COMS W3261
Computer Science Theory
Chapter 6 Notes

Alexander Roth

2014–09–27

Pushdown Automata

The context-free languages have a type of automaton that defines them. This automaton, called a "pushdown automaton," is an extension of the nondeterministic finite automaton with ϵ-transitions, which is one of the ways to define the regular languages. It is essentially an ϵ-NFA with the addition of a stack. The stack can be read, pushed, and popped only at the top, just like the "stack" data structure.

Definition of the Pushdown Automaton

Informal Induction

The pushdown automaton is in essence a nondeterministic finite automaton with ϵ-transitions permitted and one additional capability: a stack on which it can store a string of "stack symbols". This stack allows the automaton to "remember" an infinite amount of information.

Push-down automata recognize all and only the context-free languages.

A "finite-state control" reads inputs, one symbol at a time. The pushdown automaton is allowed to observer the symbol at the top of the stack and to base its transition on its current state, the input symbol, and the symbol at the top of stack. Alternatively, it may make a "spontaneous" transition, using ϵ as its input instead of an input symbol. In one transition, the pushdown automaton:

1. Consumes from the input the symbol that it uses in the transition. If ϵ is used for the input, then no input symbol is consumed.

2. Goes to a new state, which may or may not be the same as the previous state.

3. Replaces the symbol at the top of the stack by any string. The string could be ϵ, which corresponds to a pop of the stack. It could be the same

symbol that appeared at the top of the stack previously; i.e., no change to the stack is made. It could also replace the top stack symbol by one other symbol, which in effect changes the top of the stack but does not push or pop it. Finally, the top stack symbol could be replaced by two or more symbols, which has the effect of (possibly) changing the top stack symbol, and then pushing one or more new symbols onto the stack.

The Formal Definition of Pushdown Automata

Our formal notation for a *pushdown automaton* (PDA) involves seven components. We write the specification of a PDA P as follows:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

The components have the following meanings:

Q: A finite set of *states*, like the states of a finite automaton.

Σ: A finite set of *input symbols*, also analogous to the corresponding components of a finite automaton.

Γ: A finite *stack alpahbet*. This component, which has no finite-automaton analog, is the set of symbols that we are allowed to push onto the stack.

δ: The *transition function*. As for a finite automaton, δ governs the behavior of the automaton. Formally, δ takes as arguments a triple $\delta(q, aX)$, where:

1. q is a state in Q.
2. a is either an input symbol in Σ or $a = \epsilon$, the empty string, which is assumed not to be an input symbol.
3. X is a stack symbol, that is, a member of Γ.

The output of δ is a finite set of pairs (p, γ), where p is the new state, and γ is the string of stack symbols that replace X at the top of the stack.

q_0: The *start state*. The PDA is in this state before making any transitions.

F: The set of *accepting states*, or *final states*.

No "Mixing and Matching"

There may be several pairs that are options for a PDA in some situation. for instance suppose $\delta(q, aX) = \{(p, Y, Z), (r, \epsilon)\}$. When making a move of the PDA, we have to choose one pair in its entirety; we cannot pick a state from one and a stack-replacement strings from another.

A Graphical Notation for PDA's

Sometimes, a diagram, generalizing the transition diagram of a finite automaton, will make aspects of the behavior of a given PDA clearer. We shall therefore introduce and subsequently use a *transition diagram* for PDA's in which:

a) The nodes correspond to the states of the PDA>

b) An arrow labeled *Start* indicates the start state, and doubly circled states are accepting, as for finite automata.

c) The arcs correspond to transitions of the PDA in the following sense. An arc labeled a, X/α from state q to state p means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs. That is, the arc label tells what input is used and also gives the old and new tops of the stacks.

Conventionally, we use Z_0 as the start symbol for the stack.

Instantaneous Descriptions of a PDA

Intuitively, the PDA goes from configuration to configuration, in response to input symbols (or sometimes ϵ), but unlike the finite automaton, where the state is the only thing that we need to know about the automaton, the PDA's configuration involves both the state and the contents of the stack. Being arbitrarily large, the stack often more important part of the total configuration of the PDA at any time.

