CS5371 Theory of Computation

Lecture 8: Automata Theory VI (PDA, PDA = CFG)

Objectives

- Introduce Pushdown Automaton (PDA)
- Show that PDA = CFG
  - In terms of descriptive power

Pushdown Automaton (PDA)

- Roughly speaking, PDA = NFA + stack with unlimited size
- How does a PDA operate?
  - In each step, it can read a character from the input string, can pop (remove) a symbol from the stack
  - Then, depending on the character and the symbol, the PDA enters another state and can push (place) a symbol to the stack

A stack is a "last in, first out" storage device

Stack is powerful

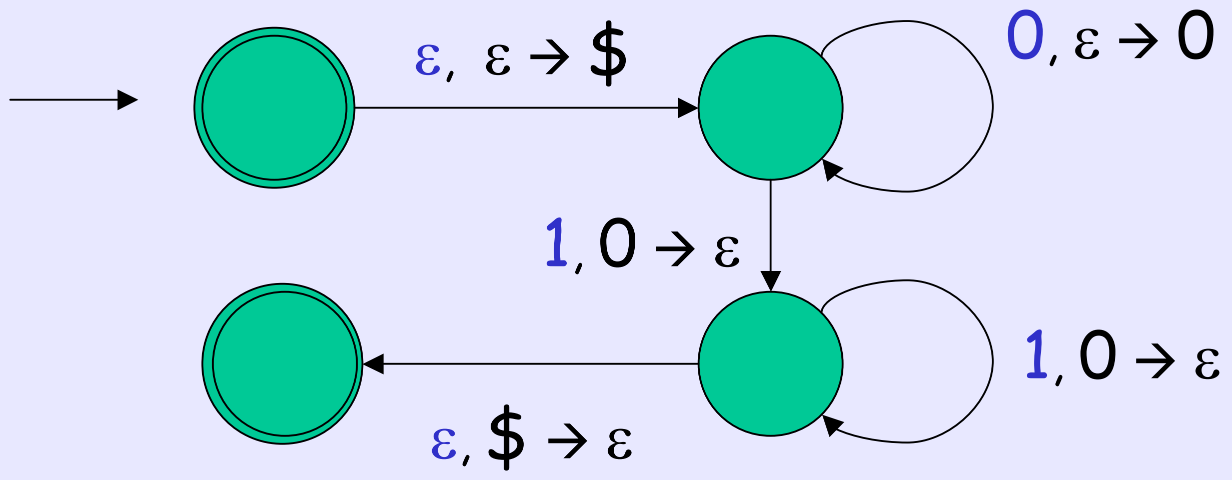
- Recall that an NFA cannot recognize the language $\{0^n1^n \mid n \geq 0\}$
- However, a PDA can do so (informally):
  - Read the characters from input
  - For any 0 it reads, push it onto the stack
  - As soon as 1s are seen**, pop a 0 off the stack for each 1 read

    ** after this point, if we read a 0 again, we reject the string immediately

  - Accept the string if the stack is just empty after the last 1 is read; Reject the string otherwise

PDA (Example)

- The following state diagram gives the PDA that recognizes $\{0^n1^n \mid n \geq 0\}$.



The notation a, b → c means that the machine reads a from input, replace b by c from the top of stack. That is, pop b then push c.

PDA (Formal Definition)

- A PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$
  - Q is a finite set of states
  - Σ is a finite set of characters
  - Γ is a finite set of stack symbols
  - δ is the transition function of the form:

    $$\delta : Q \times \Sigma' \times \Gamma' \rightarrow 2^{Q \times \Gamma'},$$

    where $\Sigma' = \Sigma \cup \{\varepsilon\}$ and $\Gamma' = \Gamma \cup \{\varepsilon\}$
  - q_0 is the start state
  - F is the set of accept states

Acceptance by PDA

- A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts the input string w if
  - w can be written as $w_1 w_2 \ldots w_m$ where w_i in Σ',
  - there exist states $r_0, r_1, \ldots, r_m$ in Q, and
  - there exist strings $s_0, s_1, \ldots, s_m$ in Γ^*

  satisfying the following three conditions: (see next slide)

Intuitively, r_i denotes the sequence of states visited by the PDA, and s_i denotes the corresponding contents in the stack (reading from top to bottom)

Acceptance by PDA (cont.)

