Deterministic context-free languages + Review of CFLs

Agenda

- Deterministic context-free languages
- Review of CFLs
- Solutions to difficult problems in HW5
- Nature of problems in Mid Term 2

Deterministic vs non-deterministic automata

- Finite automata
  - Deterministic finite automata are equivalent to non-deterministic finite automata in their language recognition power
  - They recognize the same class: that of regular languages
  - We have an algorithm for converting NFAs to DFAs
- Pushdown automata
  - Deterministic PDA are **not** equivalent to non-deterministic PDA
  - Non-deterministic PDA are more powerful than deterministic PDA
  - Certain context-free languages cannot be recognized by deterministic PDA
    - They require non-deterministic PDA

Deterministic context-free languages

- Languages recognized by deterministic PDA are called *deterministic context-free languages* (DCFL)
- DCFLs are relevant to practical applications
  - E.g. design of parsers in compilers for programming languages

Defining DPDAs

- Conform to the basic principle of determinism:
  - At each step of computation, the DPDA has at most one way to proceed according to its transition function
- More complicated than defining DFAs because DPDAs may read an input symbol without popping a stack symbol, and vice versa
- Accordingly, we allow ε-moves in the DPDA's transition function, even though ε-moves are prohibited in DFAs

ε-move types

- Take two forms
  - *ε-input moves*, corresponding to δ(q, ε, x)
  - *ε-stack moves*, corresponding to δ(q, a, ε)
- A move may combine both forms, corresponding to δ(q, ε, ε)

- If a DPDA can make an ε-move in a certain situation, it is prohibited from making a move in that same situation that involves processing a symbol instead of ε (for otherwise non-determinism will occur)

Formal definition of DPDA

A DPDA is a 6-tuple $(Q, \Sigma, T, \delta, q_0, F)$,
where Q, Σ, T, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. T is the stack alphabet,
4. $\delta: Q \times \Sigma_\varepsilon \times T_\varepsilon \rightarrow (Q \times T_\varepsilon) \cup \{\emptyset\}$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

The transition function δ must satisfy the following condition.
For every $q \in Q$, $a \in \Sigma$, and $x \in T$, exactly one of the values

- $\delta(q, a, x)$
- $\delta(q, a, \varepsilon)$
- $\delta(q, \varepsilon, x)$
- $\delta(q, \varepsilon, \varepsilon)$

is not empty set.

Output of transition function in DPDA

- The transition function may
  - output a single move of the form (r, y) or
  - indicate no action by outputting an empty set
- Example:

Suppose a DPDA M with transition function δ is

  - in state q,
  - has a as its next input symbol, and
  - has symbol x on the top of its stack

If $\delta(q, a, x) = (r, y)$ then M reads a, pops x off the stack, and enters state r, and pushes y on the stack.

Alternatively, if $\delta(q, a, \varepsilon) = \emptyset$ then when M is in state q, it has no move that reads a and pops x.

In that case, the condition on δ requires that one of $\delta(q, \varepsilon, x)$, $\delta(q, a, \varepsilon)$, or $\delta(q, \varepsilon, \varepsilon)$ is nonempty, and then M moves accordingly.

- The condition enforces deterministic behavior
- A DPDA has exactly one legal move in every situation where its stack is nonempty
- If the stack is empty, a DPDA can move only if the transition function specifies a move that pops ε.
  - Otherwise, the DPDA has no legal move and it rejects without reading the rest of the input

Acceptance in DPDA

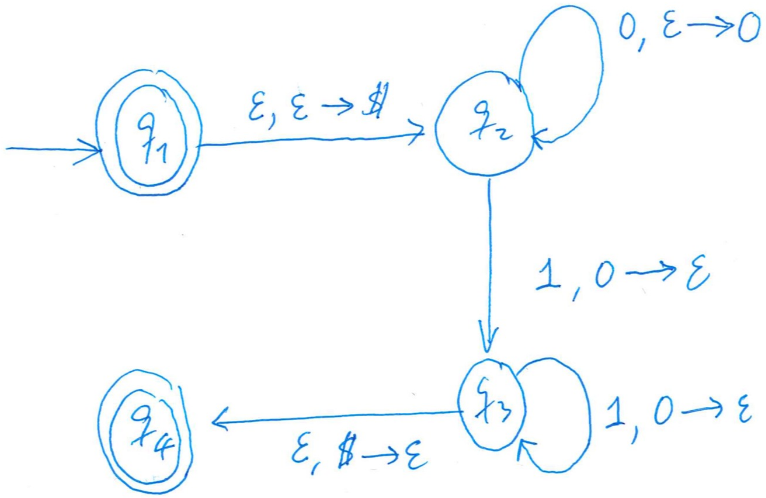
- If a DPDA enters an accept state after it has read the last symbol of an input string, it accepts that string
- In all other cases, it rejects that string

