CSC444

October 26, 2014

Contents

1 Pushdown automata

Clearly, a finite automaton cannot be a recognizer for a context-free language.

Reason: Non-regular languages, such as $\{\, a^n b^n \mid n \geq 0 \,\}$, can be represented using context-free grammars.

We will see, however, that giving a finite automaton *access to a stack* gives it enough power to recognize context-free languages.

How does a stack help?

Suppose that we are interested in recognizing the language $\{\, a^n b^n \mid n \geq 0 \,\}$. The following algorithm works:

```
if the string is empty
  then accept

while (current input symbol = a)
  push (current input symbol)

if (stack is empty)
  then reject

while (not at end of string)
  if (current input symbol != b)
    then reject
    else if (stack is empty)
      then reject
      else pop the stack and advance to the next input symbol

if (stack is empty)
  then accept
  else reject
```

To look at it another way, consider recognizing an arbitrary context-free grammar G.

If we want to deal with a rule of the form $A \to aB$, then a finite automata can easily simulate this by moving from a state A to a state B while consuming the letter a.

But what about a rule of the form $A \to aBb$? We still need to move from state A to state B and read the letter a, but what do we do with the letter b?

If we have a stack, then we can push b onto the stack and deal with it later, perhaps by checking it against a b in the input.

1.1 Definitions

Definition 1.1. A *pushdown automaton* is a sextuple $M = (K, \Sigma, \Gamma, \Delta, s, F)$ where:

- K is a finite *set of states*,
- Σ is an alphabet (the *input symbols*),
- Γ is an alphabet (the *stack symbols*),
- $s \in K$ is the *initial state*
- $F \subseteq K$ is the *set of final states*, and
- Δ, the *transition relation*, is a finite subset of $(K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*) \times (K \times \Gamma^*)$

If $((p, a, \beta), (q, \gamma)) \in \Delta$, then M, whenever it is in state p with β at the top of the stack, may read a from the input tape (note that $a = \varepsilon$ is allowed in which case the input is not consulted), replace β by γ on the top of the stack, and enter state q.

$((p, a, \beta), (q, \gamma))$ is called a *transition* of M.

Unlike finite automata, we are first giving *the nondeterministic version* of pushdown automata. There may be several transitions of M that are applicable at any point.

Terminology:

- We *push* a symbol when we add it to the top of the stack.
- We *pop* a symbol when we remove it from the top of the stack.
- A *configuration* of a pushdown automaton is a member of $K \times \Sigma^* \times \Gamma^*$. The first component is the state of the machine, the second is the portion of the input yet to be read, and the third is the contents of the pushdown store, read top-down.

For example, if (q, w, abc) were the configuration, then the a would be at the top of the stack and the c at the bottom.

More terminology:

- If (p, x, α) and (q, y, ρ) are configurations of M, we say that (p, x, α) *yields in one step* (q, y, ρ), written $(p, x, \alpha) \mapsto_M (q, y, \rho)$, if there is a transition $((p, a, \beta), (q, \gamma)) \in \Delta$ such that $x = ay$, $\alpha = \beta\tau$, and $\rho = \gamma\tau$ for some $\tau \in \Gamma^*$.
- The reflexive, transitive closure of $\mapsto_M$ is written $\mapsto_M^*$.
- M *accepts a string* $w \in \Sigma^*$ if $(s, w, \varepsilon) \mapsto_M^* (p, \varepsilon, \varepsilon)$ for some state $p \in F$.

Put another way M accepts a string $\Leftrightarrow$ there is a sequence of configurations $C_0, C_1, ..., C_n$ where $n > 0$, such that $C_0 \mapsto_M C_1 \mapsto_M \ldots \mapsto_M C_n$, $C_0 = (s, w, \varepsilon)$ and $C_n = (p, \varepsilon, \varepsilon)$.

Yet more terminology:

- Any sequence of configurations $C_0, C_1, ..., C_n$ such that $C_i \mapsto_M C_{i+1}$ for $i = 0, ..., n-1$ will be called a *computation* by M. It has *length* n or has n *steps*.
- The *language accepted by* M, denoted $\mathcal{L}(M)$, is the set of all strings accepted by M.
- As with finite automata, we will omit the M from the notation when the pushdown automata under consideration is understood.

