

DETERMINISTIC PUSHDOWN AUTOMATA

- α and β are **consistent** iff α is a prefix of β or the other way around
- Two transitions of a PDA $((p, a, \beta), (q, \gamma))$ and $((p, a', \beta'), (q, \gamma'))$ are **compatible** iff a and a' are consistent **and** β and β' are consistent; then
- A **deterministic** PDA is a PDA in which **no two distinct transitions are compatible**
- A language L is **deterministic context-free** iff $L\{\$$ is accepted by a deterministic PDA
 - A deterministic PDA is also able to sense the end of the input string

CLOSURE UNDER COMPLEMENT

- Can we just reverse final and non-final states? **No!**
 - Indeed, a PDA that accepts (for example) $\{w cw^{\mathbb{R}} : w \in \{a, b\}^*\}$ will have only one state; the stack being empty or not determines whether the run is successful or not at the end
- A configuration $C = (q, w, \alpha)$ is a **deadend** if whenever $C \vdash^* C'$ then $C' = (q', w, \alpha')$ and $|\alpha'| \geq |\alpha|$
- Consider now a **simple** deterministic PDA: we can then **detect deadends** without running the automaton
 - We do this by inspecting the current state, next input symbol, and the top of the stack:
 - (q, a, A) (**viewed as a configuration**) is a deadend iff it does not yield (p, ε, α) or (p, a, ε)
- Let D be the set of all deadend configurations
- For each (q, a, A) we then remove from Δ all transitions $((q, a, A), (p, \beta))$ and we replace them with the transition $((q, a, A), (r, \varepsilon))$ (where r is a new, non-final state)

CLOSURE UNDER COMPLEMENT (CONT'D)

- We then add the transitions $((r, a, \varepsilon), (r, \varepsilon))$ for all $a \in \Sigma$
- We finally add $((r, \$, \varepsilon), (r', \varepsilon))$ and $((r', \varepsilon, A), (r', \varepsilon))$ for each $A \in \Gamma \cup \{Z\}$ (r' is once more new, non-accepting)
- Call this new PDA M' ; it still accepts $L\{\$\}$
- **Now** we reverse r' and f' and so obtain M' which accepts $\overline{L}\{\$\}$
- **Corollary:** Deterministic context-free languages are a strict subset of context-free languages
 - When it comes to context-free languages **nondeterminism is more powerful**

ALGORITHMS FOR CONTEXT-FREE LANGUAGES

- The conversions between a context-free grammar and a pushdown automata take polynomial time (see the constructions used in the equivalence proof)
- The most practically important problem related to context-free languages is **parsing**: Given a grammar G and a string w , to determine whether $w \in \mathcal{L}(G)$
- Parsing also takes polynomial time
 - The top-down parser built in the equivalence proof takes exponential time
 - However, better house-keeping (and some canonical form of the grammar similar in spirit with the simple automaton) bring down the complexity to polynomial
 - Better house-keeping means a form of dynamic programming
 - Details for the curious are on pages 151–157

EFFICIENT PARSING

- In a real-world compiler polynomial parsing will not do
- We want instead to reach the theoretical lower bound for the problem: **linear time = deterministic PDA**
- For this to be feasible we retort to **bottom-up parsing**
 - Starting from $G = (V, \Sigma, S, R)$ we construct the automaton $M = (\{p, q\}, \Sigma, V, \Delta, s, \{q\})$ with Δ containing exactly all the following transitions:

$$\begin{array}{ll} \text{shift} & \forall a \in \Sigma : ((p, a, \varepsilon), (p, a)) \\ \text{reduce} & \forall A \rightarrow \alpha : \in R \quad \begin{array}{l} ((p, \varepsilon, \alpha^R), (p, A)) \\ ((p, \varepsilon, S), (q, \varepsilon)) \end{array} \end{array}$$

- Still nondeterministic:
 - When to shift and when to reduce?
 - * Establish a **precedence relation** $P \subseteq V \times (\Sigma \cup \{\$\})$
 - * Whenever $(\text{stack-top}, \text{input}) \in P$ we reduce and we shift otherwise

EFFICIENT PARSING (CONT'D)

- Still nondeterministic (cont'd):
 - When we reduce, with what rule we reduce?
 - * We use the **logest rule = greedy** (eat up the longest stack top)
- We get deterministic parsing for **weak-precedence grammars** = most programming languages

WEAK-PRECEDENCE GRAMMAR (EXAMPLE)

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow (E)$$

$$F \rightarrow a$$

$$((p, a, \varepsilon), (p, \varepsilon))$$

$$((p, \varepsilon, T + E), (p, E))$$

$$((p, \varepsilon, T), (p, E))$$

$$((p, \varepsilon, F * T), (p, T))$$

$$((p, \varepsilon, F), (p, T))$$

$$((p, \varepsilon,)E(), (p, F))$$

$$((p, \varepsilon, y), (p, F))$$

	()	y	+	*	\$
(
)	✓			✓	✓	✓
y	✓			✓	✓	✓
+						
*						
E						
T	✓			✓		✓
F	✓			✓	✓	✓