Preuve du splitter avec Why3

Aurélie Hurault

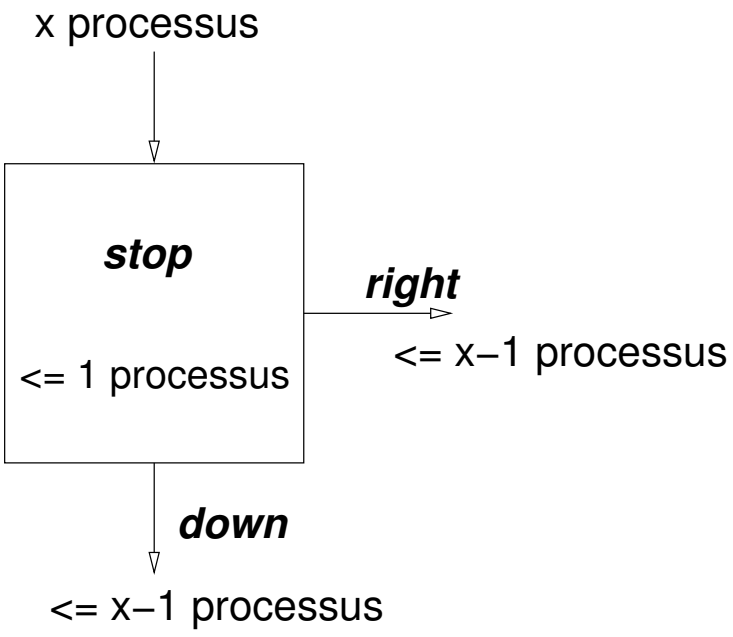
Réunion PARDI : 20 Décembre 2017

Why3

Why3 :

- est une plateforme pour la vérification de programme
- propose un langage riche de spécification (proche de Coq) et de programmation (proche d'OCaml) nommé WhyML ;
- s'appuie sur des prouveurs externes : assistants de preuve (Coq, Isabelle, ...), prouver automatiques (Alt-Ergo, Z3, CVC3, CVC4, ...) ;
- permet de décharger les différents buts sur différents prouveurs.

Le Splitter - Spécification



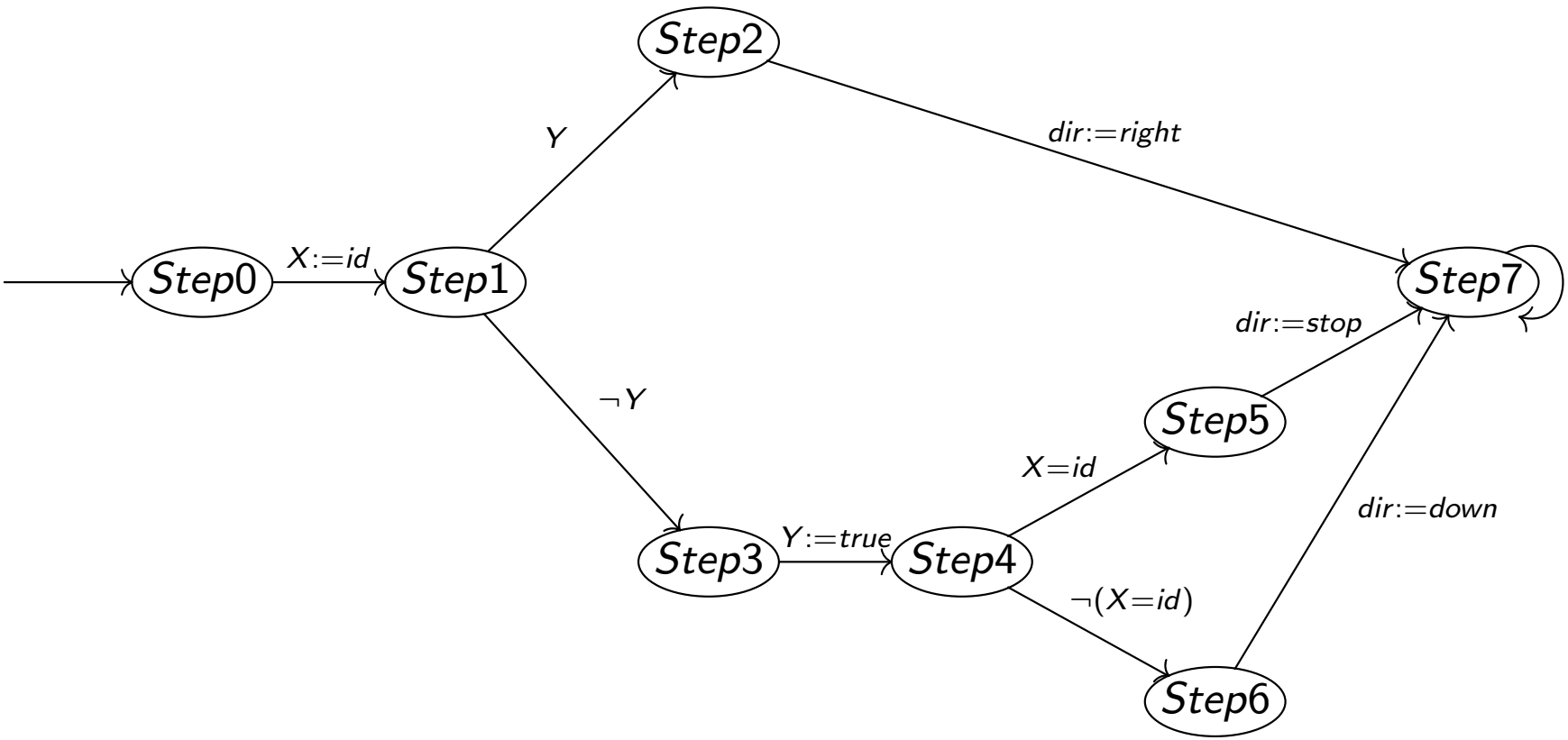
- x (indéterminé) activités appellent concurremment (ou pas) le splitter
- au plus une activité termine avec *stop*
- si $x = 1$, l'activité termine avec *stop*
- au plus $(x - 1)$ activités terminent avec *right*
- au plus $(x - 1)$ activités terminent avec *down*

