

Notes on Context Free Grammars

Geoffrey Matthews

Department of Computer Science
Western Washington University

November 5, 2018

Readings

- ▶ http://www.cs.rochester.edu/~nelson/courses/csc_173/grammars/cfg.html
- ▶ http://en.wikipedia.org/wiki/Context-free_grammar
- ▶ http://en.wikipedia.org/wiki/Context-free_language
- ▶ <http://en.wikipedia.org/wiki/Parsing>
- ▶ http://en.wikipedia.org/wiki/Pushdown_automata
- ▶ http://en.wikipedia.org/wiki/LR_parser
- ▶ <https://parasol.tamu.edu/~rwerger/Courses/434/lec12-sum.pdf>
- ▶ <http://www.cs.sunysb.edu/~cse350/slides/cfg3.pdf>

String Substitution

- ▶ Start with the string $ABBA$
- ▶ If we make the substitutions $A \rightarrow a$ and $B \rightarrow b$
- ▶ $ABBA \Rightarrow aBBA \Rightarrow abBA \Rightarrow abbA \Rightarrow abba$
- ▶ $ABBA \Rightarrow^* abba$
- ▶ If we make the substitutions $A \rightarrow ab$ and $B \rightarrow ba$
- ▶ $ABBA \Rightarrow abBBA \Rightarrow abbaBA \Rightarrow abbabaA \Rightarrow abbabaab$
- ▶ $ABBA \Rightarrow^* abbabaab$
- ▶ If we make the substitutions $A \rightarrow bab$ and $B \rightarrow bbb$
- ▶ $ABBA \Rightarrow babBBA \Rightarrow babbabbBA \Rightarrow babbabbbbaA \Rightarrow babbabbbbbbab$
- ▶ $ABBA \Rightarrow^* babbabbbbbbab$

Formal Grammars

- ▶ A set of **terminals**, e.g. $\{\text{the, cat, sat, on, mat}\}$
- ▶ A set of **nonterminals**, or **variables**, e.g. $\{S, N\}$
- ▶ A special nonterminal, the **start symbol**, e.g. S
- ▶ A set of **production rules**:

$$S \rightarrow \text{the } N \text{ sat on the } N$$
$$N \rightarrow \text{cat}$$
$$N \rightarrow \text{mat}$$

- ▶ A **derivation** is any string we get by starting with the start symbol and repeatedly making a single substitution until we only have terminals.
- ▶ $S \Rightarrow \text{the } N \text{ sat on the } N \Rightarrow \text{the cat sat on the } N \Rightarrow \text{the cat sat on the mat}$
- ▶ $S \Rightarrow \text{the } N \text{ sat on the } N \Rightarrow \text{the mat sat on the } N \Rightarrow \text{the mat sat on the mat}$

Vertical bar means “or”

This grammar:

$$S \rightarrow \text{the } N \text{ sat on the } N$$
$$N \rightarrow \text{cat}$$
$$N \rightarrow \text{mat}$$

is equivalent to this grammar:

$$S \rightarrow \text{the } N \text{ sat on the } N$$
$$N \rightarrow \text{cat} \mid \text{mat}$$

Rules can be recursive

$S \rightarrow S \text{ and } S$

$S \rightarrow \text{the } N \text{ sat on the } N$

$N \rightarrow \text{cat} \mid \text{mat}$

Context Free Grammar

A context free grammar is a grammar where all the rules are the following form:

$$S \rightarrow w$$

where S is a single nonterminal and w is a string of terminals and nonterminals.

Context Free Grammar Example 1

$$S \rightarrow aS \mid \epsilon$$

$$S \Rightarrow \epsilon$$

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaa$$

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaaaS \Rightarrow aaaaaS \Rightarrow aaaaaa$$

Context Free Grammar Example 2

$$S \rightarrow Sa \mid \epsilon$$

$$S \Rightarrow \epsilon$$

$$S \Rightarrow Sa \Rightarrow Saa \Rightarrow Saaa \Rightarrow aaa$$

$$S \Rightarrow Sa \Rightarrow Saa \Rightarrow Saaa \Rightarrow Saaaa \Rightarrow aaaa$$

