CFG and PDA Equivalence

trip today (3/9) →

← trip Friday (3/11)

CFG and PDA equivalence

- Theorem: **A language is context free if and only if some pushdown automaton recognizes it**

- Today's lecture forward direction of this result: **If a language is context-free then some pushdown automaton recognizes it**
  - we will see how to convert a CFG G into an equivalent PDA P

- Friday's we will see the other direction: **If a pushdown automaton recognizes some language, then it is context free**
  - we will see how to convert a PDA P into an equivalent CFG G

Strategy for designing P

- Let w be the input string
- The PDA P will accept w if there is a way to derive w using G
- Recall that each step of a derivation yields an **intermediate string** of variables and terminals
- We design P to determine whether some series of substitutions using the rules of G can lead from the *start variable* to w

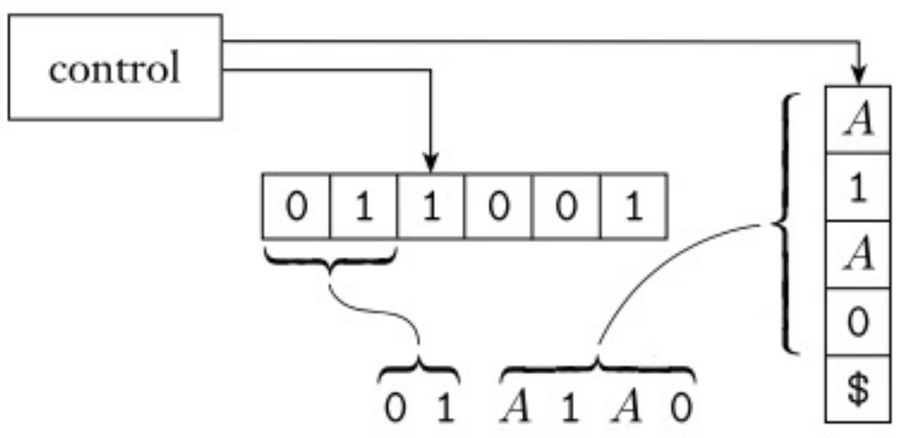
- Testing whether there is a derivation for w
  - *Issue*: figuring out which substitution to use?
  - *Solution*: selection is done non-deterministically

Strategy for designing P

- The PDA P begins by writing the start variable on its stack
- It goes through a series of intermediate strings, making one substitutions after another
- Eventually, it may arrive at a string that contains only terminal symbols
  - meaning that it has used the grammar to derive a string
- Then P accepts if this string is identical to the string it has received as input

- Implementing this strategy on a PDA requires one additional idea
  - We need to see how the PDA stores intermediate strings
  - Storing each intermediate in the stack won't work because the PDA needs to find ***the variables*** in the intermediate string to make substitutions and the PDA can access only ***the top*** of the stack
  - *Solution:* keep only part of the ***part*** of the intermediate string on the stack – the symbols starting with the first variable in the intermediate string. Terminal symbols appearing before the first variable are matched immediately with symbols in the input string.

Strategy for designing P



P representing the intermediate string 01A1A0

Informal description of P

1. Place the marker symbol $ and the start variable on the stack
2. Repeat the following steps for ever
   a. If the top of stack is a variable symbol A, non-deterministically select one of the rules for A and substitute A by the string on the RHS of the rule.
   b. If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, repeat. If they do not match, reject on this branch of nondeterminism.
   c. If the top of stack is the symbol $, enter the accept state. Doing so accepts the input if it has all been read.

Formal details of the construction of P

- $P = (Q, \Sigma, \Gamma, \delta, q_{start}, F)$
- To make the construction clearer, we use a **shorthand notation** for the transition function
  - the notation provides a way to write an entire string on the stack in one step of the machine
  - we can simulate this action by introducing additional states to write the string one symbol at a time

The shorthand

- Let q and r be the states of the PDA
- Let a be in Σ_ε and s be in Γ_ε
- Say we want the PDA to go from q to r when it reads a and pops s
- Furthermore, we want to push the entire string $u = u_1 \dots u_l$ on the stack at the same time
- We can implement this action by introducing new states $q_1, \dots, q_{l-1}$ and setting the transition function as follows:

$$
\begin{aligned}
&\delta(q, a, s) \text{ to contain } (q_1, u_l), \\
&\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, u_{l-1})\}, \\
&\delta(q_2, \varepsilon, \varepsilon) = \{(q_3, u_{l-2})\}, \\
&\qquad\qquad \vdots \\
&\delta(q_{l-1}, \varepsilon, \varepsilon) = \{(r, u_1)\}.
\end{aligned}
$$