Le Splitter - Algorithme

Deux registres partagés : X (initialisation sans importance) et Y (initialisé à faux)

```
direction(id)
  X := id
  if Y
  then dir := right
  else Y := true
       if (X=id)
       then dir := stop
       else dir := down
       endif
  endif
return dir
```

Le Splitter - Système de transition



Le Splitter en Why3 - Algorithme

```
type step = Step0 | Step1 | Step2 | Step3 | Step4 | Step5 | Step6
type direction = None | Right | Stop | Down

function pick int : int
axiom in_pick :  forall n :int. 0<= pick n < n

let splitter (n:int) (maxpas:int): array direction =
  let pc = Array.make n Step0 in
  let rval = Array.make n None in
  let x = ref 0 in
  let y = ref false in
  let nbpas = ref 0 in
  while (!nbpas < maxpas ) do
    (* On choisit de façon aléatoire un proc à faire avancer *)
    let proc = pick n in
    (* le proc fait un pas *)
    match pc[proc] with
    |Step0 -> x:= proc; pc[proc] <- Step1
    |Step1 -> if (!y) then pc[proc] <- Step2 else pc[proc] <- Step3
    |Step2 -> rval[proc] <- Right
    |Step3 -> y := true; pc[proc] <- Step4
    |Step4 -> if ((!x)=(proc)) then pc[proc] <- Step5 else pc[proc] <- Step6
    |Step5 -> rval[proc] <- Stop
    |Step6 -> rval[proc] <- Down
    end;
    nbpas := !nbpas +1;
  done;
  rval
end
```

Le Splitter en Why3 - Spécification

```
[...]

let splitter (n:int) (maxpas:int): array direction =
  requires {0<n}
  ensures {n=1
           -> (result[0]=None \/ result[0]=Stop)}
  ensures {forall i j: int. 0<=i<n /\  0<=j<n /\ result[i]=Stop  /\ result[j]=Stop
           -> i=j}
  ensures {(exists i : int. n>1 /\ 0<=i<n /\ result[i]=Right)
           -> (exists i : int. 0<=i<n /\ result[i]<>Right )}
  ensures {(exists i : int. n>1 /\ 0<=i<n /\ result[i]=Down)
           -> (exists i : int. 0<=i<n /\ result[i]<>Down )}

  let pc = Array.make n Step0 in
  [...]
  rval
end
```