- Rejection occurs
  - A) if the DPDA reads the entire input but doesn't enter an accept state when it is at the end, or
  - B) if the DPDA fails to read the entire input string
- Case B may arise
  - if the DPDA tries to pop an empty stack, or
  - if the DPDA makes an endless sequence of ε-input moves without reading the input past a certain point

Deterministic CFL

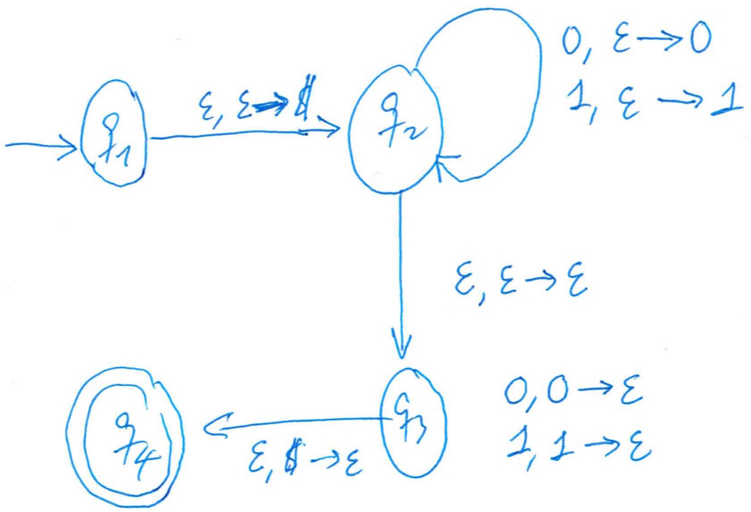
- The language of a DPDA is called a *deterministic context-free language*
- Example:
  - The language $\{0^n1^n \mid n \geq 0\}$ is a DCFL.

  We can easily modify the PDA of this language (shown at the right) to be a DPDA by adding transitions for any missing state, input symbol, and stack symbol to a “dead” state from which acceptance is not possible.



Example of a language that is not DCFL

- The language $\{ww^R \mid w \in \{0,1\}^*\}$ (which is a palindrome) is **not** a DCFL.
- Nondeterminism is necessary for recognizing this language. Challenge: can you prove that?
- The PDA is shown at the right.



Properties of DCFLs

- Theorem: The class of DCFLs is closed under complementation.

- This theorem implies that some CFLs are not DCFLs.
  - Any CFL whose complement isn't a CFL isn't a DCFL.

