Chomsky Normal Form (CNF)
is one of the most basic Normal Forms.

In CNF each production has the form

$$A \longrightarrow BC$$

$$A \longrightarrow \alpha$$

where $A, B, C \in V$ (i.e., nonterminals)
and $\alpha \in \Sigma$ (i.e., a terminal)

This can be done by simple substitution.

Consider the following

S	$\longrightarrow$	bA
S	$\longrightarrow$	aB
A	$\longrightarrow$	bAA
A	$\longrightarrow$	aS
A	$\longrightarrow$	a
B	$\longrightarrow$	aBB
B	$\longrightarrow$	bS
B	$\longrightarrow$	b

Replace terminals with *NEW* Nonterminals
Plus a rule to generate the Terminal

$$C \longrightarrow a$$
$$D \longrightarrow b$$

Note these are already in standard form.

Terminals should now only appear
in rules in CNF.

Rewriting the rules yields:

$$\begin{array}{lcl} S & \longrightarrow & DA \\ S & \longrightarrow & CB \\ A & \longrightarrow & DAA \\ A & \longrightarrow & CS \\ A & \longrightarrow & a \\ B & \longrightarrow & CBB \\ B & \longrightarrow & DS \\ B & \longrightarrow & b \\ C & \longrightarrow & a \\ D & \longrightarrow & b \end{array}$$

Only the following rules are a problem:

$$A \longrightarrow DAA$$
$$B \longrightarrow CBB$$

These can be rewritten as follows:

$$A \longrightarrow ZA$$
$$Z \longrightarrow DA$$
$$B \longrightarrow KB$$
$$K \longrightarrow CB$$

Now we have rules in CFN.

Slide Lecture 10 –277

SO WHAT?

For grammars in Chomsky Normal Form
the parse tree is always a binary tree.

We can talk about the relationship between:

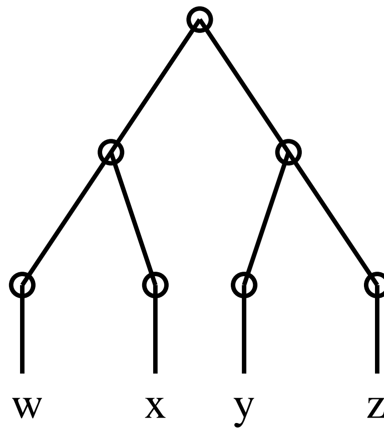
1) the depth of the parse tree

and 2) the length of its yield.

Slide Lecture 10 –278

If a parse tree for a word ω
is generated by a CNF **and** the parse tree
has a path length of at most i,
then the length of ω is at most 2^{i-1}.

Slide Lecture 10 –279



A parse tree of depth 3 with a yield
of at most 2^{3-1}.

Slide Lecture 10 –280

Griebach Normal Form (GNF)

In GNF each production has the form

$$A \longrightarrow aB$$

where $A \in V, B \in V*$
and $\alpha \in \Sigma$
(α is a terminal,
A is a nonterminal,
and B is zero or more nonterminals)
aB gives us a more immediate relationship
between the input being scanned and
the string being generated.

Slide Lecture 10 –281

We will assume G is already in Chomsky Normal Form

Let G = ($\{A_1, A_2, A_3\}$, {a, b}, R, S)

$$\begin{aligned} R \quad = \quad A_1 &\longrightarrow A_2 A_3 \\ A_2 &\longrightarrow A_3 A_1 \\ A_2 &\longrightarrow \text{b} \\ A_3 &\longrightarrow A_1 A_2 \\ A_3 &\longrightarrow \text{a} \end{aligned}$$

Slide Lecture 10 –282

Use the indices as an ordering over the nonterminals.

Rewrite the rules so that

$$A_i \longrightarrow A_j V*$$

such that $i \leq j$.