Context Free Grammar Example 3

$$S \rightarrow ABC$$
$$A \rightarrow a$$
$$B \rightarrow b$$
$$C \rightarrow c$$
$$S \Rightarrow ABC \Rightarrow aBC \Rightarrow abC \Rightarrow abc$$
$$S \Rightarrow ABC \Rightarrow AbC \Rightarrow abC \Rightarrow abc$$
$$S \Rightarrow ABC \Rightarrow aBC \Rightarrow aBc \Rightarrow abc$$
$$S \Rightarrow ABC \Rightarrow ABc \Rightarrow Abc \Rightarrow abc$$

Context Free Grammar Example 4

$$S \rightarrow AB \mid A$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow Bb \mid b$$

$$S \Rightarrow A \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow aaaa$$

$$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aaBbb \Rightarrow aabbbb$$

$$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaAB \Rightarrow aaaB \Rightarrow aaaBb \Rightarrow aaaBbb \Rightarrow aaabbbb$$

Context Free Grammar for Arithmetic Expressions

$$E \rightarrow T$$

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow (E)$$

$$T \rightarrow a$$

$$T \rightarrow b$$

$$T \rightarrow T0$$

$$T \rightarrow T1$$

$$\begin{aligned} E &\Rightarrow E + E \Rightarrow E * E + E \Rightarrow T * E + E \Rightarrow a * E + E \Rightarrow \\ &a * T + E \Rightarrow a * b + E \Rightarrow a * b + T \Rightarrow a * b + a \end{aligned}$$

Note that T could have been represented by a regular language.

Context Free Grammar for Programming Language

$$S \rightarrow \text{while } E \text{ do } S \mid \text{if } E \text{ then } S \text{ else } S \mid I := E$$
$$S \rightarrow \{ SL \}$$
$$L \rightarrow SL ; \mid \epsilon$$
$$E \rightarrow \dots$$
$$I \rightarrow \dots$$

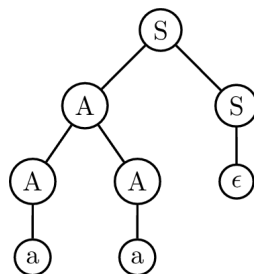
- ▶ Reference manuals for programming languages usually give the syntax of the language as a CFG.
- ▶ Note that keywords, punctuation, *etc.* can be represented by a regular language.

Derivations

- ▶ Start with S
- ▶ Find a rule for a nonterminal.
- ▶ Replace nonterminal with RHS.
- ▶ Until no more nonterminals.

$$S \rightarrow AS \mid \epsilon$$

$$A \rightarrow AA \mid a$$



Parse Tree

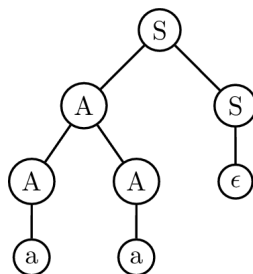
$$S \Rightarrow AS \Rightarrow AAS \Rightarrow AA \Rightarrow Aa \Rightarrow aa$$

Derivations

- ▶ Start with S
- ▶ Find a rule for a nonterminal.
- ▶ Replace nonterminal with RHS.
- ▶ Until no more nonterminals.

$$S \rightarrow AS \mid \epsilon$$

$$A \rightarrow AA \mid a$$



Parse Tree

$$S \Rightarrow AS \Rightarrow AAS \Rightarrow AA \Rightarrow Aa \Rightarrow aa$$

The **language of a grammar** is the set of all sentences for which there exists a derivation.

Derivations

- ▶ If there is more than one possible tree for some sentence, the grammar is **ambiguous**.
- ▶ There are usually many possible derivations, but only one tree.
- ▶ Important derivations are **leftmost** and **rightmost**.
- ▶ Leftmost and rightmost derivations also unique in unambiguous languages.

$$S \rightarrow AS \mid \epsilon$$

$$A \rightarrow AA \mid a$$

Leftmost:

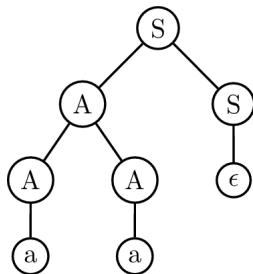
$$\underline{S} \Rightarrow \underline{A}S \Rightarrow \underline{AA}S \Rightarrow a\underline{A}S \Rightarrow aa\underline{S} \Rightarrow aa$$

