Modelling Software-based Systems

Lecture 4 Correctness by Construction with the Modelling Language Event-B using the Refinement

Telecom Nancydocu

Dominique Méry
Telecom Nancy,Université de Lorraine

2 mai 2025(10:18 A.M.)

dominique.mery@loria.fr

General Summary

Current Summary

Correctness by Construction

- Correctness by Construction is a method of building software -based systems with **demonstrable correctness** for security- and safety-critical applications.
- Correctness by Construction advocates a **step-wise refinement** process from specification to code using tools for checking and transforming models.
- Correctness by Construction is an approach to software/system construction
  - starting with an abstract model of the problem.
  - progressively adding details in a step-wise and checked fashion.
  - each step guarantees and proves the correctness of the new concrete model with respect to requirements

The Cleanroom Method as CbC

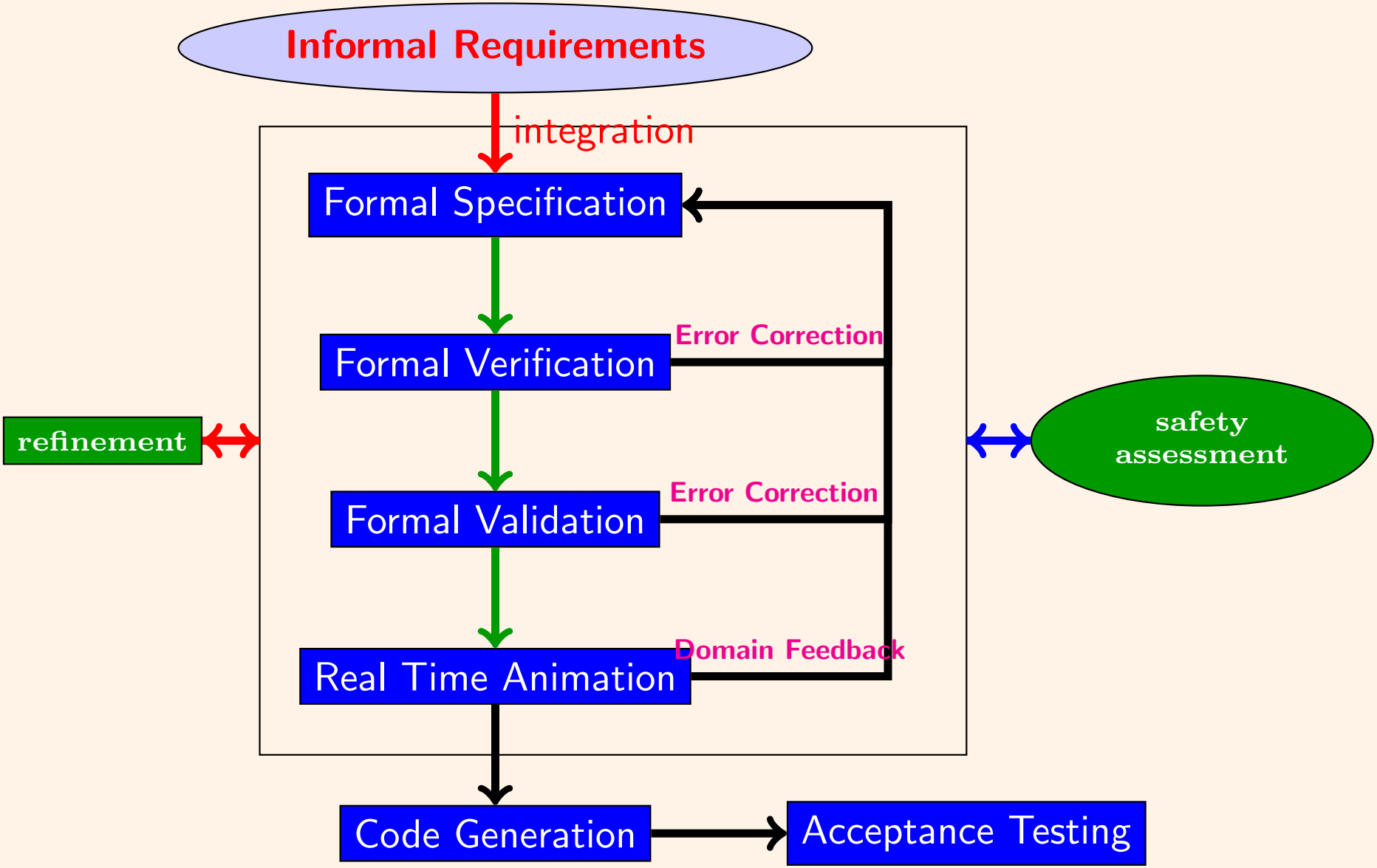
- The **Cleanroom** method, developed by Harlan Mills and his colleagues at IBM and elsewhere, attempts to do for software what cleanroom fabrication does for semiconductors : to achieve quality by keeping defects out during fabrication.
- In semiconductors, **dirt** or **dust** that is allowed to **contaminate** a chip as it is being made cannot possibly be removed later.
- But we try to do the equivalent when we write programs that are full of bugs, and then attempt to remove them all using debugging.

The Cleanroom Method as CbC

The Cleanroom method, then, uses a number of techniques to develop software carefully, in a well-controlled way, so as to avoid or eliminate as many defects as possible before the software is ever executed. Elements of the method are :

- specification of all components of the software at all levels ;
- stepwise refinement using constructs called "box structures" ;
- verification of all components by the development team ;
- statistical quality control by independent certification testing ;
- no unit testing, no execution at all prior to certification testing.

Critical System Development Life-Cycle Methodology



Current Summary

Problems for Modelling systems

- Systems are generally very complex
- Invariant should be strong enough for proving safety properties
- Problems for modelling : finding suitable mathematical structures, listing events or actions of the system, proving proof obligations, ...