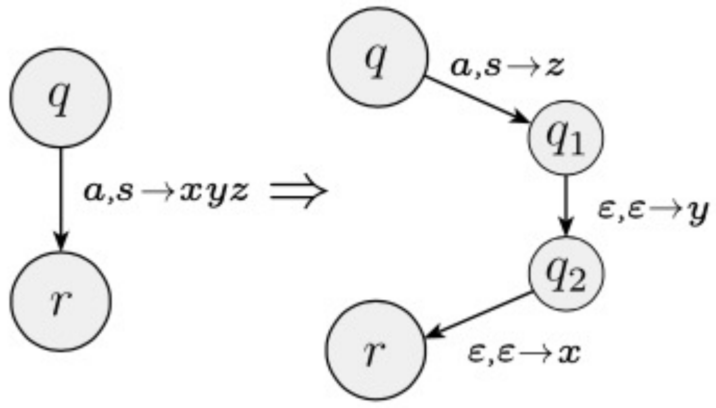
The shorthand

We use the notation (r, u) ∈ δ(q, a, s) to mean that when q is the state of the automaton, a is the next input symbol, and s is the symbol on the top of the stack, the PDA may

read the a and pop the s,

push the string u onto the stack and

go to the state r



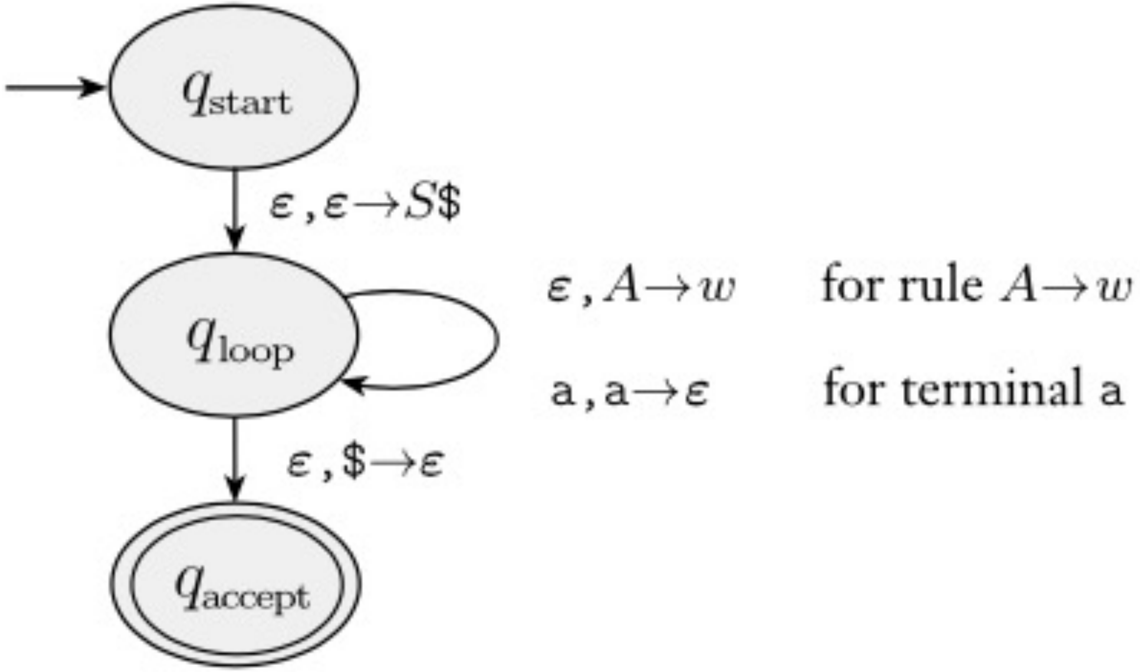
Implementing the shorthand (r, xyz) ∈ δ(q, a, s)

Coming back to the formal description of P

- The states of P are $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$, where E is the set of states we need for implementing the shorthand

- The transition function is defined as follows
  - We begin by initializing the stack to contain \$ and S: (Step 1 in the inf. desc. of P)
    - $\delta(q_{start}, \varepsilon, \varepsilon) = \{(q_{loop}, S\$)\}$
  - Then we put in transitions for the main loop (Step 2 in the inf. desc. of P)
    1. Case a: Let $\delta(q_{loop}, \varepsilon, A) = \{(q_{loop}, w) \mid \text{where } A \longrightarrow w \text{ is a rule in } R\}$
    2. Case b: Let $\delta(q_{loop}, a, a) = \{(q_{loop}, \varepsilon)\}$
    3. Case c: Let $\delta(q_{loop}, \varepsilon, \$) = \{(q_{accept}, \varepsilon)\}$

