DETERMINISTIC PUSHDOWN AUTOMATA



CS 455/555: Deterministic context-free languages

Stefan D. Bruda

Fall 2020

- 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

CS 455/555 (S. D. Bruda

Fall 2020

6

CLOSURE UNDER COMPLEMENT



CLOSURE UNDER COMPLEMENT (CONT'D)



- Can we just reverse final and non-final states? No!
 - Indeed, one can construct a PDA that accepts (for example) the language of balanced parentheses and has a single 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 $|\alpha'| \ge |\alpha|$
- 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 $(q, a, A) \in D$ 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)

- 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

CS 455/555 (S. D. Bruda) Fall 2020 2 / 6 CS 455/555 (S. D. Bruda) Fall 2020 3 / 6

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 ∈ 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
- 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
 - Top-down parsing possible in linear time for certain kind of grammars (LL(1))
 - Need to be able to decide what rule to use based on the next input symbol
 - Most programming languages have LL(1) grammars, but often they are not very readable

CS 455/555 (S. D. Bruda) Fall 2020 4 / 6



WEAK-PRECEDENCE GRAMMAR (EXAMPLE)

Grammar

$$\begin{array}{cccc}
E & \rightarrow & E + T \\
E & \rightarrow & T \\
T & \rightarrow & T * F \\
T & \rightarrow & F \\
F & \rightarrow & (E) \\
F & \rightarrow & y
\end{array}$$

Bottom-up parser

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

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

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

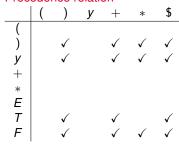
$$((p, \varepsilon, F * T), (p, T))$$

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

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

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

Precedence relation



CS 455/555 (S. D. Bruda) Fall 2020 6 /

BOTTOM-UP PARSING



- Bottom-up parsing is slightly more convenient in practice
 - 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:

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

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

CS 455/555 (S. D. Bruda) Fall 2020 5 / 6