Solution : refining models

- To understand more and more the system
- To distribute the complexity of the system
- To distribute the difficulties of the proof
- To improve explanations
- Validation (step by step)
- Refinement (invariant & behavior)

refinement between two events (I)

definition

Let x be the abstract variable (or list of variables) and $I(s, c, x)$ the abstract invariant, y the concrete variable (or list of variables) and $J(s, c, x, y)$ the concrete invariant.
Let c be a concrete event observing the variable y and a an event observing the variable x and preserving $I(s, c, x)$.
Event c refines event a with respect to x, $I(s, c, x)$, y and $J(s, c, x, y)$, if

$$AX(s, c) \vdash I(s, c, x) \wedge J(s, c, x, y) \Rightarrow [c](\neg[a](\neg J(s, c, x, y)))$$

Abstract event refined by a concret event

$$a \left\{ \begin{array}{l} \textbf{ANY } u \textbf{ WHERE} \\ \quad G(u,s,c,x) \\ \textbf{THEN} \\ \quad x :\mid ABAP(u,s,c,x,x') \\ \textbf{END} \end{array} \right. \overset{def}{=} c \left\{ \begin{array}{l} \textbf{ANY } v \textbf{ WHERE} \\ \quad H(v,s,c,y) \\ \textbf{WITNESS} \\ \quad u : WP(u,s,c,v,y) \\ \quad x' : WV(v,s,c,y',x') \\ \textbf{THEN} \\ \quad y :\mid CBAP(v,s,c,y,y') \\ \textbf{END} \end{array} \right. \overset{def}{=}$$

The two events a and c are normalised by a relationship called BA(e)(s,c,x,x'), which simplifies the notations used.
The two events a and c are equivalent to events of the following normalized form :

- a is equivalent to
  begin $x :\mid (\exists u.G(u,s,c,x) \wedge ABAP(u,s,c,x,x'))$ end
- c is equivalent to
  begin $y :\mid (\exists v.H(v,s,c,y) \wedge CBAP(v,s,c,y,y'))$ end

Explanations for the refinement

(Hypothesis)
(1) $AX(s,c) \vdash I(s,c,x) \wedge J(s,c,x,y) \Rightarrow [c](\neg[a](\neg J(s,c,x,y)))$
equivalent to
(Definition of $[a]$: $[a](\neg J(s,c,x,y)) \equiv$
$\forall x'.(\exists u.G(u,s,c,x) \wedge ABAP(u,s,c,x,x')) \Rightarrow \neg J(s,c,x',y)))$
(2) $AX(s,c) \vdash I(s,c,x) \wedge J(s,c,x,y) \Rightarrow [c](\neg(\forall x'.(\exists u.G(u,s,c,x) \wedge$
$ABAP(u,s,c,x,x')) \Rightarrow \neg J(s,c,x',y))$
equivalent to
(Transformation by simplification of logical connectives)
(3) $AX(s,c) \vdash I(s,c,x) \wedge J(s,c,x,y) \Rightarrow [c](\exists x'.(\exists u.G(u,s,c,x) \wedge$
$ABAP(u,s,c,x,x')) \wedge J(s,c,x',y))$
equivalent to
(Definition of $[c]$)

(4) $AX(s,c) \vdash$
$I(s,c,x) \wedge J(s,c,x,y) \Rightarrow (\forall y'.(\exists v.H(v,s,c,x) \wedge CBAP(v,s,c,y,y')) \Rightarrow$
$((\exists x'.(\exists u.G(u,s,c,x) \wedge ABAP(u,s,c,x,x')) \wedge J(s,c,x',y')))$
equivalent to
(Transformation by quantifier elimination $\forall$)
(5) $AX(s,c) \vdash$
$I(s,c,x) \wedge J(s,c,x,y) \Rightarrow (\exists v.H(v,s,c,y) \wedge CBAP(v,s,c,y,y')) \Rightarrow$
$((\exists x'.(\exists u.G(u,s,c,x) \wedge ABAP(u,s,c,x,x')) \wedge J(s,c,x',y')))$
equivalent to
(Transformation by elimination of connector $\wedge$)
(6) $AX(s,c) \vdash$
$I(s,c,x) \wedge J(s,c,x,y) \wedge (\exists v.H(v,s,c,y) \wedge CBAP(v,s,c,y,y')) \Rightarrow$
$((\exists x'.(\exists u.G(u,s,c,x) \wedge ABAP(u,s,c,x,x')) \Rightarrow J(s,c,x',y')))$

equivalent to
(Transformation by elimination of quantifier $\exists$)
(7)
$AX(s,c) \vdash I(s,c,x) \wedge J(s,c,x,y) \wedge H(v,s,c,y) \wedge CBAP(v,s,c,y,y') \Rightarrow$
$((\exists x'.(\exists u.G(u,s,c,x) \wedge ABAP(u,s,c,x,x')) \wedge J(s,c,x',y')))$
equivalent to
(Transformation by property of quantifier $\exists$)
(8)
$AX(s,c) \vdash I(s,c,x) \wedge J(s,c,x,y) \wedge H(v,s,c,y) \wedge CBAP(v,s,c,y,y') \Rightarrow$
$((\exists x'.((\exists u.G(u,s,c,x) \wedge ABAP(u,s,c,x,x')) \wedge J(s,c,x',y')))$

equivalent to
(Transformation by elimination of $\wedge$)
(9)

1. $AX(s,c) \vdash I(s,c,x) \wedge J(s,c,x,y) \wedge H(v,s,c,y) \wedge CBAP(v,s,c,y,y') \Rightarrow (((\exists u.G(u,s,c,x))$
2. $AX(s,c) \vdash I(s,c,x) \wedge J(s,c,x,y) \wedge H(v,s,c,x) \wedge CBAP(v,s,c,y,y') \Rightarrow ((\exists x'.\exists u.(ABAP(u,s,c,x,x')) \wedge J(s,c,x',y')))$

property refinement between events (II)

