More Context Free Languages

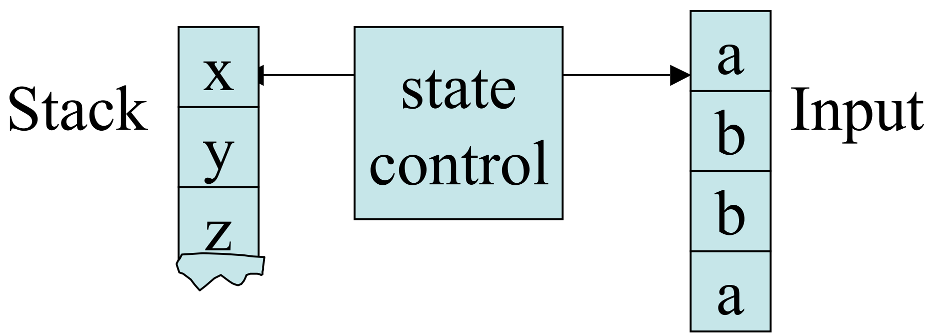
CS154
Chris Pollett
Mar 6, 2006.

Outline

- Pushdown Automata
- Equivalence

Pushdown Automata

- Our machine model is a generalization of finite automata.
- We allow our machines to have a stack:



- In a give state reading a given input symbol and a given stack symbol, the machine can switch states, advance to the next character of the input, pop the top symbol off the stack, or push a new symbol onto the stack.
- For instance, the language $\{0^n1^n \mid n>=0\}$ could be recognized by such a machine. When one reads an 0 push it onto the stack. When one starts reading 1's, if one ever sees another 0 reject, also start popping 0's off of the stack. If when one gets to the end of the string the stack is empty, then accept.

Formal Definition

- For a set of strings A. Let $A_\varepsilon = A \cup \{\varepsilon\}$.
- A **pushdown automaton** is a 6-tuple $M=(Q, \Sigma, \Gamma, \delta, q_0, F)$ where
  1. Q is the set of states
  2. Σ is the input alphabet
  3. Γ is the stack alphabet
  4. δ: $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$-->$P(Q \times \Gamma_\varepsilon)$ is the transition function
  5. $q_0 \in Q$ is the start state, and
  6. $F \subseteq Q$ is the set of accept states.
- M ***accepts*** $w= w_1w_2\ldots w_n$ where each $w_i \in \Sigma_\varepsilon$ if there is a sequence of states $r_0, r_1, \ldots, r_m$ in Q and a sequence of strings $s_0, s_1, \ldots, s_m$ in Γ^* such that (1) $r_0= q_0$, $s_0= \varepsilon$, (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 $a,b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$, and (3) $r_m \in F$.

Remarks on the Definition

- Notice the machine is a generalization of an NFA not a DFA.
- One can show deterministic pushdown automata are a strictly weaker then nondeterministic pushdown automata.

Example

- We can define a machine to recognize $\{0^n1^n \mid n>=0\}$ as $M=(Q, \Sigma, \Gamma, \delta, q_1, F)$ where:

  Q={q1, q2, q3, q4}

  Σ={0,1}

  Γ={0,$}

  F={q1, q4}

  and δ={(q1, ε, ε)->(q2, $),

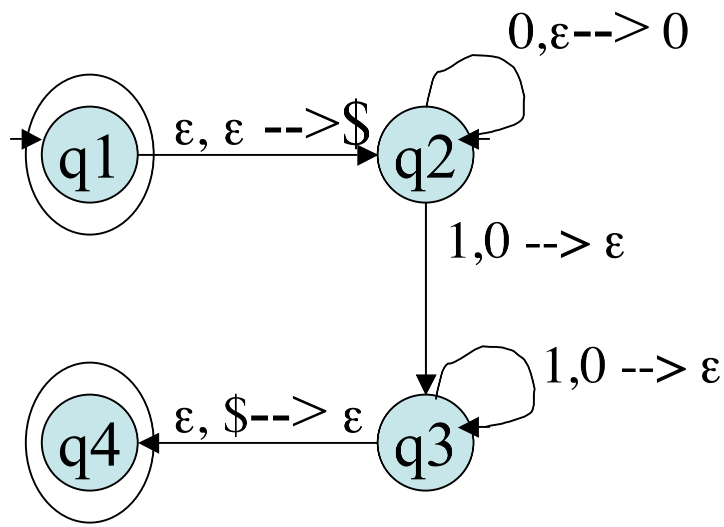
  (q2, 0, ε)-->(q2, 0),

  (q2, 1, 0)-->(q3, ε),

  (q3, 1, 0)-->(q3, ε)

  (q3, ε,$)-->(q4, ε)

  }



- Can then using the definition show this machine accepts 0011.

Equivalence

- We now works towards showing a language is context free if and only if some pushdown automata recognizes it.
- The proof split into two parts:
  - If a language is context-free then some pushdown automata recognizes it
  - If a pushdown automata recognizes some language then there is a context-free grammar that recognizes the same language.

CFL=> PDA recognizes

- Let A be a CFL. Let G be a CFG for this language, and let w be a string generated by G (and hence in A). We will have a machine with three main states $\{q_{start}, q_{loop}, q_{accept}\}$ together with some auxiliary states E.

1. We have transitions $(q_{start}, \varepsilon, \varepsilon)$-->$(q', \$)$ and $(q', \varepsilon, \varepsilon)$-->$(q_{loop}, S)$ that push the start variable S of the CFG onto our machine's stack.
2. Then what we want to do is to simulate the steps to generate w on our PDAs stack.
   a) If A is a variable of the CFG on the top fo the stack, and we are in the state q_{loop} we nondeterministically choose a rule A->$w_1 w_2 .. w_n$ and using a sequence of transitions $(q_{loop}, \varepsilon, A)$ --> (q_1, w_n) , $(q_1, \varepsilon, \varepsilon)$ -->(q_2, w_{n-1}) … $(q_n, \varepsilon, \varepsilon)$ -->(q_{loop}, w_1) We simulate this rule on the stack. Here q_i are some of the auxiliary states in E.
   b) To handle a terminal such as b on the top of the stack we have transitions (q_{loop}, b, b) --> (q_{loop}, ε).
3. Finally, we have a transition $(q_{loop}, \varepsilon, \$)$ -->$(q_{accept}, \varepsilon)$ where q_{accept} is our accept state.

PDA recognizes => CFL

- Let P be a PDA. We want to make a CFG G that generates the same language.
- For each pair of states p,q in P we will have in G a variable A_{pq}. This variable will be able to generate all strings that can take P in state p with the empty stack to state q with the empty stack.
- To simplify the problem we will assume P has been modified so that:
  - it has a single accept state
  - it empties its stack before accepting
  - each transition either pushes a symbol onto the stack or pops one off of the stack (but not both). (We might add states to make our machine have this property).
- G will have rules A_{pp} --> ε for each state p of P; A_{pq}--> $aA_{rs}b$ for each p,q, r,s such that δ(p,a, ε) contains (r,t) and δ(s,b, t) contains (q, ε), and A_{pq}-> $A_{pr}A_{rq}$for any state r.
- The start variable of G will be $A_{q0,qaccept}$.