- Condition 1: $r_0 = q_0$, $s_0 = \varepsilon$

This ensures that PDA starts at q_0, with an empty stack

- Condition 2: For $i = 0, 1, \ldots, m-1$, we have

$$(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a),$$

$$\text{where } s_i = at,\ s_{i+1} = bt \text{ for some } a, b \text{ in } \Gamma'$$

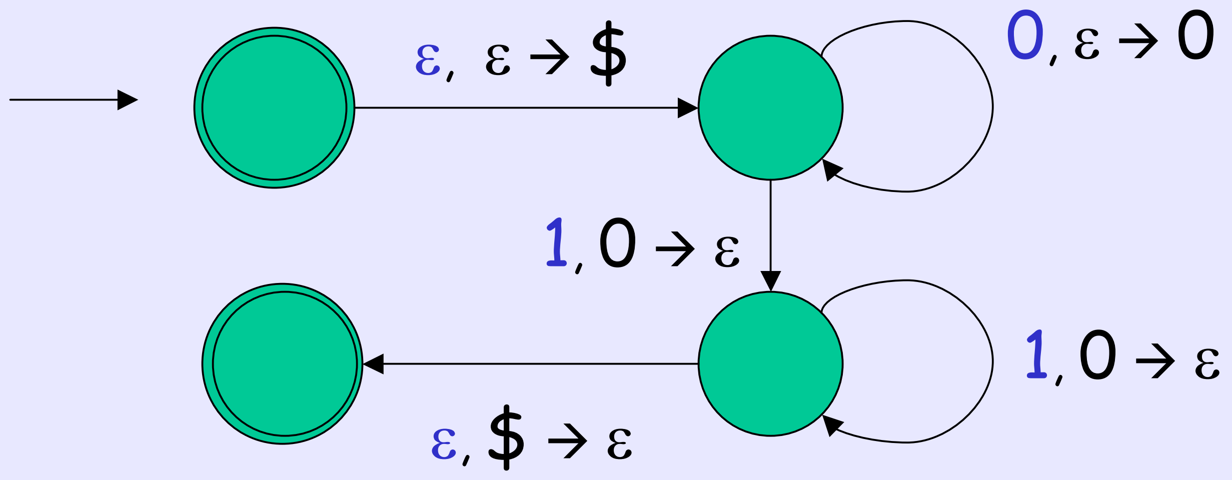
This ensures that PDA moves properly according to the state, the input character, and the stack

- Condition 3: $r_m \in F$

This ensures PDA accepts only when the PDA is in an accept state after processing the whole input string

PDA (example 1)

- Recall that the following PDA recognizes $\{0^n1^n \mid n \geq 0\}$



PDA (example 1)

Some points to notice:

- The formal definition of PDA does not allow us to test if the stack is empty
- The previous PDA tries to get the same effect by first placing \$ to the stack, so that if it ever sees \$ again, it knows the stack is empty
- Similarly, PDA cannot test if the input has all been processed
- The previous PDA can have the same effect because it can stay at the accept states only at the end of the input

PDA (example 1)

- We can also write the formal definition of the previous PDA, call it M, as follows:

  $M = (\{q_1,q_2,q_3,q_4\}, \{0,1\}, \{0,\$\}, \delta, q_1, \{q_1,q_4\})$,

  where δ is given by

  $\delta(q_1, \varepsilon, \varepsilon) = \{ (q_2, \$) \}$, $\delta(q_2, 0, \varepsilon) = \{ (q_2, 0) \}$

  $\delta(q_2, 1, 0) = \{ (q_3, \varepsilon) \}$, $\delta(q_3, 1, 0) = \{ (q_3, \varepsilon) \}$

  $\delta(q_3, \varepsilon, \$) = \{ (q_4, \varepsilon) \}$,

  and for the remaining combinations of (q, x, y),

  $\delta(q, x, y) = \{ \}$

PDA (example 2)

