Pushdown Automata

Notes on Automata and Theory of Computation

Chia-Ping Chen

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan ROC

Introduction

- Regular languages correspond to the class of dfa's. Is there a class of automata for context-free languages?
- The class of automata for context-free languages should be able to, among other things, count without limit (for language such as $\{a^n b^n : n \geq 0\}$) and store and match a sequence of symbols in the reverse order (for $\{ww^R : w \in \Sigma^*\}$). This suggests using *stack* as storage.
- The class of automata using stack as storage is called the **pushdown automata (pda)**.

Pushdown Automata

- A **nondeterministic pushdown automaton (npda)** is defined by a septuple (7-tuple)

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z, F),$$

  - Q is a finite set of states
  - Σ is a finite set called **input alphabet**
  - Γ is a finite set called **stack alphabet**
  - $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to 2^{Q \times \Gamma^*}$ is the transition function
  - $q_0 \in Q$ is the initial state
  - $z \in \Gamma$ is the **stack start symbol**
  - $F \subseteq Q$ is the set of final states

Transition Function

- Depending on the current state, the stack symbol on top, and optionally the input symbol, a pda makes a state transition and pushes a *string* of stack symbols back to the stack.
- Let's note a few things about δ,

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to 2^{Q \times \Gamma^*}.$$

  - The value is a *finite* subset of $Q \times \Gamma^*$.
  - A move is possible without using an input symbol.
  - No move is possible with an empty stack.
  - The reading of input never goes "backwards".

Examples of δ

- Suppose $\delta(q_1, a, b) = \{(q_2, cd), (q_3, \lambda)\}$ for npda M. When M is in state q_1, the input symbol is a and the top of stack is b, one of two things can happen:
    - M transits to state q_2 and replace b by cd (c on top of d).
    - M transits to state q_2 and remove b.
- A transition function can be represented by a graph, where an edge from q_i to q_j labeled by a, b, y means

$$(q_j, y) \in \delta(q_i, a, b).$$

Instantaneous Description

- The relevant information at any instant is the current state, q, the *unread* part of input string, w, and the content of stack, u. The triplet (q, w, u) is called an **instantaneous description (ID)** of an npda.
- A **move** (or step) from an ID to another is denoted by the symbol $\vdash$,

$$(q, aw, bx) \vdash (p, w, yx) \;\Leftrightarrow\; (p, y) \in \delta(q, a, b).$$

- A sequence of moves is denoted by $\overset{*}{\vdash}$, e.g.,

$$(q_1, w_1, x_1) \overset{*}{\vdash} (q_2, w_2, x_2).$$

Language Accepted by An npda

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ be an npda. The language accepted by M is the set of strings

$$L(M) = \{w \in \Sigma^* : (q_0, w, z) \overset{*}{\vdash} (p, \lambda, u), p \in F, u \in \Gamma^*\}$$

  - $L(M)$ is the set of input strings that can make M in a final state at the end of the input string.
  - The final stack content is irrelevant.
- It turns out the set of languages acceptable by npda is equal to the set of context-free languages.

An Example

- Consider the language in which each string has an equal number of a's and b's,

$$L = \{w \in \{a,b\}^* : n_a(w) = n_b(w)\}.$$

- The transition graph is drawn in Figure 7.3.

$$M = (\{q_0, q_f\}, \{a,b\}, \{0,1,z\}, \delta, q_0, z, \{q_f\}).$$

- Initially, the stack contains z. For an input a, either 0 is pushed or 1 is popped, depending on the top stack symbol. Similarly, for an input b, either 1 is pushed or 0 is popped. M transits to q_f when it finds input empty and the stack top is z.

Another Example

- Consider the set $L = \{ww^R : w \in \{a, b\}^+\}$.
- The idea for an npda to accept L is to guess the middle of the input string and matches the front substring and the rear substring. The front substring is pushed into the stack, and the rear substring is matched against the stack content (in the reverse order).
- Specifically, we construct an npda

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, z\}, \delta, q_0, z, \{q_2\}),$$

  where state q_0 signals that the input position is in the front part, and state q_1 signals that the input position is in the rear.

npda for cfg

- If L is a context-free language and $\lambda \notin L$, then there exists an npda M such that

$$L = L(M).$$

- Here we are given a cfg for L and want to construct an npda. W.l.o.g., we can assume the grammar is in Greibach normal form, say $G = (V, T, S, P)$. The idea is to make the npda simulate the leftmost derivation of G.
  - The terminal prefix of the sentential form matches the corresponding prefix of the input string.
  - The unprocessed part of sentential form is in the stack.

