Modelling and Analysis of a Cyber-Physical System with Monads

Cyber-Physical Programming — Practical Assignment 2

Melânia Pereira   Paulo R. Pereira
{pg47520, pg47554}@alunos.uminho.pt

June 17, 2022

Abstract

ola

1 The Adventurers' Problem

In the middle of the night, four adventurers encounter a shabby rope-bridge spanning a deep ravine. For safety reasons, they decide that no more than 2 people should cross the bridge at the same time and that a flashlight needs to be carried by one of them in every crossing. They have only one flashlight. The 4 adventurers are not equally skilled: crossing the bridge takes them 1, 2, 5, and 10 minutes, respectively. A pair of adventurers crosses the bridge in an amount of time equal to that of the slowest of the two adventurers.

One of the adventurers claims that they cannot be all on the other side in less than 19 minutes. One companion disagrees and claims that it can be done in 17 minutes.

Who is right? That's what we're going to find out.

2 Monadic Approach via HASKELL for Modelling the Problem

2.1 The monads used

explain the monads here

2.2 Modelling the problem

Adventurers are represented by the following data type:

$$\textbf{data}\ \mathit{Adventurer} = P_1 \mid P_2 \mid P_5 \mid P_{10}\ \textbf{deriving}\ (\mathit{Show}, \mathit{Eq})$$

Lantern is represented by the $()$ element, so we can represent all the entities by using the coproduct and defining the following data type:

$$\textbf{type}\ \mathit{Object} = \mathit{Adventurer} + ()$$
$$\mathit{lantern} = i_2\ ()$$

The names for the adventurers are quite suggestive as they are identified by the time they take to cross. However, it will be very useful to have a function that returns, for each adventurer, the time it takes to cross the bridge.

$getTimeAdv :: Adventurer \rightarrow Int$
$getTimeAdv\ P_1 = 1$
$getTimeAdv\ P_2 = 2$
$getTimeAdv\ P_5 = 5$
$getTimeAdv\ P_{10} = 10$

Now, we need to define the state of the game, i.e. the current position of each object (adventurers + the lantern). The function $\underline{False}$ represents the initial state of the game, with all adventurers and the lantern on the left side of the bridge. Similarly, the function $\underline{True}$ represents the end state of the game, with all adventurers and the lantern on the right side of the bridge. We also need to define the instances $Show$ and Eq to visualize and compare, respectively, the states of the game.

$\mathbf{type}\ State = Object \rightarrow Bool$

$\mathbf{instance}\ Show\ State\ \mathbf{where}$
$\quad show\ s = show \cdot show\ \$\ [s\ (i_1\ P_1),$
$\qquad s\ (i_1\ P_2),$
$\qquad s\ (i_1\ P_5),$
$\qquad s\ (i_1\ P_{10}),$
$\qquad s\ (i_2\ ())]$

$\mathbf{instance}\ Eq\ State\ \mathbf{where}$
$\quad (\equiv)\ s1\ s2 = and\ [s1\ (i_1\ P_1) \equiv s2\ (i_1\ P_1),$
$\qquad s1\ (i_1\ P_2) \equiv s2\ (i_1\ P_2),$
$\qquad s1\ (i_1\ P_5) \equiv s2\ (i_1\ P_5),$
$\qquad s1\ (i_1\ P_{10}) \equiv s2\ (i_1\ P_{10}),$
$\qquad s1\ (i_2\ ()) \equiv s2\ (i_2\ ())]$

$gInit :: State$
$gInit = \underline{False}$

$gEnd :: State$
$gEnd = \underline{True}$

$state2List :: State \rightarrow [Bool]$
$state2List\ s = [s\ (i_1\ P_1),$
$\quad s\ (i_1\ P_2),$
$\quad s\ (i_1\ P_5),$
$\quad s\ (i_1\ P_{10}),$
$\quad s\ (i_2\ ())]$

Changes the state of the game for a given object:

$changeState :: Object \rightarrow State \rightarrow State$
$changeState\ a\ s = \mathbf{let}\ v = s\ a\ \mathbf{in}\ (\lambda x \rightarrow \mathbf{if}\ x \equiv a\ \mathbf{then}\ \neg\ v\ \mathbf{else}\ s\ x)$

Changes the state of the game of a list of Object

$mChangeState :: [Object] \rightarrow State \rightarrow State$
$mChangeState\ os\ s = foldr\ changeState\ s\ os$

For a given state of the game, the function presents all the possible moves that the adventurers can make.