Only this rule is not in that form:

$$A_3 \longrightarrow A_1 A_2$$

Slide Lecture 10 –283

Rule

$$A_3 \longrightarrow A_1 A_2$$

Becomes:

$$A_3 \longrightarrow A_2 A_3 A_2$$

Which becomes 2 rules

$$A_3 \longrightarrow b A_3 A_2$$

$$A_3 \longrightarrow A_3 A_1 A_3 A_2$$

Slide Lecture 10 –284

We now encounter an example of
left recursion.
There must also be a rule to terminate recursion

$$A_x \longrightarrow B_k$$

$$A_x \longrightarrow A_x B_y$$

Nonterminal A_x generates

$$B_k B_y ... B_y B_y = B_k (B_y)*$$

Slide Lecture 10 –285

$$B_k (B_y)*$$

We need to move the recursion to the right,

$$A_x \longrightarrow B_k$$

$$A_x \longrightarrow B_k N_x$$

$$N_x \longrightarrow B_y N_x$$

$$N_x \longrightarrow B_y$$

Now recursion is on the right.

Slide Lecture 10 –286

Returning to our problem

$$A_3 \longrightarrow a$$

$$A_3 \longrightarrow bA_3A_2$$

$$A_3 \longrightarrow A_3A_1A_3A_2$$

Since there are two ways
to terminate the recursion
we need two rules to start recursion.

Slide Lecture 10 –287

$$A_3 \longrightarrow a$$

$$A_3 \longrightarrow bA_3A_2$$

$$A_3 \longrightarrow aN_3$$

$$A_3 \longrightarrow bA_3A_2N_3$$

$$N_3 \longrightarrow A_1A_3A_2N_3$$

$$N_3 \longrightarrow A_1A_3A_2$$

Slide Lecture 10 –288

The current set of rules:

$$A_1 \longrightarrow A_2A_3$$

$$A_2 \longrightarrow A_3A_1$$
$$A_2 \longrightarrow b$$

$$A_3 \longrightarrow bA_3A_2N_3$$
$$A_3 \longrightarrow bA_3A_2$$

$$A_3 \longrightarrow aN_3$$
$$A_3 \longrightarrow a$$

$$N_3 \longrightarrow A_1A_3A_2N_3$$
$$N_3 \longrightarrow A_1A_3A_2$$

Slide Lecture 10 –289

Since A_3 is in the correct form,
and all the original rules are expressed
in terms of "higher order" rules,
we can propagate the correct form down.

$$A_2 \longrightarrow A_3A_1$$

Becomes

$$A_2 \longrightarrow bA_3A_2N_3A_1$$
$$A_2 \longrightarrow bA_3A_2A_1$$

$$A_2 \longrightarrow aN_3A_1$$
$$A_2 \longrightarrow aA_1$$

Slide Lecture 10 –290

We can now do A_1.

$$A_1 \longrightarrow A_2 A_3$$

Becomes:

$$A_1 \longrightarrow bA_3$$

$$A_1 \longrightarrow bA_3 A_2 N_3 A_1 A_3$$

$$A_1 \longrightarrow bA_3 A_2 A_1 A_3$$

$$A_1 \longrightarrow aN_3 A_1 A_3$$

$$A_1 \longrightarrow aA_1 A_3$$

Slide Lecture 10 –291

That leaves only the new nonterminal.

$$N_3 \longrightarrow A_1 A_3 A_2 N_3$$

$$N_3 \longrightarrow A_1 A_3 A_2$$

But all the other rules have been resolved.

Since each rule goes to A_1 first and
there are now 5 rules for A_1
we get 5 new rules for each rule for N_3.

Slide Lecture 10 –292

$$N_3 \longrightarrow A_1A_3A_2N_3$$

Becomes:

$$N_3 \longrightarrow bA_3A_3A_2N_3$$

$$N_3 \longrightarrow bA_3A_2N_3A_1A_3A_3A_2N_3$$

$$N_3 \longrightarrow bA_3A_2A_1A_3A_3A_2N_3$$

$$N_3 \longrightarrow aN_3A_1A_3A_3A_2N_3$$

$$N_3 \longrightarrow aA_1A_3A_3A_2N_3$$

Slide Lecture 10 –293

$$N_3 \longrightarrow A_1A_3A_2$$

Becomes:

$$N_3 \longrightarrow bA_3A_3A_2$$

$$N_3 \longrightarrow bA_3A_2N_3A_1A_3A_3A_2$$

$$N_3 \longrightarrow bA_3A_2A_1A_3A_3A_2$$

$$N_3 \longrightarrow aN_3A_1A_3A_3A_2$$

$$N_3 \longrightarrow aA_1A_3A_3A_2$$

And we are done.