Proof that this grammar is ambiguous:

Rightmost:

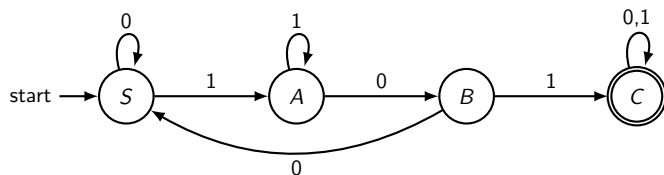
$$S \Rightarrow A\underline{S} \Rightarrow A\underline{A}S \Rightarrow A\underline{AA} \Rightarrow A\underline{a} \Rightarrow aa$$

$$\underline{S} \Rightarrow \underline{A}S \Rightarrow \underline{A} \Rightarrow \underline{AA} \Rightarrow \underline{A}a \Rightarrow aa$$



Parse Tree

All regular languages are context free languages



$$S \rightarrow 0S \mid 1A$$

$$A \rightarrow 0B \mid 1A$$

$$B \rightarrow 0S \mid 1C$$

$$C \rightarrow 0C \mid 1C \mid \epsilon$$

Generate All Possible Sentences from a CF grammar

Length 1 derivations:

$$S \rightarrow abS \mid bAc \mid d$$

$$A \rightarrow aA \mid \epsilon$$

$$S \Rightarrow abS$$

$$S \Rightarrow bAc$$

$$S \Rightarrow d$$

1. d

Generate All Possible Sentences from a CF grammar

Length 2 derivations:

$$\begin{aligned} S &\rightarrow abS \mid bAc \mid d \\ A &\rightarrow aA \mid \epsilon \end{aligned}$$

$$S \Rightarrow abS \Rightarrow ababS$$

$$S \Rightarrow abS \Rightarrow abbAc$$

$$S \Rightarrow abS \Rightarrow abd$$

$$S \Rightarrow bAc \Rightarrow baAc$$

$$S \Rightarrow bAc \Rightarrow bc$$

1. *d*

2. *abd*

3. *bc*

Generate All Possible Sentences from a CF grammar

Length 3 derivations:

$$\begin{aligned} S &\rightarrow abS \mid bAc \mid d \\ A &\rightarrow aA \mid \epsilon \end{aligned}$$

- | | |
|---|------------|
| $S \Rightarrow abS \Rightarrow ababS \Rightarrow abababS$ | 1. d |
| $S \Rightarrow abS \Rightarrow ababS \Rightarrow ababbAc$ | 2. abd |
| $S \Rightarrow abS \Rightarrow ababS \Rightarrow ababd$ | 3. bc |
| $S \Rightarrow abS \Rightarrow abbAc \Rightarrow abbaAc$ | 4. $ababd$ |
| $S \Rightarrow abS \Rightarrow abbAc \Rightarrow abbc$ | 5. $abbc$ |
| $S \Rightarrow bAc \Rightarrow baAc \Rightarrow baaAc$ | 6. bac |
| $S \Rightarrow bAc \Rightarrow baAc \Rightarrow bac$ | 7. \dots |

Generate All Possible Sentences from a CF grammar

Length 3 derivations:

$S \rightarrow abS \mid bAc \mid d$ $A \rightarrow aA \mid \epsilon$	$S \Rightarrow abS \Rightarrow ababS \Rightarrow abababS$	1. d
	$S \Rightarrow abS \Rightarrow ababS \Rightarrow ababbAc$	2. abd
	$S \Rightarrow abS \Rightarrow ababS \Rightarrow ababd$	3. bc
	$S \Rightarrow abS \Rightarrow abbAc \Rightarrow abbaAc$	4. $ababd$
	$S \Rightarrow abS \Rightarrow abbAc \Rightarrow abbc$	5. $abbc$
	$S \Rightarrow bAc \Rightarrow baAc \Rightarrow baaAc$	6. bac
	$S \Rightarrow bAc \Rightarrow baAc \Rightarrow bac$	7. \dots

Given a grammar G , is there an algorithm to find out if $s \in L(G)$?