Let x be the abstract variable (or list of variables) and $I(s, c, x)$ the abstract invariant, y the concrete variable (or list of variables) and $J(s, c, x, y)$ the concrete invariant. the concrete invariant.
Let c be a concrete event observing the variable y and a an event observing the variable x and preserving $I(s, c, x)$.
Event c refines event a with respect to x, $I(s, c, x)$, y and $J(s, c, x, y)$ *if, and only if,*

1. (GRD) $AX(s, c) \vdash I(s, c, x) \wedge J(s, c, x, y) \wedge H(v, s, c, x) \wedge CBAP(v, s, c, y, y') \Rightarrow \exists u.G(u, s, c, x)$
2. (SIM) $AX(s, c) \vdash I(s, c, x) \wedge J(s, c, x, y) \wedge H(v, s, c, x) \wedge CBAP(v, s, c, y, y') \Rightarrow ((\exists x'.\exists u.ABAP(u, s, c, x, x') \wedge J(s, c, x', y')))$

Proof obligations for Event-B refinement)

- (INIT) $AX(s,c), CInit(s,c,y') \vdash \exists x'.(AInit(s,c,x') \wedge J(s,c,x',y')$
- (GRD)
  $AX(s,c), I(s,c,x), J(s,c,x,y), H(v,s,c,x), CBAP(v,s,c,y,y') \vdash$
  $(((\exists u.G(u,s,c,x))$
- (GRD-WIT)
  $AX(s,c), I(s,c,x), J(s,c,x,y), H(v,s,c,x), CBAP(v,s,c,y,y'),$
  $WP(u,s,c,v,y) \vdash G(u,s,c,x)$
- (SIM) $AX(s,c), I(s,c,x), J(s,c,x,y), H(v,s,c,x), CBAP(v,s,c,y,y') \vdash$
  $((\exists x'.(\exists u.ABAP(u,s,c,x,x')) \wedge J(s,c,x',y')))$
- (SIM-WIT)
  $AX(s,c), I(s,c,x), J(s,c,x,y), H(v,s,c,x), CBAP(v,s,c,y,y'),$
  $WP(u,s,c,v,y), WV(v,s,c,y,x') \vdash ABAP(u,s,c,x,x')) \wedge J(s,c,x',y')$
- (WFIS-P)
  $AX(s,c) \vdash I(s,c,x) \wedge J(s,c,x,y) \wedge H(v,s,c,x) \wedge CBAP(v,s,c,y,y') \vdash$
  $\exists u.WP(u,s,c,v,y)$
- (WFIS-V)
  $AX(s,c) \vdash I(s,c,x) \wedge J(s,c,x,y) \wedge H(v,s,c,x) \wedge CBAP(v,s,c,y,y') \vdash$
  $\exists x'.WV(v,s,c,y,x')$
- (TH) $AX(s,c) \vdash I(s,c,x) \wedge J(s,c,x,y) \vdash SAFE_1(s,c,x,y)$

MACHINE CM **REFINES** AM
SEES E
VARIABLES y
INVARIANTS
$jnv_1 : J_1(s, c, x, y)$
$\ldots$
$jnv_r : J_r(s, c, x, y)$
THEOREMS
$th_1 : SAFE_1(s, c, x, y)$
$\ldots$
$th_n : SAFE_n(s, c, x, y)$
VARIANTS
$var_1 : varexp_1(s, c, y)$
$\ldots$
$var_t : varexp_t(s, c, y)$
EVENTS
EVENT initialisation
BEGIN
$y :| (CInit(s, c, y'))$
END
$\ldots$
EVENT c **REFINES** a
ANY v **WHERE**
$H(v, s, c, y)$
WITNESS
$u : WP(u, s, c, v, y)$
$x' : WV(v, s, c, y', x')$
THEN
$y :| CBAP(v, s, c, y, y')$
END
$\ldots$
END

- The machine CM is a model describing a set of events E(CM) modifying the y variable declared in the clause **VARIABLES**.
- A clause **REFINES** indicates that the CM machine refines a AM machine and E(AM) is the set of abstract events in AM .
- A particular event defines the initialisation of variable y according to the relationship $CInit(s, c, y')$.
- The property " Event c refines event a with respect to x, $I(s, c, x)$, y and $J(s, c, x, y)$" is denoted by the expression c refines a. Events a and c are attached to two machines AM and CM ; the invariant attached to each event is the invariant of its machine.

- A clause **INVARIANTS** describes the inductive invariant invariant $J(s, c, x, y)$ that this machine is assumed to respect provided that the associated verification conditions are shown to be valid in the theory induced by the context E mentioned by the clause **SEES**. $J(s, c, x, y)$ is the gluing invariant linking the variable y to the variable x.
- The clause **THEOREMS** introduces the list of safety properties derived in the theory. These properties relate to the variables y and x and must be proved valid. It is possible to add theorems about sets and constants ; this can help the proofs to be carried out during the verification process.
- To conclude this description, we would like to add that events can carry very important information for the proof process, in particular for proposing witnesses during event refinement. Furthermore, each event has a status (ordinary, convergent, anticipated) which is important in the production of verification conditions. The clause **VARIANTS** is linked to events of convergent and anticipated status. The event c (concrete) explicitly refines an event a of the AM machine.