Example 1.2. Let $L = \{\, w \in \{a, b\}^* \mid w \text{ has an equal number of } a\text{'s and } b\text{'s} \,\}$.

How will the automaton work?

- The automaton will keep either a string of a's or a string of b's on its stack.
- A string of a's indicates that M has seen more a's than b's at the current point in time. The size of the stack is the amount of the excess.
- A string of b's represents an excess of b's.
- In either case, a marker c is used to indicate that the bottom of the stack has been reached.

Let $M = (K, \Sigma, \Gamma, \Delta, s, F)$ where $K = \{s, q, f\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, c\}$, $F = \{f\}$, and Δ is given below:

- $((s,\varepsilon,\varepsilon),(q,c))$
- $((q,a,c),(q,ac))$
- $((q,a,a),(q,aa))$
- $((q,a,b),(q,\varepsilon))$
- $((q,b,c),(q,bc))$
- $((q,b,b),(q,bb))$
- $((q,b,a),(q,\varepsilon))$
- $((q,\varepsilon,c),(f,\varepsilon))$

The purpose of the transitions is the following:

- Transition 1 initializes the computation. It puts M into state q while placing a c on the bottom of the stack.
- In state q reading a, M either starts up a stack of a's from the bottom using Transition 2, adds an a to an existing stack of a's using Transition 3, or pops a b off the stack using Transition 4.
- In state q reading b, $M3$ either starts up a stack of b's from the bottom using Transition 5, adds to an existing stack of b's using Transition 6, or pops an a off the stack using Transition 7.
- When c is the topmost character on the stack and there are no characters left to read, then we can remove the c using Transition 8 and accept the string since there are no outstanding a's or b's.

Sample accepting computation: $w = aabaaabbbb$

State	Unread input	Stack	Transition	Comment
s	$aabaaabbbb$	ε	—	Initial configuration
q	$aabaaabbbb$	c	1	Bottom marker
q	$abaaabbbb$	ac	2	Start stack of a's
q	$baaabbbb$	aac	3	Continue stack of a's
q	$aaabbbb$	ac	7	Pop off an a
q	$aabbbb$	aac	3	Add another a
q	$abbbb$	$aaac$	3	Add another a
q	$bbbb$	$aaaac$	3	Add another a
q	bbb	$aaac$	7	Pop off an a
q	bb	aac	7	Pop off an a
q	b	ac	7	Pop off an a
q	ε	c	7	Pop off an a
f	ε	ε	8	Accept the string

1.2 Relationship to regular languages

We know, by direct construction, that the regular languages are a *proper subset* of the context-free languages.

- They are a subset since we can construct a grammar for every regular language.
- They are a proper subset since we can express non-regular languages using context-free grammars.

It is very easy to show Part 1 using pushdown automata, because every finite automaton can be viewed as a pushdown automaton that ignores its stack.

More formally, let $M = (K,\Sigma,\Delta,s,F)$ be a NFA. Then a pushdown automaton $M' = (K,\Sigma,\emptyset,\Delta',s,F)$ where $\Delta' = \{\, ((p,u,\varepsilon),(q,\varepsilon)) \mid (p,u,q) \in \Delta \,\}$ accepts the same language as M.

M' always pushes and pops the empty string onto its stack, but otherwise behaves like M.

2 Pushdown automata and context-free grammars

As with the regular languages, the two different representations we considered capture the same set of languages.

Theorem 2.1. *The class of languages accepted by pushdown automata is exactly the class of context-free languages.*

We will break the proof into two parts, one for each direction.

2.1 Construction of a pushdown automaton

Theorem 2.2. *Each context-free language is accepted by some pushdown automaton.*

Proof. Let $G = (V,\Sigma,R,S)$ be a context-free grammar. We must construct a pushdown automaton $M = (K,\Sigma',\Gamma,\Delta,s,F)$ s.t. $\mathcal{L}(M) = \mathcal{L}(G)$. □ M has two states p and q. M works as follows:

- M begins in state p with an empty stack.
- M begins the computation by pushing S, the start symbol of G, onto its initially empty pushdown store, and entering state q.
- On each subsequent step M either:
  - Replaces the topmost symbol A on the stack, provided that A is a non-terminal, by the right-hand side x of some rule $A \to x$; or
  - Pops the topmost symbol from the stack that matches the next input symbol, provided that it is a terminal symbol.