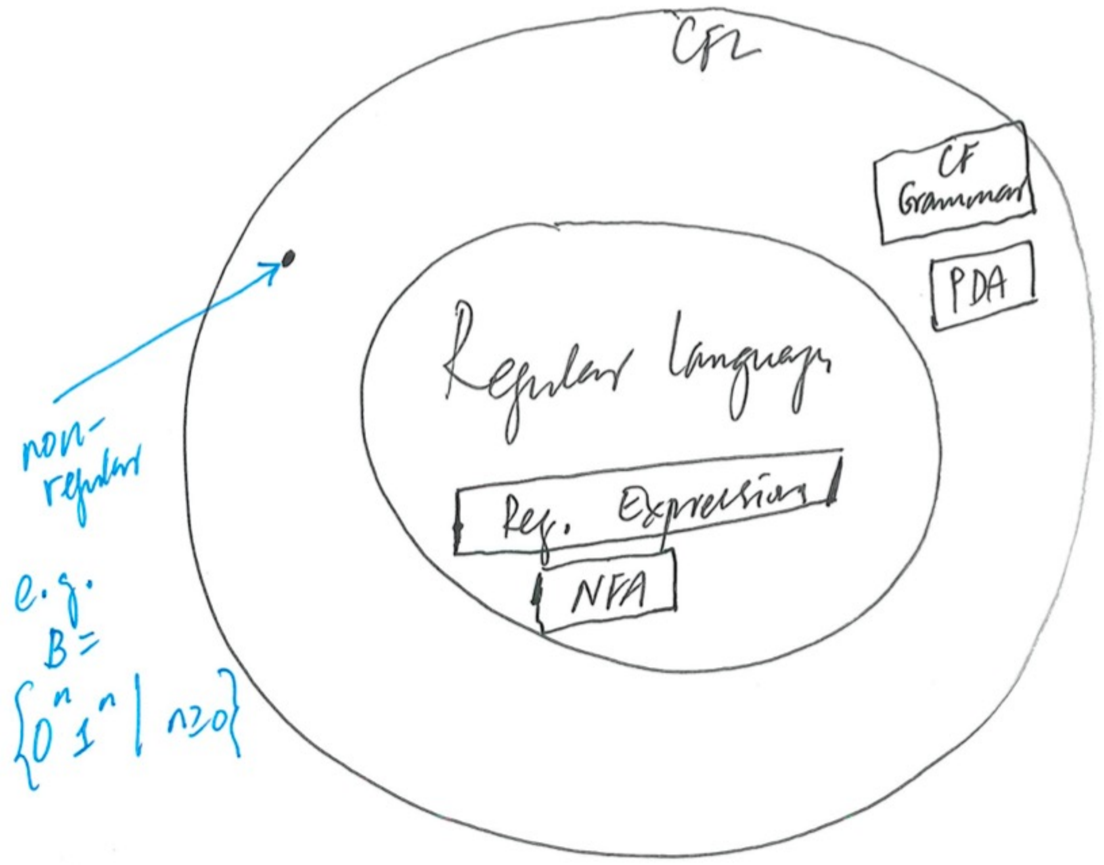
  - Thus $A = \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k \text{ where } i, j, k \geq 0\}$ is a CFL but not a DCFL.
    - Otherwise, A^{comp} would be a CFL, incorrectly implying that

      $A^{comp} \cap a^* b^* c^* = \{a^n b^n c^{n\,|} \mid n \geq 0\}$ is context free.