Give a PDA that recognizing the language

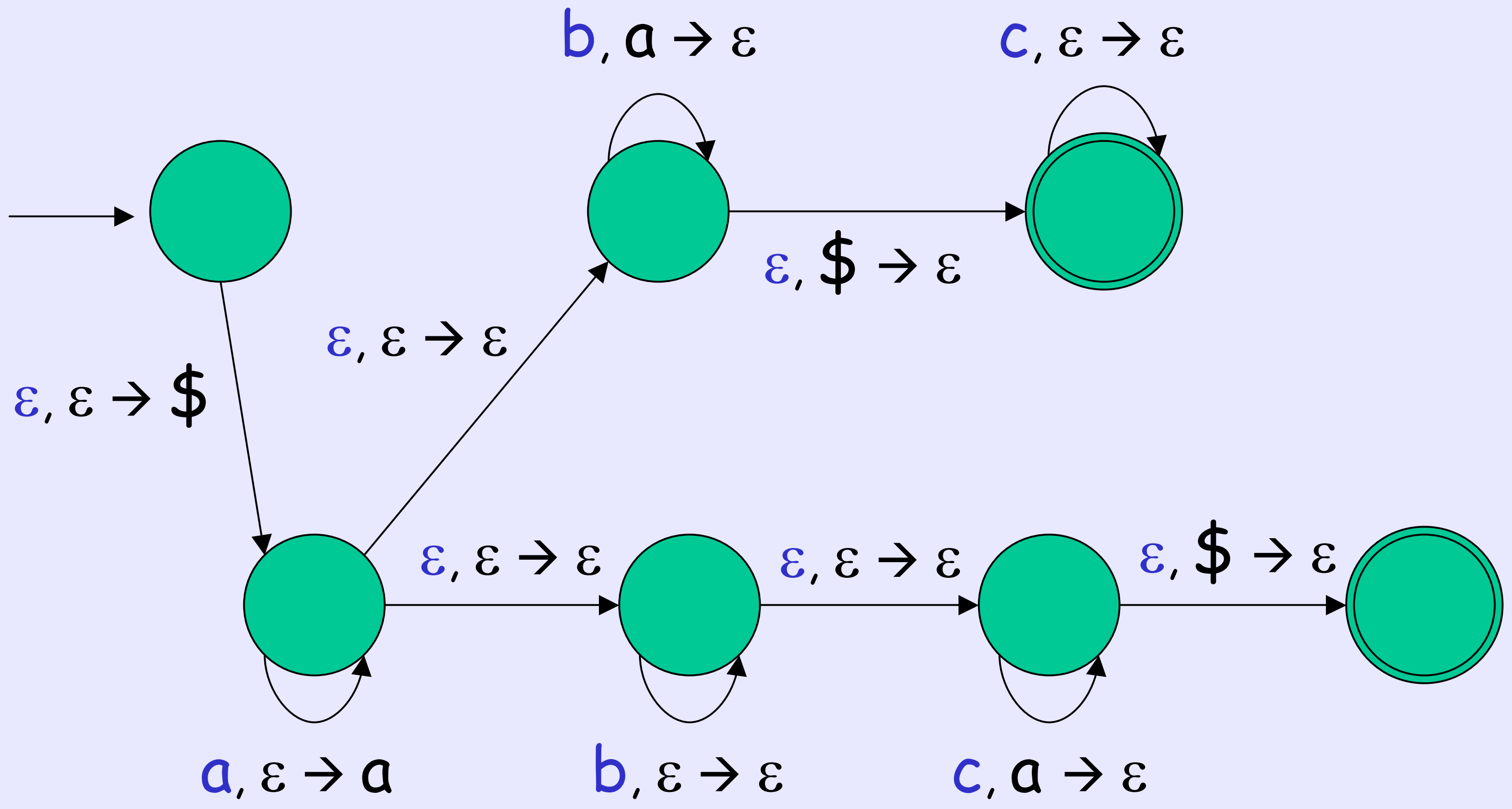
$\{ a^i b^j c^k \mid i,j,k \geq 0 \text{ and } i=j \text{ or } i=k \}$

How to construct it?

- First, for each 'a' read, we should push it onto the stack (for later matching)
- Then, we guess whether 'a' should be matched with 'b' or matched with 'c'

We can guess because of the non-deterministic nature of PDA

PDA (example 2)



PDA (example 3)

Give a PDA recognizing { ww^R | w in {0,1}* }

How to construct it?

- First, the PDA should match the first character with the last character, the second character with the second last character, and so on… So, we push each character that is read to the stack in case it will be matched later
- At each point, we guess the middle of the string has been reached. We match the remaining characters with those stored in the stack (how?)

