Mathematical Foundation of Computer Sciences V

Context-Free Grammar & PDA

Guoqiang Li

School of Software, Shanghai Jiao Tong University

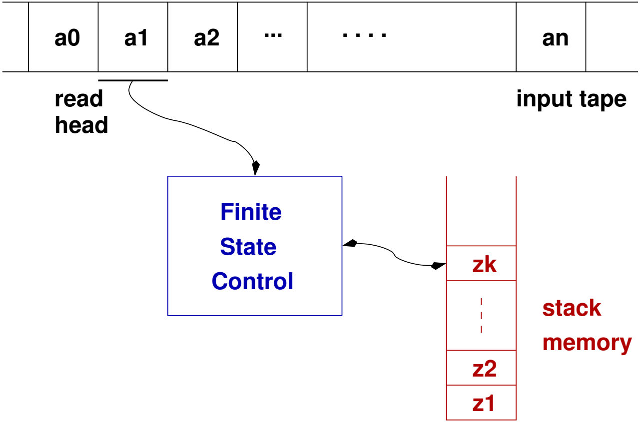
A Program Example

```
void m() {
    if (?) {
        s(); right();
        if (?) m();
    } else {
        up(); m(); down();
    }
}
```

```
void s() {
    if (?) return;
    up(); m(); down();
}

main() {
  s();
}
```

A Program Example



Context Free Languages

An Example

The grammar

$$\begin{array}{rcl} A & \to & 0A1 \\ A & \to & B \\ B & \to & \# \end{array}$$

A derivation:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000\#111.$$

An Example

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
\langle \text{NOUN-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{VERB-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{PREP-PHRASE} \rangle &\rightarrow \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle \\
\langle \text{CMPLX-NOUN} \rangle &\rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \\
\langle \text{CMPLX-VERB} \rangle &\rightarrow \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle \\
\langle \text{ARTICLE} \rangle &\rightarrow \text{a} \mid \text{the} \\
\langle \text{NOUN} \rangle &\rightarrow \text{boy} \mid \text{girl} \mid \text{flower} \\
\langle \text{VERB} \rangle &\rightarrow \text{touches} \mid \text{likes} \mid \text{sees} \\
\langle \text{PREP} \rangle &\rightarrow \text{with}
\end{aligned}
$$

An Example

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{CMPLX-NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \text{a } \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \text{a boy} \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \text{a boy} \langle \text{CMPLX-VERB} \rangle \\
&\Rightarrow \text{a boy} \langle \text{VERB} \rangle \\
&\Rightarrow \text{a boy sees.}
\end{aligned}
$$

Context-Free Grammar

Definition

A context-free grammar (CFG) is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the variables,
2. Σ is a finite set, disjoint from V, called the terminals,
3. R is a finite set of rules, with each rule being a variable and a string of variables and terminals,
4. $S \in V$ is the start variable.

Derivations

Let u, v, w be strings of variables and terminals, and

$$A \to w \quad \in R$$

Then uAv yields uwv: $uAv \Rightarrow uwv$.

u derives v, written $u \overset{*}{\Rightarrow} v$, if

- $u = v$, or
- there is a sequence $u_1, u_2, \ldots, u_k$ for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v.$$

The language of the grammar is $\{w \in \Sigma^* \mid S \overset{\star}{\Rightarrow} w\}$.

Which is a context-free language(CFL).

