Theory of Computation (IV)

Yijia Chen
Fudan University

Review

Context-Free Languages

Context-free grammars

Definition

A context-free grammar (CFL) 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, and
4. $S \in V$ is the start variable.

Derivations

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

$$A \to w$$

is a rule of the grammar. We say that uAv yields uwv, written $uAv \Rightarrow uwv$.

Say that u derives v, written $u \overset{*}{\Rightarrow} v$, if $u = v$ or if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and

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

The language of the grammar is

$$\left\{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \right\},$$

which is a context-free language (CFL).

Examples

1. The language $\left\{0^n 1^n \;\middle|\; n \geq 0\right\}$ has a grammar:

$$S_1 \to 0S_1 1|\varepsilon.$$

2. The language $\left\{1^n 0^n \;\middle|\; n \geq 0\right\}$ has a grammar:

$$S_2 \to 1S_2 0|\varepsilon.$$

3. The language

$$\left\{0^n 1^n \;\middle|\; n \geq 0\right\} \cup \left\{1^n 0^n \;\middle|\; n \geq 0\right\}$$

has a grammar:

$$\begin{aligned} S &\to S_1 \mid S_2 \\ S_1 &\to 0S_1 1|\varepsilon \\ S_2 &\to 1S_2 0|\varepsilon. \end{aligned}$$

$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle | \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle | \big(\langle \text{EXPR} \rangle \big) | \texttt{a}.$$

The string a+a×a have two different derivations:

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

Leftmost derivations

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

Ambiguity

Definition

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

Grammar G is ambiguous if it generates some string ambiguously.

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

$$\left\{\mathrm{a}^i\mathrm{b}^j\mathrm{c}^k \;\middle|\; i = j \text{ or } j = k\right\}$$

is inherently ambiguous, i.e., its every grammar is ambiguous.

Chomsky normal form

Definition

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

$$A \to BC$$
$$A \to a$$

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 $S \to \varepsilon$, where S is the start variable.

Theorem

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

Pushdown Automata

Pushdown automata

Definition

A pushdown automaton (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. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Formal definition of computation

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

1. $r_0 = q_0$ and $s_0 = \varepsilon$.
2. For $i = 0, \dots, m-1$, 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 $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$.
3. $r_m \in F$.

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

$Q = \left\{q_1, q_2, q_3, q_4\right\}$,
$\Sigma = \{0, 1\}$,
$\Gamma = \{0, \$\}$,
$F = \left\{q_1, q_4\right\}$, and

δ is given 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			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Theorem

A language is context free if 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 $u = u_1 \cdots u_\ell$ on the stack at the same time.

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

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

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

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

For the transition function,

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

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

The variables of the desired context-free grammar G are

$$\big\{A_{pq} \;\big|\; p, q \in Q\big\},$$

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

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 $A_{pq} \stackrel{*}{\Rightarrow} x$. 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 $A_{pr} \stackrel{*}{\Rightarrow} y$ and $A_{qr} \stackrel{*}{\Rightarrow} z$ 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.

Non-Context-Free Languages

The pumping lemma for context-free languages

Lemma

If A is a context-free language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided as $s = uvxyz$ satisfying the conditions

1. *for each $i \geq 0$, $uv^i xy^i z \in A$.*
2. $|vy| > 0$, *and*
3. $|vxy| \leq p$.

Proof (1)

Let G be a CFG for CFL A. Let b be the maximum number of symbols in the right-hand side of a rule. In any parse tree using this grammar, every node can have no more than b children. So, if the height of the parse tree is at most h, the length of the string generated is at most b^h. Conversely, if a generated string is at lest $b^h + 1$ long, each of its parse trees must be at least $h + 1$ high.

We choose the pumping length

$$p = b^{|V|+1}.$$

For any string $s \in A$ with $|s| \geq p$, any of its parse trees must be at least $|V| + 1$ high.

Proof (2)

Let τ be one parse tree of s with smallest number of nodes, whose height is at least $|V| + 1$. So τ has a path from the root to a leaf of length $|V| + 1$ with $|V| + 2$ nodes. One variable R must appear at least twice in the last $|V| + 1$ variables nodes on this path.

We divide s into $uvxyz$:

- u from the leftmost leaf of τ to the leaf left next to the leftmost leaf of the subtree hanging on the first R,
- v from the leftmost leaf of the subtree hanging on the first R to the leaf left next to the leftmost leaf of the subtree hanging on the second R,
- x for all the leaves of the subtree hanging on the second R,
- y from the leaf right next to the rightmost leaf of the subtree hanging on the second R to the rightmost leaf of the subtree hanging on the first R,
- z from the leaf right next to the rightmost leaf of the subtree hanging on the first R to the rightmost leaf of τ.

Proof (3)

If $|vy| = 0$, i.e., $v = y = \varepsilon$, then τ cannot have the smallest number of nodes.

To see $|vxy| \leq p = b^{|V|+1}$, note that vxy is generated by the second R, whose height is at most $|V| + 1$.

Example

$\{a^n b^n c^n \mid n \geq 0\}$ is not context free.

Proof.

