Each of the problems is worth 10 points. Please write your solutions clearly and concisely. If you do not explain your answer you will be given no credit. You are encouraged to collaborate on the homework, but you *must* write your own solutions and list your collaborators on your solution sheet. Copying someone else's solution will be considered plagiarism and may result in failing the whole course.

Please turn in the solutions by 11.59pm on Thursday 10 November. The homework should be dropped off in the box labeled CSC 3130 on the 9th floor of SHB. Late homeworks will not be accepted.

In the context-free grammar descriptions below, the start variable is always the one on the left-hand side of the first production.

Problem 1

Consider the following context-free grammar G:

$$E \to E + T \mid T$$
$$T \to \mathtt{x} \mid (E)$$

It generates expressions like `x`, `x+(x+x)+(x+x)`, and so on.

(a) Write all items in this grammar and construct an NFA for the valid item updates.

(b) Convert the NFA to a DFA. Which of the states are shift states and which are reduce states? Are there any conflicts?

(c) Using the DFA, show an execution of the LR(0) parsing algorithm on the input

$$\mathtt{x+(x+x)+(x+x)}.$$

Show the state of the stack, input, and DFA throughout the execution.

(d) Now consider the following extended context-free grammar G':

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to \mathtt{x} \mid (E)$$

Show that G' is not an LR(0) grammar.

(e) **(Extra credit)** Give an LR(1) DFA for G'. Show an execution of the LR(1) parsing algorithm on input `(x+x)*(x+x*x)`.

Problem 2

In this problem, you will design Turing machines for the following three languages:

(a) $L_1 = \{a^n \# a^n \# a^n : n \geq 0\}$, $\Sigma = \{a, \#\}$. Give both a high-level description and a state diagram of your Turing Machine.

(b) $L_2 = \{a^n b^{2^n} : n \geq 0\}$., $\Sigma = \{a, b\}$. Give only a high-level description of the Turing Machine. A state diagram is not necessary.

Problem 3

A *queue automaton* is like a push-down automaton except that the stack is replaced by a queue. A queue is a tape allowing symbols to be read only at the left end and written only at the right end. Each read operator (called a *pop*) reads and removes a symbol from the left and of the tape and each write operation (called a *push*) writes a symbol at the right end. For example, if the state of the tape is `abcaaab`, the operation **pop a** yields `bcaaab`. Now **push c** yields `bcaaabc`. Initially, the queue contains the input followed by the blank symbol $\square$. The automaton accepts (rejects) by going into a special state q_{accept} (q_{reject}). The transitions in a queue automaton are deterministic, but the automaton has the option of not popping or not pushing any symbol (that is, **pop** ε and **push** ε are allowed).

You will argue that a queue automaton is equivalent to a Turing Machine: Every queue automaton can be simulated on a Turing Machine, and vice versa.

(a) Write a formal definition of a queue automaton.

(b) Show how to simulate a queue automaton on a Turing Machine. For this, you need to specify

- how the tape of the Turing Machine will be used to represent the queue automaton;
- how the Turing Machine tape should be set up initially;
- what the Turing Machine should do when the automaton performs a push or a pop;
- what the Turing Machine should do when the queue automaton accepts / rejects.

(c) Show how to simulate a Turing Machine on a queue automaton.

Problem 4

The Church-Turing Thesis claims that Turing Machines are a universal model of computation: Any computation that can be performed on any computer we will ever build can also be done on a Turing Machine. Here are some possible objections to the Church-Turing Thesis. For each of these objections, say if you think it is reasonable or not, and explain why. (You won't be graded based on whether your answer is right or wrong, as some of the parts have no right or wrong answer. What matters is how well you explain your answer.)

(a) Suppose I want to know what is the smallest country in the world. In real life, I would use google, type in "smallest country", and I find out the answer after a few clicks. But I cannot do this on a Turing Machine. How do I even connect a Turing Machine to the internet? Since there are computations we can do in real life but not on a Turing Machine, the Church-Turing thesis is false.

(b) Look at this computer program:

```
int F(int n) {
  if (n == 1) return 1;
  else return F(n - 1) + F(n - 2);
}
```

This program uses *recursion*: A procedure makes a subroutine call to itself. But Turing Machines do not support recursion. Therefore Turing Machines are not as powerful as ordinary programming languages, and the Church-Turing thesis is false.

(c) Humans can also be modeled as computers: We take inputs from the environment (by seeing, hearing, touching) and produce outputs (via speaking and gestures). If the Church-Turing thesis is true, then any task that humans can do can also be done on a Turing Machine, and so on any machine. But there are tasks that humans are better at than machines: Learning foreign languages, identifying objects in images, winning basketball games, and so on. Therefore the Church-Turing Thesis cannot be true.