammar makes nontrivial progress toward the terminal string. Can't have subtrees yielding % or long sequences of unit productions: $A \rightarrow B \rightarrow C \rightarrow \ldots$

CFG to CNF, Step 1: Add New Start Variable*

Introduce a new start variable that rewrites to the given one. (This guarantees that the new start variable does not occur in a RHS.)

Our running example (what language does it generate?):

$S \rightarrow P \mid Q \mid bSa$
$P \rightarrow aR$
$Q \rightarrow \% \mid QS$
$R \rightarrow Sb$

$S_0 \rightarrow S$
$S \rightarrow P \mid Q \mid bSa$
$P \rightarrow aR$
$Q \rightarrow \% \mid QS$
$R \rightarrow Sb$

* Sipser does this, but Stoughton and Kozen do not (because they don't handle languages containing %).

CFG to CNF, Step 2: Remove Nullable Variables

(a) Find all **nullable variables** = those variables that can yield %.

(b) For each nullable V and production $W \rightarrow \alpha V \beta$, (where at least one of α or β is nonempty) add a production $W \rightarrow \alpha\beta$.

(c) If S_0 is nullable, add the production $S_0 \rightarrow \%$.

(d) Remove all productions of the form $V \rightarrow \%$, where $V \neq S_0$.

Our running example:

$S_0 \rightarrow S$
$S \rightarrow P \mid Q \mid bSa$
$P \rightarrow aR$
$Q \rightarrow \% \mid QS$
$R \rightarrow Sb$

$S_0 \rightarrow S$
$S \rightarrow P \mid Q \mid bSa \mid ba$
$P \rightarrow aR$
$Q \rightarrow \% \mid QS \mid Q \mid S$
$R \rightarrow Sb$

$S_0 \rightarrow \% \mid S$
$S \rightarrow P \mid Q \mid bSa \mid ba$
$P \rightarrow aR$
$Q \rightarrow QS \mid Q \mid S$
$R \rightarrow Sb$

(a) Nullable variables $= \{S_0, S, Q\}$

CFG to CNF, Step 3: Remove Unit Productions

A **unit production** is one of the form $V \rightarrow W$.

(a) For each rewrite sequence $V_1 \Rightarrow^* V_n \Rightarrow \alpha$, where α is not a single variable, add a production $V_1 \rightarrow \alpha$ (if it doesn't already exist).

(b) Remove all unit productions.

Our running example:

$S_0 \rightarrow \% \mid S$
$S \rightarrow P \mid Q \mid bSa \mid ba$
$P \rightarrow aR$
$Q \rightarrow QS \mid Q \mid S$
$R \rightarrow Sb$

(a)

$S_0 \rightarrow \% \mid S \mid aR \mid QS \mid bSa \mid ba$
$S \rightarrow P \mid Q \mid bSa \mid ba \mid aR \mid QS$
$P \rightarrow aR$
$Q \rightarrow QS \mid Q \mid S \mid aR \mid bSa \mid ba$
$R \rightarrow Sb$

(b)

$S_0 \rightarrow \% \mid aR \mid QS \mid bSa \mid ba$
$S \rightarrow aR \mid QS \mid bSa \mid ba$
$P \rightarrow aR$
$Q \rightarrow aR \mid QS \mid bSa \mid ba$
$R \rightarrow Sb$

CFG to CNF, Step 4: Completing CNF

Every production now has one of the following three forms:

(1) $V \rightarrow t$
(2) $V \rightarrow \alpha$, where α has at least two variables and/or terminals.
(3) $S_0 \rightarrow \%$

Introduce new variables and productions to replace all productions of form (2) not in CNF by CNF productions.

Our running example:

$S_0 \rightarrow \% \mid aR \mid QS \mid bSa \mid ba$
$S \rightarrow aR \mid QS \mid bSa \mid ba$
$P \rightarrow aR$
$Q \rightarrow aR \mid QS \mid bSa \mid ba$
$R \rightarrow Sb$

$S_0 \rightarrow \% \mid TR \mid QS \mid UV \mid UT$
$S \rightarrow TR \mid QS \mid UV \mid UT$
$P \rightarrow TR$
$Q \rightarrow TR \mid QS \mid UV \mid UT$
$R \rightarrow SU$
$T \rightarrow a$
$U \rightarrow b$
$V \rightarrow ST$

- In this example, we have cleverly added only three new variables, but this is not required and a straightforward implementation would add **many** more.

Simplification

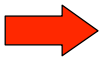
A variable in a CFG is **useless** if it can never appear in a parse tree rooted at the start symbol.

It is always safe to simplify a CFG by removing all productions that mention a useless variable.

Our running example:

$S_0 \rightarrow \% \mid TR \mid QS \mid UV \mid UT$
$S \rightarrow TR \mid QS \mid UV \mid UT$
$P \rightarrow TR$
$Q \rightarrow TR \mid QS \mid UV \mid UT$
$R \rightarrow SU$
$T \rightarrow a$
$U \rightarrow b$
$V \rightarrow ST$

P is useless



$S_0 \rightarrow \% \mid TR \mid QS \mid UV \mid UT$
$S \rightarrow TR \mid QS \mid UV \mid UT$
$Q \rightarrow TR \mid QS \mid UV \mid UT$
$R \rightarrow SU$
$T \rightarrow a$
$U \rightarrow b$
$V \rightarrow ST$

See Stoughton 4.4 for details on simplification.

Greibach Normal Form (GNF)

Another normal form for CFGs is **Greibach Normal Form (GNF)**, in which every production (except for $S_0 \rightarrow \%$) has the form:

$V_0 \rightarrow tV_1V_2...V_n$, $n \geq 0$

GNF has the nice property that every rewrite (except $S_0 \rightarrow \%$) makes progress toward the final string by adding a terminal.

Idea: To obtain GNF, start with CNF, and create a GNF production of the above form for every collection of CNF productions with the following form:

$V_0 \rightarrow W_nV_n$
$W_n \rightarrow W_{n-1}V_{n-1}$
...
$W_2 \rightarrow W_1V_1$
$W_1 \rightarrow t$

See Kozen Lecture 21 for details.