CS143 Midterm
Spring 2020

- Please read all instructions (including these) carefully.
- There are 5 questions on the exam, some with multiple parts. You have 95 minutes to both finish the exam and upload it. After 95 minutes, gradescope will close your exam and automatically submit it, so make sure submitt what you have by then.
- The exam is open note. You may use laptops, phones, e-readers, and the internet, but you may not consult anyone.
- You must upload the answer to each question as an image file or a PDF. You may submit images of hand-written answers taken with your phone (but allow yourself time to send the image to your computer, so you can upload it to gradescope). It is your responsibility, however, to ensure they are legible. Computer typed answers as a PDF are also permitted.
- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.

Problem	Max points	Points
1	20	
2	15	
3	15	
4	25	
5	25	
TOTAL	100	

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

Type your name as a signature below:

1. **Regular Expressions, NFAs and CFGs**

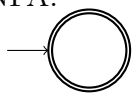
   This question asks you to fill in missing

   - English,
   - Regular Expression,
   - NFA, and
   - CFG

   descriptions of three languages. For each language, we have given you one of the four descriptions and it is your job to fill in the rest.

   For example:

   - English: The empty string
   - Regular Expression: ϵ
   - NFA:

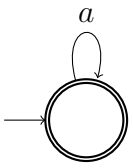
     

   - CFG: $S \to \epsilon$

   (a) **CFG:** $S \to \epsilon \mid aS$

   Solution

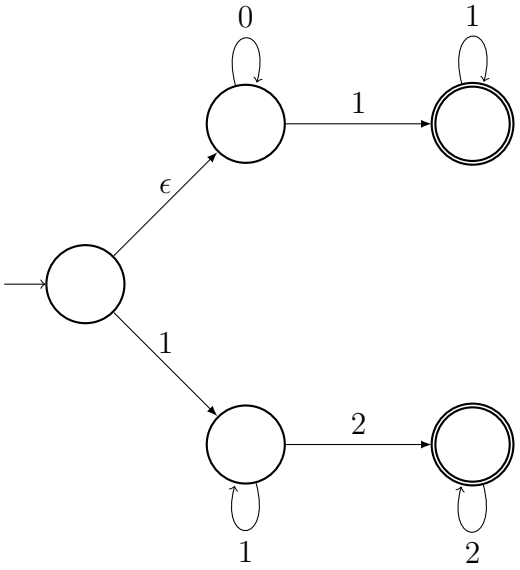
   - **English** A string consisting of only as of any length
   - **Regular Expression** a^*
   - **NFA**

(b) **Regular Expression:** $0^*1^+ \mid 1^+2^+$

Solution

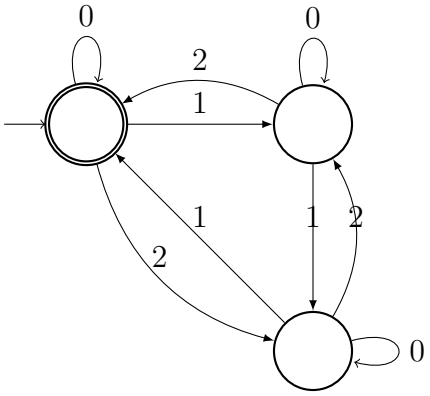
- **English** Either a string with any amount of 0's then at least one 1, or a string with at least one 1 and then at least one 2.
- **NFA**



- **CFG**

$$
\begin{aligned}
S &\rightarrow A \mid B \\
A &\rightarrow X \mid 1Y \\
B &\rightarrow 1Y \mid 2Z \\
X &\rightarrow 0X \mid \epsilon \\
Y &\rightarrow 1Y \mid \epsilon \\
Z &\rightarrow 2Z \mid \epsilon
\end{aligned}
$$

(c) **NFA** (you only need to give the English and CFG descriptions of this NFA, and *not* the regular expression):



Solution

- **English** A number where the digits add to a multiple of three. Empty string is fine.
- **CFG**

$$S \rightarrow \ \epsilon \mid S0S \mid S1S1S1S \mid S1S2S \mid S2S1S \mid S2S2S2S$$

