Specialized Grammars

Linear Grammars and Normal Forms

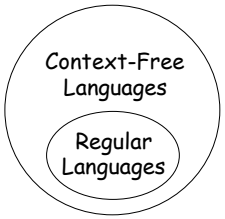
Friday, November 5, 2010
Reading: Sipser 2.1; Stoughton 4.4, 4.8-4.9; Kozen 21

CS235 Languages and Automata

Department of Computer Science
Wellesley College

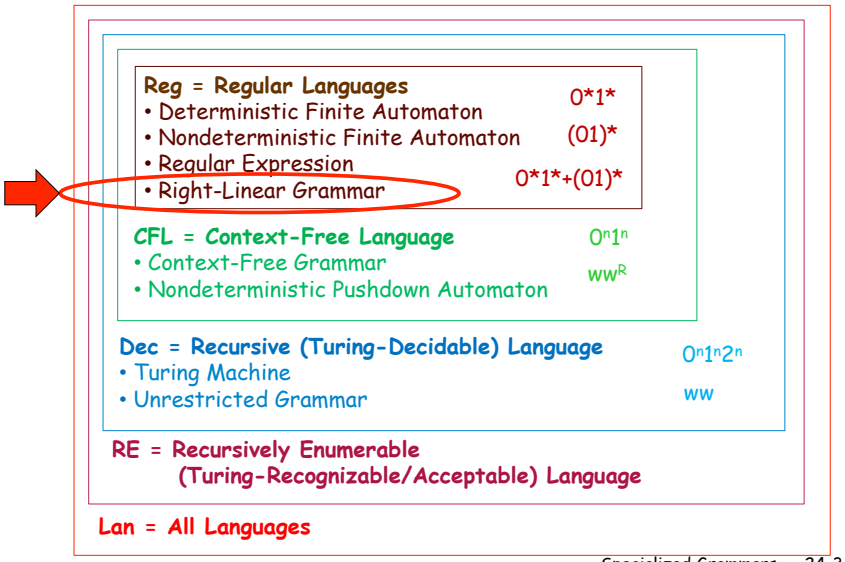
Overview

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 **Greibach Normal Form** and show how to convert any context-free grammars to these forms.

Yet Another Way to Specify Regular Languages



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 is any string of terminals.

$V \rightarrow \%$
$V \rightarrow xW$

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

$V \rightarrow \%$
$V \rightarrow Wx$

Via a simple transformation, we can also include productions with the form $V \rightarrow y$, where y is a string of terminals. How?

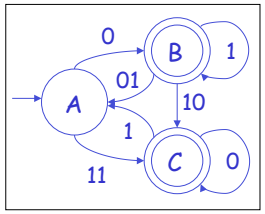
Right-Linear CFGs Generate All Regular Languages

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

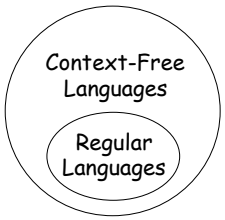
S → % | 00T | 10U
T → 01T | 1U | 111V
U → % | 0S | 000V
V → % | 11V

FA

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



right-linear grammar

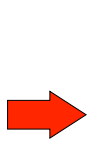


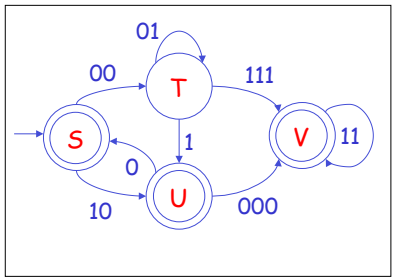
Together, (1) and (2) imply that regular languages are a subset of the context-free languages.

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





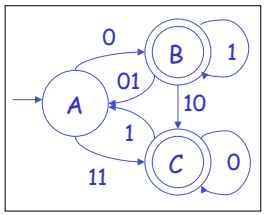
(1) The **states** of the FA are named by the variables of the CFG.

(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 of the form 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?

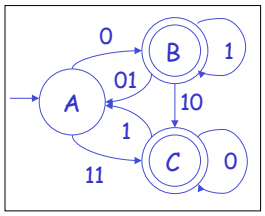
We have seen that a language is regular iff it can be expressed via a right-linear grammar.

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

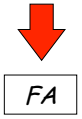
A language is regular iff it can be expressed via a left-linear grammar.

(You will work out the details on PS8.)

P → P10 | Q1 | R0
Q → % | P11 | B01 | R101
R → % | Q010 | R00



left-linear grammar



FA

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

A → % | 0B
B → A1

Chomsky Normal Form (CNF)

Sometimes it's helpful to require a CFG to be in a standard form. E.g., we'll see this next time 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:

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

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

$S \to \%$	% 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: $A \to B \to C \to \ldots$

Chomsky Normal Form: Example

$S \to \% \mid AB \mid AC$
$A \to 0$
$B \to 1$
$C \to DB$
$D \to AB \mid AC$

Here's a sample grammar in Chomsky Normal Form.

Draw derivations for the 3 shortest strings in the language generated by this grammar.