CFG = PDA

Theorem: (1) If a language is generated by a CFG, it can be recognized by some PDA. (2) If a language is recognized by a PDA, it can be generated by some CFG.

Proof: We shall prove both statements by construction.

Proof of (1)

(1) If a language is generated by a CFG, it can be recognized by some PDA.

- Let L be a language generated by a CFG G. We show how to convert G into an equivalent PDA.
- Our PDA will do the following:
  - For an input string w, it can determine whether there is a derivation for w by G

Recall that a derivation for w = a sequence of substitutions in the grammar that generates w

Proof of (1)

- The difficulty in testing whether there is a derivation for w is to figure out which substitutions to make
  - PDA can do so by guessing the sequence of correct substitutions

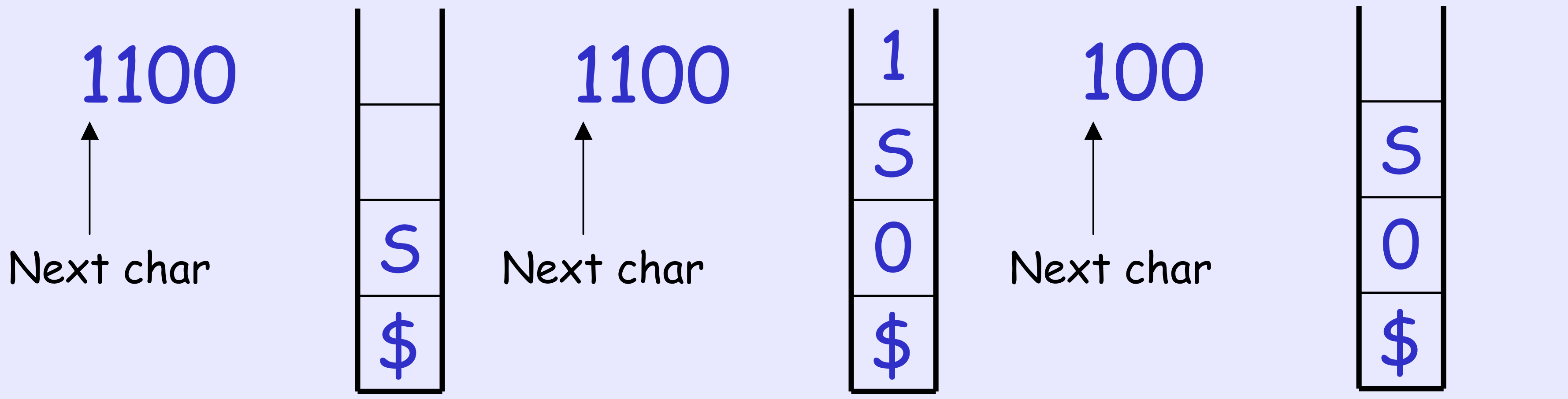
Proof of (1)

- Informally, our PDA does the following:
  - Place the mark symbol \$ and then the start variable S (of G) on the stack
  - Repeat
    - If the top of stack is a variable, say *A*, guess a rule *A* $\rightarrow$ u and substitute *A* by the string u
    - If the top of stack is a terminal, say a, read the next symbol from the input and compare it with a. If they match, repeat. Otherwise, reject this branch of non-determinism
    - If the top of stack is \$, enter the accept state