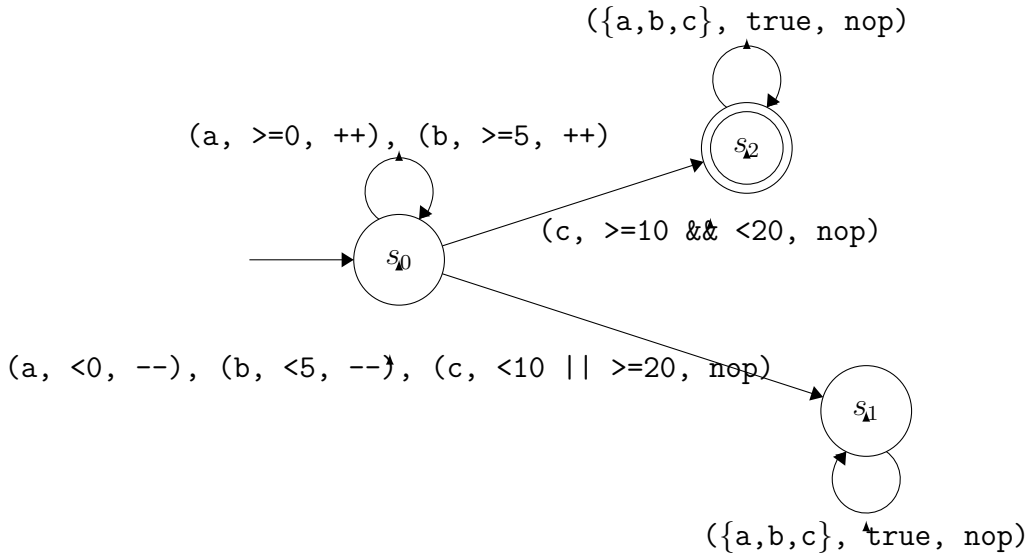
or alternatively:

$$\begin{aligned} S &\rightarrow \ 0S \mid 1A \mid 2B \mid \epsilon \\ A &\rightarrow \quad 0A \mid 2S \mid 1B \\ B &\rightarrow \quad 0B \mid 1S \mid 2A \end{aligned}$$

2. **Counting DFA**

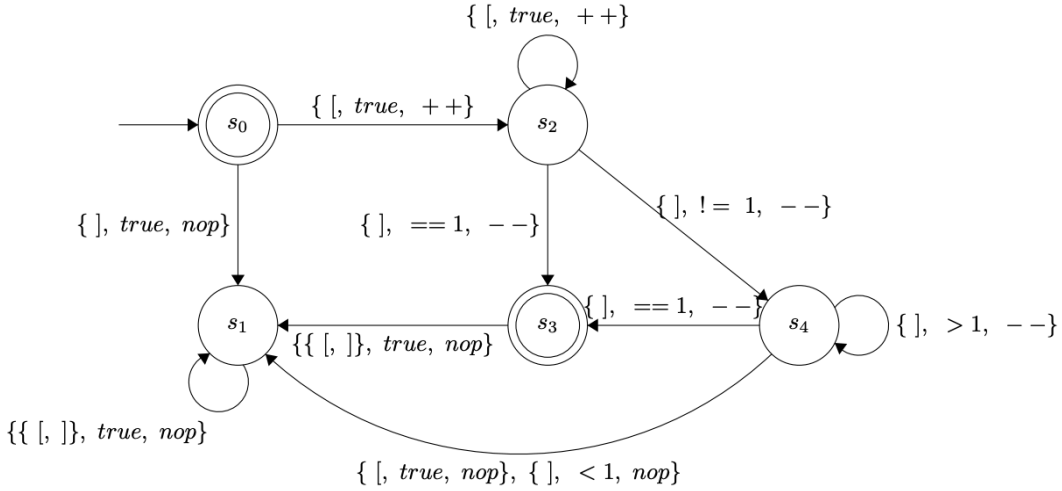
In this problem, we will augment the DFA model with a global counter. A Counting DFA contains a single global integer counter that can be updated when state transitions occur. In addition, every transition specifies a predicate on the counter that must be true for the transition to happen.

Specifically, every transition is now a three-tuple `(char, boolean expression, counter op)`. A transition is taken if the next character read from the input matches `char` and the boolean expression is satisfied with respect to the counter. Counter operations are applied after the boolean expression is evaluated and can be `++` (increment), `--` (decrement), or `nop` (no change). Each character needs a transition from each state for all possible counter values, but not all possible counter ops. The counter is set to 0 upon entering the DFA. Here is an example over the alphabet {a,b,c}, showing how transitions are specified:



Draw a Counting DFA that accepts a string of balanced square brackets: $\{[^i]^i \mid i \geq 0\}$.

