

- ① Invariant versus theorem in Event-B
- ② Designing a C program for $\lambda x.x \times x$

- ① Invariant versus theorem in Event-B
- ② Designing a C program for $\lambda x.x \times x$

$x \leq 0$ is always true !

```
VARIABLES  x
INVARIANTS
  @inv1 x ∈ ℤ
  @theproperty  $x \leq 0$ 
EVENT  INITIALISATION
  then
    @act1 x := - 1
  end
EVENT  event1
  where
    @grd1  $x \geq 0$ 
  then
    @act1 x := x + 1
  end
EVENT  event2
  then
    @act1 x := x
  end
```

- $x \leq 0$ is always true.
- $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$ is not true !

$x \leq 0$ is always true !

```
VARIABLES  x
INVARIANTS
  @inv1 x ∈ ℤ
  @theproperty  $x \leq 0$ 
EVENT  INITIALISATION
  then
    @act1 x := - 1
  end
EVENT  event1
  where
    @grd1  $x \geq 0$ 
  then
    @act1 x := x + 1
  end
EVENT  event2
  then
    @act1 x := x
  end
```

- $x \leq 0$ is always true.
- $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$ is not true !
 - ▶ $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$

$x \leq 0$ is always true !

```
VARIABLES  x
INVARIANTS
  @inv1 x ∈ ℤ
  @theproperty  $x \leq 0$ 
EVENT  INITIALISATION
  then
    @act1 x := - 1
  end
EVENT  event1
  where
    @grd1  $x \geq 0$ 
  then
    @act1 x := x + 1
  end
EVENT  event2
  then
    @act1 x := x
  end
```

- $x \leq 0$ is always true.
- $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$ is not true !
 - ▶ $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $\vdash x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$

$x \leq 0$ is always true !

```
VARIABLES  x
INVARIANTS
  @inv1 x ∈ ℤ
  @theproperty  $x \leq 0$ 
EVENT  INITIALISATION
  then
    @act1 x := - 1
  end
EVENT  event1
  where
    @grd1  $x \geq 0$ 
  then
    @act1 x := x + 1
  end
EVENT  event2
  then
    @act1 x := x
  end
```

- $x \leq 0$ is always true.
- $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$ is not true !
 - ▶ $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $\vdash x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $x \leq 0 \wedge x \geq 0 \wedge x' = x + 1 \vdash x' \leq 0$

$x \leq 0$ is always true !

```
VARIABLES  x
INVARIANTS
  @inv1 x ∈ ℤ
  @theproperty x ≤ 0
EVENT  INITIALISATION
  then
    @act1 x := - 1
  end
EVENT  event1
  where
    @grd1 x ≥ 0
  then
    @act1 x := x + 1
  end
EVENT  event2
  then
    @act1 x := x
  end
```

- $x \leq 0$ is always true.
- $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$ is not true !
 - ▶ $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $\vdash x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $x \leq 0 \wedge x \geq 0 \wedge x' = x + 1 \vdash x' \leq 0$
 - ▶ $x \leq 0 \wedge x \geq 0 \wedge x' = x + 1 \Rightarrow x' \leq 0$

$x \leq 0$ is always true !

```
VARIABLES  x
INVARIANTS
  @inv1 x ∈ ℤ
  @theproperty x ≤ 0
EVENT  INITIALISATION
  then
    @act1 x := - 1
  end
EVENT  event1
  where
    @grd1 x ≥ 0
  then
    @act1 x := x + 1
  end
EVENT  event2
  then
    @act1 x := x
  end
```

- $x \leq 0$ is always true.
- $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$ is not true !
 - ▶ $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $\vdash x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $x \leq 0 \wedge x \geq 0 \wedge x' = x + 1 \vdash x' \leq 0$
 - ▶ $x \leq 0 \wedge x \geq 0 \wedge x' = x + 1 \Rightarrow x' \leq 0$
 - ▶ $x \leq 0, x \geq 0, x' = x + 1 \vdash x' \leq 0$

$x \leq 0$ is always true !

```
VARIABLES  x
INVARIANTS
  @inv1 x ∈ ℤ
  @theproperty x ≤ 0
EVENT  INITIALISATION
  then
    @act1 x := - 1
  end
EVENT  event1
  where
    @grd1 x ≥ 0
  then
    @act1 x := x + 1
  end
EVENT  event2
  then
    @act1 x := x
  end
```

- $x \leq 0$ is always true.
- $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$ is not true !
 - ▶ $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $\vdash x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $x \leq 0 \wedge x \geq 0 \wedge x' = x + 1 \vdash x' \leq 0$
 - ▶ $x \leq 0 \wedge x \geq 0 \wedge x' = x + 1 \Rightarrow x' \leq 0$
 - ▶ $x \leq 0, x \geq 0, x' = x + 1 \vdash x' \leq 0$
 - ▶ $x \leq 0, x \geq 0, x' = x + 1 \vdash x + 1 \leq 0$

