6.035 Fall 2019

Exam I

You have 50 minutes to finish this quiz.

Write your name and kerberos on this cover sheet.

Some questions may be harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

This exam is open book and open laptop. Additionally, you may access the course website, but aside from that you may NOT USE THE NETWORK.

Please do not write in the boxes below.

Q1 (/ 10)	Q2 (/ 16)	Q3 (/ 10)	Q4 (/ 20)	Q5 (/ 24)	Total (/ 80)

Name:

Kerberos:

Question 1: Regular Expressions and DFAs (10 points)

Let L be the language of binary strings that **do not** contain three consecutive 1's. For example, 11000001, ε, and 00 are in L, but 00110**111**0 is not.

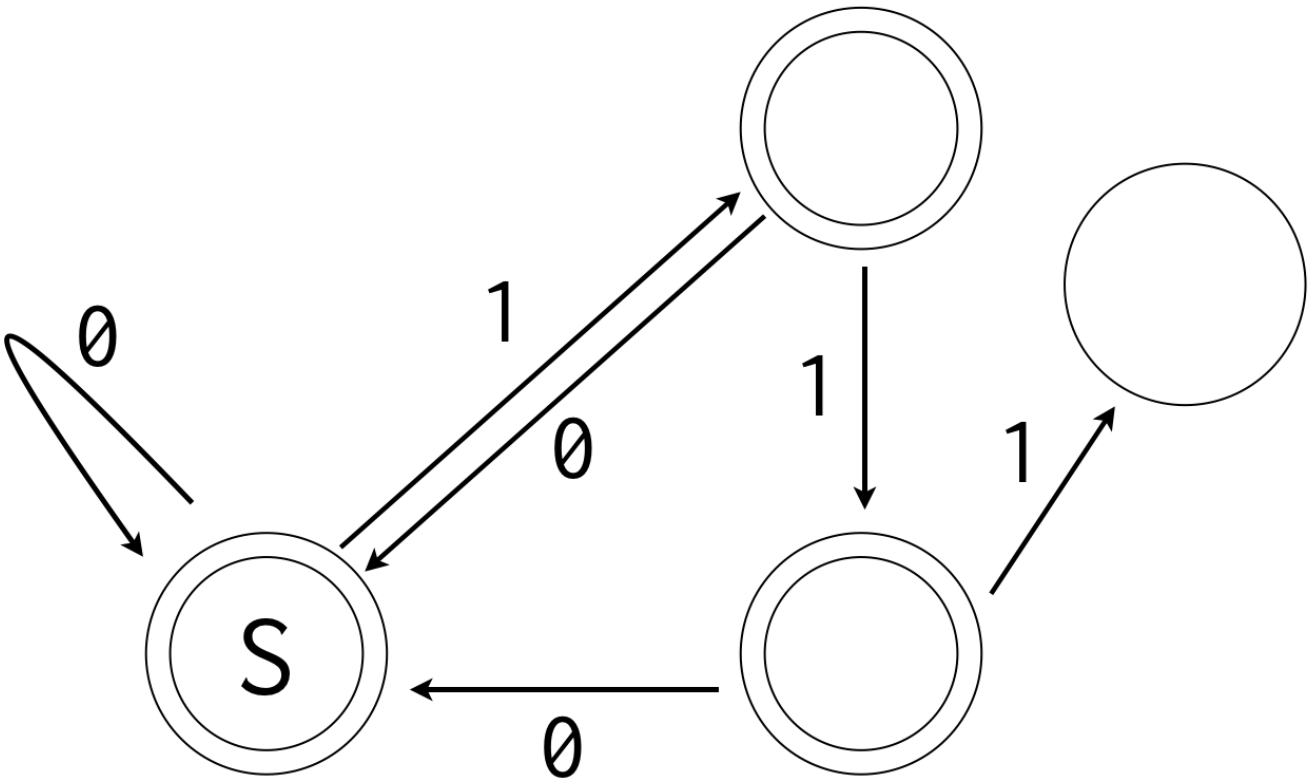
(i) (5 points) Write a regular expression that recognizes language L. (Use the regular expression syntax used in lecture: alphabet symbols, |, *, ε, and parentheses.)

```
-1 if * instead of + or similarly minor mistakes
-2 if the repeating component is correct but non-repeating is not
```

```
(1|ε)(1|ε)(0|01|011)*
```