Assume otherwise, and let p be the pumping length. Consider $s = a^p b^p c^p$ and divide it to $uvxyz$ according to the Pumping Lemma.

- When both v and y contain only one type of symbols, i.e., one of a, b, c, then uv^2xy^2z cannot contain equal number of a's, b's, and c's.
- If either v or y contains more than one type of symbols, then uy^2wx^2y would have symbols interleaved.

□

Example

$\{ww \mid w \in \{0,1\}^*\}$ is not context free.

Proof.

Assume otherwise, and let p be the pumping length. Consider $s = 0^p1^p0^p1^p$ and divide it to $s = uvxyz$ with $|vxy| \leq p$.

- If vxy occurs only in the first half of s, then the second half of uv^2xy^2z must start with a 1. This is impossible.
- Similarly vxy cannot occur only in the second half of s.
- If vxy straddles the midpoint of s, then pumping s to uyz with the form $0^p1^i0^j1^p$ cannot ensure $i = j = p$.

□

Part Two.
Computability Theory

The Church – Turing Thesis

Turing Machines

Turing machines

Alan Turing in 1936 proposed Turing machines M:

- M uses an infinite tape as its unlimited memory, with a tape head reading and writing symbols and moving around on the tape.

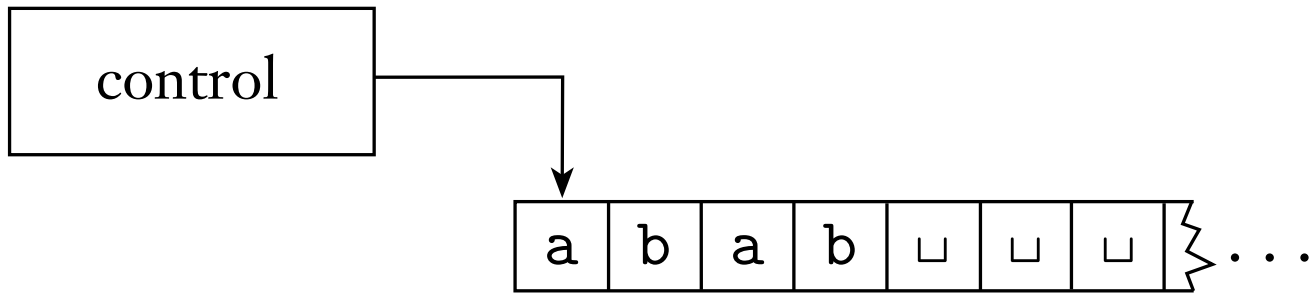
  The tape initially contains only the input string and is blank everywhere else.

- If M needs to store information, it may write this information on the tape. To read the information that it has written, M can move its head back over it.

- M continues computing until it decides to produce an output. The outputs accept and reject are obtained by entering designated accepting and rejecting states.

- If M doesn't enter an accepting or a rejecting state, it will go on forever, never halting.

Schematic of a Turing machine



The difference between finite automata and Turing machines

1. A Turing machine can both write on the tape and read from it.
2. The read-write head can move both to the left and to the right.
3. The tape is infinite.
4. The special states for rejecting and accepting take effect immediately.

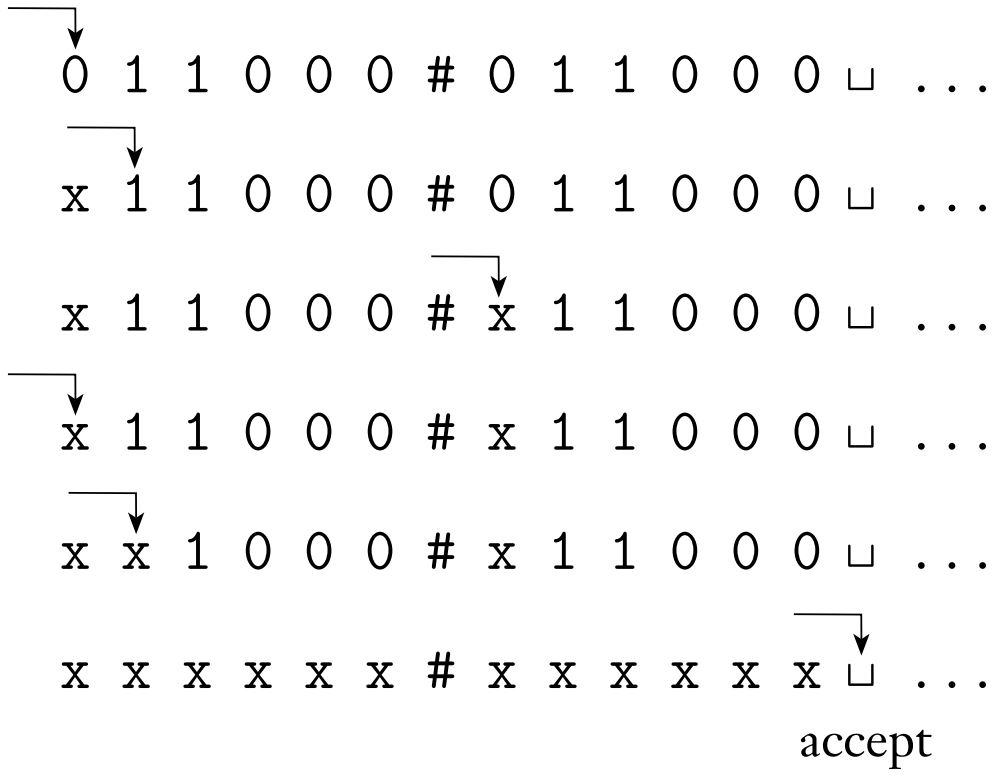
$B = \{w\#w \mid w \in \{0,1\}^*\}$

M_1 on input string w:

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, reject. Cross off symbols as they are checked to keep track of which symbols correspond.