Thus, we shall represent the configuration of a PDA by a triple (q, w, γ), where

1. q is the state,
2. w is the remaining input, and
3. γ is the stack contents.

Conventionally, we show the top of the stack at the left end of γ and the bottom at the right end. Such a triple is called an *instantaneous description*, or ID, of the pushdown automaton.

For finite automata, the $\hat{\delta}$ notation was sufficient to represent sequences of instantaneous descriptions through which a finite automaton moved since the ID for a finite automaton is just its state. However, for PDA's we need a notation that describes changes in the state, the input, and stack. Thus we adopt the "turnstile" notation for connection pairs of ID's that represent one or many moves of a PDA.

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Define $\underset{P}{\vdash}$, or just $\vdash$ when P is understood as follows. Suppose $\delta(q, a, X)$ contains (p, α). The for all strings w in Σ^* and β in Γ^*:

$$(q, aw, X\beta) \vdash (p, w, \alpha\beta)$$

This move reflects the idea that, by consuming a (which may be ϵ) from the input and replacing X on top of the stack by α, we can go from state q to state p.

We also use the symbols $\overset{*}{\underset{P}{\vdash}}$, or $\overset{*}{\vdash}$ when the PDA P is understood, to represent zero or more moves of the PDA.

There are three important principles about ID's and their transitions that we shall need in order to reason about PDA's:

1. If a sequence of ID's (*computation*) is legal for a PDA P, then the computation formed by adding the same additional input string to the end of the input (second component) in each ID is also legal.

2. If a computation is legal for a PDA P, then the computation formed by adding the same additional stack symbols below the stack in each ID is also legal.

3. If a computation is legal for a PDA P, and some tail of the input is not consumed, then we can remove this tail from the input in each ID, and the resulting computation will still be legal.

Intuitively, data that P never looks at cannot affect its computation. Thus, we conclude:

Theorem 1. If $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA, and $(q, x, \alpha) \overset{*}{\underset{P}{\vdash}} (p, y, \beta)$, then for any strings w in Σ^* and γ in Γ^*, it is also true that

$$(q, xw, \alpha\gamma) \overset{*}{\underset{P}{\vdash}} (p, yw, \beta\gamma)$$

Note that if $\gamma = \epsilon$, then we have a formal statement of principle (1) above, and if $w = \epsilon$, then we have the second principle.

Theorem 2. If $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA, and

$$(q, xw, \alpha) \overset{*}{\underset{P}{\vdash}} (p, yw, \beta)$$

then it is also true that $(q, x, \alpha) \overset{*}{\underset{P}{\vdash}} (p, y, \beta)$.

The Languages of a PDA

We have assumed that a PDA accepts its input by consuming it and entering an accepting state. We call this approach "acceptance by final state." There is a second approach to defining the language of a PDA that has important applications. We may also define for any PDA the language "accepted by empty stack," that is, the set of strings that cause the PDA to empty its stack, starting from the initial ID.

These two methods are equivalent, in the sense that a language L has a PDA that accepts it by final state if and only if L has a PDA that accepts it by empty stack. However, for a given PDA P, the languages that P accepts by final state and by empty stack are usually different.

Acceptance by Final State

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then $L(P)$, the *language accepted by P by final state* is

$$\{w \mid (q_0, w, Z_0) \overset{*}{\underset{P}{\vdash}} (q, \epsilon, \alpha)\}$$

for some state q in F and any stack string α. That is, starting in the initial ID with w waiting on the input, P consumes w from the input and enters an accepting state. The contents of the stack at that time is irrelevant.

Acceptance by Empty Stack

For each PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, we also define

$$N(P) = \{w \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \epsilon)$$

for any state q. That is, $N(P)$ is the set of inputs w that P can consume and at the same time empty its stack.

Since the set of accepting states is irrelevant, we shall sometimes leave off the last (seventh) component from the specification of a PDA P, if all we care about is the language that P accepts by empty stack. Thus, we would write P as a six-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0)$.