Examples

1. Language $\{0^n1^n \mid n \geq 0\}$, grammar

$$S_1 \to 0S_11 \mid \epsilon.$$

2. Language $\{1^n0^n \mid n \geq 0\}$, grammar

$$S_2 \to 1S_20 \mid \epsilon.$$

3. Language $\{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$, grammar

$$\begin{array}{rcl} S & \to & S_1 \mid S_2 \\ S_1 & \to & 0S_11 \mid \epsilon \\ S_2 & \to & 1S_20 \mid \epsilon. \end{array}$$

Ambiguity

$$\langle EXPR \rangle \to \langle EXPR \rangle + \langle EXPR \rangle \mid \langle EXPR \rangle \times \langle EXPR \rangle \mid (\langle EXPR \rangle) \mid a$$

The string $a + a \times a$ have two different derivations:

1. $\langle EXPR \rangle \Rightarrow \langle EXPR \rangle \times \langle EXPR \rangle \Rightarrow \langle EXPR \rangle + \langle EXPR \rangle \times \langle EXPR \rangle \overset{*}{\Rightarrow} a + a \times a$
2. $\langle EXPR \rangle \Rightarrow \langle EXPR \rangle + \langle EXPR \rangle \Rightarrow \langle EXPR \rangle + \langle EXPR \rangle \times \langle EXPR \rangle \overset{*}{\Rightarrow} a + a \times a$

Leftmost derivations

A derivation of a sting w in a grammar G is a leftmost derivation if at every step the leftmost remaining variable is the one replaced.

Ambiguity

A string w is derived ambiguously is a context free grammar G if it has two or more different leftmost derivations.

Grammar G is ambiguous if it generates some string ambiguously..

$\{a\}$ has two different grammars $S_1 \to S_2 \mid a; S_2 \to a$ and $S \to a$. The first is ambiguous, while the second is not.

$\{a^i b^j c^k \mid i = j \ \vee \ j = k\}$ is inherently ambiguous,i.e., its every grammar is ambiguous.

Chomsky Normal Form

A context-free grammar is in Chomsky normal form if every rule is of the form

$$\begin{array}{rcl} A & \rightarrow & BC \\ A & \rightarrow & a \end{array}$$

where a is any terminal and A, B and C are any variables, except that B and C may be not the start variable.

In addition, we permit the rule $S \rightarrow \epsilon$, where S is the start variable.

Theorem

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Proof of the Theorem

1. Add a new start variable S_0 with the rule $S_0 \to S$, where S is the original start variable.
2. Remove every $A \to \epsilon$, where $A \neq S$.
   For each occurrence of A on the right-hand side of a rule, we add a new rule with that occurrence deleted.
   a) $R \to uAv$ will be replace by $R \to uv$;
   b) Do the above operation for each occurrence of A: e.g. $R \to uAvAw$, will be replaced by $R \to uvAw \mid uAvw \mid uvw$.
   c) For $R \to A$, we add $R \to \epsilon$ unless we had previously removed $R \to \epsilon$.
3. Remove every $A \to B$.
   Whenever a rule $B \to u$ appears, where u is a string of variables and terminals, we add the rule $A \to u$ unless this was previously removed.

Proof of the Theorem (cont.)

1. New start variable S_0.
2. Remove every $A \to \epsilon$.
3. Remove every $A \to B$.
4. Replace each rule $A \to u_1 u_2 \cdots u_k$ with $k \geq 3$ and each u_i is a variable or terminal with the rules

$$A \to u_1 A_1, A_1 \to u_2 A_2, A_2 \to u_2 A_3, \cdots, \text{ and } A_{k-2} \to u_{k-1} u_k.$$

   The A_is are new variables. We replace any terminal u_i with the new variable U_i and add $U_i \to u_i$.

An Example

Applying the first step to make a new start variable appears on the right.

$$\begin{aligned} S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

An Example

Remove ε-rules $B \to \varepsilon$ on the left, and $A \to \varepsilon$ on the right.