$x \leq 0$ is always true !

```
VARIABLES  x
INVARIANTS
  @inv1 x ∈ ℤ
  @theproperty x ≤ 0
EVENT  INITIALISATION
  then
    @act1 x := - 1
  end
EVENT  event1
  where
    @grd1 x ≥ 0
  then
    @act1 x := x + 1
  end
EVENT  event2
  then
    @act1 x := x
  end
```

- $x \leq 0$ is always true.
- $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$ is not true !
 - ▶ $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $\vdash x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $x \leq 0 \wedge x \geq 0 \wedge x' = x + 1 \vdash x' \leq 0$
 - ▶ $x \leq 0 \wedge x \geq 0 \wedge x' = x + 1 \Rightarrow x' \leq 0$
 - ▶ $x \leq 0, x \geq 0, x' = x + 1 \vdash x' \leq 0$
 - ▶ $x \leq 0, x \geq 0, x' = x + 1 \vdash x + 1 \leq 0$
 - ▶ $x \leq 0, x \geq 0, x = 0, x' = x + 1 \vdash x + 1 \leq 0$

$x \leq 0$ is always true !

```
VARIABLES  x
INVARIANTS
  @inv1 x ∈ ℤ
  @theproperty x ≤ 0
EVENT  INITIALISATION
  then
    @act1 x := - 1
  end
EVENT  event1
  where
    @grd1 x ≥ 0
  then
    @act1 x := x + 1
  end
EVENT  event2
  then
    @act1 x := x
  end
```

- $x \leq 0$ is always true.
- $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$ is not true !
 - ▶ $x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $\vdash x \leq 0 \wedge BA(event1)(x, x') \Rightarrow x' \leq 0$
 - ▶ $x \leq 0 \wedge x \geq 0 \wedge x' = x + 1 \vdash x' \leq 0$
 - ▶ $x \leq 0 \wedge x \geq 0 \wedge x' = x + 1 \Rightarrow x' \leq 0$
 - ▶ $x \leq 0, x \geq 0, x' = x + 1 \vdash x' \leq 0$
 - ▶ $x \leq 0, x \geq 0, x' = x + 1 \vdash x + 1 \leq 0$
 - ▶ $x \leq 0, x \geq 0, x = 0, x' = x + 1 \vdash x + 1 \leq 0$
 - ▶ $x \leq 0, x \geq 0, x = 0, x' = x + 1 \vdash 1 \leq 0!$

$x \leq 0$ is always true !

```
VARIABLES  x
INVARIANTS
  @inv1 x ∈ ℤ
  @inv2 x = -1
theorem @safety1 x ≤ 0
EVENT  INITIALISATION
  then
    @act1 x := -1
  end
EVENT  event1
  where
    @grd1 x ≥ 0
  then
    @act1 x := x + 1
  end
EVENT  event2
  then
    @act1 x := x
end
```

- $x \leq 0$ is always true.
- $x = -1 \wedge BA(event1)(x, x') \Rightarrow x' = -1$ is correct
- $x \leq 0$ is a theorem
- $x = -1$ is an inductive invariant.

Current Summary

- ② Designing a C program for $\lambda x.x \times x$

Designing a C program for $\lambda x.x \times x$

Computing $\lambda x.x \times x$ with only addition

The problem is to derive a C program which is computing the function $\lambda x.x \times x$ using only addition.

```
#ifndef _A_H
#define _A_H
#include <limits.h>
/*@ axiom rule1: \forall int n; n > 0 ==> n*n == (n-1)*(n-1)+2*n-1;
    @ */

/*@ requires 0 <= x;
    requires x <= INT_MAX;
    requires x*x <= INT_MAX;
    assigns \nothing;
    ensures \result == x*x;
 */
int power2(int x);
#endif
```

Context for computing $\lambda x.x \times x$

```
CONTEXT power20
CONSTANTS n0 v w s
AXIOMS
  @axm1 n0 ∈ ℕ // precondition
  @axm2 w ∈ ℕ → ℤ
  @axm3 w(0) = 0
  @axm4 ∀ n. n ∈ ℕ ⇒ w(n + 1) = w(n) + 2
  @axm5 v ∈ ℕ → ℤ
  @axm6 v(0) = 0
  @axm7 ∀ n. n ∈ ℕ ⇒ v(n + 1) = v(n) + w(n) + 1
  @axm8 s ∈ ℕ → ℕ ∧ (∀ i. i ∈ ℕ ⇒ s(i) = i + 1)
  @axm9 ∀ A. A ⊆ ℕ ∧ 0 ∈ A ∧ s[A] ⊆ A ⇒ ℕ ⊆ A
  theorem @axm10 ∀ n. n ∈ ℕ ⇒ w(n) = 2 * n
  theorem @axm11 ∀ n. n ∈ ℕ ⇒ v(n) = n * n
  @axm12 n0 ≥ 3
end
```


