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, and three \$1 bills (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.*

Solution. The 6 propositions needed are:

1. `ten`: represents \$10 bill
2. `five`: represents \$5 bill
3. `one`: represents \$1 bill
4. `cookie`: represents one cookie box
5. `candy`: represents one candy bar
6. `chips`: represents one packet of chips

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

Solution. The rules are defined as

$$\frac{\texttt{ten}}{\texttt{cookie} \quad \texttt{five} \quad \texttt{one} \quad \texttt{one}}\ \textsc{10-Cookie} \qquad \frac{\texttt{ten}}{\texttt{candy} \quad \texttt{five} \quad \texttt{one} \quad \texttt{one} \quad \texttt{one}}\ \textsc{10-Candy}$$

$$\frac{\texttt{ten}}{\texttt{chips} \quad \texttt{five} \quad \texttt{one} \quad \texttt{one} \quad \texttt{one}}\ \textsc{10-Chips} \qquad \frac{\texttt{five}}{\texttt{cookie} \quad \texttt{one} \quad \texttt{one}}\ \textsc{5-Cookie}$$

$$\frac{\texttt{five}}{\texttt{candy} \quad \texttt{one} \quad \texttt{one} \quad \texttt{one}}\ \textsc{5-Candy} \qquad \frac{\texttt{five}}{\texttt{chips} \quad \texttt{one} \quad \texttt{one} \quad \texttt{one}}\ \textsc{5-Chips}$$

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.*

Solution. This is impossible. Consider the following derivation:

$$\begin{array}{l}
\qquad\qquad\qquad\qquad\texttt{ten} \\ \hline
\qquad\texttt{cookie}\quad\texttt{five}\quad\texttt{one}\quad\texttt{one} \qquad\qquad \textsc{10-Cookie} \\ \hline
\texttt{cookie}\quad\texttt{candy}\quad\texttt{one}\quad\texttt{one}\quad\texttt{one}\quad\texttt{one}\quad\texttt{one} \qquad \textsc{5-Candy}
\end{array}$$

At this point, there is no way to generate `chips`. The same would have happened if we had applied the 5-Candy rule. Hence, it is impossible to produce this derivation.

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 ... 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 ... 1 dec.

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

Solution. The rules for dec are as follows:

$$\frac{1 \quad \mathsf{dec}}{0}\ \textsc{1-Dec} \qquad\qquad \frac{0 \quad \mathsf{dec}}{\mathsf{dec} \quad 1}\ \textsc{0-Dec} \qquad\qquad \frac{\$ \quad 0}{\$}\ \textsc{0-Trim}$$

The last rule trims the leading 0.

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* dec dec *and derive the final state. You should apply the rules sequentially (i.e., process the first* dec*, then process the second* dec*).*

Solution.

$$\begin{array}{l}
\$\quad 1\quad 0\quad 0\quad \mathsf{dec}\quad \mathsf{dec} \\ \hline
\$\quad 1\quad 0\quad \mathsf{dec}\quad 1\quad \mathsf{dec} \qquad \textsc{0-Dec} \\ \hline
\$\quad 1\quad \mathsf{dec}\quad 1\quad 1\quad \mathsf{dec} \qquad \textsc{0-Dec} \\ \hline
\$\quad 0\quad 1\quad 1\quad \mathsf{dec} \qquad \textsc{1-Dec} \\ \hline
\$\quad 1\quad 1\quad \mathsf{dec} \qquad \textsc{0-Trim} \\ \hline
\$\quad 1\quad 0 \qquad \textsc{1-Dec}
\end{array}$$

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?*

Solution. Applying the two rules concurrently will also lead to the same result. There are no rules that mix the interaction of the two dec propositions. So, the order does not matter.

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 $\oplus\{\mathsf{left} : A, \mathsf{right} : B\}$.