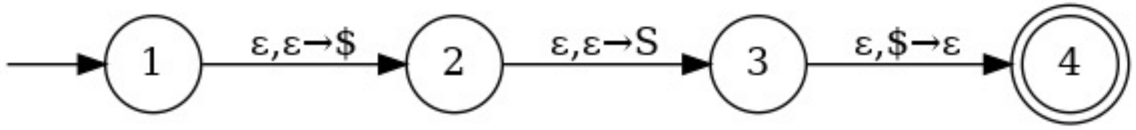
State diagram of P



Example Conversion

- Consider the following CFG from Problem 3 in the Homework
  - $S \rightarrow aSb \mid bY \mid Ya$
  - $Y \rightarrow aY \mid bY \mid \varepsilon$
  - (Note: this problem originally asks for an English description, and a grammar for the complement)
- Suppose we had to transform this grammar into a PDA. How do we do it?
- Start with making four states:
  - State 1: Start state
  - State 2: From state 1 after pushing stack symbol $
  - State 3: From state 2 after pushing S, the start variable in the grammar
  - State 4: Accept state. From state 3 after popping stack symbol $

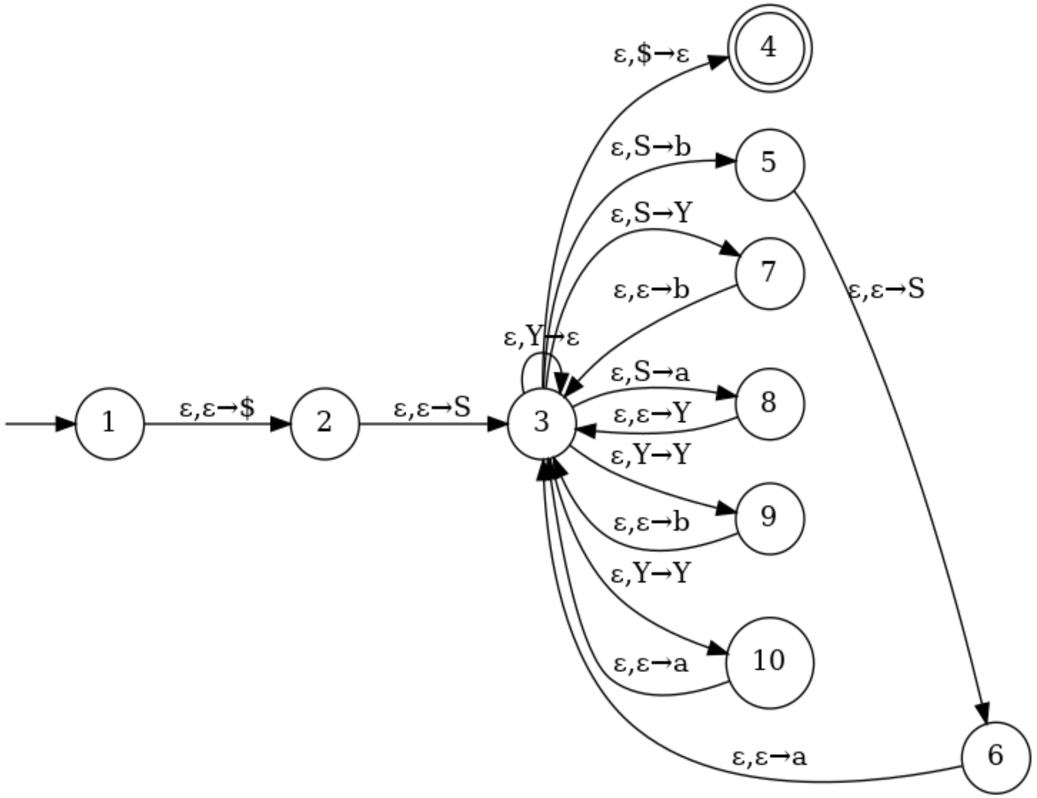
Example Conversion (cont)



- This PDA represents a grammar with no rules
- The stack symbol is never popped, since S is always on top, thus we never reach the accept state.
- We want to replace symbols in the grammar to process the entire string, and accept IFF a string is in the language the grammar generates.
- First step is to create replacement rules for each non-terminal.
- From state 3 (we'll call the "loop state") for each rule in the CFG, push symbols (terminals and non-terminals alike) to stack in reverse order. Create new states if multiple symbols need to be pushed.

