Written Assignment 6

CS 538, Spring 2020

April 16, 2020

In this homework, you will be working with a core, concurrent language, also known as a *process calculus*. Our version is loosely modeled after Robin Milner's CCS. We first set up the language grammar.

$$
\begin{aligned}
\text{Num } n &::= 0 \mid 1 \mid 2 \mid \cdots \\
\text{Var } v &::= x \mid y \mid z \mid \cdots \\
\text{Exp } e &::= v \mid n \mid e_1 + e_2 \mid e_1 \times e_2 \mid \cdots \\
\text{BExp } b &::= \mathit{true} \mid \mathit{false} \mid e_1 < e_2 \mid \cdots \\
\text{Chn } C &::= A \mid B \mid C \mid \cdots \\
\text{Prc } P, Q &::= \mathit{done} \mid v \Leftarrow C; P \mid e \Rightarrow C; P \mid \mathit{mk}\ C\{P\} \mid [b]\, P \mid P + Q \mid P|Q
\end{aligned}
$$

In words, we have the following components:

- Numbers (0, 1, ...)
- Variables (x, y, ...)
- Expressions possibly involving variables ($x + 0$, 3×5)
- Boolean expressions ($x < 0$)
- Channel names (A, B, ...)

Processes are built out of several components. Following the order in the grammar:

- The done process, which doesn't do anything.
- Receiving a message from a channel, and using it in another process.
- Sending a message to a channel, then continuing as another process.
- Declaring a new channel for use in a process.
- Running a process if some boolean condition is true.
- Running one process or another process.
- Running two processes in parallel.

We will work with an operational semantics for processes, except steps may have *labels*:

- $P \xrightarrow{A,5} P'$ means that P sends 5 along channel A and steps to P'.
- $Q \xrightarrow{\bar{A},5} Q'$ means that Q receives 5 along channel A and steps to Q'.
- $P \longrightarrow Q$ means that P steps to Q without sending or receiving anything externally. (Components within P may exchange messages before stepping to Q.)

We write L for any label: send, receive, or none. We assume that we have a standard step relation for expressions and boolean expressions: $e \to e'$.

$$\frac{e \to e'}{e \Rightarrow A; P \longrightarrow e' \Rightarrow A; P} \qquad \frac{}{n \Rightarrow A; P \xrightarrow{A,n} P} \qquad \frac{}{x \Leftarrow A; P \xrightarrow{\bar{A},n} P[x \mapsto n]} \qquad \frac{P \xrightarrow{L} Q \qquad A, \bar{A} \notin L}{mk\ A\{P\} \xrightarrow{L} mk\ A\{Q\}}$$

$$\frac{b \to b'}{[b]\,P \longrightarrow [b']\,P} \qquad \frac{}{[true]\,P \longrightarrow P} \qquad \frac{}{[false]\,P \longrightarrow done} \qquad \frac{}{P + Q \longrightarrow P} \qquad \frac{}{P + Q \longrightarrow Q}$$

$$\frac{P \xrightarrow{L} P'}{P|Q \xrightarrow{L} P'|Q} \qquad \frac{Q \xrightarrow{L} Q'}{P|Q \xrightarrow{L} P|Q'} \qquad \frac{P \xrightarrow{A,n} P' \qquad Q \xrightarrow{\bar{A},n} Q'}{P|Q \longrightarrow P'|Q'}$$

1 Stepping processes (15)

Step the following processes. Note that there are many possible traces for each process. Just show one trace where (i) the first step has a label of the form $\bar{I}, m$, and (ii) the last step has a label of the form O, n. (Perhaps I and O are special input/output channels, like the input and output of the main thread.)

We've done the first one for you.

0. $x \Leftarrow I; x \Rightarrow A; done | y \Leftarrow A; y + 1 \Rightarrow O; done$

$$\begin{aligned}
&\xrightarrow{\bar{I},42} 42 \Rightarrow A; done | y \Leftarrow A; y + 1 \Rightarrow O; done \\
&\longrightarrow done | 42 + 1 \Rightarrow O; done \\
&\longrightarrow done | 43 \Rightarrow O; done \\
&\xrightarrow{O,43} done | done
\end{aligned}$$

1. $x \Leftarrow I; x + 1 \Rightarrow O; done$
2. $x \Leftarrow I; ([x > 12]\,100 \Rightarrow O; done) + ([x \leq 12]\,5 \Rightarrow O; done)$
3. $x \Leftarrow I; x - 1 \Rightarrow O; done | x \Leftarrow I; x + 1 \Rightarrow O; done$
4. $x \Leftarrow I; x \Rightarrow A; done | y \Leftarrow A; y + y \Rightarrow O; done$
5. $mk\ A\{x \Leftarrow I; x \Rightarrow A; done | y \Leftarrow A; y + y \Rightarrow O; done\}$

2 Writing processes (10)

Write down a process that models the following scenarios. Demonstrate how the first process works by giving one trace. For the second process, explain why it cannot make any progress.

1. Two communicating threads. The first adds one to the input and passes to the second, the second adds two and passes it back to the first, and the first adds three and sends to output.
2. Two deadlocked threads. Both threads should be trying to receive on a channel, but the combined process should not be able to reduce. Note: your process should not be able to reduce, even if there are some external inputs or outputs on channels besides I and O.

3 Recursive processes (5)

To handle recursive processes, we extend the grammar with process variables and definitions, and allow process variables to show up in processes:

$$\begin{aligned} \text{PVar} &::= X \mid Y \mid Z \mid \cdots \\ \text{PDef} &::= X \triangleq P \\ \text{Prc } P, Q &::= X \mid \cdots \end{aligned}$$

Note that process definitions can be recursive—the process P on the right-hand side of $X \triangleq P$ can mention X (or other process variables). We also introduce a step rule to unfold a definition:

$$\frac{X \triangleq P}{X \longrightarrow P}$$

1. A doubling server. Write a process that continually listens to a channel A, receives a value, doubles the value and outputs it on a channel B. Your process should loop: it should be able to handle an arbitrary number of inputs to A, not just one input.

2. A stateful server. So far, all our processes have been *state-less*: they do not remember previous messages that have been sent or received. While we could augment processes with a notion of local state by adding a store, it turns out hat we can borrow an idea from the λ calculus and use the process itself to represent the state.

   To see how this works, write a process that continually listens to a channel A. For every odd (first, third, fifth, ...) incoming request, your process should output zero on channel B. For every even (second, fourth, sixth, ...) incoming request, your process should double the input and output on channel B. (Hint: you should probably write two definitions, representing the behavior in odd counts, and the behavior in even counts.)

For each part, your solution should consist of (i) one or more definitions, and (ii) a main process (which might be just a process variable). Demonstrate how your processes work by composing in parallel with a client process and showing a trace. Your client processes do not need to be very complicated, perhaps they just issue 1-2 requests. You can skip over uninteresting steps in the reduction.