$allValidPlays :: State \rightarrow ListLogDur\ State$
$allValidPlays\ s = LSD\ \$\ \mathsf{map}\ \ Duration\ \$\ \mathsf{map}\ \ (id \times \langle toTrace\ s, id\rangle \cdot (mCS\ s))\ t\ \mathbf{where}$
$\quad t = (\mathsf{map}\ \ (addLantern \cdot addTime) \cdot combinationsUpTo2 \cdot advsWhereLanternIs)\ s$

$mCS = flip\ mChangeState$
$toTrace\ s\ s' = printTrace\ (state2List\ s, state2List\ s')$

$addTime :: [Adventurer] \rightarrow (Int, [Adventurer])$
$addTime = \langle maximum \cdot (\mathsf{map}\ \ getTimeAdv), id\rangle$

$addLantern :: (Int, [Adventurer]) \rightarrow (Int, [Object])$
$addLantern = id \times ((lantern:) \cdot \mathsf{map}\ \ i_1)$

$advsWhereLanternIs :: State \rightarrow [Adventurer]$
$advsWhereLanternIs\ s = filter\ ((\equiv s\ lantern) \cdot s \cdot i_1)\ [P_1, P_2, P_5, P_{10}]$

$combinationsUpTo2 :: Eq\ a \Rightarrow [a] \rightarrow [[a]]$
$combinationsUpTo2 = \mathsf{conc} \cdot \langle f, g\rangle$ **where**
$f\ t =$ **do** $\{x \leftarrow t; return\ [x]\}$
$g\ t =$ **do** $\{x \leftarrow t; y \leftarrow (remove\ x\ t); return\ [x, y]\}$
$remove\ x\ [\,] = [\,]$
$remove\ x\ (h : t) =$ **if** $x \equiv h$ **then** t **else** $remove\ x\ t$

```
> combinationsUpTo2 [1,2,3]
[[1], [2], [3], [1,2], [1,3], [2,3]]
```

2.2.1 The trace log

As we saw, our monad $ListLogDur$ keeps the trace by calling the function $toTrace :: State \rightarrow State \rightarrow String$. But what does it do?

First, we can see that, according to the representation of the state, adventurers can be represented by indexes. We take advantage of this to be able to present an elegant trace of the moves. For example, if the previous state is $[False, False, False, False, False]$ and the current state is $[True, True, False, False, True]$, we know that P_1 and P_2 have crossed (because the first two and the last elements and diferent). So, we can simply compare element to element and, if they are different, we keep the index. In the previous example, it would return $[0, 1, 4]$ — index 4 represents the lantern, and because we assume that the movements are always valid, we can ignore that.

$index2Adv :: Int \rightarrow String$
$index2Adv\ 0 = \texttt{"P1"}$
$index2Adv\ 1 = \texttt{"P2"}$
$index2Adv\ 2 = \texttt{"P5"}$
$index2Adv\ 3 = \texttt{"P10"}$

$indexesWithDifferentValues :: Eq\ a \Rightarrow ([a], [a]) \rightarrow [Int]$
$indexesWithDifferentValues\ (l_1, l_2) = aux\ l_1\ l_2\ 0$ **where**
$aux :: Eq\ a \Rightarrow [a] \rightarrow [a] \rightarrow Int \rightarrow [Int]$
$aux\ [\,]\ l\ \_ = [\,]$
$aux\ l\ [\,]\ \_ = [\,]$
$aux\ (h_1 : t1)\ (h_2 : t2)\ index =$ **if** $h_1 \not\equiv h_2$ **then** $index : aux\ t1\ t2\ (index + 1)$
else $aux\ t1\ t2\ (index + 1)$

The result $[0, 1, 4]$ means that "P_1 and P_2 crosses". We now have automate this (pretty) print. We only need to ignore the lantern index (4), convert the indexes to the respective adventurers and define a print function for them.