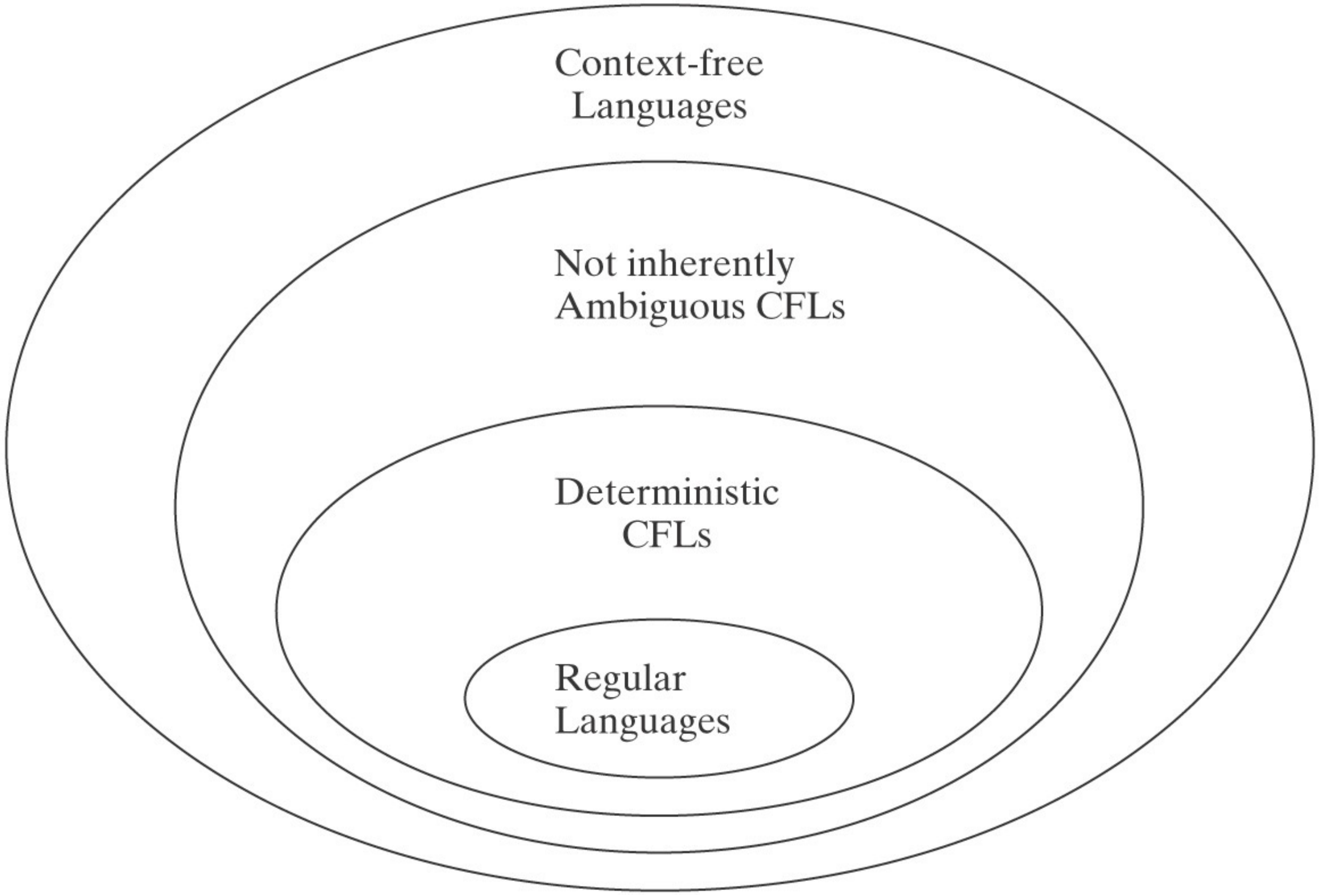
Properties of DCFLs

- The class of DCFLs is ***not*** closed under
  - Union
  - Intersection
  - Star
  - Reversal

Recap


CFL
CF Grammar
PDA
Regular languages
Reg. Expression
NFA
non-regular
e.g. B = {0^n 1^n | n≥0}

Recap


Context-free
Languages
Not inherently
Ambiguous CFLs
Deterministic
CFLs
Regular
Languages

Topics covered (Chapter 2 of Sipser)

- **2. Context-free Languages**
  - 2.1. Context-Free Grammars
    - Formal definition of CFG [lecture on Feb 25]
    - Examples of CFGs [lecture on Feb 25]
    - Designing CFGs [lecture on Feb 28]
    - Ambiguity [lecture on Feb 28]
    - Chomsky Normal Form [lecture on Mar 2, Mar 4]
  - 2.2. Pushdown Automata
    - Formal definition of PDA [lecture on Mar 4]
    - Examples of PDA [lecture on Mar 7]
    - CFG to PDA conversion [lecture on Mar 9]
    - PDA to CFG conversion [lecture on Mar 11]
  - 2.4. Deterministic Context-Free Languages
    - Properties of DCFLs [lecture on March 23]

DFA Equivalence and Minimization
(The Table Filling Algorithm)
- [Lecture on March 21]

Some solutions from HW5: CFG→CNF

- Problem 4a provides a relatively simple grammar with a difficult solution:
  - $S \rightarrow R1R1R1R$
  - $R \rightarrow 0R \mid 1R \mid \varepsilon$
- Replacing start variable unnecessary, since we don't have recursion to S.
- Second step is eliminating ε. This is where things become complicated.
- $S \rightarrow R1R1R1R$ contains 4 R's. Each can be ε or not, leading to 2^4 or 16 possible combinations.
- Consider all binary numbers 0000 to 1111. Replace 0's with empty string, 1's with R's, and put 1's in between all of them.
  - Example: binary number 1010 looks like R11R1

CFG→CNF solution continued.