An Example Run

Input: 1100

CFG: S → SS | 1S0 | 10



At the beginning

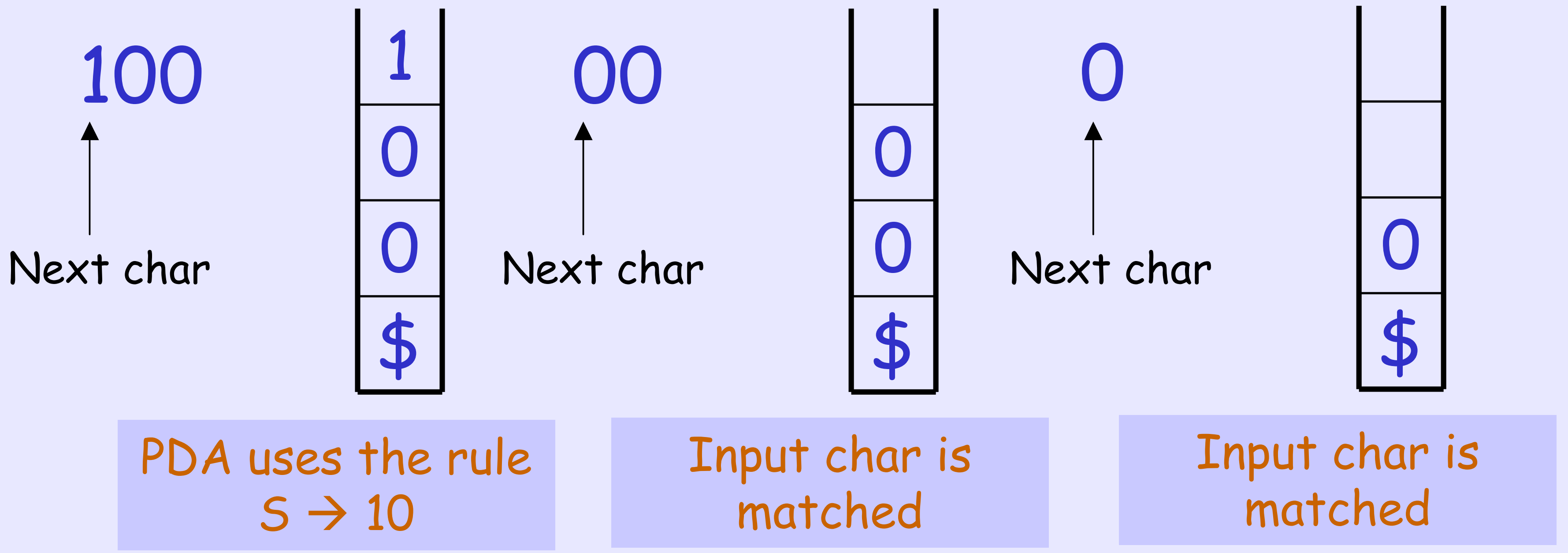
PDA uses the rule S → 1S0

Input char is matched

An Example Run (cont.)

Input: 1100

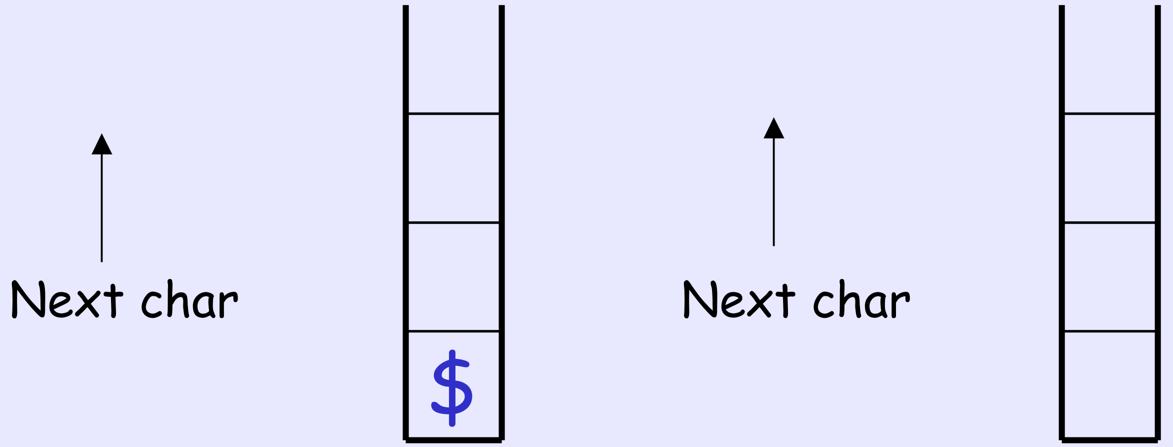
CFG: S → SS | 1S0 | 10



An Example Run (cont.)