Le Splitter en Why3 - Preuve de terminaison

```
[...]

let splitter (n:int) (maxpas:int): array direction =
  requires {0<n}
  ensures [...]

  let pc = Array.make n Step0 in
  [...]
  while (!nbpas < maxpas ) do
    variant {maxpas-(!nbpas)}
    [..]
  done;
  rval
end
```

Le Splitter en Why3 - Preuve de la première postcondition

```
[...]

let splitter (n:int) (maxpas:int): array direction =
  requires {0<n}
  ensures {n=1
           -> (result[0]=None \/ result[0]=Stop)}
  [...]

  let pc = Array.make n Step0 in
  [...]
  while (!nbpas < maxpas ) do
    invariant {(n=1 /\ pc[0]=Step0) -> (not (!y))}
    invariant {(n=1 /\ pc[0]=Step1) -> (not (!y))}
    invariant {n=1 -> (!x = 0)}
    invariant {n=1 ->
        pc[0]=Step0\/pc[0]=Step1\/pc[0]=Step3\/pc[0]=Step4\/pc[0]=Step5\/pc[0]=Step7}
    invariant {n=1 -> (rval[0]=None \/ rval[0]=Stop)}
    [..]
  done;
  rval
end
```

Le Splitter en Why3 - Preuve des autres postconditions

```
[...]
let splitter (n:int) (maxpas:int): array direction =
  requires {0<n}
  ensures [...]

  let pc = Array.make n Step0 in
  [...]
  while (!nbpas < maxpas ) do
    [..]
    (****** Généralités ******)
    invariant {forall i :int. 0<=i<n /\ pc[i]=Step0 -> rval[i]=None}
    invariant {forall i :int. 0<=i<n /\ pc[i]=Step1 -> rval[i]=None}
    invariant {forall i :int. 0<=i<n /\ pc[i]=Step2 -> rval[i]=None}
    invariant {forall i :int. 0<=i<n /\ pc[i]=Step3 -> rval[i]=None}
    invariant {forall i :int. 0<=i<n /\ pc[i]=Step4 -> rval[i]=None}
    invariant {forall i :int. 0<=i<n /\ pc[i]=Step5 -> rval[i]=None}
    invariant {forall i :int. 0<=i<n /\ pc[i]=Step6 -> rval[i]=None}
    invariant {forall i :int. 0<=i<n /\ rval[i]=Right -> pc[i]=Step7}
    invariant {forall i :int. 0<=i<n /\ rval[i]=Stop -> pc[i]=Step7}
    invariant {forall i :int. 0<=i<n /\ rval[i]=Down -> pc[i]=Step7}
    invariant {forall i :int. 0<=i<n /\ pc[i]=Step7 -> rval[i]<>None}

    invariant {0 <= (!x) <n}
    [..]
  done;
  rval
end
```

Le Splitter en Why3 - Preuve de la deuxième postcondition

```
[...]
let splitter (n:int) (maxpas:int): array direction =
 requires {0<n}
 ensures {forall i j: int. 0<=i<n /\  0<=j<n /\ result[i]=Stop  /\ result[j]=Stop
          -> i=j}
 [...]
 let pc = Array.make n Step0 in
 [...]
 while (!nbpas < maxpas ) do
  [..]
  invariant {(exists i : int. 0<=i<n /\ pc[i]=Step4) -> (!y)}
  invariant {(exists i : int. 0<=i<n /\ pc[i]=Step5) -> (!y)}
  invariant {(exists i : int. 0<=i<n /\ pc[i]=Step7 /\ rval[i]=Stop) -> (!y)}
  invariant {(exists i : int. 0<=i<n /\ rval[i]=Stop) -> (!y)}
  invariant {forall i : int. 0<=i<n /\ pc[i]=Step5
     -> (!x=i \/ pc[!x]=Step1 \/ pc[!x]=Step2 \/ (pc[!x]=Step7 /\ rval[!x]=Right))}
  invariant {forall i : int. 0<=i<n /\ pc[i]=Step7 /\ rval[i]=Stop
     -> (!x=i \/ pc[!x]=Step1 \/ pc[!x]=Step2 \/ (pc[!x]=Step7 /\ rval[!x]=Right))}
  invariant {forall i j: int. 0<=i<n /\  0<=j<n /\ pc[i]=Step5  /\ pc[j]=Step5 -> i=j}
  invariant {not(exists i j: int. 0<=i<n/\0<=j<n/\pc[i]=Step5/\pc[j]=Step7/\rval[j]=Stop)}
  invariant {forall i j: int. 0<=i<n/\0<=j<n/\pc[i]=Step7/\rval[i]=Stop
     /\pc[j]=Step7/\rval[j]=Stop->i=j}
  invariant {forall i j: int. 0<=i<n/\0<=j<n/\rval[i]=Stop/\rval[j]=Stop -> i=j}
  [..]
 done;
 rval
end
```