Slide Lecture 10 –294

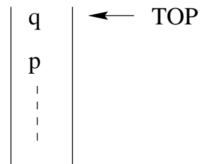
The textbook

"Introduction to Automata Theory,
Languages and Computation."
covers Griebach Normal Form

page 96-97, Hopcroft and Ullman,
1979, Addison-Wesley

Context Free Grammars require a limited form of memory.

To add memory to a finite automaton
and produce a simple yet more general machine,
we add **a pushdown stack**
A pushdown stack is a LIFO queue
which is operated by the primitives
"push" and "pop".

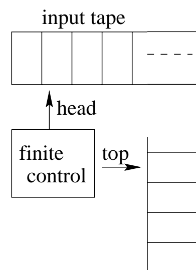


The depth of the stack is assumed always to be "sufficient" (virtually infinite).

This is a "one stack automaton".

Adding t or more stacks results in distinct classes of automata.

Slide Lecture 10 –297



Slide Lecture 10 –298

Formally, a pushdown automaton is a 6-tuple

$$M = (K, \Sigma, T\ \Delta, S, F)$$

K set of states

Σ input alphabet

T stack alphabet

S initial state

F set of final states

Δ the transition function

a subset of $(K \times \Sigma^* \times T^*) \times (K \times T^*)$

K ... current state

Σ^* ... input sequence

T^* ... stack sequence

Δ maps triples composed of

- a current state
- a sequence of input symbols and
- the top elements of the stack

TO

- a new state and
- new stack contents

$((p,$	$\mu,$	$\beta)$	$(q,$	$\gamma))$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
state 1	input	top of stack ‡	next state	new stack symbol in old position ‡

‡: we will allow λ here

push (x)	$((p, \mu, \lambda), (q, x))$
pop (x)	$((p, \mu, x), (q, \lambda))$

Slide Lecture 10 –301

Configurations

1. current state
2. unscanned portion of the input string
3. contents of the stack
   (K, Σ^*, T^*)

$$(s, \omega, \lambda) \overset{*}{\underset{m}{\vdash}} (f, \lambda, \lambda)$$

for some state $f \in F$

This implies computations $C_0 \underset{m}{\vdash} C_1 \underset{m}{\vdash} \cdots \underset{m}{\vdash} C_n$

where $C_0 = (s, \omega, \lambda)$, $C_n = (f, \lambda, \lambda)$

Slide Lecture 10 –302

Actually we can define acceptance in two ways:

- "empty store" . . . all input scanned, stack empty
- designate a set of final states

We can show these are equivalent.

Slide Lecture 10 –303

Example 1.

$$L = \{\omega c \omega^R \mid \omega \in (a,b)^*, c = \text{“ c”}, \omega^R \text{ is the reverse of } \omega\}$$

$$\Sigma = \{ a, b, c\}$$

abcba $\in$ L, acb $\notin$ L

Slide Lecture 10 –304

$$
\begin{aligned}
M &= (K, \Sigma, T, \Delta, S, F) \text{ where} \\
K &= \{s, f\} \\
\Sigma &= \{a, b, c\} \\
T &= \{a, b\} \\
S &= \{s\} \\
F &= \{f\}
\end{aligned}
$$

Slide Lecture 10 –305

Δ	$\{((S, a, \lambda), (S, a))$	(1)	scan and push a
	$((S, b, \lambda), (S, b))$	(2)	scan and push b
	$((S, c, \lambda), (f, \lambda))$	(3)	
	$((f, a, a), (f, \lambda))$	(4)	
	$((f, b, b), (f, \lambda))\}$	(5)	

Note: this automaton accepts by "empty store". Being in f, in this case, is **not** sufficient.