Generate All Possible Sentences from a CF grammar

Length 3 derivations:

$S \Rightarrow abS \Rightarrow ababS \Rightarrow abababS$ 1. d

$S \Rightarrow abS \Rightarrow ababS \Rightarrow ababbAc$ 2. abd

$S \Rightarrow abS \Rightarrow ababS \Rightarrow ababd$ 3. bc

$S \Rightarrow abS \Rightarrow abbAc \Rightarrow abbaAc$ 4. $ababd$

$S \Rightarrow abS \Rightarrow abbAc \Rightarrow abbc$ 5. $abbc$

$S \Rightarrow bAc \Rightarrow baAc \Rightarrow baaAc$ 6. bac

$S \Rightarrow bAc \Rightarrow baAc \Rightarrow bac$ 7. \dots

$S \rightarrow abS \mid bAc \mid d$

$A \rightarrow aA \mid \epsilon$

Given a grammar G , is there an algorithm to find out if $s \in L(G)$?

Almost, but not quite.

CFG problems

- ▶ Any regular language.
- ▶ $a^n b^n$
- ▶ $a^n b^{2n}$
- ▶ $a^n b^{3n}$
- ▶ $a^{4n+5} b^{3n+2}$
- ▶ $a^n b^m a^n$
- ▶ Even length palindromes
- ▶ Odd length palindromes
- ▶ All palindromes
- ▶ All strings with the same number of a 's and b 's
- ▶ $\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$

Removing ϵ from grammars

$$S \rightarrow aDaE$$

$$D \rightarrow bD \mid E$$

$$E \rightarrow cE \mid \epsilon$$

- ▶ Find all nonterminals N such that $N \Rightarrow^* \epsilon$.
- ▶ Make new rules from old by removing one or more of the null nonterminals.
- ▶ Remove all null productions $N \rightarrow \epsilon$.
- ▶ May have to keep $S \rightarrow \epsilon$, but only if $\epsilon \in L(S)$.

Removing ϵ from grammars

- ▶ Null nonterminals: D and E

	Original Production	New Productions
	$S \rightarrow aDaE$	$S \rightarrow aaE \mid aDa \mid aa$
	$D \rightarrow bD$	$D \rightarrow b$
	$D \rightarrow E$	$D \rightarrow \epsilon$
$S \rightarrow aDaE$	$E \rightarrow cE$	$E \rightarrow c$
$D \rightarrow bD \mid E$	$E \rightarrow \epsilon$	none
$E \rightarrow cE \mid \epsilon$		

Final grammar:

$$\begin{aligned} S &\rightarrow aDaE \mid aaE \mid aDa \mid aa \\ D &\rightarrow bD \mid b \mid E \\ E &\rightarrow cE \mid c \end{aligned}$$

Chomsky Normal Form

- ▶ All rules must be in one of these forms:

$$A \rightarrow BC$$

$$A \rightarrow a$$

$$S \rightarrow \epsilon$$

- ▶ A , B and C are nonterminals, a is a single terminal, and S is the start symbol.
- ▶ The last rule is necessary only if the language contains ϵ .
- ▶ If a grammar is in CNF, what do we know about the tree?
- ▶ If a grammar is in CNF, how long is a derivation?

Converting to Chomsky Normal Form

1. Add S_0 , a new start symbol, and the rule $S_0 \rightarrow S$.
 - ▶ Only necessary if S is recursive
2. Eliminate ϵ rules.
 - ▶ Except possibly $S_0 \rightarrow \epsilon$
3. Eliminate **unit** rules, $A \rightarrow B$:
 - ▶ Find all rules $B \rightarrow W$, where W is a string longer than one.
 - ▶ Add $A \rightarrow W$ for all of them.
4. Fix longer rules:
 - ▶ Replace $A \rightarrow UVWXYZ$ with
 - ▶ $A \rightarrow UA_1$, $A_1 \rightarrow VA_2$, $A_2 \rightarrow WA_3$, $A_3 \rightarrow XA_4$, $A_4 \rightarrow YZ$.
5. For each terminal x , add a rule $X \rightarrow x$ and replace all terminals in long (length two) strings with the corresponding nonterminals.