$printTrace :: ([Bool], [Bool]) \rightarrow String$
$printTrace = prettyLog \cdot (\mathsf{map}\ \ index2Adv) \cdot init \cdot indexesWithDifferentValues$

$$
\begin{aligned}
&\mathit{prettyLog} :: [\mathit{String}] \to \mathit{String}\\
&\mathit{prettyLog} = (>1) \cdot \mathit{length} \to f\ ,\ (+\!\!+\texttt{" cross\textbackslash n"}) \cdot \mathit{head}\ \textbf{where}\\
&\quad f = (+\!\!+\texttt{" crosses\textbackslash n"}) \cdot \mathsf{conc} \cdot ((\mathit{concat} \cdot \mathsf{map}\ (+\!\!+\texttt{" and "})) \times \mathit{id}) \cdot \langle \mathit{init}, \mathit{last} \rangle
\end{aligned}
$$

Let's see the result of applying the function $\mathit{printTrace}$ with the previous example.

```
> t = ([False,False,False,False,False],[True,True,False,False,True])
> printTrace t
"P1 and P2 crosses\n"
```

Finnaly, using the function putStr, we get a pretty nice log:

```
> putStr $ printTrace t
P1 and P2 crosses
```

In the next subsection, we'll see the trace of the optimal play which shows how elegant the log is.

2.3 Solving the problem

For a given number n and initial state, the function calculates all possible n-sequences of moves that the adventures can make

$$
\begin{aligned}
&\mathit{exec} :: \mathit{Int} \to \mathit{State} \to \mathit{ListLogDur}\ \mathit{State}\\
&\mathit{exec}\ 0\ s = \mathit{allValidPlays}\ s\\
&\mathit{exec}\ n\ s = \textbf{do}\ \mathit{ps} \leftarrow \mathit{exec}\ (n-1)\ s\\
&\quad \mathit{allValidPlays}\ \mathit{ps}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{execPred} :: (\mathit{State} \to \mathit{Bool}) \to \mathit{State} \to (\mathit{Int}, \mathit{ListLogDur}\ \mathit{State})\\
&\mathit{execPred}\ p\ s = \mathit{aux}\ p\ s\ 0\ \textbf{where}\\
&\quad \mathit{aux}\ p\ s\ \mathit{it} = \textbf{let}\ \mathit{st} = \mathit{exec}\ \mathit{it}\ s\\
&\qquad \mathit{res} = \mathit{filter}\ \mathit{pred}\ (\mathsf{map}\ \ \mathit{remDur}\ (\mathit{remLSD}\ \mathit{st}))\ \textbf{in}\\
&\qquad \textbf{if}\ \mathit{length}\ (\mathit{res}) > 0\ \textbf{then}\ ((\mathit{it}+1), \mathit{LSD}\ (\mathsf{map}\ \ \mathit{Duration}\ \mathit{res}))\\
&\qquad \textbf{else}\ \mathit{aux}\ p\ s\ (\mathit{it}+1)\ \textbf{where}\\
&\qquad\quad \mathit{remDur}\ (\mathit{Duration}\ a) = a\\
&\qquad\quad \mathit{pred}\ (\_, (\_, s)) = p\ s
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{leqX} :: \mathit{Int} \to (\mathit{Int}, \mathit{Bool})\\
&\mathit{leqX}\ \ n = \textbf{if}\ \mathit{res}\ \textbf{then}\ (\mathit{it}, \mathit{res})\\
&\qquad \textbf{else}\ (0, \mathit{res})\ \textbf{where}\\
&\qquad\quad \mathit{res} = \mathit{length}\ (\mathit{filter}\ p\ (\mathsf{map}\ \ \mathit{remDur}\ (\mathit{remLSD}\ l))) > 0\\
&\qquad\quad (\mathit{it}, l) = \mathit{execPred}\ (\equiv \mathit{gEnd})\ \mathit{gInit}\\
&\qquad\quad p\ \ (d, (\_, \_)) = d \leqslant n\\
&\qquad\quad \mathit{remDur}\ (\mathit{Duration}\ a) = a
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{lX} :: \mathit{Int} \to (\mathit{Int}, \mathit{Bool})\\
&\mathit{lX}\ \ n = \textbf{if}\ \mathit{res}\ \textbf{then}\ (\mathit{it}, \mathit{res})\\
&\quad \textbf{else}\ (0, \mathit{res})\ \textbf{where}\\
&\qquad \mathit{res} = \mathit{length}\ (\mathit{filter}\ p\ (\mathsf{map}\ \ \mathit{remDur}\ (\mathit{remLSD}\ l))) > 0\\
&\qquad (\mathit{it}, l) = \mathit{execPred}\ (\equiv \mathit{gEnd})\ \mathit{gInit}\\
&\qquad p\ \ (d, (\_, \_)) = d < n\\
&\qquad \mathit{remDur}\ (\mathit{Duration}\ a) = a
\end{aligned}
$$