(ii) (5 points) Draw a state diagram of a deterministic finite-state automaton (DFA) that recognizes language L. Remember to indicate starting and accepting states.

```
-3 if L is not accepted
-2 if it is not deterministic
-1 if minor detail wrong
```



Question 2: Parse Trees (16 points)

This question uses the following grammar (where ∗ is a multiplication symbol, not a regex repetition operator):

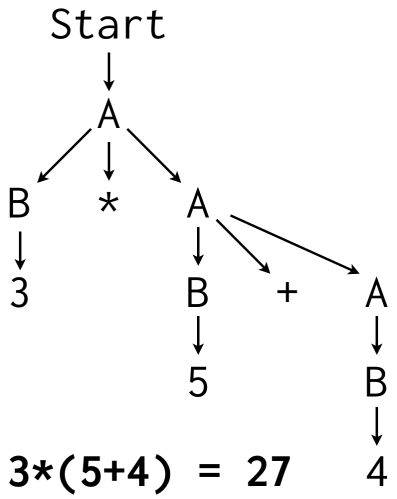
Start → A
A → B ∗ A
A → B + A
A → B
B → [0-9]+

For each of the following expressions,

- Draw a parse tree using the grammar above, or state why it cannot be done. (If there are multiple valid parses, you need only give one.) (6 points each)
- Give the value of the expression when evaluated using the groupings determined by your parse tree. (Assume numbers and operators do the thing you expect.) (2 points each)