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid aB \mid a \\ A &\to B \mid S \mid \varepsilon \\ B &\to b \mid \varepsilon \end{aligned}$$

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid aB \mid a \mid AS \mid SA \mid S \\ A &\to B \mid S \mid \varepsilon \\ B &\to b \end{aligned}$$

An Example

Remove unit rules $S \to S$ on the left, and $S_0 \to S$ on the right.

$$\begin{array}{rcl} S_0 & \to & S \\ S & \to & ASA \mid aB \mid a \mid AS \mid SA \\ A & \to & B \mid S \\ B & \to & b \end{array}$$

$$\begin{array}{rcl} S_0 & \to & S \mid ASA \mid aB \mid a \mid AS \mid SA \\ S & \to & ASA \mid aB \mid a \mid AS \mid SA \\ A & \to & B \mid S \\ B & \to & b \end{array}$$

An Example

Remove unit rules $A \to B$ on the left, and $A \to S$ on the right.

$$
\begin{aligned}
S_0 &\to ASA \mid aB \mid a \mid AS \mid SA \\
S &\to ASA \mid aB \mid a \mid AS \mid SA \\
A &\to B \mid S \mid b \\
B &\to b
\end{aligned}
$$

$$
\begin{aligned}
S_0 &\to ASA \mid aB \mid a \mid AS \mid SA \\
S &\to ASA \mid aB \mid a \mid AS \mid SA \\
A &\to S \mid b \mid ASA \mid aB \mid a \mid AS \mid SA \\
B &\to b
\end{aligned}
$$

An Example

Convert the remaining rules into the proper form by adding additional variables and rules.

$$
\begin{aligned}
S0 &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\
S &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\
A &\rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS \\
A_1 &\rightarrow SA \\
U &\rightarrow a \\
B &\rightarrow b
\end{aligned}
$$

Efficient Derivation

Theorem

If G is a context-free grammar in Chomsky normal form then any $w \in L(G)$ such that $w \neq \varepsilon$ can be derived from the start state in exactly $2|w| - 1$ steps.

Pushdown automata

Definition

A pushdown automata (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite set of input alphabet,
3. Γ is a finite set of stack alphabet,
4. $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $F \subseteq Q$ is the set of accept states.

Formal Definition of Computation

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a pushdown automaton. M accepts input w if w can be written as $w = w_1 \ldots w_m$, and sequences of states $r_0, r_1, \ldots, r_m \in Q$ and strings $s_0, s_1, \ldots, s_m \in \Gamma^*$ exist that satisfy the following three conditions.

1. $r_0 = q_0$ and $s_0 = \epsilon$.
2. For $i = 0, \ldots, m-1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\epsilon$ and $t \in \Gamma^*$.
3. $r_m \in F$.

PDA for $\{0^n 1^n \mid n \geq 0\}$

$$
\begin{array}{lcl}
Q & = & \{q_1, q_2, q_3, q_4\}, \\
\Sigma & = & \{0, 1\}, \\
\Gamma & = & \{0, \$\}, \\
q_1 & & \text{is the start state} \\
F & = & \{q_1, q_4\}
\end{array}
$$

The transition function is defined by the following table, wherein blank entries signify $\emptyset$

Input:	0			1			ϵ		
Stack:	0	\$	ϵ	0	\$	ϵ	0	\$	ϵ
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \epsilon)\}$					
q_3				$\{(q_3, \epsilon)\}$				$\{(q_4, \epsilon)\}$	
q_4									

Equivalence of CFL and PDA

Theorem

A language is context free if and only if some pushdown automaton recognizes it.

Every Context-Free Language Can Be Recognized by a PDA

1. Place the marker symbol \$ and the start variable on the stack.
2. Repeat the following steps forever.
   2.1 If the top of stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule.
   2.2 If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, repeat.If they do not match, reject on this branch of the nondeterminism.
   2.3 If the top of stack is the symbol \$, enter the accept state. Doing so accepts the input if it has all been read.

Push a long string in "one step"

Let q and r be states of the PDA and let $a \in \Sigma_\varepsilon$ and $s \in \Gamma_\varepsilon$.