Input: 1100

CFG: S → SS | 1S0 | 10



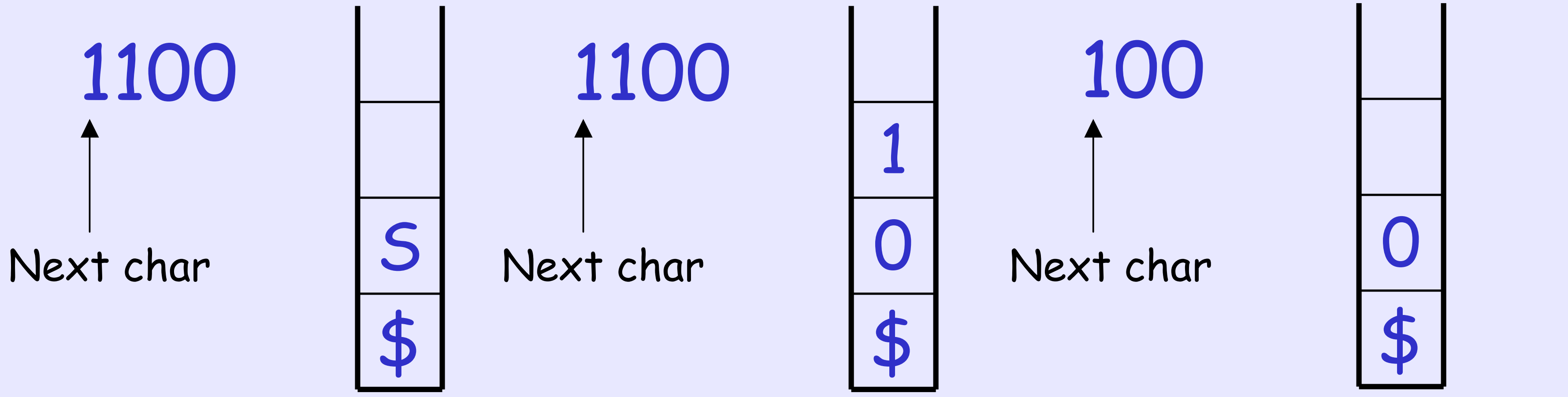
Whole input string is processed

The $ in the top of stack tells PDA the stack is empty. PDA accepts the string

Another Example Run

Input: 1100

CFG: S → SS | 1S0 | 10



At the beginning

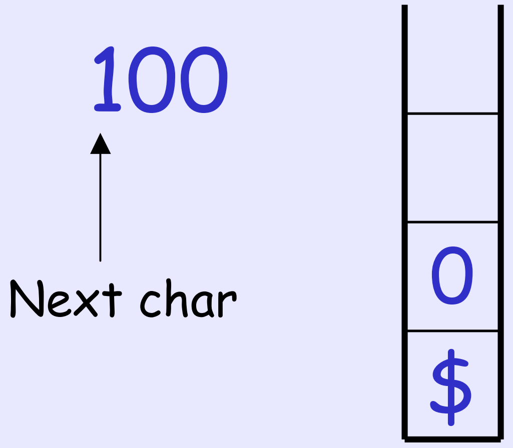
PDA uses the rule S → 10

Input char is matched

Another Example Run (cont.)

Input: 1100

CFG: $S \rightarrow SS \mid 1S0 \mid 10$



Input char cannot match top of stack. What will PDA do?

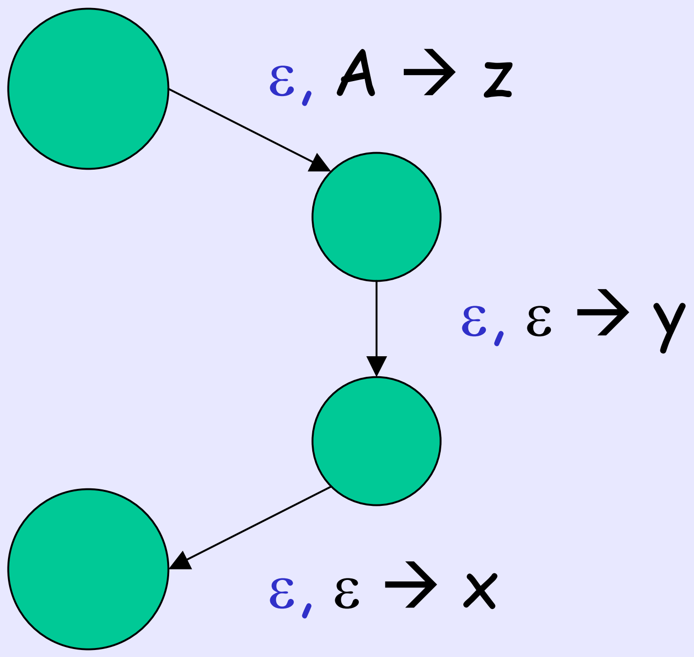
It stops exploring this branch of computation