Construction

- We construct an npda

$$M = (\{q_0, q_1, q_f\}, T, V \cup \{z\}, \delta, q_0, z, \{q_f\}).$$

- The key is δ. Initially, M pushes the start symbol S to the stack and transits to state q_1. That is,

$$\delta(q_0, \lambda, z) = \{(q_1, Sz)\}.$$

For any rule $A \to au$ in P, M reads input a, replace A by u in stack,

$$(q_1, u) \in \delta(q_1, a, A).$$

Finally, M moves to q_f if the input is empty and the stack contains only z.

Proof $L(G) \subseteq L(M)$

- Consider partial derivation

$$S \overset{*}{\Rightarrow} a_1 \ldots a_n A A_2 \ldots A_m \Rightarrow a_1 \ldots a_n b B_1 \ldots B_k A_2 \ldots A_m$$

  By construction of M, if the stack content is $AA_2 \ldots A_m$ after reading $a_1 \ldots a_n$ then it is $B_1 \ldots B_k A_2 \ldots A_m$ after reading $a_1 \ldots a_n b$.
- That is, the stack content matches the variable string in the sentential form, and the input position matches the terminal prefix of the sentential form. This can be formally proved by induction on the number of steps.
- Thus, if $S \overset{*}{\Rightarrow} w$, then eventually M will empty the stack and end up in q_f using w as input.

Proof $L(M) \subseteq L(G)$

- Suppose $(q_0, w, z) \overset{*}{\vdash} (q_f, \lambda, z)$ and $w = a_1 \ldots a_n$. After the initial move, let the first step be

$$(q_1, a_1 \ldots a_n, Sz) \vdash (q_1, a_2 \ldots a_n, u_1 z).$$

  Then there exists a rule $S \rightarrow a_1 u_1$, and $S \Rightarrow a_1 u_1$.

- Let $u_1 = A u_2$. Let the next move be

$$(q_1, a_2 \ldots a_n, A u_2 z) \vdash (q_1, a_3 \ldots a_n, u_3 u_2 z).$$

  Then there exists a rule $A \rightarrow a_2 u_3$, and $S \overset{*}{\Rightarrow} a_1 a_2 u_3 u_2$.

- At any point, the stack content is identical with the unmatched part of the sentential form, so eventually

$$S \overset{*}{\Rightarrow} a_1 \ldots a_n = w$$

cfg for npda

- If $L = L(M)$ for an npda M, then L is context-free. That is, there exists a cfg G such that $L(G) = L$.
- First, we state without proof that for any npda there exists an equivalent one (accepting the same language) with the following properties.
  - It has a single final state q_f that is entered if and only if the stack is empty.
  - Each move either increases or decreases the stack by a single symbol. That is,

    $\delta(q_i, a, A)$ contains objects like (q_j, λ) or (q_j, BC).
- We are given one such npda (δ) and we want to construct a grammar.

Basic Idea

- Here we want the sentential form to represent stack content.
- The grammar we construct uses variables of the form (q_iAq_j). We require the following relation between grammar and npda.
  $(q_iAq_j) \overset{*}{\Rightarrow} v$ *if and only if the npda erases* A *from the stack while reading* v *and going from* q_i *to* q_j.
  Here "erasing" means bringing up the variable under A in stack.

Construction

- For $\delta(q_i, a, A) = (q_j, \lambda)$, we add to the production

$$(q_i A q_j) \rightarrow a.$$

- For $\delta(q_i, a, A) = (q_j, BC)$, we add to the production

$$(q_i A q_k) \rightarrow a(q_j B q_l)(q_l C q_k),$$

  where q_l, q_k take on all values in Q. That is, to erase A from stack, we first replace A by BC and subsequently erase B and C.

- The start symbol is $(q_0 z q_f)$.

Proof

- We show that for all $q_i, q_j \in Q, u, v \in \Sigma^*, A \in \Gamma, X \in \Gamma^*$,

$$(q_i, uv, AX) \overset{*}{\vdash} (q_j, v, X) \;\Leftrightarrow\; (q_i A q_j) \overset{*}{\Rightarrow} u,$$

  which implies $(q_0, w, z) \overset{*}{\vdash} (q_f, \lambda, \lambda) \;\Leftrightarrow\; (q_0 z q_f) \overset{*}{\Rightarrow} w$.
- Suppose $|u| = n$. Then there are $n - 1$ increments and n decrements of stack size. For each of these increment/decrement we have a corresponding rule. Stringing these rules together we have $(q_i A q_j) \overset{*}{\Rightarrow} u$.