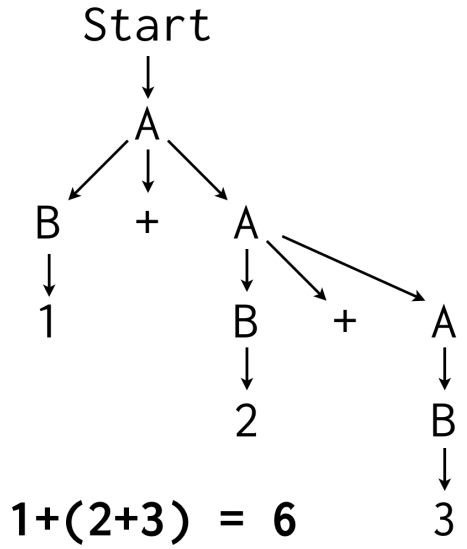
`-0 if not a fully CST`

(i) 3 ∗ 5 + 4



3*(5+4) = 27

(ii) 1 + 2 + 3



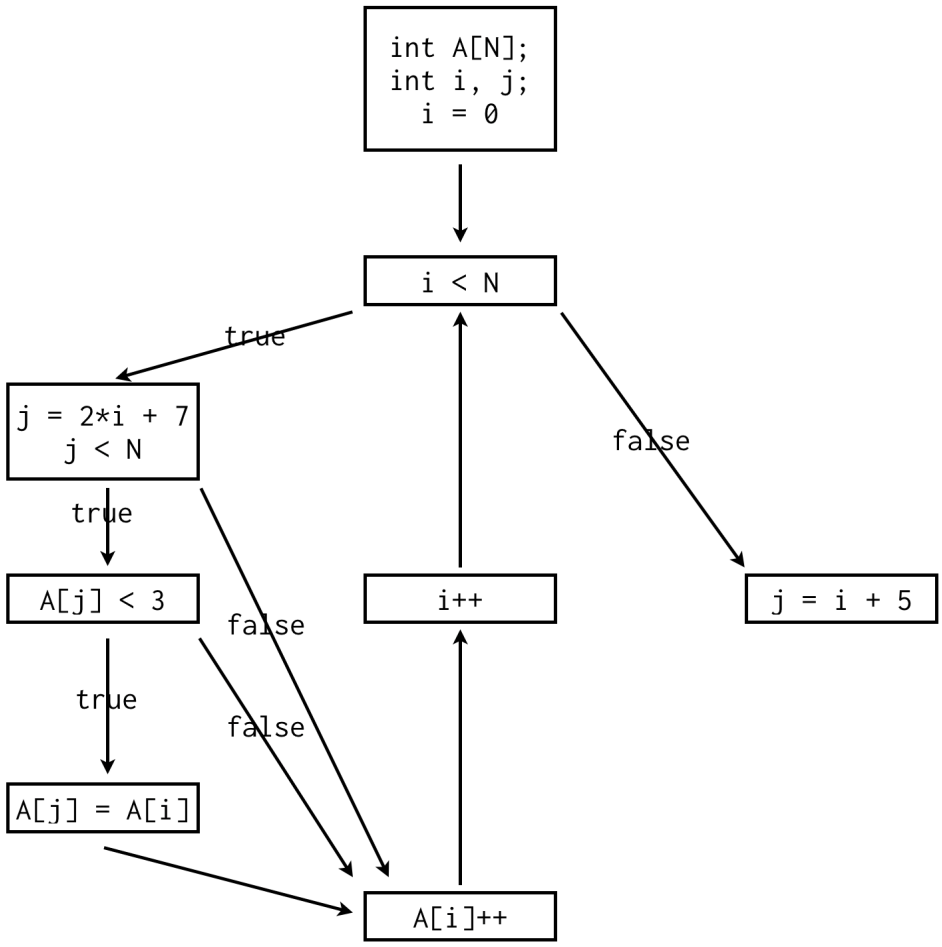
1+(2+3) = 6

Question 3: Control Flow Graphs (10 points)

Draw the control flow graph for the following section of code, using short circuit evaluation where applicable. (For ease of grading, please label your branches with true and false.)

```
int A[N];
int i, j;
for (i=0; i<N; i++) {
  j = 2*i + 7;
  if (j<N && A[j]<3) {
    A[j] = A[i];
  }
  A[i]++;
}
j = i + 5;
```

-3 if does not short circuit



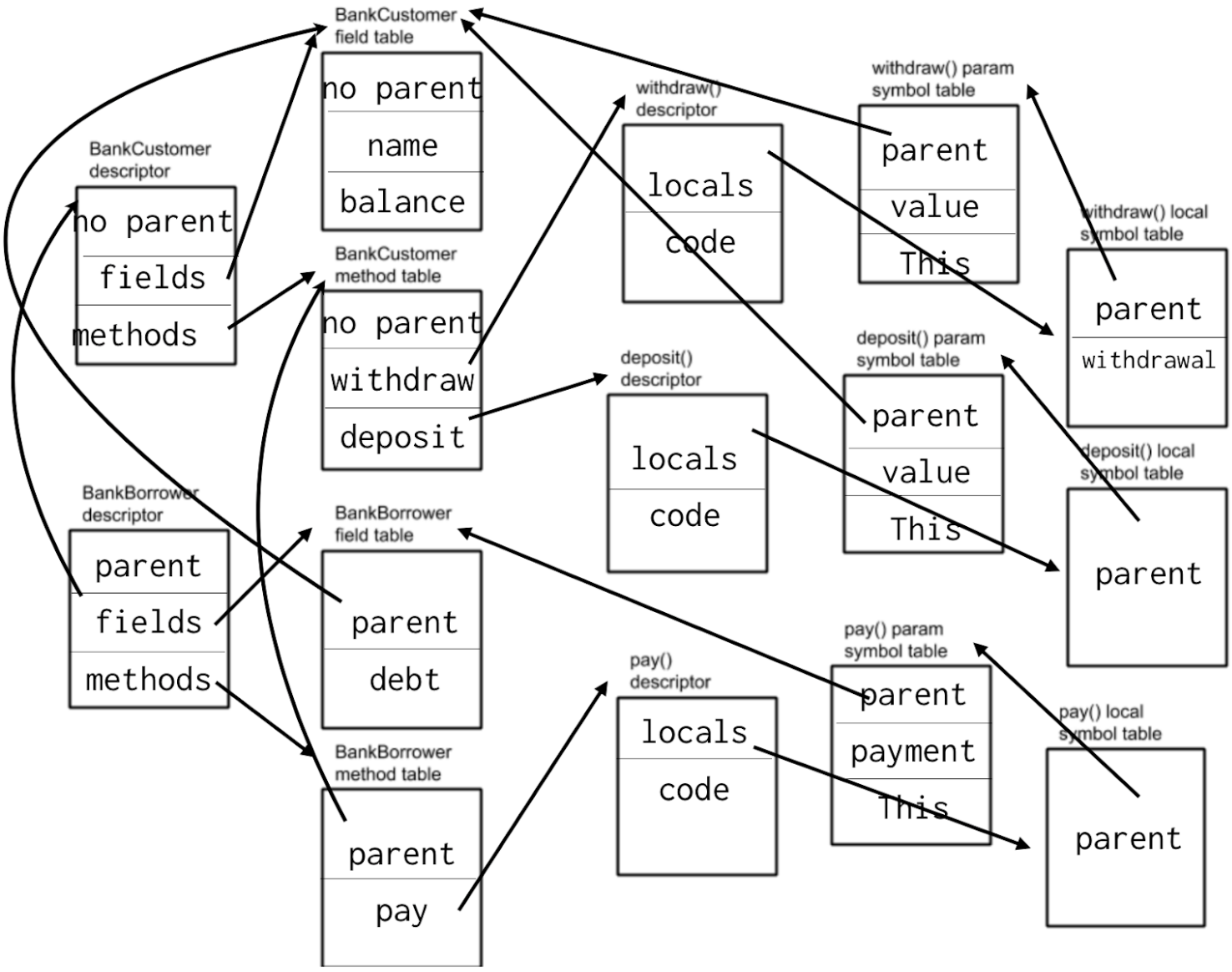
Question 4: Descriptors and Symbol Tables (20 points)