Example converting to Chomsky Normal Form

$$S \rightarrow aSb \mid T$$

$$T \rightarrow cT \mid \epsilon$$

Step 1: $S_0 \rightarrow S$
 $S \rightarrow aSb \mid T$
 $T \rightarrow cT \mid \epsilon$

Step 4: $S_0 \rightarrow aD \mid ab \mid cT \mid c \mid \epsilon$
 $S \rightarrow aD \mid ab \mid cT \mid c$
 $D \rightarrow Sb$
 $T \rightarrow cT \mid c$

Step 2: $S_0 \rightarrow S \mid \epsilon$
 $S \rightarrow aSb \mid ab \mid T$
 $T \rightarrow cT \mid c$

Step 5: $S_0 \rightarrow AD \mid AB \mid CT \mid c \mid \epsilon$
 $S \rightarrow AD \mid AB \mid CT \mid c$
 $D \rightarrow SB$
 $T \rightarrow CT \mid c$

Step 3: $S_0 \rightarrow aSb \mid ab \mid cT \mid c \mid \epsilon$
 $S \rightarrow aSb \mid ab \mid cT \mid c$
 $T \rightarrow cT \mid c$

$$A \rightarrow a$$
$$B \rightarrow b$$
$$C \rightarrow c$$

Note on Eliminating Unit Rules

If we have two mutually recursive unit rules, for example in the following grammar:

$$\begin{aligned}A &\rightarrow B \mid XY \mid UVW \\ B &\rightarrow A \mid DE \mid FGH\end{aligned}$$

It doesn't hurt to eliminate both of them, so long as the remainder of the clauses is added to each:

$$\begin{aligned}A &\rightarrow XY \mid UVW \mid DE \mid FGH \\ B &\rightarrow DE \mid FGH \mid XY \mid UVW\end{aligned}$$

Any derivation that used the two unit rules, such as

$$A \Rightarrow B \Rightarrow A \Rightarrow B \Rightarrow FGH$$

Can be replaced by a short-circuited version

$$A \Rightarrow FGH$$

Consequences of Chomsky Normal Form

- ▶ What do all trees look like?
- ▶ What do all derivations look like?
- ▶ How long are the derivations?

Consequences of Chomsky Normal Form

- ▶ What do all trees look like?
- ▶ What do all derivations look like?
- ▶ How long are the derivations?
- ▶ There are finitely many derivations for a given string.
- ▶ To find a parse we can exhaustively search all derivations of this length or less.

Greibach Normal Form

- ▶ All rules must be in one of these forms:

$$A \rightarrow aB_1B_2 \dots B_n$$

$$S \rightarrow \epsilon$$

- ▶ $B_1B_2 \dots B_n$ is a (possibly empty) string of nonterminals.
- ▶ The last rule is needed only if the language contains ϵ .
- ▶ There can be no left recursion.
- ▶ What do the trees look like?
- ▶ How long are the derivations?

Converting to Greibach Normal Form

1. Add S_0 , a new start symbol, and the rule $S_0 \rightarrow S$.
2. Eliminate **unit** rules, $A \rightarrow B$.
3. Remove left recursion.
4. Eliminate ϵ rules.
5. Make substitutions as needed.
 - ▶ Can be very difficult and explode to many rules.

Consequences of Greibach Normal Form

- ▶ All derivations of a string of length n have n steps.
- ▶ There are only finitely many derivations of length n or less.
- ▶ To find a parse we can exhaustively search all derivations of length n or less.
- ▶ There exists an **effective procedure** to find a parse for a CFL.

Pumping Lemma for CF Languages

- ▶ If a CFL is infinite, it must have recursion in it somewhere:

$$S \rightarrow uNy$$

$$N \rightarrow vNx \mid w$$

- ▶ This means derivations like this are possible:

$$S \Rightarrow uNy \Rightarrow uvNxy \Rightarrow uvvNxxxy \Rightarrow \dots \Rightarrow uv^4Nx^4y \Rightarrow uv^4wx^4y$$