Machine power21 for stating the pre/post specification

```

MACHINE  power21  SEES  power20

VARIABLES  r ok n
INVARIANTS
  @inv1 r ∈ ℤ
  @inv2 n ∈ ℤ
  @inv3 ok ∈ BOOL
  @inv4 ok = TRUE ⇒ r = n0 × n0
  @inv5 n = n0

```

- Defining variables and invariant
- r is the variable for the result.
- ok is the boolean variable used for expressing the process termination.
- n is the variable containing the input of the process.

Machine power21 for stating the pre/post specification

```

EVENT INITIALISATION
  then
    @act1  $r : \in \mathbb{Z}$ 
    @act2  $ok := FALSE$ 
    @act3  $n := n0$ 
  end
EVENT final
  where
    @grd1  $ok = FALSE$ 
  then
    @act1  $r := v(n)$ 
    @act2  $ok := TRUE$ 
  end

```

- INITIALISATION is setting variables especially $n = n_0$
- final is observed and gets the value $v(n)$ which is sound since $v(n) = n \times n$.
- ok* controls the observation of the event final at most one time.
- iteration is anticipating an hidden iteration.

Machine power21 for stating the pre/post specification

```

EVENT INITIALISATION
  then
    @act1  $r : \in \mathbb{Z}$ 
    @act2  $ok := FALSE$ 
    @act3  $n := n0$ 
  end
EVENT final
  where
    @grd1  $ok = FALSE$ 
  then
    @act1  $r := v(n)$ 
    @act2  $ok := TRUE$ 
  end
end

```

- INITIALISATION is setting variables especially $n = n_0$
- final is observed and gets the value $v(n)$ which is sound since $v(n) = n \times n$.
- *ok* controls the observation of the event final at most one time.
- iteration is anticipating an hidden iteration.

```

anticipated EVENT iteration
  then
    @act1  $r, ok, n \mid (r' \in \mathbb{Z} \wedge (ok' = TRUE \Rightarrow r' = n0 * n0) \wedge n' = n0)$ 
  end

```

Machine power22 for stating the computing process

```
MACHINE power22 REFINES power21 SEES power20
VARIABLES r vv k ww ok n
INVARIANTS
@inv1 vv ∈ ℕ ↔ ℤ
@inv2 ww ∈ ℕ ↔ ℤ
@inv3 k ∈ ℕ
@inv4 ∀ i. i ∈ dom(vv) ⇒ vv(i) = v(i)
@inv5 ∀ i. i ∈ dom(ww) ⇒ ww(i) = w(i)
@inv6 dom(vv) = 0..k
@inv7 dom(ww) = 0..k
@inv8 k ≤ n
theorem @safe1 ∀ i. i ∈ dom(vv) ⇒ vv(i) = i * i
theorem @safe2 ∀ i. i ∈ dom(ww) ⇒ ww(i) = 2 * i
@inv11 k < n ⇒ ok = FALSE
```

- Two new variables vv and ww are introduced for storing the two sequences v and w by iterating over k
- Condition of termination is that $n \in \text{dom}(vv)$
- $vv(i) = v(i)$ and $ww(i) = w(i)$ are expressing the relationship between computed values and mathematically defined values of the two sequences.

Machine power²² for stating the computing process

```

EVENT INITIALISATION
  then
    @act1  $r : \in \mathbb{Z}$ 
    @act2  $vv := \{0 \mapsto 0\}$ 
    @act3  $ww := \{0 \mapsto 0\}$ 
    @act4  $k := 0$ 
    @act5  $ok := FALSE$ 
    @act6  $n := n0$ 
  end

```

- INITIALISATION is setting variables especially ww and vv
- Sequences v and w are used for intialisation.

Machine power²² for stating the computing process

```

EVENT final  REFINES final
  where
    @grd1  $n \in \text{dom}(vv)$ 
    @grd2  $ok = \text{FALSE}$ 
    then
      @act1  $r := vv(n)$ 
      @act2  $ok := \text{TRUE}$ 
    end
end

convergent EVENT step – computing
REFINES iteration
  where
    @grd1  $n \notin \text{dom}(vv)$ 
    @grd2  $ok = \text{FALSE}$ 
    then
      @act1  $vv(k+1) := vv(k) + ww(k) + 1$ 
      @act2  $k := k + 1$ 
      @act3  $ww(k+1) := ww(k) + 2$ 
    end
end

VARIANT  $n - k$ 