Slide Lecture 10 –306

Example of operation of M:

ω = abcba

State	Unread Input	Stack	Transition Used
S	abcba	λ	-
S	bcba	a	1
S	cba	ba	2
f	ba	ba	3
f	a	a	5
f	λ	λ	4

L = ωcω^R is easy.

Slide Lecture 10 –307

What about L = $\omega\omega^R$?

When do we start popping instead of pushing?

A nondeterminism!

$$
\begin{aligned}
L &= \{\omega\omega^* \mid \omega \in \{a,b\}^*\} \\
M' &= (K, \Sigma, T, \Delta, S, F) \\
K &= (S, F) \\
\Sigma &= \{a,b\} \\
T &= \{a,b\} \\
F &= \{f\}
\end{aligned}
$$

Slide Lecture 10 –308

$$
\begin{aligned}
\Delta \quad = \quad & \{((S, a, \lambda), (S, a)) && (1)\\
& ((S, b, \lambda), (S, b)) && (2)\\
& ((S, \lambda, \lambda), (f, \lambda)) && (3)\\
& ((f, a, a), (f, \lambda)) && (4)\\
& ((f, b, b), (f, \lambda))\} && (5)
\end{aligned}
$$

Consider ((s, λ, λ), (f, λ))

Don't Push, Don't Pop, Don't Scan.

This is clearly nondeterministic. Again accepts by empty store.

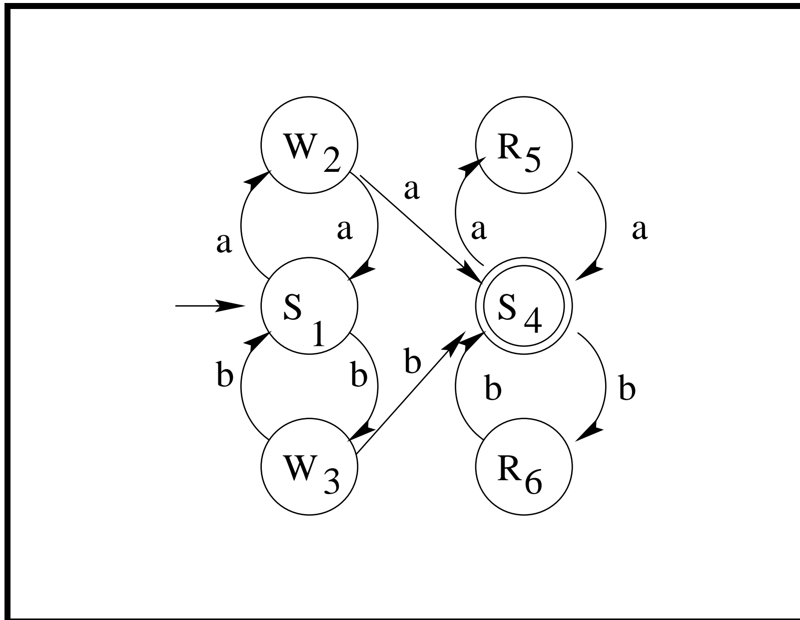
Slide Lecture 10 –309

What about the grammar for L?

G = (V, Σ, R, S) where

$$
\begin{aligned}
V &= \{S\} \bigcup \Sigma \\
\Sigma &= \{a, b\} \\
R &= \{S \longrightarrow aSa \\
& \quad S \longrightarrow bSb \\
& \quad S \longrightarrow \lambda\}
\end{aligned}
$$

Slide Lecture 10 –310



Slide Lecture 10 –311

Program [finite control]

1)	scan	(a, 2)	(b, 3)
2)	write	(a, 1)	(a, 4)
3)	write	(b, 3)	(b, 4)
4)	scan	(a, 5)	(b, 6)
5)	read	(a, 4)	
6)	read	(b, 4)	

Slide Lecture 10 –312