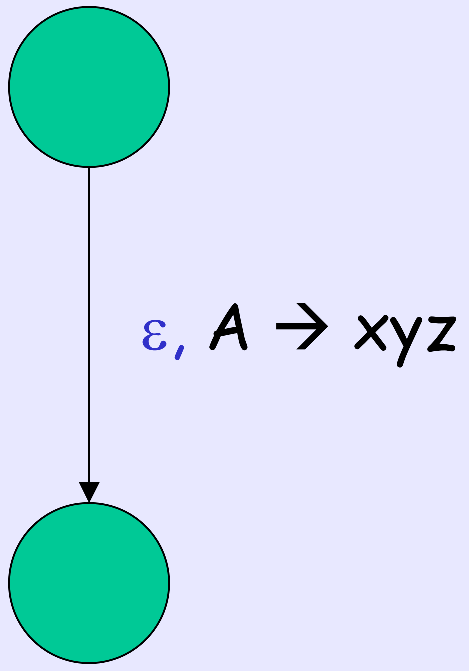
Implementation Details

- We can see that, an input string w is accepted by our PDA if and only if there is a derivation from S to w
- What remains is to show that such a PDA can be constructed
- Pushing \$, pushing S, or matching input chars with terminals in the stack is easy
- Difficulty: How to replace a variable in the top of stack by right side of a corresponding rule? (E.g., top of stack is *A* and we have with a rule *A* → xyz. How to replace *A* by xyz?)

