Homework 4

Theory of Computation (CSCI 2210)

due: 2023-11-08

Problem 1
Give a complete description (either the formal components or a state-transition graph) of a *pushdown automaton* that decides the language $\mathrm{L}_{0^n,1^{2n}}$, containing strings consisting of a run of 0s followed by a run of twice as many 1s.

examples: ε, 0 1 1, 0 0 1 1 1 1

counterexamples: 0, 1, 1 1 0, 1 0 0, 0 1 1 0 1 1

Hint: recall the PDA for the language $\mathrm{L}_{0^n,1^n}$ that we defined in lecture and think about what needs to be changed.

Problem 2
Consider the *context-free grammar* G implied by the following set of production rules:

$$\begin{array}{lrcl} r_1: & \text{EXPR} & \text{->} & \text{EXPR} + \text{TERM} \\ r_2: & \text{EXPR} & \text{->} & \text{TERM} \\ r_3: & \text{TERM} & \text{->} & \text{TERM} \times \text{FCTR} \\ r_4: & \text{TERM} & \text{->} & \text{FCTR} \\ r_5: & \text{FCTR} & \text{->} & (\,\text{EXPR}\,) \\ r_6: & \text{FCTR} & \text{->} & x \end{array}$$

For each of the following strings, either give a *parse tree* if it is in the language of G, or if it is not then briefly explain why.

(a) $x + x$

(b) $x \times x + x$

(c) $((x))$

(d) $x + x + x$

Problem 3
Consider the following TM with input alphabet $\Sigma := \{0, 1\}$:

working alphabet: $\Gamma := \Sigma \cup \{\mathrm{X}, \mathrm{Y}\}$

state set: $\{q_0, q_1, q_2, q_3, q_4, q_h\}$

start state: q_0

halt state: q_h

state transition partial function:

δ	0	1	X	Y	$\sqcup$
q_0	$(q_0\,,0\,,\mathrm{R})$	$(q_0\,,1\,,\mathrm{R})$	–	–	$(q_1\,,\sqcup\,,\mathrm{L})$
q_1	$(q_2\,,\mathrm{X}\,,\mathrm{R})$	$(q_3\,,\mathrm{Y}\,,\mathrm{R})$	–	–	$(q_h\,,\sqcup\,,\mathrm{R})$
q_2	$(q_2\,,0\,,\mathrm{R})$	$(q_2\,,1\,,\mathrm{R})$	–	–	$(q_4\,,0\,,\mathrm{L})$
q_3	$(q_3\,,0\,,\mathrm{R})$	$(q_3\,,1\,,\mathrm{R})$	–	–	$(q_4\,,1\,,\mathrm{L})$
q_4	$(q_4\,,0\,,\mathrm{L})$	$(q_4\,,1\,,\mathrm{L})$	$(q_1\,,0\,,\mathrm{L})$	$(q_1\,,1\,,\mathrm{L})$	–
q_h	–	–	–	–	–

(a) Draw the state transition graph for this TM.

(b) Does this machine accept the input 0 1 1? If so, what are the contents of the memory when it halts?

Problem 4

When discussing how we can simulate a two-ended Turing machine with a one-ended Turing machine, we asserted that the following two functions can be implemented as one-ended Turing machines:

memory right-shift: move the contents of the TM memory one cell to the right, duplicating the contents of the left-most cell (if any), and return the memory pointer to its starting position.

memory left-shift: move the contents of the TM memory one cell to the left, deleting the contents of the left-most cell (if any), and return the memory pointer to its starting position.

For example:

input	right-shifted	left-shifted
ε	ε	ε
0	0 0	ε
0 1	0 0 1	1
0 1 0 1	0 0 1 0 1	1 0 1

Implement these two operations as Turing machines in the concrete syntax of the TM simulator at `https://turingmachine.io/` as programs `shift_r.tm` and `shift_l.tm`, respectively. Do this using *only* the transitions that are allowed in a one-ended TM:

- do not *push* (make a transition $(\sqcup \mapsto x\,,d)$) at the left end of the memory, unless it is also the right end of the memory; i.e., the memory is empty,
- do not *pop* (make a transition $(x \mapsto \sqcup\,,d)$) at the left end of the memory, unless it is also the right end of the memory; i.e., the memory is a single cell.

You may assume an input alphabet of $\Sigma := \{0,1\}$.

Problem 5

It is an interesting fact that a *doubly-linked list* can be implemented using two stacks. We'll call the two stacks *left* and *right*. The idea is as follows.

- The top element, if any, of the right stack represents the *current cell* of the doubly-linked list.
- If the right stack is empty, that indicates that we have reached the *right end* of the doubly-linked list.
- If the left stack is empty, that indicates that we have reached the *left end* of the doubly-linked list.
- To *move one cell to the right* in the doubly-linked list we pop an element from the right stack and push it onto the left stack.
- To *move one cell to the left* in the doubly-linked list we pop an element from the left stack and push it onto the right stack.

- To *grow the right end* of the doubly-linked list by one cell we push a new element onto an empty right stack.
- To *grow the left end* of the doubly-linked list by one cell we push a new element onto the right stack with an empty left stack.
- To *shrink the right end* of the doubly-linked list by one cell we pop the only element of the right stack and discard it.
- To *shrink the left end* of the doubly-linked list by one cell we pop the top element of the right stack and discard it with an empty left stack.

An *n-stack pushdown automaton* (n-PDA) is a generalization of the PDA model of computation. In each state transition it can optionally read an input symbol and optionally attempt to pop an element from each of its n stacks to examine. Based on its current state and the result of these operations it decides which state to transition to, and what, if anything, to push onto each of its stacks.

Give a high-level description of how we could simulate a Turing machine using a 2-PDA by answering the following questions:

(a) What features of a 2-PDA can we use to simulate the TM's memory?

(b) How can we load a 2-PDA's input string into the simulator of the TM's memory so that it is in the right order with the memory pointer of the simulated TM at the correct starting position?

(c) How can we simulate a single step of TM computation with a 2-PDA?
Hint: consider each of the possible kinds of transition that a TM can make and think about how to simulate each one.