Exercises

Concurrency and Distributed Systems

January 2023

Contents

- Vending machines
- Lifts

Vending machines

Write an alternative version of your process `VM1` using `let` and `within`. Call this process `VM1a`. Check that it has exactly the same traces as `VM1` using the following pair of assertions:

```
assert VM1 [T= VMa

assert VMa [T= VM1
```

We can add another event to our description:

- `cancel`: having inserted a coin, the user presses a button to cancel the request; the machine will then return the coin.

We will not model the action of returning the coin.

Declare this new event, and write a new process `VMcancel` to describe the behaviour of the machine in terms of `coin`, `tea`, `coffee`, and `cancel`.

Finite capacity

In practice, the vending machine is capable of serving only a certain number of cups of tea or coffee before it runs out of water and needs to be refilled.

> `refill`: refill the machine so that it is full to capacity

We will ignore the supply of the other ingredients, and assume that the availability of water is the only limitation.

We will assume that the machine starts operation with a full tank of water.

Complete the following definition to define a process that describes the behaviour of a vending machine that can hold enough water for exactly `max` cups of tea or coffee.

```
VMF(max) =
  let
    State(n) =
      ...
  within
    State(max)
```

Here, `State(n)` describes its behaviour when it has enough water left for `n` cups.

Graph and probe your new process.

Nondeterministic capacity

Using the same `refill` event as before, write a process `VMR(max)` to describe the behaviour of a vending machine of unknown and possibly-variable capacity.

All that we know is that the capacity is bounded by `max`: the machine cannot dispense more than `max` cups without being refilled.

This machine should not allow the user to insert a `coin` if it would be unable to dispense a cup of tea or coffee. It should not allow the user to `cancel`.

Lifts

Consider the following, parameterised version of our lift controller process:

```
LiftController(max) =
  let
    AtFloorOpen(n) =
      close -> AtFloorClosed(n)
      []
      (n < max) & up -> GoingUp(n)
      []
      (n > 0) & down -> GoingDown(n)

    AtFloorClosed(n) =
      open -> AtFloorOpen(n)
      []
```

```
      (n < max) & up -> GoingUp(n)
      []
      (n > 0) & down -> GoingDown(n)

    GoingUp(n) =
      arrive -> AtFloorOpen(n+1)

    GoingDown(n) =
      arrive -> AtFloorOpen(n-1)
  within
    AtFloorOpen(0)
```

Graph and probe your new process.