The formal definition of M is the following:

- $K = \{p,q\}$
- $s = p$

- $F = \{q\}$
- $\Gamma = V$, that is, M will use the set of terminal and non-terminals as the stack alphabet
- $\Sigma = \Sigma'$

All that remains is to define the transition relation.

Δ contains the following transitions:

- $((p,\varepsilon,\varepsilon),(q,S))$
- $((q,\varepsilon,A),(q,x))$ for each rule $A \to x$ in R
- $((q,a,a),(q,\varepsilon))$ for each $a \in \Sigma$

Intuitively these transitions do the following:

- Transition 1 pushes the start symbol onto the stack.
- Transitions of type 2 correspond to replacing the top non-terminal on the stack with its expansion.
- Transitions of type 3 correspond to popping the terminals off the stack to expose the next non-terminal.

Claim 2.3. *Let $w \in \Sigma^*$ and $\alpha \in (V-\Sigma)V^* \cup \{\varepsilon\}$. Then $S \overset{L^*}{\Rightarrow} w\alpha$ if and only if $(q,w,S) \mapsto_M^* (q,\varepsilon,\alpha)$*

This claim suffices to show that $\mathcal{L}(G) = \mathcal{L}(M)$ since if we take $\alpha = \varepsilon$ then the claim states that $S \overset{L^*}{\Rightarrow} w$ if and only if $(q,w,S) \mapsto_M^* (q,\varepsilon,\varepsilon)$.

Proof. Postponed until after our discussion of parse trees. □

2.1.1 An example

Consider the grammar $G = (V, \Sigma, R, S)$ with $V = \{S, a, b, c\}$, $\Sigma = \{a, b, c\}$ and $R = \{S \to aSa, S \to bSb, S \to c\}$.

Sample derivations:

- $S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abacaba$
- $S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aacaa$

$\mathcal{L}(G) = \{\, wcw^R \mid w \in \{a, b\}^* \,\}$.

The corresponding pushdown automaton using the above construction is $M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$ where Δ contains the following transitions:

- $((p, \varepsilon, \varepsilon), (q, S)$
- $((q, \varepsilon, S), (q, aSa))$
- $((q, \varepsilon, S), (q, bSb))$
- $((q, \varepsilon, S), (q, c))$
- $((q, a, a), (q, \varepsilon))$
- $((q, b, b), (q, \varepsilon))$
- $((q, c, c), (q, \varepsilon))$

As an example, the string *abacaba* is accepted by M through the following sequence of moves:

State	Unread input	Stack	Transition
p	*abacaba*	ε	—
q	*abacaba*	S	1
q	*abacaba*	aSa	2
q	*bacaba*	Sa	5
q	*bacaba*	$bSba$	3
q	*acaba*	Sba	6
q	*acaba*	$aSaba$	2
q	*caba*	$Saba$	5
q	*caba*	$caba$	4
q	*aba*	aba	7
q	*ba*	ba	5
q	*a*	a	6
q	ε	ε	5

Let's compare this with the computation of the pushdown automaton that we designed before.

$M = (K, \Sigma, \Gamma, \Delta, s, F)$ where $K = \{s, f\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{a, b\}$, $F = \{f\}$, and Δ contains the following five transitions:

- $((s, a, \varepsilon), (s, a))$
- $((s, b, \varepsilon), (s, b))$
- $((s, c, \varepsilon), (f, \varepsilon))$
- $((f, a, a), (f, \varepsilon))$
- $((f, b, b), (f, \varepsilon))$

Accepting computation for $w = abacaba$

State	Unread input	Stack	Transition
s	*abacaba*	ε	—
s	*bacaba*	a	1
s	*acaba*	ba	2
s	*caba*	aba	1
f	*aba*	aba	3
f	*ba*	ba	4
f	*a*	a	5
f	ε	ε	4

2.2 Construction of a context-free grammar

Theorem 2.4. *If a language is accepted by a pushdown automaton, it is a context-free language.*

It is helpful for this construction to consider a restricted form of pushdown automata.

2.2.1 Simple pushdown automata

Definition 2.5. A pushdown automaton is *simple* if whenever $((q, a, \beta), (p, \gamma))$ is a transition and q is not the start state, then $\beta \in \Gamma$ and $|\gamma| \le 2$.

A simple pushdown automaton always consults its topmost stack symbol (and no symbols below it), and replaces it either with ε, or with a single stack symbol or with two stack symbols.