On the next page, fill in the descriptors and symbol tables for the classes, methods, fields, parameters, and locals in the code shown below. Use arrows to represent when one descriptor/table points to another. To reduce clutter, omit type descriptors and variable descriptors; you need only fill in the given boxes.

Note: the ? and : below represent C's ternary operator, but you shouldn't need to be familiar with it for the question.

```
class BankCustomer {
    string name;
    int balance;

    void deposit (int value) {
        this.balance += value;
    }

    // withdraw no more than balance, and return the amount withdrawn
    int withdraw (int value) {
        int withdrawal = (value <= this.balance) ? value : this.balance;
        this.balance -= withdrawal;
        return withdrawal;
    }
}

// borrowers inherit from customers, and also have some debt
class BankBorrower : BankCustomer {
    int debt;

    void pay (int payment) {
        this.balance -= payment;
        this.debt -= payment;
    }
}
```

-1 for every missing table entry
-1 for every missing arrow
-0 if we've already penalized for same mistake in different place
-0 for pointing to unnecessary things
-0 for not showing parent slot when it's empty
(If table is only partially filled in, points prior to penalties are proportional to the amount filled in. Penalties can't take score below zero. So e.g. a blank submission is worth 0.)



Question 5: Shift Reduce Parsing (24 points)

For all parts of this question, use the following grammar (rules numbered for reference):

