Supplementary Lecture F

Deterministic Pushdown Automata

A *deterministic pushdown automaton* (DPDA) is an octuple

$$M = (Q, \Sigma, \Gamma, \delta, \bot, \dashv, s, F),$$

where everything is the same as with NPDAs, except:

(i) $\dashv$ is a special symbol not in Σ, called the *right endmarker*, and

$$\delta \subseteq (Q \times (\Sigma \cup \{\dashv\} \cup \{\epsilon\}) \times \Gamma) \times (Q \times \Gamma^*).$$

(ii) δ is *deterministic* in the sense that exactly one transition applies in any given situation. This means that for any $p \in Q$, $a \in \Sigma \cup \{\dashv\}$, and $A \in \Gamma$, δ contains exactly one transition of the form $((p, a, A), (q, \beta))$ or $((p, \epsilon, A), (q, \beta))$.

(iii) δ is restricted so that $\bot$ is always on the bottom of the stack. The machine may pop $\bot$ off momentarily, but must push it directly back on. In other words, all transitions involving $\bot$ must be of the form $((p, a, \bot), (q, \beta\bot))$.

The right endmarker $\dashv$ delimits the input string and is a necessary addition. With NPDAs, we could guess where the end of the input string was, but with DPDAs we have no such luxury.

The restriction in (iii) is so that the machine never deadlocks by emptying its stack. This assumption is without loss of generality; even if the machine

did not obey it, we could make it do so by a construction involving a new stack symbol $\perp\!\!\!\perp$, as in Supplementary Lecture E. In that construction, we must modify the transitions (E.2) to read the symbol $\dashv$.

We consider only acceptance by final state. One can define acceptance by empty stack and prove that such machines are equivalent. The assumption (iii) would have to be modified accordingly.

The definitions of configurations and acceptance by final state are the same as with NPDAs. The start configuration on input $x \in \Sigma^*$ is $(s, x \dashv, \perp)$, and x is accepted iff

$$(s, x \dashv, \perp) \xrightarrow[M]{*} (q, \epsilon, \beta)$$

for some $q \in F$ and $\beta \in \Gamma^*$.

A language accepted by a DPDA is called a *deterministic context-free language* (DCFL). Surely every DCFL is a CFL, since every DPDA can be simulated by an NPDA. In this lecture we will show that the family of DCFLs is closed under complement. We know that there exist CFLs whose complements are not CFLs; for example,

$$\{a, b\}^* - \{ww \mid w \in \{a, b\}^*\}.$$

These CFLs cannot be DCFLs. Thus, unlike finite automata, nondeterminism in PDAs gives strictly more power.

To show that the DCFLs are closed under complement, we will construct, given any DPDA M with input alphabet Σ, a new DPDA M' such that $L(M') = \Sigma^* - L(M)$.

We would like to build M' to simulate M and accept iff M does not accept. Unfortunately, we cannot just switch accept and nonaccept states like we did with DFAs. The main difficulty here is that unlike finite automata, DPDAs need not scan all of their input; they may loop infinitely on inputs they do not accept without reading the entire input string. The machine M' will have to detect any such pathological behavior in M, since M' will have to scan the entire input and enter an accept state on all those inputs that are not accepted by M, including those inputs on which M loops prematurely.

We solve this problem by showing how to modify M to detect such spurious looping and deal with it gracefully. After each modification step, we will argue that the resulting machine still accepts the same set as M and is still deterministic.

Checking for End of Input

It will be useful to include one bit of information in the finite control of M to remember whether or not M has seen the endmarker $\dashv$ yet. Formally, we duplicate the finite control Q to get a new copy $Q' = \{q' \mid q \in Q\}$ disjoint from Q and add a new transition

$$((p', a, A), (q', \beta))$$

for each transition $((p, a, A), (q, \beta)) \in \delta$. We remove any transition of the form $((p, \dashv, A), (q, \beta))$ and replace it with $((p, \dashv, A), (q', \beta))$. The primed states thus behave exactly like the unprimed original states, except that we jump from an unprimed state to a primed state when we scan the endmarker. The start state will still be s, but we will take as final states all primed states corresponding to final states in the old machine M; that is, the new set of final states will be

$$F' = \{q' \mid q \in F\}.$$

The new machine is still deterministic, since there is still exactly one transition that applies in any configuration. It accepts the same set, since if

$$(s, x \dashv, \bot) \xrightarrow[M]{*} (q, \epsilon, \gamma), \quad q \in F$$

in the old machine, then there must be some intermediate transition that reads the $\dashv$:

$$(s, x \dashv, \bot) \xrightarrow[M]{*} (p, \dashv, A\beta) \xrightarrow[M]{1} (r, \epsilon, \alpha\beta) \xrightarrow[M]{*} (q, \epsilon, \gamma).$$