Why can't q be the start state?

Because then the automaton could not work on an empty stack, rendering it useless.

Note. This definition of simple is a bit different than the one in the textbook. It is instructive to look at the definition in the book and the resulting proof.

Claim 2.6. *If a language is accepted by an unrestricted pushdown automaton, then it is accepted by a simple pushdown automaton.*

Proof. Let $M = (K, \Sigma, \Gamma, \Delta, s, F)$ be any pushdown automaton. We will construct a simple pushdown automaton M' so that $\mathcal{L}(M) = \mathcal{L}(M')$.

$M' = (K', \Sigma, \Gamma \cup \{Z\}, \Delta', s', \{f'\})$ where s' and f' are new states not in K, and Z is a new stack symbol, *the bottom stack symbol*.

Initially, Δ' contains *all the transitions of* Δ plus the following additional transitions:

We first add to Δ' the transition $((s', \varepsilon, \varepsilon), (s, Z))$.

This transition starts the computation *by placing the stack bottom symbol* at the bottom of the stack.

The stack bottom symbol will remain at the bottom of the stack throughout the computation.

No transition of Δ' will ever push a Z onto the stack except to replace it at the bottom of the stack.

We also add to Δ' the transitions $((f, \varepsilon, Z), (f', \varepsilon))$ for each $f \in F$.

These transitions end the computation *by removing* Z from the bottom of the stack.

We shall next *replace all transitions in* Δ' *that violate the simplicity condition* by equivalent transitions that satisfy the simplicity condition.

We will do that in *three stages*:

- Replace all transitions with $|\beta| \ge 2$.
- Get rid of all transitions with $|\gamma| > 2$ without introducing any transitions that have $|\beta| \ge 2$.
- Remove all transitions with $\beta = \varepsilon$ without introducing any transitions with $|\beta| \ge 2$ or $|\gamma| > 2$.

Stage 1: Consider any transition $((q, a, \beta), (p, \gamma)) \in \Delta'$ where $\beta = B_1 B_2 \ldots B_n$ with $n > 1$.

It will be replaced with new transitions that pop each of the B_i's sequentially rather than in one step.

Specifically, we add the transitions:

$((q, \varepsilon, B_1), (q_{B_1}, \varepsilon))$

$((q_{B_1}, \varepsilon, B_2), (q_{B_1 B_2}, \varepsilon))$

...

$((q_{B_1 B_2 \dots B_{n-2}}, \varepsilon, B_{n-1}), (q_{B_1 B_2 \dots B_{n-1}}, \varepsilon))$

$((q_{B_1 B_2 \dots B_{n-2} B_{n-1}}, a, B_n), (p, \gamma))$

where for $i = 1, ..., n-1$, $q_{B_1 B_2 \dots B_i}$ is a new state with the intuitive meaning "state q after the symbols $B_1, B_2, ..., B_i$ have been popped".

We repeat this with all transitions that have $|\beta| > 1$. It is clear that these modifications do not change the language recognized by the pushdown automaton.

Stage 2: We now replace transitions $((q, a, \beta), (p, \gamma))$ where $\gamma = C_1 C_2 ... C_m$ and $m \geq 2$ by the transitions:

$((q, a, \beta), (r_1, C_m))$

$((r_1, \varepsilon, \varepsilon), (r_2, C_{m-1}))$

$((r_2, \varepsilon, \varepsilon), (r_3, C_{m-2}))$

...

$((r_{m-2}, \varepsilon, \varepsilon), (r_{m-1}, C_2))$

$((r_m, \varepsilon, \varepsilon), (p, C_1))$

where $r_1, ..., r_m$ are new states.

Note. At this point all transitions have $|\gamma| \leq 1$, which if not corrected, would be a loss of generality. This will be fixed in the next stage.

Clearly no transitions with $|\beta| > 1$ were added.

Stage 3: Consider any transition $((q, a, \varepsilon), (p, \gamma))$ with $q \neq s'$. These are the only possible remaining violations of the simplicity condition.

Replace any such transition with all transitions of the form $((q, a, A), (p, \gamma A))$ for all $A \in \Gamma \cup \{Z\}$.

This means that if the automaton could move without consulting its stack, it can also move by consulting the top stack symbol, whatever it may be, and then replacing it immediately.