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

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

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

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

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

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

CFG to CNF, Step 2: Find Nullable Variables

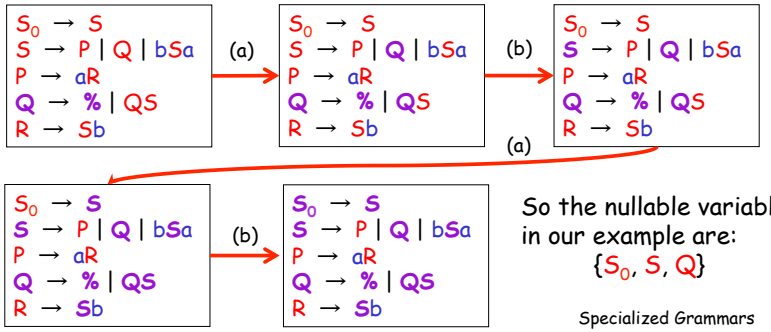
$S_0 \to S$
$S \to P \mid Q \mid bSa$
$P \to aR$
$Q \to \% \mid QS$
$R \to Sb$

A variable is **nullable** iff it can yield **%**.

To find all **nullable** variables, first color every **%**.

Then repeat the following steps until there are no changes (an **iterative fixed point technique**)

(a) color a LHS variable if one of its RHSs is completely colored;
(b) color every occurrence of a colored LHS variable in a RHS.



So the nullable variables in our example are:
$\{S_0, S, Q\}$

CFG to CNF, Step 3: Remove Nullable Variables

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

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

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

Our running example:

Nullable variables = {S_0, S, Q}

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

(a)

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

(b), (c)

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

CFG to CNF, Step 4: 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 isn't a single variable, add a production $V_1 \rightarrow \alpha$ (if it doesn't already exist).

(b) Remove all unit productions.

Our running example:

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

(a)

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

(b)

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

CFG to CNF, Step 5: 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:

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

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

- In this example, we cleverly added only three new variables, but 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:

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

P is useless

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

See Stoughton 4.4 for details on simplification.

CNF in Forlan

```
- val L1gram = Gram.input "L1.gram";
val L1gram = - : gram

- Gram.output ("", L1gram);
{variables}
A, B, S
{start variable}
S
{productions}
A -> % | 0A1; B -> % | 1B0; S -> AB
val it = () : unit

- val L1gramCNF = Gram.chomskyNormalForm L1gram;
val L1gramCNF = - : gram

- Gram.output ("", L1gramCNF);
{variables}
<1,A>, <1,B>, <1,S>, <2,0>, <2,1>, <3,A1>, <3,B0>
{start variable}
<1,S>
{productions}
<1,A> -> <2,0><2,1> | <2,0><3,A1>; <1,B> -> <2,1><2,0> | <2,1><3,B0>;
<1,S> -> <1,A><1,B> | <2,0><2,1> | <2,0><3,A1> | <2,1><2,0> | <2,1><3,B0>;
<2,0> -> 0; <2,1> -> 1; <3,A1> -> <1,A><2,1>; <3,B0> -> <1,B><2,0>
val it = () : unit
```


Forlan Grammar Canonicalization

```
- Gram.output ("", (Gram.renameVariablesCanonically L1gramCNF));
{variables}
A, B, C, D, E, F, G
{start variable}
C
{productions}
A -> DE | DF; B -> ED | EG; C -> AB | DE | DF | ED | EG; D -> 0; E -> 1;
F -> AE; G -> BD
val it = () : unit
```

(* Note: Forlan's CNF *cannot* generate % even though it was in the original language! *)

Using the above CNF grammar, what is the parse tree for 001110?

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:

$V_0 \rightarrow tV_1V_2...V_n$, $n \geq 0$ (where t is a single terminal)

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

Not-quite-right 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:

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

$\Rightarrow$

$V_0 \rightarrow tV_1V_2...V_n$
$W_n \rightarrow tV_1V_2...V_{n-1}$
$\vdots$
$W_2 \rightarrow tV_1$
$W_1 \rightarrow t$

In fact, it's more complicated than this. See Kozen Lecture 21 for details.

Greibach Normal Form: Example

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

expand initial T, U

$S_0 \rightarrow \% \mid aR \mid QS \mid bV \mid bT$
$S \rightarrow aR \mid QS \mid bV \mid bT$
$Q \rightarrow aR \mid QS \mid bV \mid bT$
$R \rightarrow SU \mid b$
$T \rightarrow a$
$U \rightarrow b$
$V \rightarrow ST$

expand initial Q *

$S_0 \rightarrow \% \mid aR \mid aRS \mid bVS \mid bTS \mid bV \mid bT$
$S \rightarrow aR \mid aRS \mid bVS \mid bTS \mid bV \mid bT$
$Q \rightarrow aR \mid aRS \mid bVS \mid bTS \mid bV \mid bT$
$R \rightarrow SU \mid b$
$T \rightarrow a$
$U \rightarrow b$
$V \rightarrow ST$

expand initial S and remove Q (now useless)

$S_0 \rightarrow \% \mid aR \mid aRS \mid bVS \mid bTS \mid bV \mid bT$
$S \rightarrow aR \mid aRS \mid bVS \mid bTS \mid bV \mid bT$
$R \rightarrow aRU \mid aRSU \mid bVSU \mid bTSU \mid bVU \mid bTU \mid b$
$T \rightarrow a$
$U \rightarrow b$
$V \rightarrow aRT \mid aRST \mid bVST \mid bTST \mid bVT \mid bTT$

* This step is glossing over some details ....