From Empty Stack to Final State

Theorem 3. If $L = N(P_N)$ for some PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$, then there is a PDA P_F such that $L = L(P_F)$.

From Final State to Empty Stack

Theorem 4. Let L be $L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$.

Equivalence of PDA's and CFG's

To prove that the languages defined by PDA's are exactly the context-free languages, we must prove the following three classes of languages:

1. The context-free languages, i.e., the languages defined by CFG's.
2. The languages that are accepted by final state by some PDA.
3. The languages that are accepted by empty stack by some PDA.

are all the same class.

From Grammars to Pushdown Automata

Given a CFG G, we construct a PDA that simulates the leftmost derivations of G. Any left-sentential form that is not a terminal string can be written as $xA\alpha$, where A is the leftmost variable, x is whatever terminals appear to its left, and α is the string of terminals and variables that appear to the right of A. We call $A\alpha$ the *tail* of this left-sentential form. If a left-sentential form consists of terminals only, then its tail is ϵ.

The idea behind the construction of a PDA from a grammar is to have the PDA simulate the sequence of left-sentential forms that the grammar uses to generate a given terminal string w. The tail of each sentential form $xA\alpha$ appears on the stack, with A at the top. At that time, x will be "represented" by our having consumed x from the input, leaving whatever of w follows its prefix x. That is, if $w = xy$, then y will remain on the input.

Suppose that PDA is in an ID $(q, y, A\alpha)$, representing a left-sentential form $xA\alpha$. It guesses the production to use to expand A, say $A \to \beta$. The move of the PDA is to replace A on the top of the stack by β, entering ID $(q, y, \beta\alpha)$. Note that there is only one state, q, for this PDA.

Now $(q, y, \beta\alpha)$ may not be a representation of the next left-sentential form, because β may have a prefix of terminals. In fact, β may have a prefix of terminals. Whatever terminals appear at the beginning of $\beta\alpha$ need to be removed, to expose the next variable at the top of the stack. These terminals are compared against the next input symbols, to make sure our guesses at the leftmost derivation of input string w are correct; if not, this branch of the PDA dies.

If we succeed in this way to guess a leftmost derivation of w, then we shall eventually reach the left-sentential form w. At that point, all the symbols on the stack have either been expanded (if they are variables) or matched against the input (if they are terminals). The stack is empty, and we accept by empty stack.

Let $G = (V, T, Q, S)$ be a CFG. Construct the PDA P that accepts $L(G)$ by empty stack as follows:

$$P = (\{q\}, T, V \cup T, \delta, q, S)$$

where transition function δ is defined by:

1. For each variable A,

$$\delta(q, \epsilon, A) = \{(q, \beta) \mid A \to \beta \text{ is a production of } G\}$$

2. For each terminal a, $\delta(q, a, a) = \{(q, \epsilon)\}$.

Theorem 5. If PDA P is constructed from CFG G by the construction above, then $N(P) = L(G)$.

From PDA's to Grammars

Now, we complete the proofs of equivalence by showing that for every PDA P, we can find a CFG G whose language is the same language that P accepts

by empty stack. The idea behind the proof is to recognize that the fundamental event in the history of a PDA's processing of a given input is the net popping of one symbol off the stack, while consuming some input. A PDA may change state as it pops stack symbols, so we should also note the state it enters when it finally pops a level off its stack.

For example, some input x_1 is read while Y_1 is popped. We should emphasize that this "pop" is the net effect of (possibly) many moves. For example, the first move may change Y_1 to some other symbol Z. The next move may replace Z by UV, later moves have the effect of popping U, and then other moves pop V. The net effect is that Y_1 has been replaced by nothing; i.e., it has been popped, and all the input symbols consumed so far constitute x_1.

We suppose that the PDA starts out in state p_0, with Y_1 at the top of the stack. After all the moves whose net effect is to pop Y_1, the PDA is in state p_1. It then proceeds to (net) pop Y_2, while reading input string X_2 and winding up, perhaps after many moves in state p_2 with Y_2 off the stack. The computation proceeds until each of the symbols on the stack is removed.