Question: Is it possible for all adventurers to be on the other side in $\leqslant$ 17 minutes and not exceeding 5 moves?

$$leq17 :: Bool$$
$$leq17 = \pi_2\ (leqX\ 17) \wedge \pi_1\ (leqX\ 17) \leqslant 5$$

Question: Is it possible for all adventurers to be on the other side in <17 minutes?

$$l17 :: Bool$$
$$l17 = \pi_2\ (lX\ 17)$$

As we saw, it is possible for all adventurers to be on the other side in $\leqslant$ 17 minutes and not exceeding 5 moves (actually we exactly 5 moves). We also prooved that it isn't possible for all adventurers to be on the other side in <17 minutes. So, one could get that information by executing the following function *optimalTrace*.

$$
\begin{aligned}
& optimalTrace :: \mathsf{IO}\ ()\\
& optimalTrace =\\
& \quad putStrLn \cdot t \cdot \mathsf{map}\ \ remDur \cdot remLSD \cdot \pi_2\ \$\ execPred\ (\equiv gEnd)\ gInit\ \mathbf{where}\\
& \quad t = prt \cdot \langle head \cdot \mathsf{map}\ \ \pi_1, \mathsf{map}\ \ (\pi_1 \cdot \pi_2)\rangle \cdot pairFilter \cdot \langle minimum \cdot \mathsf{map}\ \ \pi_1, id\rangle\\
& \quad remDur\ (Duration\ a) = a\\
& \quad pairFilter\ (d, l) = filter\ (\lambda(d', (\_, \_)) \rightarrow d \equiv d')\ l\\
& \quad p = (>1) \cdot length \rightarrow p'\ ,\ head\\
& \quad p' = \mathsf{conc} \cdot \langle concat \cdot \mathsf{map}\ \ ((+\!\!+ (\texttt{"\textbackslash nOR\textbackslash n\textbackslash n"}))) \cdot init, last\rangle\\
& \quad prt\ (d, l) = (p\ l) +\!\!+ \texttt{"\textbackslash nin "} +\!\!+ (show\ d) +\!\!+ \texttt{" minutes."}
\end{aligned}
$$

Result:

```
> optimalTrace
P1 and P2 crosses
P1 cross
P5 and P10 crosses
P2 cross
P1 and P2 crosses

OR

P1 and P2 crosses
P2 cross
P5 and P10 crosses
P1 cross
P1 and P2 crosses

in 17 minutes.
```

3 Comparative Analysis and Final Comments