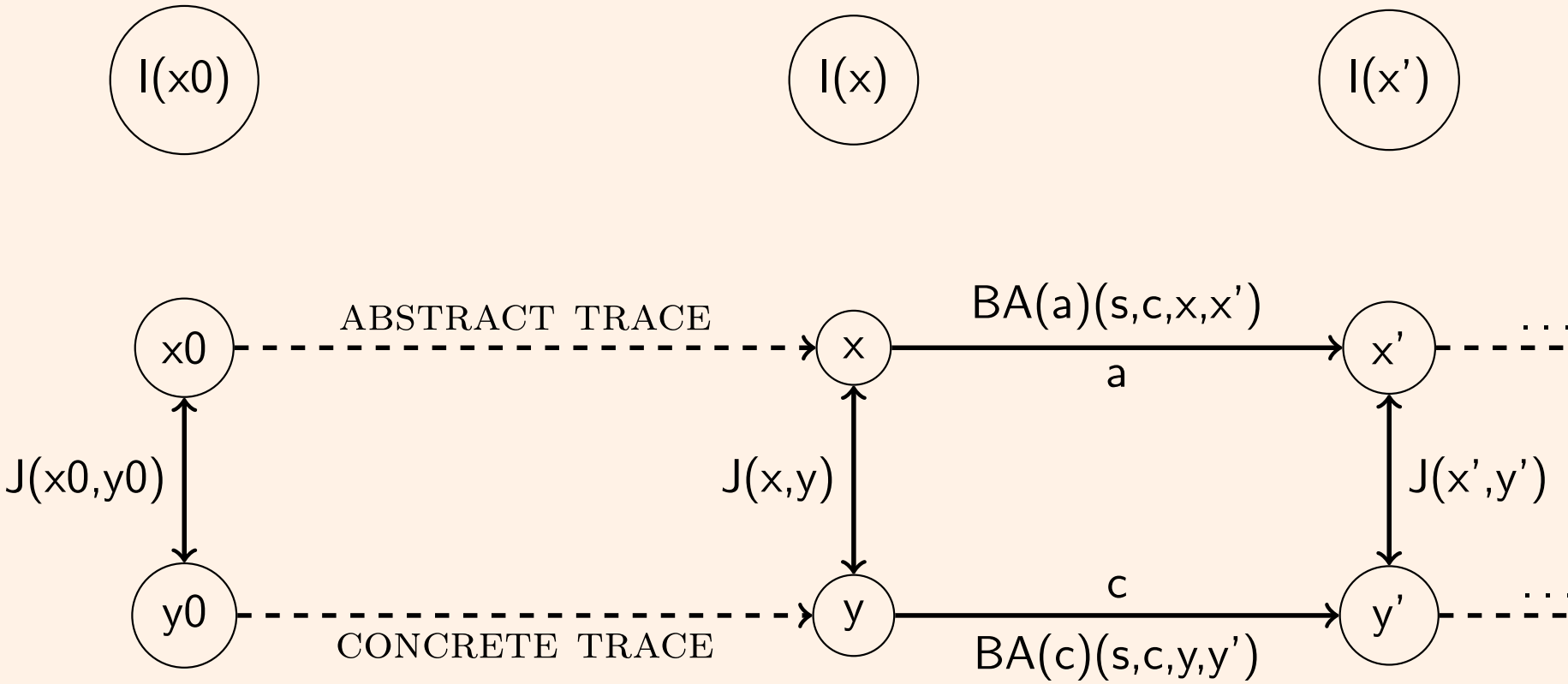
Event-B machine refinement REFINES

definition

The machine CM refines the machine AM , if any event c of CM refines an event a of AM :
$\forall$c.c $\in$ E(CM) $\Rightarrow$ $\exists$a.a $\in$ E(AM) $\wedge$ e refines a.

- Each machine has an event skip which does not modify the machine's variables.
- A concrete event c can refine an event skip whose effect is not to modify x in the abstract machine AM.
- The invariant of AM is $I(s, c, x)$ and that the initialisation of AM is $AInit(s, c, x')$.
- The proof witnesses are used to give properties of the parameter u and the variable x which have disappeared in the machine CM but for which the user must give an expression according to the state of CM .

Refinement between two machines



Current Summary

Example of a clock

- A machine M1 models hours or a machine M1 reports observations of hours

Example of a clock

- A machine M1 models hours or a machine M1 reports observations of hours
- and a machine M2 reports hours and minutes.
- A very special case of refinement called *superposition* and the proof is fairly straightforward.

CONTEXT C

CONSTANTS H M

AXIOMS
$@axm1\ H = 0..23$
$@axm2\ M = 0..59$
end

MACHINE $M1$ SEES C

VARIABLES h

INVARIANTS
$@inv1\ h \in H$

EVENTS
EVENT $INITIALISATION$
then
$@act1\ h :\in H$
end

EVENT $h1$
where
$@grd1\ h < 23$
then
$@act1\ h := h + 1$
end

EVENT $h2$
where
$@grd1\ h = 23$
then
$@act1\ h := 0$
end
end

MACHINE $M2$
REFINES $M1$
SEES C

VARIABLES $h\ m$

INVARIANTS
$@inv1\ m \in M$
$theorem\ @inv2\ h \in H$

EVENTS
EVENT $INITIALISATION$
then
$@act1\ h :\in H$
$@act2\ m :\in M$
end

EVENT $h1m1$
where
$@grd1\ h < 23$
$@grd2\ m < 59$
then
$@act2\ m := m + 1$
end

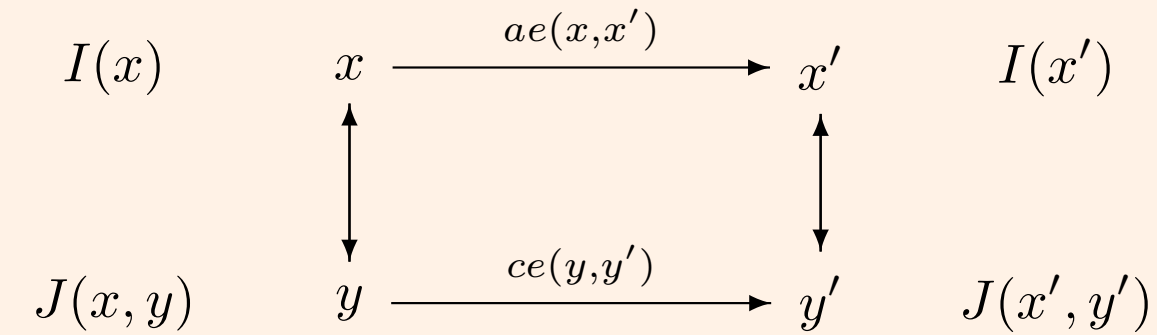
EVENT $h1m2$ REFINES $h1$
where
$@grd1\ h < 23$
$@grd2\ m = 59$
then
$@act1\ h := h + 1$
$@act2\ m := 0$
end