Solution


{ [, true, ++}
s_0
{ [, true, ++}
s_2
{], true, nop}
{], == 1, −−}
{], != 1, −−}
s_1
s_3
{], == 1, −−}
s_4
{], > 1, −−}
{{ [,]}, true, nop}
{{ [,]}, true, nop}
{ [, true, nop}, {], < 1, nop}

3. **Syntax-Directed Translation**

Suppose you are a taxi driver in a city laid out as a grid. You can drive north (n), south (s), east (e), and west (w). The following grammar describes the routes you take that bring you back to where you started (these are not all possible routes):

$$
\begin{aligned}
S &\rightarrow PS \\
&\mid \epsilon \\
P &\rightarrow nPs \\
&\mid sPn \\
&\mid ePw \\
&\mid wPe \\
&\mid \epsilon
\end{aligned}
$$

Add semantic actions that, given a route, computes the maximal distance west of your starting point that your route took you. For example, `wwnsee` takes you two distance units west.

Solution

```
S -> PS {
  $$.west = max($1.west, $2.west)
} | epsilon {
  $$.west = 0
}
P -> nPs {
  $$.west = $2.west
}
| sPn {
  $$.west = $2.west
}
| ePw {
  $$.west = max(0, $2.west - 1)
}
| wPe {
  $$.west = $2.west + 1
}
| epsilon {
  $$.west = 0
}
```

4. **Top-Down Parsing**

Produce an LL(1) parsing table for the language described by the following CFG with terminals $\{a, b, (,)\}$.

$$\begin{aligned} S &\rightarrow ST \mid \epsilon \\ T &\rightarrow (a) \mid (b) \end{aligned}$$

Hint: you may have to factor the grammars.

Solution

Left-factored grammar where left recursion is also removed:

$$\begin{aligned} S &\rightarrow TS \mid \epsilon \\ T &\rightarrow (T' \\ T' &\rightarrow a) \mid b) \end{aligned}$$

Table (the only required part of the answer):

	(	**a**	**b**	**)**	**$**
S	TS				ϵ
T	(T'				
T′		a)	b)		

5. **Bottom-Up Parsing**

Below is the skeleton of an LR parsing automaton, showing (numbered) states and transitions arrows, but not items or transition labels.

(a) Provide the missing right-hand sides of the following grammar of the state diagram. There should be three terminals.

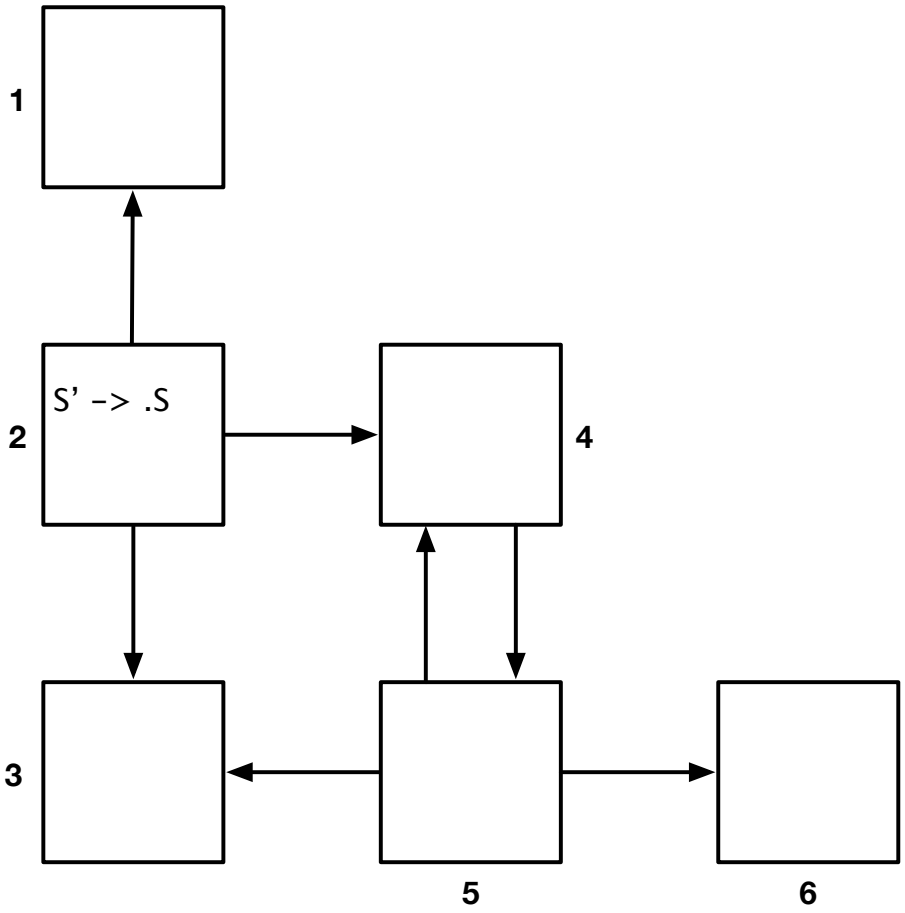
$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow abS \\ S &\rightarrow c \end{aligned}$$