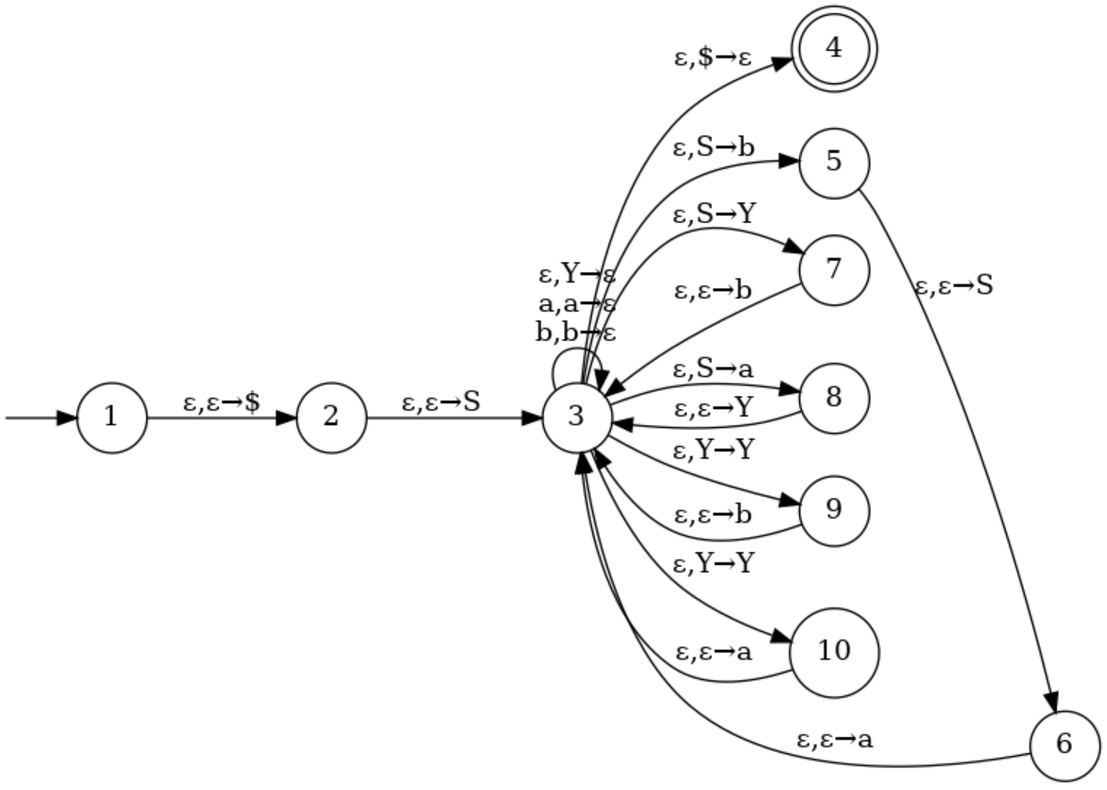
Example Conversion (cont)

- Recall the syntax for PDA transitions: input, stack pop → stack push
- Our PDA now pops non-terminals from the stack, so should be able to reach accept state if the empty string is in the language (it isn't).
- All paths from the start now result in a terminal symbol on the top of the stack.
- Furthermore, we have not yet used the input parameter to our PDA.
- Solution: Add a new transition from loop state to itself, reading every terminal symbol and popping it from the stack



Example Conversion (cont)

- Final PDA should accept all strings the CFG generates, and reject all strings it doesn't generate.
- PDA works by pushing and popping the non-terminals of its derivation in CFG.
- If the string is in the language, all symbols are read and nothing remains on the stack but $, taking us to the accept state.
- Otherwise, we are left with something on the stack, and cannot accept.



Homework Hints

- Most problems are not meant to be difficult, but a few are somewhat tedious (having group members helps, I'd hope)
- Problems 2 and 3 may be the exception here, since they aren't solved just by following an algorithm.
- Problem 2: Showing a grammar is ambiguous requires finding a string that has two different derivations.
  - Hint: We have two very similar non-terminals in <IF> and <IF-ELSE>
  - What if a string had both an if statement and an if-else statement (i.e. a nested if statement)?
- Problem 3: Needs an English description of a language, and also needs to find the complement of the language.
  - Hint: Knowing either the English description or the complement makes the other trivial. You can describe a language by what's not in it!

Homework Hints (cont)

- Problems 4-6 should all be able to be solved algorithmically.
- Follow the procedure described today to handle 5 and 6. You may end up with a *lot* of states in problem 6, so get started on it soon if you haven't already.
- For problem 4, We have provided a document describing the procedure for transforming CFG → CNF on Canvas.
  - Note: When describing the types of rules NOT allowed in CNF, we forgot a major one.
  - A → ab is an example of another invalid rule (multiple terminals)
  - When in doubt, only the three types of rules described as valid (two non-terminals, one terminal, or start to epsilon) should be used.
- Problem 4a is probably the most tedious. If each of the Rs in R1R1R1R can be replaced with ε, then this will be turned into 16 different rules. Please don't wait until the last minute to attempt this!