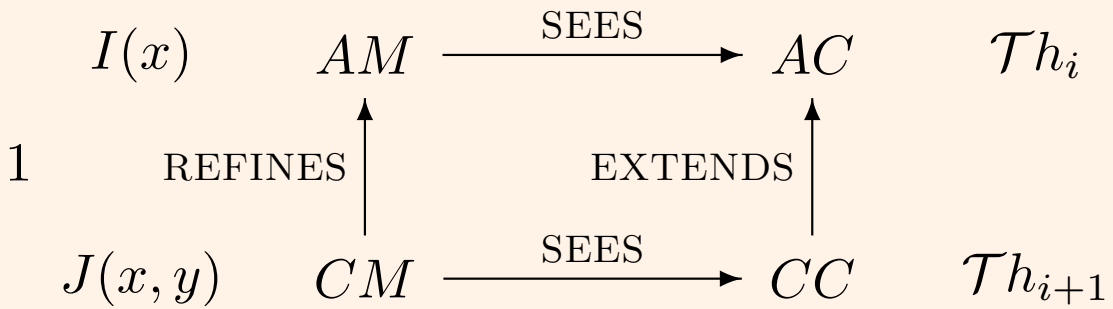
EVENT $h2m1$ REFINES $h2$
where
$@grd1\ h = 23$
$@grd2\ m = 59$
then
$@act1\ h := 0$
$@act2\ m := 0$
end

EVENT $h2m2$
where
$@grd1\ h = 23$
$@grd2\ m < 59$
then
$@act1\ m := m + 1$
end
end

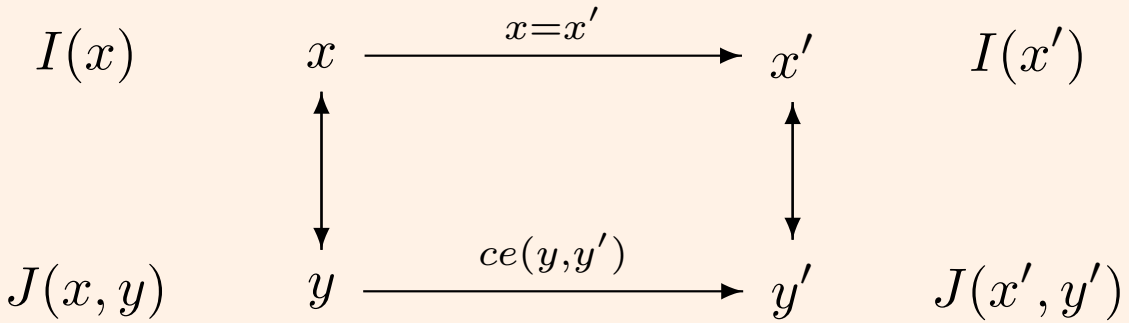
Current Summary

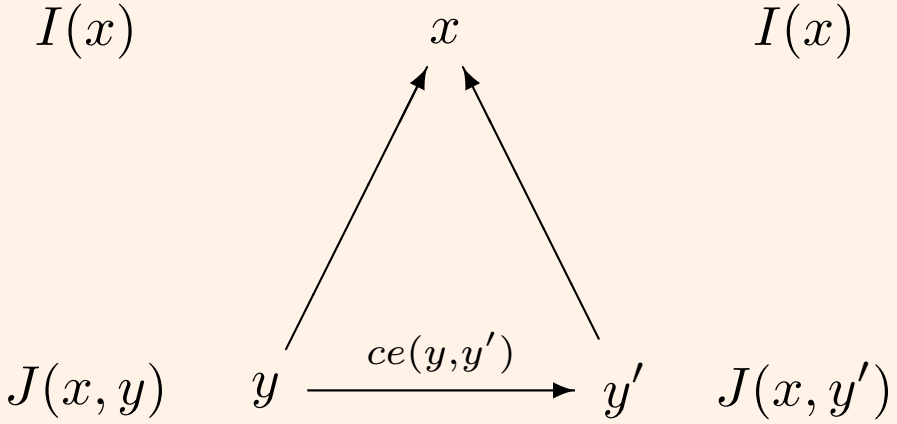
Refinement of a model by another one (I)





Refinement of a model by another one (II)





Proof obligations for refinement

(REF1) : refinement of initial conditions

$$\textsc{Initc}(y) \Rightarrow \exists x * (\textsc{Init}(x) \wedge \mathrm{J}(x, y)) :$$

The initial condition of the refinement model imply that there exists an abstract value in the abstract model such that that value satisfies the initial conditions of the abstract one and implies the new invariant of the refinement model.

Proof obligations for refinement

(REF2) : refinement of events

$$\mathrm{I}(x) \quad \wedge \quad \mathrm{J}(x,y) \quad \wedge \quad ce(y,y') \quad \Rightarrow \quad \exists x'.(ae(x,x') \quad \wedge \quad \mathrm{J}(x',y')) \quad :$$

The invariant in the refinement model is preserved by the refined event and the activation of the refined event triggers the corresponding abstract event.