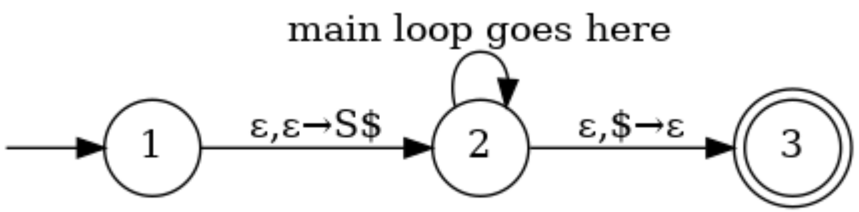
- Resulting grammar from previous step looks as follows:
  - $S \rightarrow 111 \mid 111R \mid 11R1 \mid 11R1R \mid 1R11 \mid 1R11R \mid 1R1R1 \mid 1R1R1R \mid R111 \mid R111R \mid R11R1 \mid R11R1R \mid R1R11 \mid R1R11R \mid R1R1R1 \mid R1R1R1R$
  - $R \rightarrow 0R \mid 1R \mid 0 \mid 1$
- This grammar doesn't have unit rules (i.e. $A \rightarrow B$), so what remains is adding new rules to make everything conform correctly.
- Best to start with rules for terminals:
  - $Z \rightarrow 1$ (**Z**ero)
  - $O \rightarrow 0$ (**O**ne)
- Everything else is to replace common patterns we see. For instance:
  - $A \rightarrow RO$ (R1 originally)
  - $B \rightarrow OR$ (1R originally)
  - $C \rightarrow OO$ (11 originally)

CFG→CNF solution continued.

- Final grammar for 4a is shown as follows:
  - $S \rightarrow GD \mid CO \mid CB \mid AC \mid CD \mid BC \mid EB \mid EA \mid ED \mid AC \mid FB \mid FA \mid FD \mid AF \mid GB \mid GA$
  - $R \rightarrow ZR \mid OR \mid 0 \mid 1$
  - $O \rightarrow 1$
  - $Z \rightarrow 0$
  - $A \rightarrow RO$
  - $B \rightarrow OR$
  - $C \rightarrow OO$
  - $D \rightarrow AR$
  - $E \rightarrow BO$
  - $F \rightarrow AO$
  - $G \rightarrow AA$

More solutions from HW5: CFG→PDA

- Problems 5 and 6 can be tedious or straightforward depending on whether shorthand syntax is used.
- For our example, we will consider problem 6 in shorthand, then problem 5 with a more complete PDA.
- To start with shorthand, make three states: 1, 2, and 3.
  - Transition from 1 → 2 always has the label ε, ε → S\$. We start with empty stack, and push stack symbol \$, then start variable S.
  - Transition from 2 → 3 always has the label ε, \$ → ε. We detect the stack now only has the stack symbol left, pop it, and accept.