2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, reject; otherwise, accept.

$B = \{w\#w \mid w \in \{0,1\}^*\}$ (cont'd)



Formal definition of a Turing machine

Definition

A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{rejcet}})$, where Q, Σ, Γ are all finite and

1. Q is set of states,
2. Σ is the input alphabet not containing the blank symbol $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Computation by M

- Initially, M receives its input $w = w_1 w_2 \ldots w_n \in \Sigma^*$ on the leftmost n squares of the tape, and the rest of the tape is blank (i.e., filled $\sqcup$).
- The head starts on the leftmost square of the tape.
- As Σ does not contain $\sqcup$, so the first blank appearing on the tape marks the end of the input.
- Once M has started, the computation proceeds according to the rules described by the transition function.
- If M ever tries to move its head to the left off the left-hand end of the tape, the head stays in the same place for that move, even though the transition function indicates L.
- The computation continues until it enters either the accept or reject states, at which point it halts. If neither occurs, M goes on forever.

Configurations

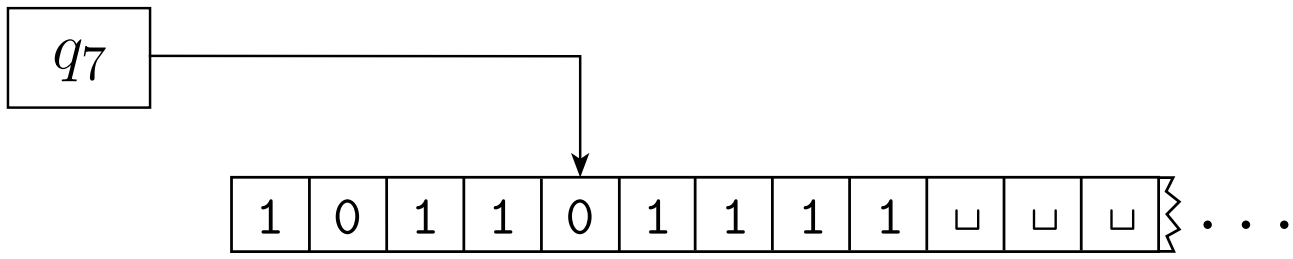
A configuration of a Turing machine consists of

- the current state,
- the current tape contents, and
- the current head location.

By $u\ q\ v$ we mean the configuration where

- the current state is q,
- the current tape contents is uv, and
- the current head location is the first symbol of v.
- The tape contains only blanks following the last symbol of v.

Configurations (cont'd)



A Turing machine with configuration 1011 q_7 01111

Formal definition of computation

Let $a, b, c \in \Gamma$, $u, v \in \Gamma^*$, and $q_i, q_j \in Q$.

1. If $\delta(q_i, b) = (q_j, c, L)$, then

   $ua\ q_i\ bv$ yields $u\ q_j\ acv$.

2. If $\delta(q_i, b) = (q_j, c, R)$, then

   $ua\ q_i\ bv$ yields $uac\ q_j\ v$.

Special cases occur when the head is at one of the ends of the configuration:

1. For the left-hand end, the configuration $q_i\ bv$ yields $q_j\ cv$ if the transition is left moving (because we prevent the machine from going off the left-hand end of the tape), and it yields $c\ q_j\ v$ for the right-moving transition.

2. For the right-hand end, the configuration $ua\ q_i$ is equivalent to $ua\ q_i\ \sqcup$ because we assume that blanks follow the part of the tape represented in the configuration.

Special configurations

- The start configuration of M on input w is the configuration $q_0 w$.
- In an accepting configuration, the state of the configuration is q_{accept}.
- In a rejecting configuration, the state of the configuration is q_{reject}.
- Accepting and rejecting configurations are halting configurations and do not yield further configurations.

Formal definition of computation (cont'd)

M accepts w if there are sequence of configurations $C_1, C_2, \ldots, C_k$ such that

1. C_1 the start configuration of M on w.
2. Each C_i yields C_{i+1}, and
3. C_k is an accepting configuration.

The collection of strings that M accepts is the language of M, or the language recognized by M, denoted $L(M)$.

Definition

A language is Turing-recognizable, if some Turing machine recognizes it.

On an input, the machine M may accept, reject, or loop. By loop we mean that the machine simply does not halt.

If M always halts, then it is a decider. A decider that recognizes some language is said to decide that language.

Definition

A language is Turing-decidable or simply decidable if some Turing machine decides it.