RL1 - Solution Sketches

March 2, 2020

We sketch here the solutions for the exercises of lecture RL1 in a brief manner. Note that a proper solution would require more detailed descriptions, explanations, and in some cases examples. Some of the exercises may have more than one solution, and we just show one of them.

Exercise RL1.1. Wolf-goat-cabbage

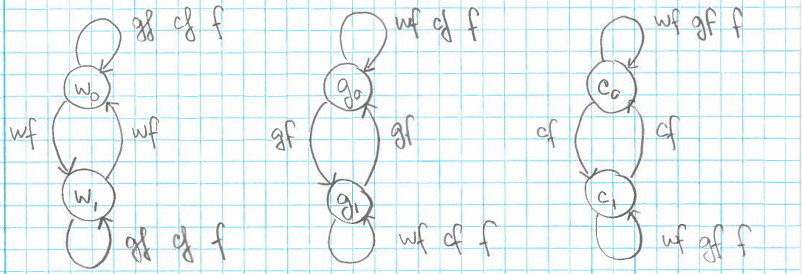
02141 /HRW

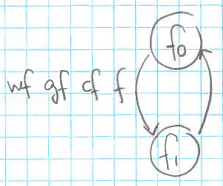
The wolf-goat-cabbage problem

First we construct automata for each of the four individuals. Each automaton has two state, one for each side of the river. We have four actions

wf : wolf and ferryman cross the river
gf : goat and ferryman cross the river
cf : cabbage and ferryman cross the river
f : ferryman alone crosses the river.

The four automata are





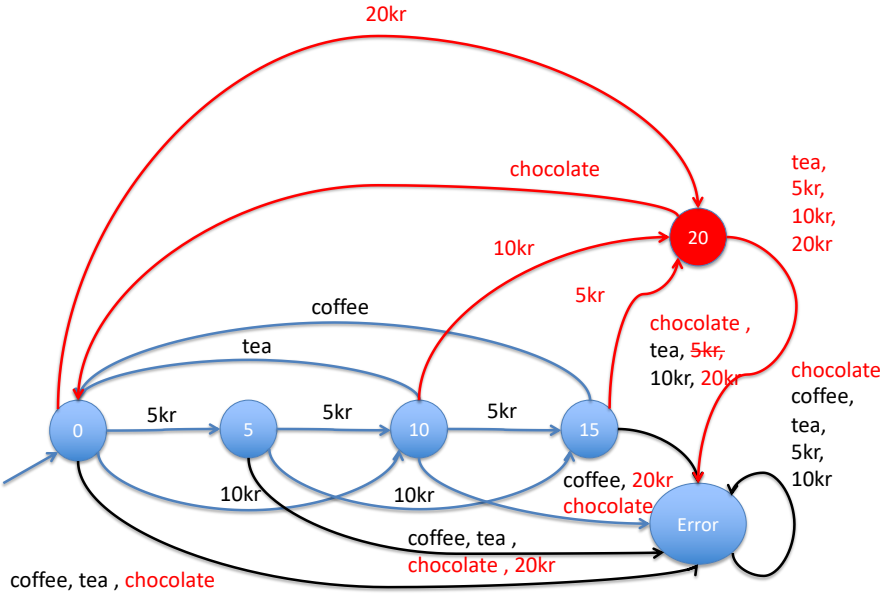
The product automaton has $16 = 2 \cdot 2 \cdot 2 \cdot 2$ states - each describing where the individuals are. The automaton is as follows:

p.1

Exercise RL1.2. Vending Machine in F# One possibility is to encode the automaton / vending machine as a follows:

```
// States are implemented as union types
type State = S0 | S5 | S10 | S15 | ERROR

// Input symbols are implemented as union types
type Symbol = KR5 | KR10 | TEA | COFFEE

// Transitions are implemented as a function
let nextState state symbol =
    match state , symbol with
    // Transitions from state S0
    | S0 , KR5    -> S5
    | S0 , KR10   -> S10
    | S0 , TEA    -> ERROR
    | S0 , COFFEE -> ERROR
    // Transitions from state S0
    | S5 , KR5    -> S10
    | S5 , KR10   -> S15
    | S5 , TEA    -> ERROR
    | S5 , COFFEE -> ERROR
    // Transitions from state S0
    | S10 , KR5    -> S15
    | S10 , KR10   -> ERROR
    | S10 , TEA    -> S0
    | S10 , COFFEE -> ERROR
    // Transitions from state S0
    | S15 , KR5    -> ERROR
    | S15 , KR10   -> ERROR
    | S15 , TEA    -> ERROR
    | S15 , COFFEE -> S0
      // Transitions from state S0
    | ERROR , KR5    -> ERROR
    | ERROR , KR10   -> ERROR
    | ERROR , TEA    -> ERROR
    | ERROR , COFFEE -> ERROR

// Running machine on a sequence of input symbols can be implemented recursively:
let rec run state input =
    match input with
    | [] -> state
    | symbol::more -> run (nextState state symbol) more

printfn "Running...\nFinished at state %A" (run S0 (KR5::KR10::COFFEE::[]))
```

Exercise RL1.3. Extension of the Vending Machine In the picture the main changes with respect to the original vending machine are highlighted in red.



Exercise RL1.4. 2x2 puzzle The following automaton models the entire puzzle. The states are the possible placement of the tiles (1, 2 and 3) and the hole (denoted with a blank space). The actions are up, down, left, right and correspond to moving a tile to the hole.

Starting in any state you can solve the puzzle by reaching the state (1,2,3,blank).

Note that that there are two disconnected parts in the automaton. The one above is the actual one that you can play regularly. The one below is one that you could obtain by physically extracting a tile from the board and replacing it by an adjacent one. In the new board obtained there is no way to solve the puzzle.