By using dummy states

Using dummy states

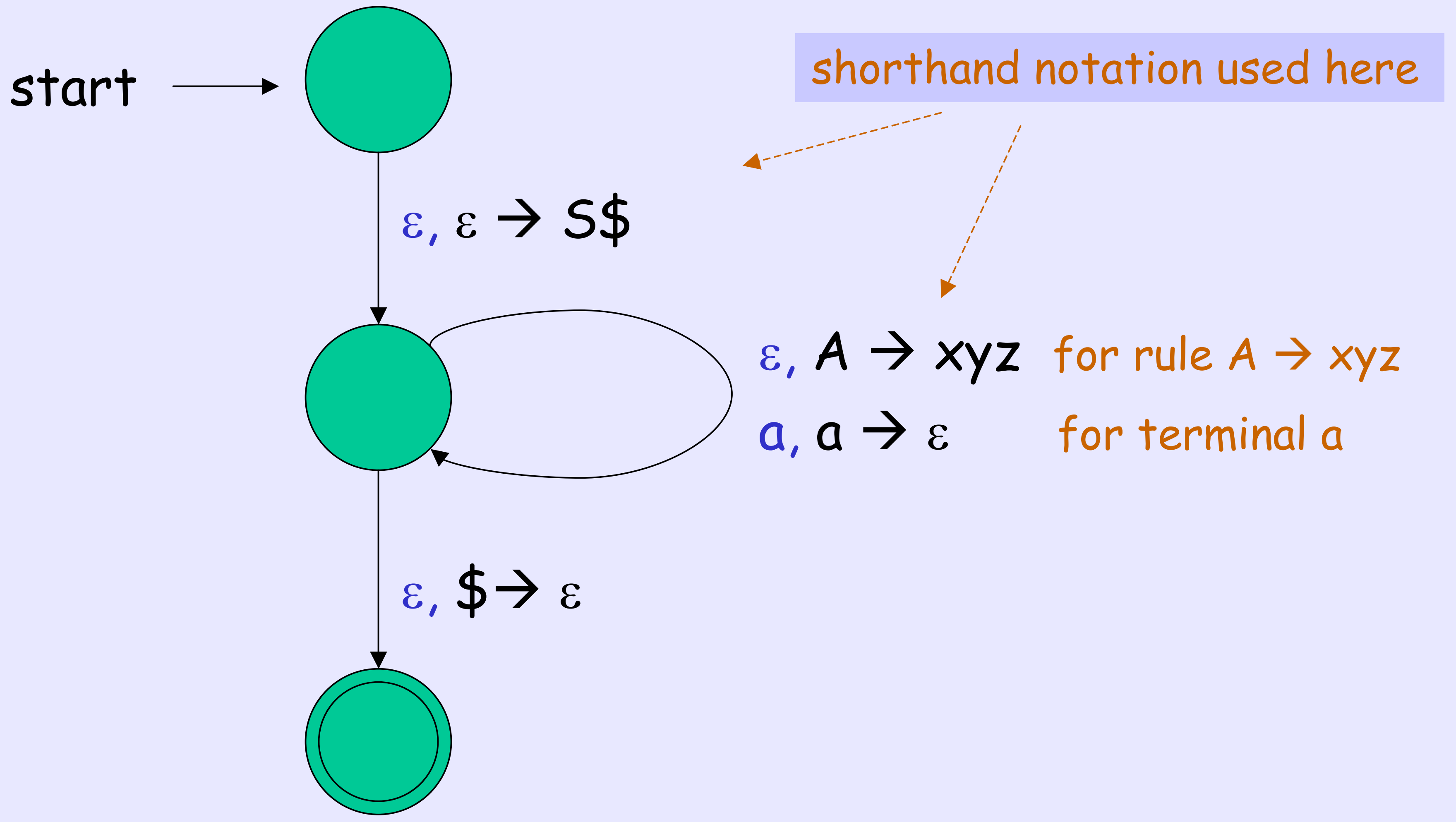




Using two dummy states to replace A by xyz at the top of the stack

A shorthand notation

PDA for Proof of (1)



Example of Conversion

- Convert the following CFG into an equivalent PDA

$$S \rightarrow aTb \mid b$$
$$T \rightarrow Ta \mid \varepsilon$$

Proof of (2)

(2) If a language is recognized by a PDA, it can be generated by some CFG.

- We show how to convert a PDA into an equivalent CFG.
- Let L be the language, and P be the PDA recognizing L.

Proof of (2)

- We first change P slightly so that:
  - It has a single accept state, q_{accept}
  - It empties the stack before accept
  - Each transition either pushes a symbol on the stack, or pops a symbol off the stack, but not both

first and second changes are easy

For the third change, we replace each transition in P that

(i) pushes and pops at the same time with a two transition sequence that goes through a new state,

(ii) neither pushes or pops with a two-transition sequence that pushes then pops a dummy stack symbol

Proof of (2)

- Next, for each pair of states p, q in P, we create a CFG variable A_{pq}
- Our target is to make A_{pq} generate exactly those strings that can bring P from p with an empty stack to q with an empty stack

- How to do so?

Creating A_{pq}

- Note that PDA can get from p (with an empty stack) to q (with an empty stack) in two ways:

(1) The stack gets empty before reaching q

- This implies we get from p to some r (with empty stack) and then to q

(2) The stack never gets empty before reaching q

- This implies at p, we push some char t in stack, and then at q, we pop the same char t

Creating A_{pq} (cont.)

- For each p, q, r, add the rule

$$A_{pq} \to A_{pr}A_{rq}$$

That is, if we can get from p to r, and also from r to q, then we can get from p to q (Here, all starts and ends are with empty stack)

- For each p, q, r, s with $(r, t) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, t)$, add the rule

$$A_{pq} \to aA_{rs}b$$

That is, if we can get from p to r by reading a and pushing t, and we can get from s to q by reading b and popping t, then, if we start with p with an empty stack, we can reach q with an empty stack by reading a, going from r to s, then reading b.

Creating A_{pq} (cont.)

- For each p, we also add the rule

$$A_{pp} \rightarrow \varepsilon$$

That is, if we can get from p to p by reading nothing

If A_{pq} really generates exactly those strings that brings P from p to q (with empty stacks), then what is $A_{q_{start}, q_{accept}}$?

where q_{start} denotes the start state of PDA

Proof of (2)

- So, in our grammar, we will have all the rules A_{pq} and set $A_{q_{start}, q_{accept}}$ as the start variable
- What remains is to prove A_{pq} generates exactly those strings that brings P from p to q (with empty stacks)
- That is, we need to prove
  - If A_{pq} generates x, x brings P from p to q (with empty stacks)
  - If x brings P from p to q (with empty stacks, A_{pq} generates x

Exercise: Prove the above two statements (Hint: by induction)

Next Time

- Pumping Lemma for CFL
- Non-CFL