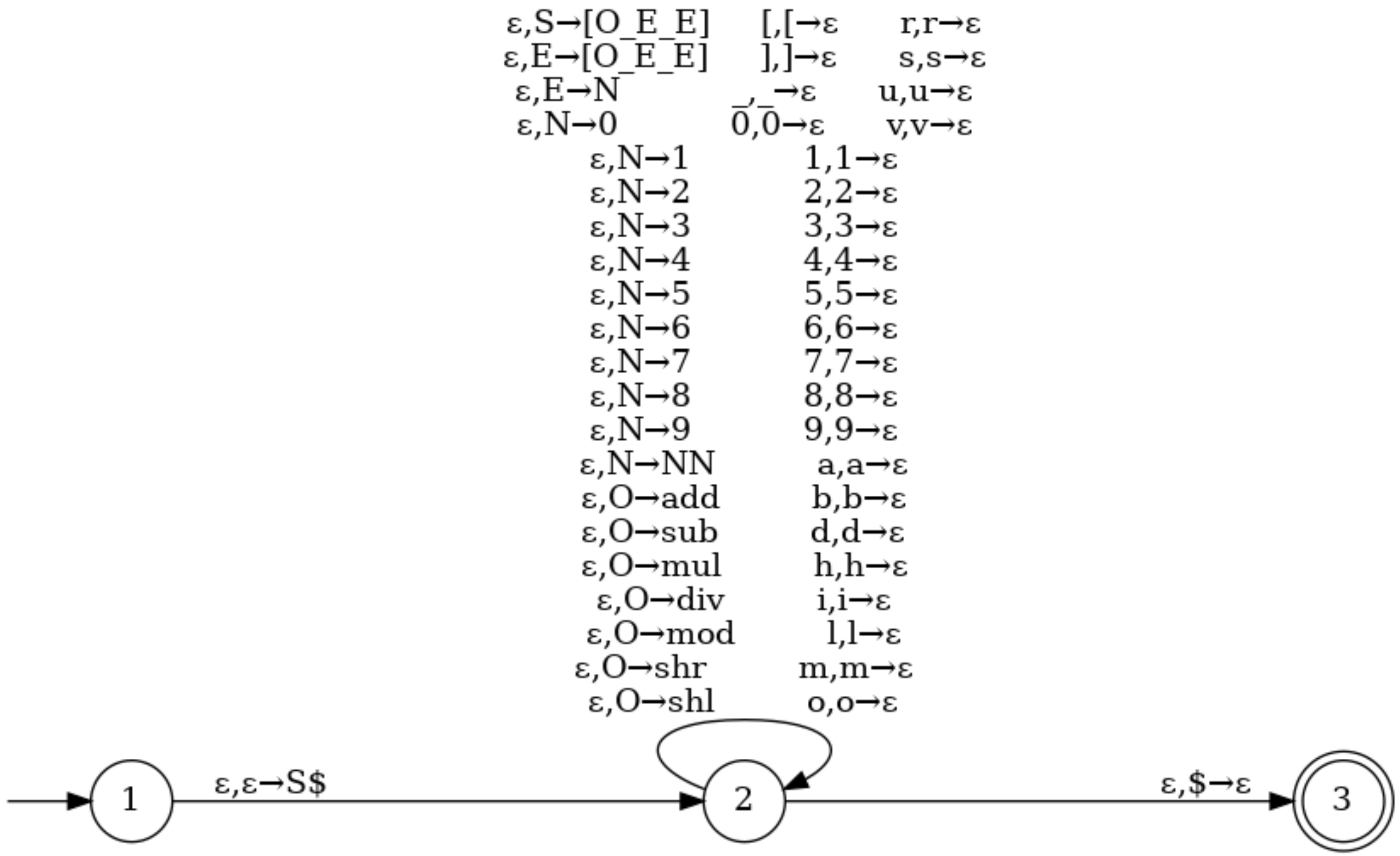


Problem 6 shorthand continued

- For the main loop, we add in two types of rules:
  - For every rule $A \rightarrow B$ in the grammar, the rule ε, $A \rightarrow B$
  - For every terminal a in the input alphabet, the rule a, $a \rightarrow \varepsilon$
- What is our alphabet?
  - We have [,], and _ from the [O_E_E] construction
  - We have the digits from variable N
  - Finally, from the rule $O \rightarrow$ add | sub | mul | div | mod | shr | shl, we have the following letters: a, b, d, h, i, l, m, o, r, s, u, v
- Now just all these rules to the one loop from $2 \rightarrow 2$

Problem 6 shorthand continued

- Final PDA for problem 6 might look as follows:

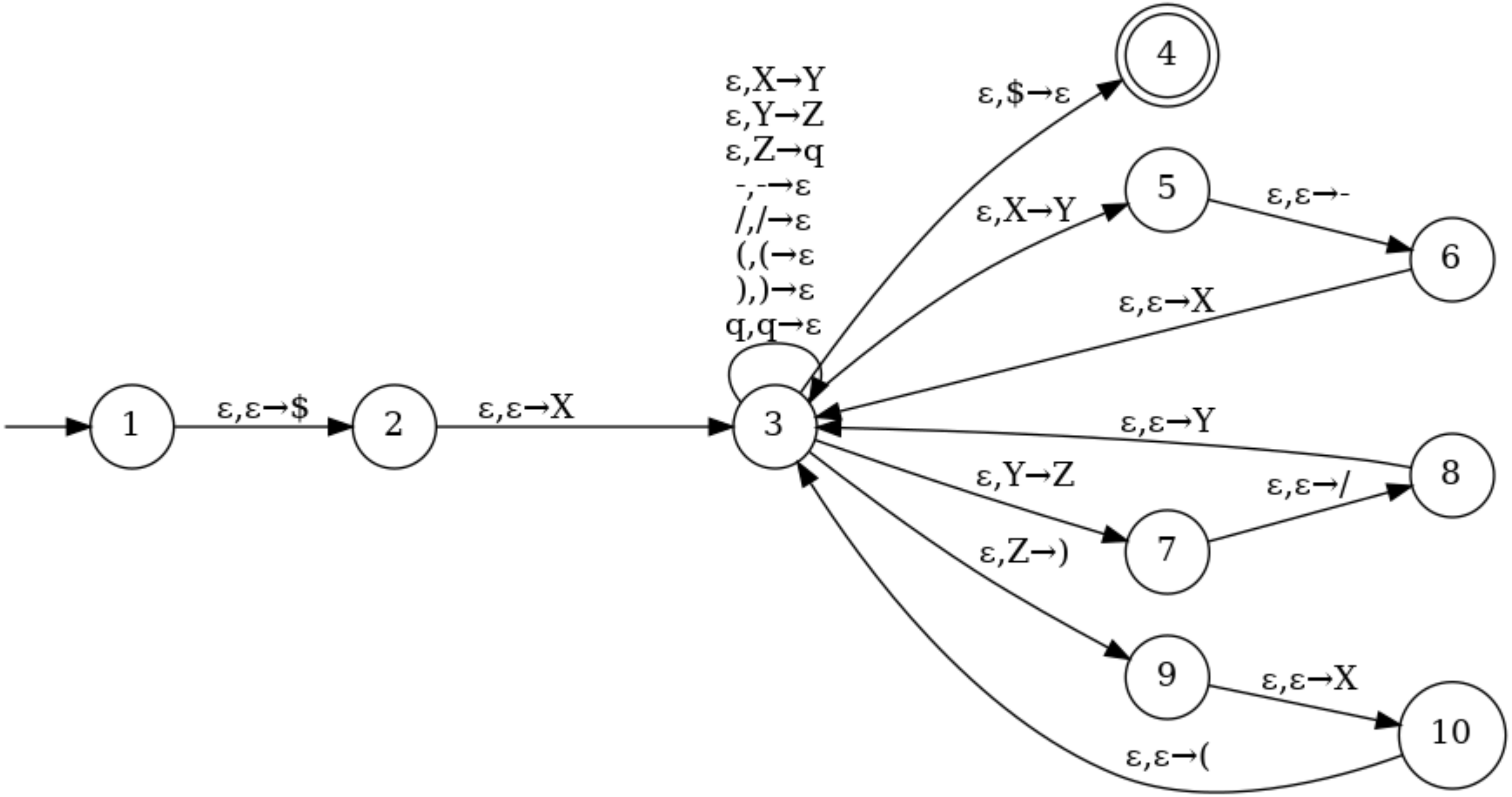


CFG→PDA non-shorthand solution

- What if we don't want to/can't use the shorthand?
- For the next example, we consider problem 5, which has the following grammar:
  - $X \rightarrow X - Y \mid Y$
  - $Y \rightarrow Y / Z \mid Z$
  - $Z \rightarrow (X) \mid q$
- Setup is similar to shorthand, but with 4 states, as \$ and S are pushed separately.
- Same rules for terminals apply, since we read and pop in one transition.
- For rules in CFG, $A \rightarrow B$ may take multiple transitions, however.
  - If rule is $A \rightarrow x_1\ x_2\ ..\ x_n$, then add n-1 states.
  - Transition from state 3 (loop state) to new state by replacing A with x_n, then to each new state doing nothing but pushing x_{n-k}.
  - From last state in the loop, transition back to state 3 pushing x_1.

CFG→PDA non-shorthand solution

- Final PDA for Problem 5 looks as follows:



Mid-term 2: Fri Mar 25

- Will have 5 problems
  - 1 of them has two parts (a and b)
  - 3 of them have just one part
  - 1 of them has five True/False questions (a – e)
- Each of the 5 problems has a weight of 20
- The problems/prompt will fit on one page
- First page will have instructions
- The exam is in-class, in-person
- The exam is closed-book, closed-notes