Our construction of an equivalent grammar uses variables each of which represents an "event" consisting of:

1. The net popping of some symbol X from the stack, and
2. A change in state from some p at the beginning to q when X has finally been replaced by ϵ on the stack.

We represent such a variable by the composite symbol $[pXq]$. Remember that this sequence of characters is our way of describing *one* variables it is not five grammar symbols.

Theorem 6. Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ be a PDA. Then there is a context-free grammar G such that $L(G) = N(P)$.

Deterministic Pushdown Automata

While PDA's are by definition allowed to be nondeterministic, the deterministic subcase is quite important. In particular, parsers generally behave like deterministic PDA's, so the class of languages that can be accepted by these automata is interesting for the insights it give us into what constructs are suitable for use in programming languages.

Definition of a Deterministic PDA

Intuitively, a PDA is deterministic if there is never a choice of move in any situation. These choices are of two kinds. If $\delta(q, a, X)$ contains more than one pair, then surely the PDA is nondeterministic becuase we can choose mong these pairs when deciding on the next move. However, even if $\delta(q, a, X)$ is always a singleton, we could still have a choice between using a real input symbol or

making a move on ϵ. Thus, we define a PDA $P = (Q, \Sigma, \Gamma, q_0, Z_0, F)$ to be *deterministic* (DPDA), if and only if the following conditions are met:

1. $\delta(q, a, X)$ has at most one member for any q in Q, a in Σ or $a = \epsilon$, and X in Γ.
2. If $\delta(q, a, X)$ is nonempty, for some a in Σ, then $\delta(q, \epsilon, X)$ must be empty.

Regular Langauges and Deterministic PDA's

The DPDA's accept a class of languages that is between the regular languages and the CFL's.

Theorem 7. If L is a reuglar language, then $L = L(P)$ for some DPDA P.

If we want the DPDA to accept by empy stack, then we find that our language-recognizing capability is rather limited. Say that language L has the *prefix property* if there are no two different strings x and y in L such that x is a prefix of y.

Theorem 8. A language L is $N(P)$ for some DPDA P if and only if L has the prefix property and L is $L(P')$ for some DPDA P'.

DPDA's and Context-Free Languages

The languages accepted by DPDA's by final state properly include the regular languages, but are properly included in the CFL's.

DPDA's and Ambiguous Grammars

Theorem 9. If $L = N(P)$ for some DPDA P, then L has an unambiguous context-free grammar.

Theorem 10. If $L = L(P)$ for some DPDA P, then L has an unambiguous CFG.

Summary of Chapter 6

Pushdown Automata A PDA is a nondeterministic finite automaton coupled with a stack that can be used to store a string of arbitrary lenght. The stack can be read and modified only at its top.

Moves of a Pushdown Automata A PDA chooses its next move based on its current state, the next input symbol, and the symbol at the top of its stack. It may also choose to make a move independent of the input symbol and without consuming that symbol from the input. Being nondeterministic, the PDA may have some finite number of choices of move; each is a new state and a string of stack symbols with which to replace the symbol currently on top of the stack.

Acceptance by Pushdown Automata There are two ways in whcih we may allow the PDA to signal acceptance. One is by entering an accepting state; the other by emptying its stack. These methods are equivalent, in the sense that any language accepted by one method is acceted (by some other PDA) by the other method.

Instantaneous Descriptions We use an ID consisting of the state, remaining input, and stack contents to describe the "current condition" of a PDA. A transition function $\vdash$ between ID's represents sinle moves by a PDA.

Pushdown Automata and Grammars The languages accepted by PDA's either by final state or by empty stack, are exactly the context-free languages.

Deterministic Pushdown Automata The two modes of acceptance – final state and empty stack – are not the same for DPDA's. Rather, the languages accepted by empty stack are exactly those of the languages accepted by final state that have the prefix property: no string in the language is a prefix of another word in the language.

The Languages Accepted by DPDA's All the regular languages are accepted (by final state) by DPDA's and there are nonregular languages accepted by DPDA's. The DPDA languages are context-free languages, and in fact are languages that have unambiguous CFG's. Thus teh DPDA languages lie strictly between the regular languages and the context-free languages.