Pumping Lemma for CF Languages

Theorem

If a language L is context-free, then there exists some $p \geq 1$ such that any string s in L with $|s| \geq p$ can be written as

$$s = abcde$$

with substrings $a, b, c, d, e \in (V \cup \Sigma)^$, such that*

1. $|bcd| \leq p$
2. $|bd| \geq 1$
3. ab^ncd^ne is in L for all $n \geq 0$

Proof that $L = 0^n 1^n 2^n$ is not CF

- ▶ Suppose L is CF, then p exists as in the theorem.
- ▶ $0^p 1^p 2^p \in L$ by definition.
- ▶ From theorem, $0^p 1^p 2^p = abcde$ and $ab^n cd^n e \in L$ for all n , but we don't know which parts are where.
- ▶ Case 1: b or d contains two different digits.
 - ▶ Then $ab^2 cd^2 e$ must have letters out of numerical order.
 - ▶ Then $ab^2 cd^2 e \notin L$, contradiction.
- ▶ Case 2: b and d each contain only one kind of digit.
 - ▶ Then $ab^2 cd^2 e$ contains more of 1 or 2 kinds of digit, not all three.
 - ▶ Then $ab^2 cd^2 e \notin L$, contradiction.

Pumping Lemma exercises (some are hard)

Show that each of the following languages is not CF.

- ▶ 1^n where n is prime
- ▶ 1^m where $m = n^2$
- ▶ $0^\ell 1^m 2^n$ where $\ell < m < n$
- ▶ $0^n 1^n 2^i$ where $i \leq n$
- ▶ ww where $w \in (0 + 1)^*$
- ▶ $0^n 1^n 2^n$

CF Languages are closed under union, product, and closure

- ▶ Let S_1 and S_2 be the start symbols for L_1 and L_2 .
- ▶ A grammar for $L_1 \cup L_2$ can be constructed starting with

$$S \rightarrow S_1 \mid S_2$$

- ▶ A grammar for $L_1 L_2$ can be constructed starting with

$$S \rightarrow S_1 S_2$$

- ▶ A grammar for L_1^* can be constructed starting with

$$S \rightarrow S_1 S \mid \epsilon$$

CF languages are NOT closed under complement

- ▶ $L = a^\ell b^m c^n$ where either $\ell \neq m$ or $m \neq n$ is CF
 - ▶ (exercise)
- ▶ The complement of L is $a^n b^n c^n$, which is not CF
 - ▶ (see previous)

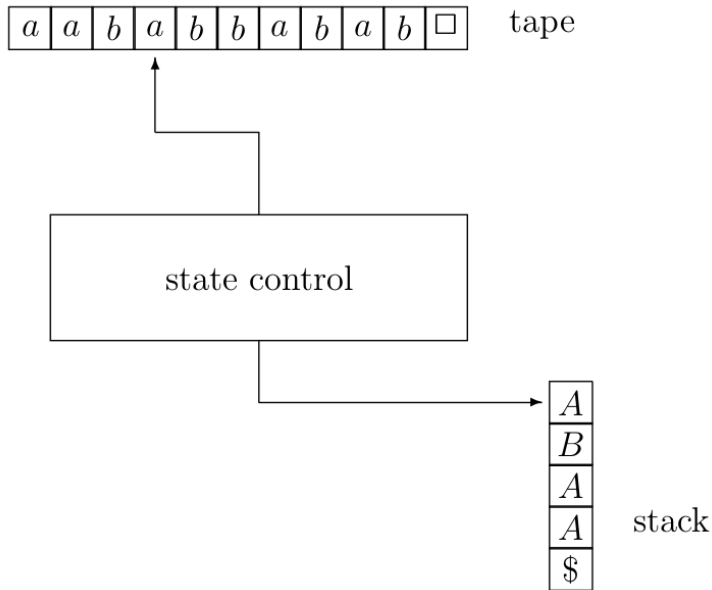
CF languages are NOT closed under intersection

- ▶ $L_1 = a^m b^m c^n$ is CF
 - ▶ (exercise)
- ▶ $L_2 = a^m b^n c^n$ is CF
 - ▶ (exercise)
- ▶ $L_1 \cap L_2 = a^n b^n c^n$, which is not CF
 - ▶ (see previous)

The intersection of a CF language and a regular language is CF

- ▶ Given a PDA for one and a DFSA for the other:
- ▶ Create a new PDA with states that are the cross product of the states of the two machines.
- ▶ As input is processed, run both machines in parallel.
- ▶ Accept if both accept.