Le Splitter en Why3 - Preuve de la troisième postcondition

```
[...]

let splitter (n:int) (maxpas:int): array direction =
  requires {0<n}
  ensures {(exists i : int. n>1 /\ 0<=i<n /\ result[i]=Right)
           -> (exists i : int. 0<=i<n /\ result[i]<>Right )}
  [...]

  let pc = Array.make n Step0 in
  [...]
  while (!nbpas < maxpas ) do
    [..]
    invariant {(exists i : int. 0<=i<n /\ pc[i]=Step2) -> (!y)}
    invariant {(exists i : int. 0<=i<n /\ rval[i]=Right ) -> (!y)}
    invariant {(!y) -> (   (exists i : int. 0<=i<n /\ pc[i]=Step4 /\ rval[i]=None)
                        \/ (exists i : int. 0<=i<n /\ pc[i]=Step5 /\ rval[i]=None)
                        \/ (exists i : int. 0<=i<n /\ pc[i]=Step6 /\ rval[i]=None)
                        \/ (exists i : int. 0<=i<n /\ pc[i]=Step7 /\ rval[i]=Down)
                        \/ (exists i : int. 0<=i<n /\ pc[i]=Step7 /\ rval[i]=Stop))}
    [..]
  done;
  rval
end
```

Le Splitter en Why3 - Preuve de la quatrième postcondition

```
[...]

let splitter (n:int) (maxpas:int): array direction =
  requires {0<n}
  ensures {(exists i : int. n>1 /\ 0<=i<n /\ result[i]=Down)
           -> (exists i : int. 0<=i<n /\ result[i]<>Down )}
  [...]

  let pc = Array.make n Step0 in
  [...]
  while (!nbpas < maxpas ) do
    [..]
    invariant {   (pc[!x]=Step0 /\ rval[!x]=None)
              \/ (pc[!x]=Step1 /\ rval[!x]=None)
              \/ (pc[!x]=Step2 /\ rval[!x]=None)
              \/ (pc[!x]=Step3 /\ rval[!x]=None)
              \/ (pc[!x]=Step4 /\ rval[!x]=None)
              \/ (pc[!x]=Step5 /\ rval[!x]=None)
              \/ (pc[!x]=Step7 /\ rval[!x]=Right)
              \/ (pc[!x]=Step7 /\ rval[!x]=Stop)}
    [..]
  done;
  rval
end
```

Le Splitter en Why3 - Démonstration

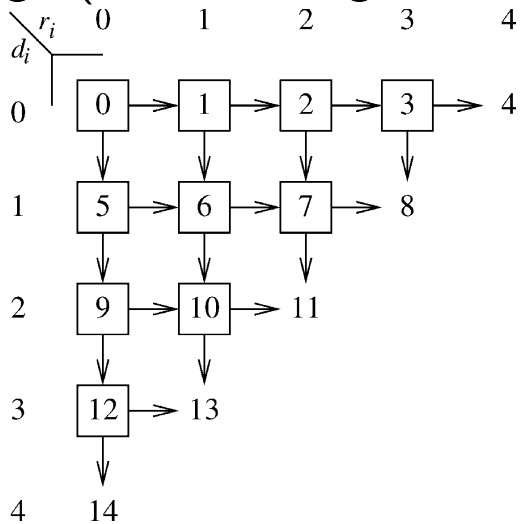
Démonstration

Conclusion et Perspective

Conclusion

- Pas si compliqué que ça...
- Pas eu besoin de variables "histoires" (fantôme)

Perspective : le renommage (utilise une grille de splitter)



- Beaucoup plus compliqué
- Les invariants de boucles sont compliqués à écrire car on aurait envie d'utiliser de la logique temporelle
- Est-ce que Why3 est le bon outil?