(0) Start → Y$
(1) Y → Y+X
(2) Y → X
(3) X → a
(4) X → X * a

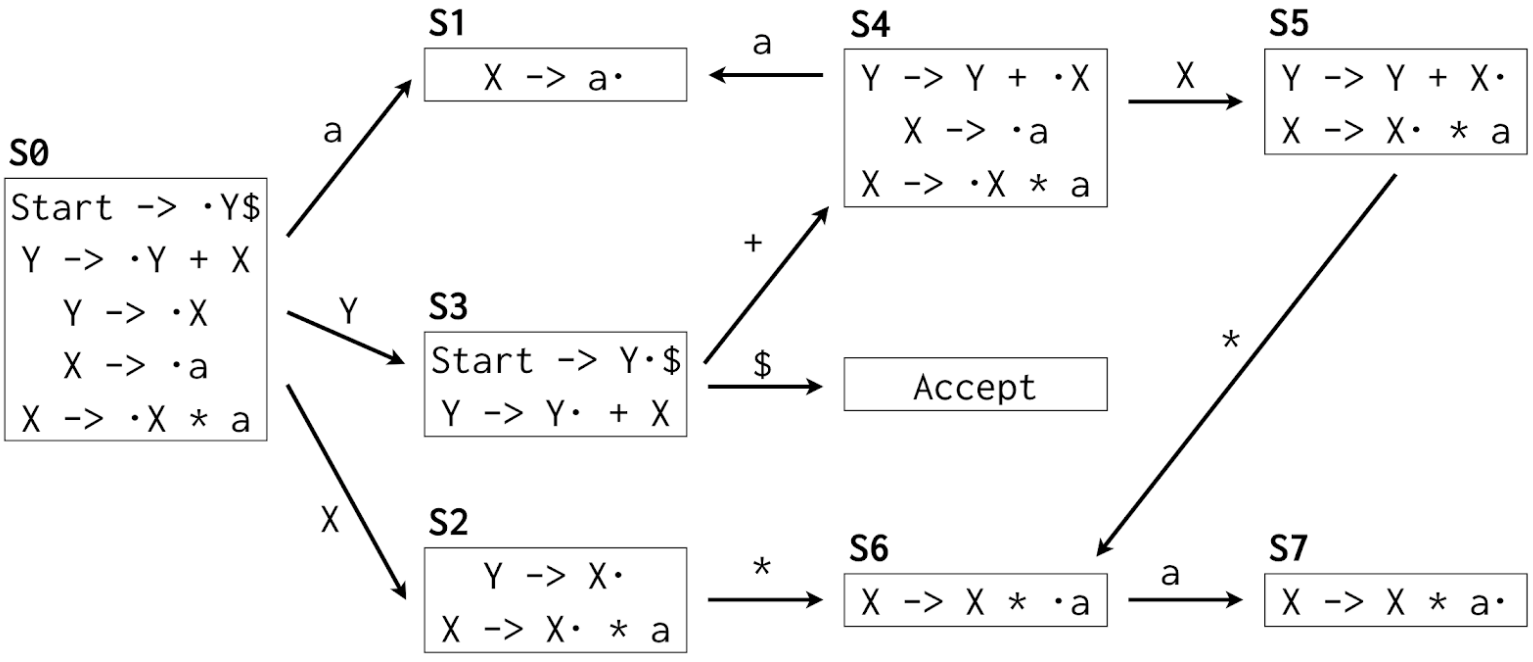
a, +, *, and $ are terminal symbols; X and Y are nonterminals.

(i) (3 points) Write down 3 distinct strings that this grammar recognizes.

```
-1 for every wrong string
-0 for missing dollar
"a$", "a+a$", "a*a*a$"
```

(ii) (9 points) Fill in the DFA below with items such that each transition represents the Goto of a state with the labeled symbol. We have given you one, but not necessarily all, of the items in S0 to start you off. (The state labels S0, … , S7 will be useful in part (iii).

```
-1 for every wrong state
```



(iii) (12 points) Fill in the shift-reduce table, based on the DFA. We have already provided rows S2 and S4 completely. If there's no goto action, leave it blank. Entries should be one of {shift <state>, reduce (<rule #>), goto <state>, error, accept, <blank>}.

2 points per row; this means if row is filled in, -1/3 for every wrong cell, if not filled in then -2

(Round the sum of the -1/3 penalties towards zero)

state	a	*	+	$	X	Y
S0	SHIFT S1	ERROR	ERROR	ERROR	GOTO S2	GOTO S3
S1	REDUCE (3)	REDUCE (3)	REDUCE (3)	REDUCE (3)		
S2	REDUCE (2)	SHIFT S6	REDUCE (2)	REDUCE (2)		
S3	ERROR	ERROR	SHIFT S4	ACCEPT		
S4	SHIFT S1	ERROR	ERROR	ERROR	GOTO S5	
S5	REDUCE (1)	SHIFT S6	REDUCE (1)	REDUCE (1)		
S6	SHIFT S7	ERROR	ERROR	ERROR		
S7	REDUCE (4)	REDUCE (4)	REDUCE (4)	REDUCE (4)		