Pushdown automata



Tape drives



Deterministic pushdown automata

Definition 3.5.1 A *deterministic pushdown automaton* is a 5-tuple $M = (\Sigma, \Gamma, Q, \delta, q)$ where

1. Σ is a finite set, called the *tape alphabet*; the blank symbol \square is not in Σ .
2. Γ is a finite set, called the *stack alphabet*; this alphabet contains the special symbol $\$$.
3. Q is a finite set, whose elements are called *states*.
4. q is an element of Q , called the *start state*.
5. δ is the *transition function*, which is a function

$$\delta : Q \times (\Sigma \cup \{\square\}) \times \Gamma \rightarrow Q \times \{N, R\} \times \Gamma^*$$

Given a state, a tape letter, and the top letter on the stack, (r, a, A) , δ determines the next state, whether or not to move the tape head, and the replacement for A on the stack:

$$\delta(r, a, A) = (r', \sigma, w)$$

Computation and Termination

Start: The automaton is in start state q , the tape head is at the leftmost symbol of the input string $a_1a_2\dots a_n$, and the stack contains only the special symbol $\$$.

Computation: The state, input and the stack are transformed, step by step, by following the transition function δ .

Termination: The PDA *terminates* when the stack becomes empty.

Acceptance: The PDA *accepts* $a_1a_2\dots a_n$ if

1. the automaton terminates on this input, and
2. at the time of termination, the tape head is immediately to the right of a_n .

Properly nested parentheses

$qa\$ \rightarrow qR\S	because of the a , S is pushed onto the stack
$qaS \rightarrow qRSS$	because of the a , S is pushed onto the stack
$qbS \rightarrow qR\epsilon$	because of the b , the top element is popped from the stack
$qb\$ \rightarrow qN\epsilon$	the number of bs read is larger than the number of as read; the stack is made empty (hence, the computation terminates before the entire string has been read), and the input string is rejected
$q\Box\$ \rightarrow qN\epsilon$	the entire input string has been read; the stack is made empty, and the input string is accepted
$q\Box S \rightarrow qNS$	the entire input string has been read, it contains more as than bs ; no changes are made (thus, the automaton does not terminate), and the input string is rejected

$0^n 1^n$

$q_0 0\$ \rightarrow q_0 R\S	push S onto the stack
$q_0 0S \rightarrow q_0 RSS$	push S onto the stack
$q_0 1\$ \rightarrow q_0 N\$$	first symbol in the input is 1; loop forever
$q_0 1S \rightarrow q_1 R\epsilon$	first 1 is encountered
$q_0 \square\$ \rightarrow q_0 N\epsilon$	input string is empty; accept
$q_0 \square S \rightarrow q_0 NS$	input only consists of 0s; loop forever
$q_1 0\$ \rightarrow q_1 N\$$	0 to the right of 1; loop forever
$q_1 0S \rightarrow q_1 NS$	0 to the right of 1; loop forever
$q_1 1\$ \rightarrow q_1 N\$$	too many 1s; loop forever
$q_1 1S \rightarrow q_1 R\epsilon$	pop top symbol from the stack
$q_1 \square\$ \rightarrow q_1 N\epsilon$	accept
$q_1 \square S \rightarrow q_1 NS$	too many 0s; loop forever

b in the middle

$qa\$ \rightarrow qR\S	push S onto the stack
$qaS \rightarrow qRSS$	push S onto the stack
$qb\$ \rightarrow q'R\$$	reached the middle
$qb\$ \rightarrow qR\S	did not reach the middle; push S onto the stack
$qbS \rightarrow q'RS$	reached the middle
$qbS \rightarrow qRSS$	did not reach the middle; push S onto the stack
$q\Box\$ \rightarrow qN\$$	loop forever
$q\Box S \rightarrow qNS$	loop forever
$q'a\$ \rightarrow q'N\epsilon$	stack is empty; terminate, but reject, because the entire input string has not been read
$q'aS \rightarrow q'Re$	pop top symbol from stack
$q'b\$ \rightarrow q'N\epsilon$	stack is empty; terminate, but reject, because the entire input string has not been read
$q'bS \rightarrow q'Re$	pop top symbol from stack
$q'\Box\$ \rightarrow q'N\epsilon$	accept
$q'\Box S \rightarrow q'NS$	loop forever