We want the PDA to go from q to r when it reads a and pops s.
Furthermore, we want it to push the entire string $u = u_1 \dots u_l$ on the stack at the same time.

$$
\begin{aligned}
&(q_1, u_l) \in \delta(q, a, s)\\
&\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, u_{l-1})\}\\
&\delta(q_2, \varepsilon, \varepsilon) = \{(q_3, u_{l-2})\}\\
&\vdots\\
&\delta(q_{l-1}, \varepsilon, \varepsilon) = \{(r, u_1)\}
\end{aligned}
$$

We use the abbreviation

$$(r, u) \in \delta(q, a, s)$$

Proof

We construct a pushdown automaton P as follows.

The states of P are

$$Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$$

where E is the set of states we need for the construction in the previous slide.

For the transition function,

- $\delta(q_{start}, \varepsilon, \varepsilon) = \{(q_{loop}, S\$)\}$
- $\delta(q_{loop}, \varepsilon, A) = \{(q_{loop}, w) \mid A \to w \text{ is a rule in the given grammar}\}$
- $\delta(q_{loop}, a, a) = \{(q_{loop}, \varepsilon)\}$
- $\delta(q_{loop}, \varepsilon, \$) = \{(q_{accept}, \varepsilon)$

Every Language Recognized by a PDA is Context Free

Let P be a PDA. For each pair of states p and q, the grammar has a variable A_{pq} which generates

all strings taking P from p with an empty stack to q with an empty stack.

We modify P such that:

1. It has a single accept state q_{accept}.
2. It empties its stack before accepting.
3. Each transition either pushes a symbol onto the stack or pops one off the stack, but it does not do both at the same time.

Inductive Definition of A_{pq}

Two possibilities occur during P's computation on an input string x.

1. The symbol popped at the end is the symbol that was pushed at the beginning. Then, we have a rule $A_{pq} \to aA_{rs}b$.
2. Otherwise, we have a rule $A_{pq} \to A_{pr}A_{rq}$.

Proof (1)

Assume $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$.

The variables of the desired context-free grammar G are

$$\{A_{pq} \mid p, q \in Q\}$$

in which the start variable is $A_{q_0, q_{accept}}$.

For the rules:

R1 For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $(r, u) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, u)$, then G has the rule

$$A_{pq} \to aA_{rs}b$$

R2 For each $p, q, r \in Q$, G has the rule

$$A_{pq} \to A_{pr}A_{rq}$$

R3 For each $p \in Q$, G has the rule

$$A_{pp} \to \varepsilon$$

Proof (2)

Claim

If A_{pq} generates x, the x can bring P from p with empty stack to q with empty stack.

Basis: The derivation has 1 step. A derivation with a single step must use a rule whose right-hand side contains no variables. The only rules in G where no variables occur on the right-hand side are $A_{pp} \to \varepsilon$.

Induction step: The derivation has $k+1$ step with $A_{pq} \Rightarrow^* x$. Thus, either $A_{pq} \Rightarrow aA_{rs}b$ or $A_{pq} \Rightarrow A_{pr}A_{rq}$.

In case $A_{pq} \Rightarrow aA_{rs}b$ the claim follows from (R1) and the induction hypothesis.

For $A_{pq} \Rightarrow A_{pr}A_{rq}$, there exist y and z with $x = yz$ such that $A_{pr} \Rightarrow^* y$ and $A_{qr} \Rightarrow^* z$ both in at most k steps. The claim then again follows from the induction hypothesis.

Proof (3)

Claim

If x can bring P from p with empty stack to q with empty stack, then A_{pq} generates x.

Basis: The computation has 0 steps.

$x = \varepsilon$ and we have $A_{pp} \to \varepsilon$.

Induction step:

If the stack is always non-empty in the middle of the computation, then:

- There is a u which is pushed in the first move and popped in the last move.
- In the first move, let a be the input and r be the state after; in the last move let b be the input and s be the state before.
- We deduce $(r, u) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, u)$. Hence, G has the rule $A_{pq} \to aA_{rs}b$.

We can conclude by the induction hypothesis.

If the stack becomes empty in the middle of the computation, the claim then again follows from the induction hypothesis.