CS 599 D1: Mock Mid-Term

Total: 80 pts

Ankush Das

1 Linear Inference [20 pts]

In this section, we will solve the 'Vending Machine' problem using rules of linear inference. In front of you is a vending machine that accepts \$1, \$5, and \$10 bills. The vending machine contains chips packets, candy bars, and cookie boxes each costing \$2, \$2, and \$3 respectively. Here's how the vending machine works:

1. You insert *exactly one* bill into the machine.
2. You choose *exactly one* item you want to purchase.
3. The vending machine dispenses the item you wished to purchase.
4. The vending machine returns your change (in case you overpaid) *in as few bills as possible.* For instance, if you insert a \$10 bill and purchase a packet of chips, the vending machine will return one \$5 bill, one \$2 bill, and one \$1 bill (instead of returning eight \$1 bills).

Your task is to define the mechanism of purchasing items from the vending machine as rules of linear inference.

Problem 1 (2 pts) *Define the propositions you will need in the inference rules. Briefly describe the intuitive meaning of each proposition.*

Problem 2 (10 pts) *Define the rules of linear inference for the vending machine problem.*

Problem 3 (8 pts) *Suppose you start with a \$10 bill and you intend to purchase one packet of chips, one candy bar, and one cookie box. Is this possible using the rules above? If yes, provide a derivation from the initial state to the final state. If this is not possible, explain why.*

2 Ordered Inference [20 pts]

In class, we looked at incrementing binary numbers using rules of ordered inference. In this problem, you need to define the rules for decrementing a binary number. A binary number is written from left to right as $\$ \ 1 \ 0 \ \ldots \ 1$ where the \$ at the left end indicates the start of the number. Consider all the digits (including \$) as ordered propositions. To decrement a number, we introduce a new proposition called dec, which starts at the right end of the binary number.

In other words, to decrement a binary number, we will write the initial set of propositions as $\$ \ 1 \ 0 \ \ldots \ 1 \ \mathsf{dec}$.

Problem 4 (10 pts) *Define the rules of ordered inference for* dec.

Problem 5 (5 pts) *Consider the binary number* $\$ \ 1 \ 0 \ 0$. *Now, we need to decrement this number twice. Apply the rules defined in the previous problem twice, i.e., start with* $\$ \ 1 \ 0 \ 0 \ \mathsf{dec} \ \mathsf{dec}$ *and derive the final state. You should apply the rules sequentially (i.e., process the first* dec*, then process the second* dec*).*

Problem 6 (10 pts) *Now try to apply the two rules concurrently, i.e., process the two* dec *propositions simultaneously. Do you arrive at the same result? Why or why not?*

3 Linear Proofs [20 pts]

Problem 7 (20 pts) *For the following problems, determine if the propositions are provable or not. If they are provable, show the derivation and construct the corresponding session-typed proof term by giving appropriate channel names to each proposition.*

1. $A \vdash A \otimes A$
2. $A \otimes A \vdash A$
3. $A \otimes (B \oplus C) \vdash (A \otimes B) \oplus (A \otimes C)$
4. $A \oplus (B \otimes C) \vdash (A \oplus B) \otimes (A \oplus C)$
5. $A \otimes (B \otimes C) \vdash (A \otimes B) \otimes C$

To provide a session-typed term for types of the form $A \oplus B$, consider adding the following labels: Instead of using $A \oplus B$, use $\oplus\{\mathsf{left} : A, \mathsf{right} : B\}$.

4 Session-Typed Programming [20 pts]

In the following set of problems, you are expected to write programs in the session-typed programming language introduced in lecture. Recall the syntax of the language

$$\begin{array}{lrl} \text{Expressions} & P ::= & x.\mathsf{k}\,;\,P \mid \mathsf{case}\ x\ (\ell \Rightarrow P_\ell)_{\ell \in L} \mid y \leftarrow \mathsf{recv}\ x\,;\,P \mid \mathsf{send}\ x\ y\,;\,P \mid \mathsf{wait}\ x\,;\,P \mid \mathsf{close}\ x \\ & \mid & x \leftrightarrow y \mid x \leftarrow f\ \overline{y}\,;\,P \\ \text{Types} & A, B ::= & \oplus\{\ell : A_\ell\}_{\ell \in L} \mid \&\{\ell : A_\ell\}_{\ell \in L} \mid A \otimes B \mid A \multimap B \mid \mathbf{1} \end{array}$$

To get some programming experience, we will implement standard processes for lists. The type of list is defined as

$$\mathsf{type}\ \mathtt{list}_A = \oplus\{\mathbf{cons} : A \otimes \mathtt{list}_A, \mathbf{nil} : 1\}$$

Problem 8 (10 pts) *Define a process called* `append` *with the following type*

$$\mathsf{decl}\ \mathtt{append} : (l_1 : \mathtt{list}_A), (l_2 : \mathtt{list}_A) \vdash (l : \mathtt{list}_A)$$

The process appends list l_2 to the end of list l_1 and produces the output list on l.

Problem 9 (10 pts) *Define a process called* `alternate` *with the following signature:*

$$\mathsf{decl}\ \mathtt{alternate} : (l_1 : \mathtt{list}_A), (l_2 : \mathtt{list}_A) \vdash (l : \mathtt{list}_A)$$

This process outputs the elements of list l_1 and l_2 on the output list l in an alternate fashion, i.e., one element from l_1, next one from l_2 and so on. In other words, if $l_1 = [1; 2; 3]$ and $l_2 = [4; 5; 6; 7; 8]$, then $l = [1; 4; 2; 5; 3; 6; 7; 8]$. If the two lists are of unequal size, then l contains the leftover elements from the larger list in order as shown in the example above.