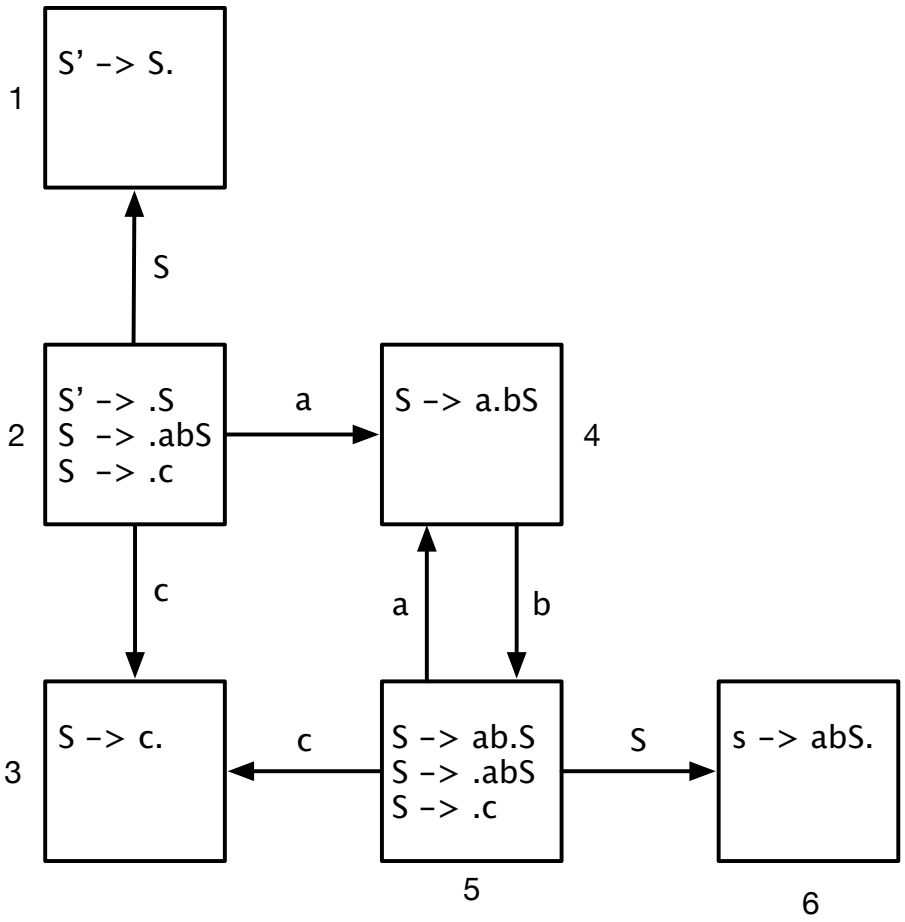
Solution

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow \\ S &\rightarrow \end{aligned}$$

(b) For each state, provide the LR(0) items from your grammar, and for each label, provide the correct transition symbols. There should be no shift-reduce or reduce-reduce conflicts in your automaton (under LR(0) parsing rules). You may not add any additional states or transitions to the diagram.

In your answer, you may use the numbers to identify states (e.g., 1: *items*) and transitions (e.g., $2 \rightarrow 1$: *transition*).




1
S' -> S.
S
2
S' -> .S
S -> .abS
S -> .c
a
S -> a.bS
4
c
a
b
3
S -> c.
c
S -> ab.S
S -> .abS
S -> .c
S
s -> abS.
5
6