Solution. Below are the solutions:

1. Not provable.

2. Not provable.

3.

$$
\dfrac{
\dfrac{
\dfrac{x : B \vdash (y \leftrightarrow x) :: y : B}
{\begin{array}{c} z : A, x : B \vdash \\ (\mathsf{send}\ y\ z\ ;\ y \leftrightarrow x) \\ :: y : (A \otimes B) \end{array}} \otimes\mathrm{R}
}
{\begin{array}{c} z : A, x : B \vdash \\ (y.\mathsf{left}\ ;\ \mathsf{send}\ y\ z\ ;\ y \leftrightarrow x) \\ :: y : (A \otimes B) \oplus (A \otimes C) \end{array}} \oplus\mathrm{R1}
\quad
\dfrac{
\dfrac{x : C \vdash (y \leftrightarrow x) :: y : C}
{\begin{array}{c} z : A, x : C \vdash \\ (\mathsf{send}\ y\ z\ ;\ y \leftrightarrow x) \\ :: y : (A \otimes C) \end{array}} \otimes\mathrm{R}
}
{\begin{array}{c} z : A, x : C \vdash \\ (y.\mathsf{right}\ ;\ \mathsf{send}\ y\ z\ ;\ y \leftrightarrow x) \\ :: y : (A \otimes B) \oplus (A \otimes C) \end{array}} \oplus\mathrm{R2}
}
{
\dfrac{\begin{array}{c} z : A, x : B \oplus C \vdash \\ \mathsf{case}\ x\ (\mathsf{left} \Rightarrow y.\mathsf{left}\ ;\ \mathsf{send}\ y\ z\ ;\ y \leftrightarrow x \mid \mathsf{right} \Rightarrow y.\mathsf{right}\ ;\ \mathsf{send}\ y\ z\ ;\ y \leftrightarrow x) \\ :: y : (A \otimes B) \oplus (A \otimes C) \end{array}}
{\begin{array}{c} x : A \otimes (B \oplus C) \vdash \\ z \leftarrow \mathsf{recv}\ x\ ;\ \mathsf{case}\ x\ (\mathsf{left} \Rightarrow y.\mathsf{left}\ ;\ \mathsf{send}\ y\ z\ ;\ y \leftrightarrow x \mid \mathsf{right} \Rightarrow y.\mathsf{right}\ ;\ \mathsf{send}\ y\ z\ ;\ y \leftrightarrow x) \\ :: y : (A \otimes B) \oplus (A \otimes C) \end{array}} \otimes\mathrm{L}
} \oplus\mathrm{L}
$$

4. Not provable.

5.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{x : C \vdash (y \leftrightarrow x) :: y : C}
{w : B, x : C \vdash (\mathsf{send}\ y\ w\ ;\ y \leftrightarrow x) :: y : (B \otimes C)} \otimes\mathrm{R}
}
{z : A, w : B, x : C \vdash (\mathsf{send}\ y\ z\ ;\ \mathsf{send}\ y\ w\ ;\ y \leftrightarrow x) :: y : A \otimes (B \otimes C)} \otimes\mathrm{R}
}
{z : A, x : B \otimes C \vdash (w \leftarrow \mathsf{recv}\ x\ ;\ \mathsf{send}\ y\ z\ ;\ \mathsf{send}\ y\ w\ ;\ y \leftrightarrow x) :: y : A \otimes (B \otimes C)} \otimes\mathrm{L}
}
{x : A \otimes (B \otimes C) \vdash (z \leftarrow \mathsf{recv}\ x\ ;\ w \leftarrow \mathsf{recv}\ x\ ;\ \mathsf{send}\ y\ z\ ;\ \mathsf{send}\ y\ w\ ;\ y \leftrightarrow x) :: y : A \otimes (B \otimes C)} \otimes\mathrm{L}
$$

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

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

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.

Solution.

proc $l \leftarrow$ append l_1 l_2 =
  case l_1
  (nil $\Rightarrow$ wait l_1 ; $l \leftrightarrow l_2$
  | cons $\Rightarrow$ $x \leftarrow$ recv l_1 ; l.cons ; send l x ; $l \leftarrow$ append l_1 l_2
)

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.

Solution.

proc $l \leftarrow$ `alternate` l_1 l_2 =
  case l_1
  (nil $\Rightarrow$ wait l_1 ; $l \leftrightarrow l_2$
  | cons $\Rightarrow$ $x \leftarrow$ recv l_1 ; l.cons ; send l x ; $l \leftarrow$ alternate l_2 l_1
)