Deterministic Case

- A **deterministic pushdown acceptor (dpda)** never has a choice for its move. We require for dpda that
  - $\delta(q, a, b)$ contains at most one element.
  - If $\delta(q, \lambda, b)$ is non-empty, then $\delta(q, c, b)$ is empty for all $c \in \Sigma$,
- Note that for dpda we allow λ-transition or no move, unlike the case of dfa.
- dpda and npda are not equal in their descriptive powers. There are languages that can be accepted by npda but not by any dpda. This is also different from the case of dfa and nfa.

Deterministic cfg

- A language L is said to be a **deterministic context-free language** if there exists a dpda M such that $L = L(M)$. For example, $\{a^n b^n : n \geq 0\}$ is a deterministic cfg.
- The other familiar example $\{ww^R : w \in \Sigma^*\}$ is known to be non-deterministic.
- A context-free language L can be shown, indirectly, to be non-deterministic by assuming it to be deterministic and reaching a contradiction.

$\{a^n b^n : n \geq 0\} \cup \{a^n b^{2n} : n \geq 0\}$

- This language is a cfl but not dcfl.
- It is a cfl since $L = L_1 \cup L_2$ and both $L_1 = \{a^n b^n : n \geq 0\}$ and $L_2 = \{a^n b^{2n} : n \geq 0\}$ are cfls.
- Consider $\widehat{L} = L \cup \{a^n b^n c^n : n \geq 0\}$. It can be shown by pumping lemma for cfg that $\widehat{L}$ cannot be a cfl.
- Suppose L is dcfl. Then we can construct an npda $\widehat{M}$ for $\widehat{L}$ from a dpda M for L, as depicted in Figure 7.4. That's a contradiction.

Grammars for Compiler Design

- The decision whether a string is in the language of a dpda can be made quickly as there is no backtracking involved.
- In order to decide if a string is in a context-free grammar, if there were some way to pinpoint the next rule to use in the derivation, then the parsing process would also be very fast.
- The material introduced here is important for the study of compilers, but not directly related to later subjects in this course.

LL Grammar

- Recall that in the s-grammar, the parsing is linear-time. There is only one candidate rule to use when we look at the first variable in sentential form and the first symbol in the unmatched part of input string.
- s-grammar is a special case of the class of LL grammars. LL means we are scanning the symbols from left-to-right and we are constructing the leftmost derivation.
- More generally, with a $LL(k)$ grammar, we can identify the correct rule given the next k symbols in the input. Apparently the parsing can be done very quickly.

Pumping Lemma

- Let L be an infinite context-free language. Then there exists some positive integer m such that any $w \in L$ with $|w| \geq m$ can be decomposed as

  $$w = uvxyz,$$

  with

  $$|vxy| \leq m, \quad |vy| \geq 1,$$

  such that

  $$uv^ixy^iz \in L \;\text{ for } i = 0, 1, 2, \ldots .$$

- This is known as the pumping lemma for context-free languages.

Proof

- Assume the grammar is in the Chomsky normal form, so there is no λ-production.
- Consider a very big derivation tree where the depth is more than the number of variables in V. Then at least one of the variables, say A, is repeated in the longest path from root.
- Suppose the terminal string under the last appearance of A is x. Let the partial derivation tree under the first appearance A be vxy, so uAz is a sentential form in the derivation of w. Then clearly uv^ixy^iz is in the language.

Closure Properties

- The set of context-free languages is closed under union, concatenation, and star closure.
- The set of context-free languages is not closed under intersection and complementation.

Proof

- Suppose that L_1, L_2 are context-free languages. $U = L_1 \cup L_2$, $C = L_1 \cdot L_2$, and $S = L_1^*$. The variable sets are disjoint.
- The grammar for U is to add a new start variable S_U and a rule $S_U \to S_1|S_2$.
- The grammar for C is to add a new start variable S_C and a rule $S_C \to S_1 S_2$.
- The grammar for S is to add a new start variable S_U and a rule $S_S \to S_1 S_S|\lambda$.
- Counter-examples can be given to prove the non-closedness of intersection and complementation.

Regular Intersection

- Let L_1 be context-free and L_2 be regular. Then $L_1 \cap L_2$ is context-free.
- To prove, we construct an npda given the npda M_1 for L_1 and dfa M_2 for L_2, which simulates simultaneous execution of M_1 and M_2.

Decidable Properties

- There exists an algorithm for deciding whether $L(G) = \emptyset$. Recall that an algorithm has to answer correctly for every instance of the problem.
- We remove the useless symbols and rules. S is useless iff $L(G)$ is empty.
- There exists an algorithm for deciding whether $L(G)$ is infinite.
- Remove useless variables, rules, unit productions and λ-productions. $L(G)$ is infinite iff there is a cycle in the dependency graph, where an edge from A to B means there is a rule

$$A \to xBy.$$