Then in the new machine,

$$(s, x \dashv, \bot) \xrightarrow[M']{*} (p, \dashv, A\beta) \xrightarrow[M']{1} (r', \epsilon, \alpha\beta) \xrightarrow[M']{*} (q', \epsilon, \gamma).$$

Conversely, if

$$(s, x \dashv, \bot) \xrightarrow[M']{*} (q', \epsilon, \gamma), \quad q' \in F'$$

in the new machine, then removing the primes gives a valid accepting computation sequence

$$(s, x \dashv, \bot) \xrightarrow[M]{*} (q, \epsilon, \gamma), \quad q \in F$$

in the old machine.

Now we can tell from information in the state whether we have seen $\dashv$ or not. With this information, we can make the machine remain in an accept state if it has already scanned the $\dashv$ and accepted. This is done by deleting every primed transition $((p', \epsilon, A), (q', \beta))$ with $p' \in F'$ and replacing it with $((p', \epsilon, A), (p', A))$.

Getting Rid of Spurious Loops

We include two new nonaccept states r and r' and transitions

$$\begin{array}{ll} ((r,a,A),\ (r,A)), & a \in \Sigma,\ A \in \Gamma, \\ ((r,\dashv,A),\ (r',A)), & A \in \Gamma, \\ ((r',\epsilon,A),\ (r',A)), & A \in \Gamma. \end{array}$$

We can think of r' as a reject state. If the machine is in state r, it will always scan to the end of the input and enter state r', leaving the stack intact. Once the machine is in state r', it stays there. (So far there is no way for the machine to reach state r or r', but just wait...)

We will now show how to modify the machine so that for all inputs, the machine scans the entire input and enters either an accept state or the state r'. Let x be any input. Because of determinism, there is a unique infinite sequence of configurations the machine goes through on input x. Let γ_i denote the stack contents at time $i \in \{0,1,2,\ldots\}$. There exists an infinite sequence of times $i_0 < i_1 < i_2 < \cdots$ such that for all i_k,

$$|\gamma_{i_k}| \leq |\gamma_i|, \quad i \geq i_k. \tag{F.1}$$

We can take $i_0 = 0$, since $\gamma_0 = \perp$ and $|\gamma_0| = 1$, and the machine never empties its stack. Proceeding inductively, we can take i_{k+1} to be the earliest time after i_k such that $|\gamma_{i_{k+1}}|$ is minimum among all $|\gamma_i|$, $i > i_k$.

Now we pick an infinite subsequence $j_0 < j_1 < j_2 < \cdots$ of $i_0 < i_1 < i_2 < \cdots$ such that the same transition, say $((p,\epsilon,A),\ (q,\beta))$, is applied at times $j_0, j_1, j_2, \ldots$. Such a subsequence exists by the pigeonhole principle: there are only finitely many transitions in δ, so at least one must be applied infinitely often. The states p, q can be primed or unprimed. The transition must be an ϵ-transition, since it is applied infinitely often, and there are only finitely many input symbols to scan.

By (F.1), the machine never sees any stack symbol below the top symbol of γ_{j_k} after time j_k, and the top symbol is A. Thus the only stack symbols it sees after time j_k are those it pushes after time j_k. Since the machine is deterministic, once it applies transition $((p,\epsilon,A),\ (q,\beta))$, it is in a loop and will go through the same periodic sequence of ϵ-transitions repeated forever, since it sees nothing that can force it to do anything different. Moreover, this behavior is independent of the input. Thus if p is not an accept state, then the input is not accepted. We might as well remove the transition $((p,\epsilon,A),\ (q,\beta))$ from δ and replace it with the transition $((p,\epsilon,A),\ (r,A))$ if p is an unprimed state or $((p,\epsilon,A),\ (r',A))$ if p is a primed state. The language accepted by the automaton is not changed.

If this is done for all transitions $((p,\epsilon,A),\ (q,\beta))$ causing such spurious loops, we obtain a machine equivalent to M that on any input scans the

entire input string and the endmarker $\dashv$ and enters either an accept state or the state r'. To get a machine M' accepting the complement of $L(M)$, make r' the unique accept state of M'.

Historical Notes

Deterministic PDAs were first studied by Fischer [37], Schützenberger [112], Haines [54], and Ginsburg and Greibach [44].