Proof obligations for refinement

(REF3) : refinement of stuttering steps

$$\mathrm{I}(x) \quad \wedge \quad \mathrm{J}(x,y) \quad \wedge \quad ce(y,y') \;\Rightarrow\; \mathrm{J}(x,y') \quad :$$

The invariant in the refinement model is preserved by the refined event but the event of the refinement model is a new event which was not visible in the abstract model ; the new event refines $skip$.

Proof obligations for refinement

(REF4) : Refinement does not introduce more blocking states

$$\mathrm{I}(x) \quad \wedge \quad \mathrm{J}(x,y) \quad \wedge \quad (G_1(x) \vee \ldots \vee G_n(x)) \quad \Rightarrow \quad H_1(y) \vee \ldots \vee H_k(y) \quad :$$

The guards of events in the refinement model are strengthened and we have to prove that the refinement model is not more blocked than the abstract.

Proof obligations for refinement

(REF5) : Well-definedness of variant

$$\mathrm{I}(x) \quad \wedge \quad \mathrm{J}(x, y)) \quad \Rightarrow \quad V(y) \in \mathbb{N}$$

Proof obligations for refinement

(REF6) : Well behaviour of new events

$$\mathrm{I}(x) \quad \wedge \quad \mathrm{J}(x,y) \quad \wedge \quad ce(y,y') \;\Rightarrow\; V(y') < V(y) \quad :$$

New events should not block forever abstract ones.

Proof obligations for refinement

(REF7) : Feasibility of refined events

$$\Gamma(s,c) \vdash I(x) \wedge J(x,y) \wedge grd(E) \Rightarrow \exists y' \cdot P(y,y')$$

Current Summary

Current Summary

Designing a algorithm for computing $\lambda x.x!$

Computing $\lambda x.x!$

The problem is to derive an algorithm which is computing the function $\lambda x.x!$.

```
#ifndef _A_H
#define _A_H
/*@ axiomatic mathfact {
  @ logic integer mathfact(integer n);
  @ axiom mathfact_1: mathfact(0) == 1;
  @ axiom mathfact_rec: \forall integer n; n >= 1
  ==> mathfact(n) == n * mathfact(n-1);
  @ } */

/*@ requires n >= 0;
  ensures \result == mathfact(n);
*/
int codefact(int n);
#endif
```

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

CONTEXT $A - functions$

CONSTANTS $factorial\ n$

AXIOMS

$@axm1\ n \in \mathbb{N}$

$@axm2\ factorial \in \mathbb{N} \leftrightarrow \mathbb{N}$

$@axm3\ 0 \mapsto 1 \in factorial$

$@axm4\ \forall a.\forall b\,.\,a \in \mathbb{N} \land b \in \mathbb{N} \land a \mapsto b \in factorial \Rightarrow a+1 \mapsto (a+1)*b \in factorial$

$@axm5\ \forall f\,.\,f \in \mathbb{N} \leftrightarrow \mathbb{N} \land 0 \mapsto 1 \in f \land (\forall a,b\,.\,a \in \mathbb{N} \land b \in \mathbb{N} \land a \mapsto b \in f \Rightarrow a+1 \mapsto (a+1)*b \in f) \Rightarrow factorial \subseteq f$

$theorem\ @th1\ factorial \in \mathbb{N} \to \mathbb{N}$

$theorem\ @th2\ factorial(0) = 1$

$theorem\ @th3\ \forall u\,.\,u \in \mathbb{N} \land u \neq 0 \Rightarrow factorial(u) = u * factorial(u-1)$

$@axm6\ n > 3$

end

Machine B-prepost for stating the pre/post specification

MACHINE $B - prepost$ SEES $A - functions$
VARIABLES r
INVARIANTS
$@inv1\ r \in \mathbb{Z}$
EVENTS
EVENT $INITIALISATION$
then
$@act1\ r :\in \mathbb{Z}$
end
EVENT $computing1$
then
$@act1\ r := factorial(n)$
end
end

- Defining variables and invariant
- r is the variable for the result.
- n is the constant containing the input of the process.

Machine C-computing for stating the computing process

MACHINE $C-computing$ REFINES $B-prepost$
SEES $A-functions$
VARIABLES $r\ fac\ x$
INVARIANTS
$@inv1\ fac \in \mathbb{N} \pfun \mathbb{N}$
$@inv2\ dom(fac) \subseteq 0..n$
$@inv3\ dom(fac) \neq \varnothing$
$@inv4\ \forall\ i\ .\ i \in dom(fac) \Rightarrow fac(i) = factorial(i)$
$@inv5\ x \in dom(fac)$
$@inv6\ dom(fac) = 0..x$
EVENT $INITIALISATION$ REFINES $INITIALISATION$
then
$@act1\ r :\in \mathbb{Z}$
$@act2\ fac := \{0 \mapsto 1\}$
$@act3\ x := 0$
end

- Two new variables x and fac are introduced for storing the sequence $factorial$ by iterating over x
- Condition of termination is that $n \in dom(fac)$
- $fac(i) = factorial(i)$ is expressing the relationship between computed values and mathematically defned values of the sequence.

Machine C-computing for stating the computing process

EVENT $computing2$ REFINES $computing1$
where
$@grd1\ n \in dom(fac)$
then
$@act1\ r := fac(n)$
end

$convergent$ EVENT $step2$
where
$@grd11\ x \in dom(fac)$
$@grd12\ x+1 \notin dom(fac)$
$@grd13\ n \notin dom(fac)$
then
$@act11\ fac(x+1) := (x+1) * fac(x)$
$@act1\ x := x+1$
end
VARIANT $0..n \setminus dom(fac)$

- 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 is decreasing.
- it refines skip

Machine D-prealgo for getting an algorithmic process

MACHINE $D-prealgo$ REFINES $C-computing$
SEES $A-functions$
VARIABLES $r\ vfac\ cfac\ fac\ x$
INVARIANTS
$@inv1\ vfac \in \mathbb{N}$
$@inv2\ cfac \in \mathbb{N}$
$@inv3\ cfac \leq n$
$@inv4\ cfac \geq 0$
$@inv6\ cfac \in dom(fac)$
$@inv5\ vfac = fac(cfac)$
$@inv7\ cfac+1 \notin dom(fac)$
$@inv8\ dom(fac) = 0..cfac$
$@inv9\ x = cfac$

EVENT $INITIALISATION$
then
$@act1\ r :\in \mathbb{N}$
$@act2\ fac := \{0 \mapsto 1\}$
$@act3\ cfac := 0$
$@act4\ vfac := 1$
$@act5\ x := 0$
end

- Two new variables are introduced for storing really useful data namely the last computed values of the two sequences.
- Obviously, $vfac$ and $cfac$ satsify $vfac = fac(cfac)$
- Previous properties of abstrcat variables are safety properties which are no more to be reproved, thanks to refienement.