We know that there will always be at least one symbol on the stack during the computation, namely the bottom of stack marker Z.

Note. At this stage we may introduce γ where $|\gamma| = 2$. This doesn't violate simplicity, but it is necessary to obtain general pushdown automata.

It is easy to see that this construction results in a simple pushdown automaton M' s.t. $\mathcal{L}(M') = \mathcal{L}(M)$.

This completes the claim that every pushdown automaton can be simulated by a simple pushdown automaton. $\square$

2.2.2 Proof of Theorem 3.4.2

Assume we are given a pushdown automaton $M = (K, \Sigma, \Gamma, \Delta, s, F)$ and that we have converted it into a simple pushdown automaton $M' = (K', \Sigma, \Gamma \cup \{Z\}, \Delta', s', \{f'\})$.

We will construct a context-free grammar G s.t. $\mathcal{L}(G) = \mathcal{L}(M')$.

Let $G = (V, \Sigma, R, S)$ where V contains, in addition to a new symbol S and the symbols in Σ, a new symbol $\langle q, A, p \rangle$ for all $q, p \in K'$ and each $A \in \Gamma \cup \{\varepsilon, Z\}$.

If $A \in \Gamma$, then the non-terminal $\langle q, A, p \rangle$ represents any portion of the input string that might be read between a point in time when M' is in state q with A on top of its stack, and a point in time when M' removes that occurrence of A from the stack and enters state p.

If $A = \varepsilon$ then $\langle q, \varepsilon, p \rangle$ denotes a portion of the input string that might be read between a time when M' is in state q and a time when it is in state p with the same stack, without in the interim changing or consulting that part of the stack.

The rules of R are of four types:

- $S \to \langle s, Z, f' \rangle$ where s is the start state of M and f' the new final state.
- For each transition $((q, a, B), (r, C))$ where $q, r \in K'$, $a \in \Sigma \cup \{\varepsilon\}$, and $B, C \in \Gamma \cup \{\varepsilon\}$ and for each $p \in K'$, we add the rule $\langle q, B, p \rangle \to a \langle r, C, p \rangle$.

- For each transition $((q, a, B), (r, C_1 C_2))$ where where $q, r \in K'$, $a \in \Sigma \cup \{\varepsilon\}$, $B \in \Gamma \cup \{\varepsilon\}$, and $C_1, C_2 \in \Gamma$ and for each $p, p' \in K'$, we add the rule $\langle q, B, p \rangle \to a \langle r, C_1, p' \rangle \langle p', C_2, p \rangle$.
- For each $q \in K'$, we add the rule $\langle q, \varepsilon, q \rangle \to \varepsilon$

Note. Because M' is simple, all of its transitions must be either Type 2 or Type 3.

The rules of R can be understood intuitively as follows:

- Rules of Type 1 state that any input string which can be read by M' passing from state s to the final state, while at the same time the net effect on the stack is that the stack bottom symbol was popped, is a string in the language $\mathcal{L}(M')$.
- Rules of Type 4 say that no computation is needed to go from a state to itself without changing the stack.
- A rule of Type 2 or a rule of Type 3 says that, if $((q, a, B), (p, \gamma)) \in \Delta'$, then one of the possible computations that lead from state q to state p while consuming B from the top of the stack, starts by reading a, replacing B by γ, passing to state r, and then going on to consume γ and end up in state p.

Since M' is simple $\gamma = \varepsilon, \gamma = C$ or $\gamma = C_1 C_2$ so Types 2 and 3 cover all the possibilities.

If $\gamma = C_1 C_2$ then this last computation can, in principle, pass through any state p' immediately after C_1 is popped.

The following claim completes the proof:

Claim 2.7. *For any $q, p \in K'$, $A \in \Gamma \cup \{\varepsilon\}$, and $x \in \Sigma^*$, $\langle q, A, p \rangle \Rightarrow_G^* x$ if and only if $(q, x, A) \mapsto_{M'}^* (p, \varepsilon, \varepsilon)$.*

This claim suffices since $\langle s, \varepsilon, f \rangle \Rightarrow_G^* x$ for some $f \in F$ if and only if $(s, x, \varepsilon) \mapsto_{M'}^* (f, \varepsilon, \varepsilon)$.

This means that $x \in \mathcal{L}(G)$ if and only if $x \in \mathcal{L}(M')$.

Proof. Omitted. $\square$