Assignment 3
Linear Message-Passing Programs

15-836: Substructural Logics
Frank Pfenning

Due Monday, September 25, 2023
80 points

In this assignment we explore linear message-passing programs, based on the propositions-as-types correspondence between linear logic and session types.

You should hand in two files:

- `hw3.pdf` with the solutions to Problem 2.
- `hw3.mps` with the programs written in the MPASS language as specified in Problem 1. This file is autograded. If you have difficulty with the syntax or semantics of the language, please use Piazza for help.

The handout has the following structure:

- `src/`, the implementation of MPass.
- `starter/`, the starter code for this assignment, including a template for `hw3.mps`
- `hw3.pdf`, this assignment spec
- `*.{tex,sty}`, LaTeX source, macros, and style files

1 Linear Message-Passing Programs (50 pts)

In all the problems below, you may define auxiliary processes as you with. Efficiency is not an consideration, just correctness. In lecture, we use metavariable like A and B to stand for arbitrary propositions (= types), but these are not supported in the current implementation so we use lists and stores of binary numbers instead.

Task 1 We define Booleans with

```
type bool = +{ 'false : 1, 'true : 1 }
```

Define each of the following processes on Booleans.

(i) $a : \mathsf{bool} \vdash \mathit{not} :: (c : \mathsf{bool})$

(ii) $a : \mathsf{bool}, b : \mathsf{bool} \vdash \mathit{and} :: (c : \mathsf{bool})$

(iii) $a : \mathsf{bool}, b : \mathsf{bool} \vdash \mathit{or} :: (c : \mathsf{bool})$

Task 2 We define the binary numbers with

```
type bin = +{ 'e : 1, 'b0 : bin, 'b1 : bin }
```

where the least significant bit is the first message. The numbers should have no "leading" zeros, that is, ϵ 1 1 0 is six, but ϵ 0 1 1 0 is disallowed.

Define the predecessor *pred*

$$x : \mathsf{bin} \vdash \mathit{pred} :: (y : \mathsf{bin})$$

of zero to be zero. You may assume x does not send any leading zeros, and must ensure it for y.

Task 3 We define lists of binary numbers with

```
type list = +{ 'nil : 1, 'cons : bin * list }
```

Define a process *reverse* that reverses a given input list l to the output list r.

$$l : \mathsf{list} \vdash \mathit{reverse} :: (r : \mathsf{list})$$

Task 4 We cannot define the usual functional map

$$\mathit{map}\ (f : A \to B)\ (l : \mathsf{list} A) : \mathsf{list}\ B$$

because f may be used many times, including zero if l is empty. But we can use iteration instead and remain linear.

```
type iterator = &{ 'next : bin -o (bin * iterator) , 'done : 1 }
```

(i) Define a different version of *map*

$$\mathit{iter} : \mathsf{iterator}, l : \mathsf{list} \vdash \mathit{map} :: (r : \mathsf{list})$$

that applies the given iterator to each element in l in order to yield r.

(ii) Define *isucc*

$$\cdot \vdash \mathit{isucc} :: (i : \mathsf{iterator})$$

that increments each element.

Task 5 We define a store interface with

```
type store = &{ 'ins : bin -o store,
                'del : +{ 'none : 1, 'some : bin * store } }
```

In lecture, we gave an implementation that behaved as a stack. Give an alternative implementation that behaves like a queue.

(i) $\cdot \vdash \mathit{empty\_queue} :: (s : \mathsf{store})$

(ii) $x : \mathsf{bin}, t : \mathsf{store} \vdash \mathit{node\_queue} :: (s : \mathsf{store})$

2 Weakening and Contraction (30 pts)

In (purely) linear logic, we can define propositions *inductively*

$$\text{propositions} \quad A, B \quad ::= \quad P \mid A \otimes B \mid A \multimap B \mid \mathbf{1} \mid A \oplus B \mid \mathbf{0} \mid A \mathbin{\&} B \mid \top$$

Now consider the rules of weakening and contraction, which are not sound for linear logic.

$$\frac{\Delta \vdash C}{\Delta, A \vdash C}\ \mathsf{weaken} \qquad\qquad \frac{\Delta, A, A \vdash C}{\Delta, A \vdash C}\ \mathsf{contract}$$

Task 6 Give an inductive characterization of a class of propositions in linear logic for which both weakening and contraction are *admissible*, that is, every instance of the rule is correct. It does not need to be the largest such class, but it should have a simple definition.

Task 7 Prove the admissibility of *weakening* for propositions in your class. You should clearly state the structure of your proof. You may freely use the admissibility of cut and identity. If your proof is by induction, show one base case and one inductive case.

Task 8 Prove the admissibility of *contraction* for propositions in your class. You should clearly state the structure of your proof. You may freely use the admissibility of cut and identity. If your proof is by induction, show one base case and one inductive case.

Task 9 In MPASS we do not have atomic types P but we have recursive types. How does the class of propositions admitting weakening and contraction change? Speculate about how you may or may not want to extend MPASS to take advantage of this property. Use examples to illustrate your thoughts.