Machine D-prealgo for getting an algorithmic process

EVENT $computing3$ REFINES $computing2$
where
$@grd2\ cfac = n$
then
$@act1\ r := vfac$
end

$convergent$ EVENT $step3$ REFINES $step2$
where
$@grd1\ cfac \neq n$
then
$@act1\ vfac := (cfac + 1) * vfac$
$@act2\ cfac := cfac + 1$
$@act3\ fac(cfac + 1) := (cfac + 1) * fac(cfac)$
$@act4\ x := x + 1$
end

- The two events SIMulate the abstract events.
- However, the guards are strengthened and are made closer to an implementation : $cfac < n$ implies $n \notin dom(fac)$ and $cfac = n$ implies that $n \in dom(fac)$.

Machine E-algo for getting an algorithmic machine

VARIABLES $r\ vfac\ cfac$
INVARIANTS
$theorem\ @thm1\ vfac = factorial(cfac)$
$@inv1\ r \in \mathbb{N}$
$@computation\_inv1\ fac \in \mathbb{N} \pfun \mathbb{N}$
$@inv2\ dom(fac) \subseteq 0..n$
$@inv3\ dom(fac) \neq \varnothing$
$@inv4\ \forall\ i\ .\ i \in dom(fac) \Rightarrow fac(i) = factorial(i)$
$@inv5\ x \in dom(fac)$
$@inv6\ dom(fac) = 0..x$
$@algorithm\_inv1\ vfac \in \mathbb{N}$
$@algorithm\_inv2\ cfac \in \mathbb{N}$
$@algorithm\_inv3\ cfac \leq n$
$@algorithm\_inv4\ cfac \geq 0$
$@algorithm\_inv6\ cfac \in dom(fac)$
$@algorithm\_inv5\ vfac = fac(cfac)$
$@inv7\ cfac + 1 \notin dom(fac)$
$@inv8\ dom(fac) = 0..cfac$
$@inv9\ x = cfac$

VARIANT $n - cfac$

- The variables fac is now hidden and they disappear from the machine.
- It is playing the role of model variables as ghost variables.
- Invariants and safety properties are preserved through refinement.

Machine E-algo for getting an algorithmic machine

EVENT $INITIALISATION$
then
$@act1\ r :\in \mathbb{N}$
$@act3\ cfac := 0$
$@act4\ vfac := 1$
end

EVENT $computing4$ REFINES $computing3$
where
$@grd2\ cfac = n$
then
$@act1\ r := vfac$
end

$convergent$ EVENT $step4$ REFINES $step3$
where
$@grd1\ cfac \neq n$
then
$@act1\ vfac := (cfac + 1) * vfac$
$@act2\ cfac := cfac + 1$
end

- Assignments of fac are removed.

Current Summary

Simple Form of an Event

- An event of the simple form is denoted by :

$< event\_name > \widehat{=}$
WHEN
$< condition >$
THEN
$< action >$
END

where
- $< event\_name >$ is an identifier
- $< condition >$ is the firing condition of the event
- $< action >$ is a generalized substitution (parallel "assignment")

Non-deterministic Form of an Event

- An event of the non-deterministic form is denoted by :

$< event\_name > \widehat{=}$
ANY $< variable >$ **WHERE**
$< condition >$
THEN
$< action >$
END

where
- $< event\_name >$ is an identifier
- $< variable >$ is a (list of) variable(s)
- $< condition >$ is the firing condition of the event
- $< action >$ is a generalized substitution (parallel "assignment")

Shape of a Generalized Substitution

A generalized substitution can be

- Simple assignment : $x := E$
- Generalized assignment : $x : P(x, x')$
- Set assignment : $x :\in S$
- Parallel composition : $\begin{array}{l} T \\ \cdots \\ U \end{array}$

,

Invariant Preservation Verification (0)

$$\text{INVARIANT} \quad \wedge \quad \text{GUARD}$$
$$\Longrightarrow$$
$$\text{ACTION establishes INVARIANT}$$

Invariant Preservation Verification (1)

- Given an event of the simple form :

EVENT EVENT $\widehat{=}$
WHEN
$G(x)$
THEN
$x := E(x)$
END

and invariant $I(x)$ to be preserved, the statement to prove is :

$$I(x) \quad \wedge \quad G(x) \quad \Longrightarrow \quad I(E(x))$$

Invariant Preservation Verification (2)

- Given an event of the simple form :

EVENT EVENT $\widehat{=}$
WHEN
$G(x)$
THEN
$x :| P(x, x')$
END

and invariant $I(x)$ to be preserved, the statement to prove is :

$$I(x) \quad \wedge \quad G(x) \quad \wedge \quad P(x, x') \quad \Longrightarrow \quad I(x')$$

Invariant Preservation Verification (3)

- Given an event of the simple form :

EVENT EVENT $\widehat{=}$
WHEN
$G(x)$
THEN
$x :\in S(x)$
END

and invariant $I(x)$ to be preserved, the statement to prove is :

$$I(x) \quad \wedge \quad G(x) \quad \wedge \quad x' \in S(x) \quad \Longrightarrow \quad I(x')$$

