Modelling Software-based Systems

Tutorial

Telecom Nancydocu

Dominique Méry
Telecom Nancy,Université de Lorraine

27 mars 2025(5:59 P.M.)

dominique.mery@loria.fr

General Summary

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

Current Summary

1. 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.

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

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

CONTEXT $power20$
CONSTANTS $n0\ v\ w\ s$
AXIOMS
$@axm1\ n0 \in \mathbb{N}$ // $precondition$
$@axm2\ w \in \mathbb{N} \rightarrow \mathbb{Z}$
$@axm3\ w(0) = 0$
$@axm4\ \forall n.\ n \in \mathbb{N} \Rightarrow w(n+1) = w(n) + 2$
$@axm5\ v \in \mathbb{N} \rightarrow \mathbb{Z}$
$@axm6\ v(0) = 0$
$@axm7\ \forall n.\ n \in \mathbb{N} \Rightarrow v(n+1) = v(n) + w(n) + 1$
$@axm8\ s \in \mathbb{N} \rightarrow \mathbb{N} \wedge (\forall i.\ i \in \mathbb{N} \Rightarrow s(i) = i + 1)$
$@axm9\ \forall A.\ A \subseteq \mathbb{N} \wedge 0 \in A \wedge s[A] \subseteq A \Rightarrow \mathbb{N} \subseteq A$
$theorem\ @axm10\ \forall n.n \in \mathbb{N} \Rightarrow w(n) = 2 * n$
$theorem\ @axm11\ \forall n.n \in \mathbb{N} \Rightarrow v(n) = n * n$
$@axm12\ n0 \geq 3$
end

Machine power21 for stating the pre/post specification

MACHINE $power21$ SEES $power20$

VARIABLES $r\ ok\ n$
INVARIANTS
$@inv1\ r \in \mathbb{Z}$
$@inv2\ n \in \mathbb{Z}$
$@inv3\ ok \in BOOL$
$@inv4\ ok = TRUE \Rightarrow r = n0 \times n0$
$@inv5\ n = n0$

- 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 :∈ 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 $n = n0$
- final is observed and gets the value $v(n)$ which is sound since $v(n) = n \times n$.
- ok controls the obsedrvation of the event final at most one time.

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 $n = n0$
- final is observed and gets the value $v(n)$ which is sound since $v(n) = n \times n$.
- ok controls the obsedrvation 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

- INITIALISATION is setting variables especially $n = n0$
- final is observed and gets the value $v(n)$ which is sound since $v(n) = n \times n$.
- ok controls the obsedrvation of the event final at most one time.
- iteration is anticipating an hidden iteration.

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

Machine power22 for stating the computing process

MACHINE $power22$ REFINES $power21$ SEES $power20$
VARIABLES $r\ vv\ k\ ww\ ok\ n$
INVARIANTS
$@inv1\ vv \in \mathbb{N} \pfun \mathbb{Z}$
$@inv2\ ww \in \mathbb{N} \pfun \mathbb{Z}$
$@inv3\ k \in \mathbb{N}$
$@inv4\ \forall i.\ i \in dom(vv) \Rightarrow vv(i) = v(i)$
$@inv5\ \forall i.\ i \in dom(ww) \Rightarrow ww(i) = w(i)$
$@inv6\ dom(vv) = 0..k$
$@inv7\ dom(ww) = 0..k$
$@inv8\ k \leq n$
$theorem\ @safe1\ \forall i.\ i \in dom(vv) \Rightarrow vv(i) = i * i$
$theorem\ @safe2\ \forall i.\ i \in dom(ww) \Rightarrow ww(i) = 2 * i$
$@inv11\ k < n \Rightarrow ok = FALSE$

- 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 $n \in dom(vv)$
- $vv(i) = v(i)$ and $ww(i) = w(i)$ are expressing the relationship between computed values and mathematically defned 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
$@grd1\ n \in dom(vv)$
$@grd2\ ok = FALSE$
then
$@act1\ r := vv(n)$
$@act2\ ok := TRUE$
end

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

VARIANT $n-k$

- the event final is controled by the condition $n \in dom(vv)$ meaning that we have finally reached the computing goal.
- SIM proof obligations are generated.
- the event step-computng 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
$@inv1\ cv \in \mathbb{Z}$
$@inv2\ cv = vv(k)$
$@inv3\ cw \in \mathbb{Z}$
$@inv4\ cw = ww(k)$
$theorem\ @inv5\ k \in 0..n$
$theorem\ @inv6\ cw = 2*k$
$theorem\ @inv7\ cv = k*k$
$theorem\ @inv8\ 4*cv = cw*cw$

- Two new variables are introduced for storing really useful data namey the last computed values of the two sequences.
- Obviously, $cv = vv(k)$ and $cw = ww(k)$
- Previous properties of abstrcat variables are safety properties which are no more to be reproved, thanks to refienement.
- 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 $INITIALISATION$
then
$@act1\ r :\in \mathbb{Z}$
$@act2\ vv := \{0 \mapsto 0\}$
$@act3\ k := 0$
$@act4\ cv := 0$
$@act5\ ww := \{0 \mapsto 0\}$
$@act6\ cw := 0$
$@act7\ ok := FALSE$
$@act8\ n := n0$
end

- Initialisation of new variables according to the invariant.

Machine power23 for getting an algorithmic process

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

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
$theorem\ @th1\ cw = 2*k$
$theorem\ @th2\ cv = k*k$
$theorem\ @inv1th3\ 4*cv = cw*cw$

- 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

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

- Assignments of vv and ww are removed.

Translating the machine power24 to an algorithm

$begin$
$int\ r, k\ := 0,\ cv\ := 0,\ cw\ := 0,\ ok\ := FALSE, n\ := n0;$
$while\ k\ < n\ \{$
$(\ k, cv, cv)\ := (k+1, cv+cw+1, cw\ := cw+2);$
$\}\ ;$
$r\ := cv;$
$ok\ := TRUE$
end

Translating the machine power24 to an algorithm

```
#include <limits.h>
#include "power2.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 0 <= cv && 0 <= cw && 0 <= k;
        @ 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; */
  while (k<x)
    {
    ok=k;ocv=cv;ocw=cw;
      k=ok+1;
      cv=ocv+ocw+1;
      cw=ocw+2;

    }
  r=cv;
  return(r);
}
```