The Problem of the Dutch National Flag

Wouter Swierstra
Vector Fabrics

FP Dag 2010
There is a row of buckets numbered from 1 to n. It is given that:

- each bucket contains one pebble
- each pebble is either red, white, or blue.

A mini-computer is placed in front of this row of buckets and has to be programmed in such a way that it will rearrange (if necessary) the pebbles in the order of the Dutch national flag.

A Discipline of Programming, E.W. Dijkstra
Specification

- The mini-computer supports two commands:
  - swap \((i,j)\) exchanges the pebbles in buckets numbered \(i\) and \(j\) for \(1 \leq i,j \leq n\);
  - read \((i)\) returns the colour of the pebble in bucket number \(i\) for \(1 \leq i \leq n\).

- Solution should use one pass only and constant memory.
The Problem of the Dutch National Flag

Wouter Swierstra
AIM X
The Problem of the
Dutch National Flag

Indonesian

Wouter Swierstra
AIM X
Known to be white
Known to be white

Known to be red
Known to be white

Known to be red
Known to be white

Known to be red
Known to be white

Known to be red
Known to be white

Known to be red
Known to be white

Known to be red
Known to be white

Known to be red
Known to be white

Known to be red
Known to be white

Known to be red
Verified Solution

- Implement the mini-computer in the dependently typed language Agda;
- Write a total solution for the Problem of the Dutch National Flag;
- Formally prove our solution is correct.
Pebbles and Buckets

data Pebble : Set where
    Red : Pebble
    White : Pebble

data Buckets : Nat -> Set where
    Nil : Buckets Zero
    Cons : Pebble -> Buckets n -> Buckets (Succ n)
Indices

data Fin : Nat -> Set where
  Fz : Fin (Succ n)
  Fs : Fin n -> Fin (Succ n)
Indices

data Fin : Nat -> Set where
  Fz : Fin (Succ n)
  Fs : Fin n -> Fin (Succ n)
The state monad

State : Nat -> Set -> Set
State n a =
    Buckets n
    -> Pair a (Buckets n)
Reading

read : Fin n -> State n Pebble
read i bs = (bs ! i , bs)

where

(Cons p ps) ! Fz = p
(Cons p ps) ! (Fs i) = ps ! i
Swap

swap : Fin n -> Fin n
    -> State n Unit

swap i j =
    read i >>= \pi ->
    read j >>= \pj ->
    write i pj >>
    write j pi
Back to the problem
An approximation

sort :: Int -> Int -> IO ()

sort w r =
    if w == r then return ()
else case read w of
    White -> sort (w + 1) r
    Red   -> swap w r >>
            sort w (r - 1)
An approximation

Why does this terminate?

sort :: Int -> Int -> IO ()
sort w r =
  if w == r then return ()
  else case read w of
    White -> sort (w + 1) r
    Red  -> swap w r >>
            sort w (r - 1)
sort :: Int -> Int -> IO ()
sort r w =
  if r == w then return ()
  else case read r of
    White -> sort (w + 1) r
    Red -> swap r w >>
         sort w (r - 1)
An approximation

sort :: Int -> Int -> IO ()

Only terminates if $w \leq r$

sort r w =
  if r == w then return ()
  else case read r of
    White -> 
    Red ->  swap r w >>
             sort r (w - 1)
             sort (w + 1) r
    sort w (r - 1)
Manipulating Fin n

sort :: Int -> Int -> IO ()
sort r w =
  if r == w then return ()
  else case read r of
    White -> sort (w + 1) w
    Red   -> swap r w >>
            sort r (r - 1)
Two problems

• We need to increment and decrement inhabitants of \texttt{Fin n};

• We need to prove that our algorithm terminates.
Fs : Fin n -> Fin (Succ n)
Injection

\[ \text{inj} : \text{Fin } n \to \text{Fin } (\text{Succ } n) \]
\[ \text{inj } Fz = Fz \]
\[ \text{inj } (Fs \ i) = Fs \ (\text{inj } i) \]
Fs or inj

0 1 2 3

Fs

0 1 2 3

inj

0 1 2 3
Idea

• Only increment the image of $\text{inj}$;
• Only decrement the image of $\text{Fs}$.
data Diff : (i j : Fin n) → Set where
    Base : (i : Fin (Succ n) → Diff i i
    Step : (i j : Fin n) → Diff i j → Diff (inj i) (Fs j)
Sort – Base case

sort : (w r : Fin n) ->
    Diff w r ->
    State n Unit

sort i i Base = return unit
sort : (w r : Fin n) ->
  Diff w r ->
  State n Unit
sort (inj w) (Fs r) (Step w r p)
  = read (inj w) >>= \p ->
    case p of
    White -> sort (Fs w) (Fs r) ?
    Red ->
    swap (inj w) (Fs r) >>
    sort (inj w) (inj r) ?
Lemmas

- We need to prove a few useful lemmas:
  - \( \text{Diff } i \ j \rightarrow \text{Diff } (F\text{s } i) \ (F\text{s } j) \)
  - \( \text{Diff } i \ j \rightarrow \text{Diff } (\text{inj } i) \ (\text{inj } j) \)
Verification
Verification

the easy part
Correctness Theorem

forall (h : Buckets n) (w r : Fin n),
(p : Diff w r) ->
(forall i -> i < w -> h ! i == White) ->
(forall i -> r < i -> h ! i == Red) ->
exists (m : Fin n),
    let h’ = sort w r p h in
    forall i -> i < m -> h’ ! i == White
    && forall i -> i > m -> h’ ! i == Red)
Proof sketch

- Proof proceeds by induction on $\text{Diff}$
- Distinguish three cases:
  - Base case (trivial);
  - No swap happens (not too hard);
  - Swap happens (a bit trickier).
- In the latter two cases, we establish the invariant holds and make a recursive call.
Conclusions

- It is possible to reason about “impure” computations using Agda;
- A simple algorithm leads to simple proofs.