Invariant Preservation Verification (4)

- Given an event of the non-deterministic form :

EVENT EVENT $\widehat{=}$
ANY v **WHERE**
$G(x, v)$
THEN
$x := E(x, v)$
END

and invariant $I(x)$ to be preserved, the statement to prove is :

$$I(x) \quad \wedge \quad G(x, v) \quad \Longrightarrow \quad I(E(x, v))$$

Refinement Technique (1)

- Abstract models works with variables x, and concrete one with y
- A gluing invariant $J(x, y)$ links both sets of vrbls
- Each abstract event is refined by concrete one (see below)

Refinement Technique (2)

- Some new events may appear : they refine “skip”
- Concrete events must not block more often than the abstract ones
- The set of new event alone must always block eventually

Correct Refinement Verification (1)

- Given an abstract and a corresponding concrete event

EVENT ea $\widehat{=}$
WHEN
$G(x)$
THEN
$x := E(x)$
END

EVENT ec $\widehat{=}$
WHEN
$H(y)$
THEN
$y := F(y)$
END

and invariants $I(x)$ and $J(x, y)$, the statement to prove is :

$$I(x) \ \wedge \ J(x, y) \ \wedge \ H(y) \implies G(x) \ \wedge \ J(E(x), F(y))$$

Correct Refinement Verification (2)

- Given an abstract and a corresponding concrete event

EVENT ea $\widehat{=}$
ANY v **WHERE**
$G(x, v)$
THEN
$x := E(x, v)$
END

EVENT ec $\widehat{=}$
ANY w **WHERE**
$H(y, w)$
THEN
$y := F(y, w)$
END

$$I(x) \quad \wedge \quad J(x, y) \quad \wedge \quad H(y, w)$$
$$\Longrightarrow$$
$$\exists v \cdot (\, G(x, v) \quad \wedge \quad J(E(x, v), F(y, w)) \,)$$

Correct Refinement Verification (3)

- Given a NEW event

EVENT EVENT $\widehat{=}$
  WHEN
    $H(y)$
  THEN
    $y := F(y)$
  END

and invariants $I(x)$ and $J(x, y)$, the statement to prove is :

$$I(x) \quad \wedge \quad J(x, y) \quad \wedge \quad H(y) \quad \Longrightarrow \quad J(x, F(y))$$

Current Summary

General form of proof obligations for an event e

- *INIT/I/INV* : $C(s,c), INIT(c,s,x) \vdash I(c,s,x)$
- *e/I/INV* : $C(s,c), I(c,s,x), G(c,s,t,x), P(c,s,t,x,x') \vdash I(c,s,x')$
- *e/act/FIS* : $C(s,c), I(c,s,x), G(c,s,t,x) \vdash$
- *e/act/WD* : $C(s,c), I(c,s,x), G(c,s,t,x) \vdash \exists x'.P(c,s,t,x,x')$

Well-definedness of an Axiom	m / WD	m is the axiom name
Well-definedness of a Derived Axiom	m / WD	m is the axiom name
Derived Axiom	m / THM	m is the axiom name
Well-definedness of an Invariant	v / WD	v is the invariant name
Well-definedness of a Derived Invariant	m / WD	m is the invariant name
Well-definedness of an event Guard	t / d / WD	t is the event name d is the action name
Well-definedness of an event Action	t / d / WD	t is the event name d is the action name
Feasibility of a non-det. event Action	t / d / FIS	t is the event name d is the action name
Derived Invariant	m / THM	m is the invariant name
Invariant Establishment	INIT. / v / INV	v is the invariant name
Invariant Preservation	t / v / INV	t is the event name v is the invariant name

Current Summary

algorithms by J.-R. Abrial

WHEN P Q **THEN** S **END**

WHEN P $\neg Q$ **THEN** T **END**

are merged into

WHEN P **THEN** **WHILE** Q **DO** S **END**; T **END**

Side Conditions :

- P must be invariant under S.
- The first event must have been introduced at one refinement step below the second one.
- Special Case : If P is missing the resulting "event" has no guard

algorithms by J.-R. Abrial

```
WHEN
  P
  Q
THEN
  S
END
```

```
WHEN
  P
  ¬Q
THEN
  T
END
```

are merged into

```
WHEN
  P
THEN
  IF Q THEN S
  ELSE T
  END;
END
```

Side Conditions :

- The disjunctive negation of the previous side conditions
- Special Case : If P is missing the resulting "event" has no guard

Deriving an algorithm

precondition : $n \in \mathbb{N}$
postcondition : $result = factorial(n)$
local variables : $vfac, cfac \in \mathbb{N}$

$cfac := 0; vfac := 1; result :\in \mathbb{N};$
while $cfac \neq n$ **do**
  Invariant : $vfac = fac(cfac)$
  $vfac := (cfac + 1) * vfac; cfac := cfac + 1;$
;
$result := vfac;$

Translating the machine E-algo to an algorithm

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

int codefact(int n) {
  int cfac=0;
  int vfac= 1;
  /*@ loop invariant cfac >= 0 && cfac <= n &&  mathfact(cfac) ==
    loop assigns cfac, vfac;
    loop variant n-cfac;
   */
  while (cfac != n) {
    vfac = (cfac+1)*vfac;
    cfac  = cfac + 1;
  };
  return vfac;
}
```

Current Summary

Conclusion

- Refinement helps in discovering invariants
- Refinement helps in proving invariants
- The choice of the *good* abstraction is not very simple and is a challenge by itself

Current Summary

Summary on refinement

- Refining means making models more deterministic
- Refining means adding new variable and new events
- Refining is simulating
- Refining preserves safety properties of the refined model.
- The very abstract model is crucial.
- The process should be incremental to make proofs easier for the proof tool.
- Problem : Preserving the liveness properties