```

- the event final is controlled by the condition $n \in \text{dom}(vv)$ meaning that we have finally reached the computing goal.
- SIM proof obligations are generated.
- the event step-computing is refining iteration and when it observed, the variant $n - k$ is decreasing.
- it refines iteration

Machine power²³ for getting an algorithmic process

```

MACHINE power23
  REFINES power22
  SEES power20

VARIABLES  r vv k cv ww cw ok n

INVARIANTS
  @inv1 cv ∈ ℤ
  @inv2 cv = vv(k)
  @inv3 cw ∈ ℤ
  @inv4 cw = ww(k)
  theorem @inv5 k ∈ 0..n
  theorem @inv6 cw = 2 * k
  theorem @inv7 cv = k * k
  theorem @inv8 4 * cv = cw * cw

```

- Two new variables are introduced for storing really useful data namely the last computed values of the two sequences.
- Obviously, $cv = vv(k)$ and $cw = ww(k)$
- Previous properties of abstract variables are safety properties which are no more to be reproved, thanks to refinement.
- We can get extra properties that are relating the variables as $4 \times cv = cw \times cw$.

Machine power23 for getting an algorithmic process

```
EVENT final REFINES final
  where
    @grd1  $k = n$ 
    then
      @act1  $r := cv$ 
      @act2  $ok := TRUE$ 
    end
  end

convergent EVENT step - prealgo
  REFINES step - computing
    where
      @grd1  $k < n$ 
      then
        @act1  $vv(k+1) := vv(k) + ww(k) + 1$ 
        @act2  $k := k + 1$ 
        @act3  $cv := cv + cw + 1$ 
        @act4  $ww(k+1) := ww(k) + 2$ 
        @act5  $cw := cw + 2$ 
      end
    end

  VARIANT  $n - k$ 
```

- The two events SIMulate the abstrcat events.
- However, the guards are strengthened and are made closer to an implmentation :
 $k < n$ implies $n \notin dom(vv)$ and
 $k = n$ implies that
 $n \in dom(vv)$.

Machine power²⁴ for getting an algorithmic machine

MACHINE *power24* REFINES *power23*
SEES *power20*

VARIABLES r k cv cw ok n

INVARIANTS

$$\textit{theorem} \ @th1 \ cw = 2 * k$$
$$\text{theorem @th2 } cv = k * k$$
$$\text{theorem @inv1th3 } 4 * cv = cw * cw$$

- The two variables vv and ww are now hidden and they disappear from the machine.
- They are playing the role of model variables as ghost variables.
- Invariants and safety properties are preserved through refinement.

Machine power²⁴ for getting an algorithmic machine

```

EVENT INITIALISATION
  then
    @act1  $r \in \mathbb{Z}$ 
    @act5  $k := 0$ 
    @act8  $cv := 0$ 
    @act10  $cw := 0$ 
    @act11  $ok := FALSE$ 
    @act12  $n := n0$ 
  end

```

- INITILISATION is the same event without vv and ww .

Machine power²⁴ for getting an algorithmic machine

```

EVENT final REFINES final
  where
    @grd1  $k = n$ 
    then
      @act1  $r := cv$ 
      @act2  $ok := TRUE$ 
    end
  end

convergent EVENT step
REFINES step - prealgo
  where
    @grd1  $k < n$ 
    then
      @act4  $k := k + 1$ 
      @act5  $cv := cv + cw + 1$ 
      @act7  $cw := cw + 2$ 
    end
  end

```

- Assignments of vv and ww are removed.

Translating the machine power24 to an algorithm

```

begin
  int  $r, k$  := 0,  $cv$  := 0,  $cw$  := 0,  $ok$  := FALSE,  $n$  :=  $n_0$ ;
  while  $k < n$  {
    ( $k, cv, cw$ ) := ( $k + 1, cv + cw + 1, cw := cw + 2$ );
  };
   $r := cv$ ;
   $ok := TRUE$ 
end

```

Translating the machine power24 to an algorithm

```
#include <limits.h>
#include "ppower2.h"

int power2(int x)
{
    int r, k, cv, cw, or, ok, ocv, ocw;
    r=0; k=0; cv=0; cw=0; or=0; ok=k; ocv=cv; ocw=cw;
    /*@ loop invariant cv == k*k;
       @ loop invariant k <= x;
       @ loop invariant cw == 2*k;
       @ loop invariant 4*cv == cw*cw;
       @ loop assigns k, cv, cw, or, ok, ocv, ocw;
       @ loop variant x-k;
    */
    while (k<x)
    {
        ok=k; ocv=cv; ocw=cw;
        k=ok+1;
        cv=ocv+ocw+1;
        cw=ocw+2;
    }
    r=cv;
    return (r);
}
```