

Modelling Software-based Systems

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

Telecom Nancy

Dominique Méry
Telecom Nancy, Université de Lorraine

26 mars 2025(1:37 P.M.)
dominique.mery@loria.fr

General Summary

- 1 Correctness by Construction
- 2 The refinement of models
- 3 Example of the clock
- 4 Summary of the refinement
- 5 Example of the factorial function refined into an algorithm
- 6 Review of Event-B
- 7 Intermezzo on the Event B modelling notation
- 8 Transformations of Event-B models
- 9 Conclusion
- 10 The Inductive Paradigm
- 11 Summary

Current Summary

- ## 9 Conclusion

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.

- Informal Requirements : Restricted form of natural language.
- Formal Specification : Modeling language like Event-B , Z, ASM, VDM, TLA+...
- Formal Verification : Theorem Prover Tools like PVS, Z3, SAT, SMT Solver...
- Formal Validation : Model Checker Tools like ProB, UPPAAL , SPIN, SMV ...
- Real-time Animation : **Our proposed approach ... Real-Time Animator ...**
- Code Generation : **Our proposed approach ... EB2ALL : EB2C, EB2C++, EB2J, EB2C# ...**
- Acceptance Testing : Failure Mode, Effects and Critically analysis(FMEA and FMEA), System Hazard Analyses(SHA)

- *Colin Boyd and Anish Mathuria. Protocols Authentication and Key Establishment. Springer 2003.*
- *C. C. Marquezan and L. Z. Granville. Self-* and P2P for Network Management - Design Principles and Case Studies. Springer Briefs in Computer Science. Springer, 2012.*
- *Pacemaker Challenge Contribution*

Current Summary

- ## 9 Conclusion

- 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)

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

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

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

Explanations for the refinement

(Hypothesis)

$$(1) \quad AX(s, c) \vdash I(s, c, x) \wedge J(s, c, x, y) \Rightarrow [c](\neg[a](\neg J(s, c, x, y)))$$

equivalent to

(Definition of $[a]$: $[a](\neg J(s, c, x, y)) \equiv$

$$\forall x'. (\exists u. G(u, s, c, x) \wedge ABAP(u, s, c, x, x')) \Rightarrow \neg J(s, c, x', y))$$

$$(2) \quad AX(s, c) \vdash I(s, c, x) \wedge J(s, c, x, y) \Rightarrow [c](\neg(\forall x'. (\exists u. G(u, s, c, x) \wedge ABAP(u, s, c, x, x')) \Rightarrow \neg J(s, c, x', y)))$$

equivalent to

(Transformation by simplification of logical connectives)

$$(3) \quad AX(s, c) \vdash I(s, c, x) \wedge J(s, c, x, y) \Rightarrow [c](\exists x'. (\exists u. G(u, s, c, x) \wedge ABAP(u, s, c, x, x')) \wedge J(s, c, x', y))$$

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)

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

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

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

$jnv_1 : J_1(s, c, x, y)$

...

$jnv_r : J_r(s, c, x, y)$

THEOREMS

$th_1 : SAFE_1(s, c, x, y)$

...

$th_n : SAFE_n(s, c, x, y)$

VARIANTS

$var_1 : varexp_1(s, c, y)$

...

$var_t : varexp_t(s, c, y)$

EVENTS

EVENT initialisation

BEGIN

$y : |(CInit(s, c, y'))$

END

...

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

...

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 $J(s, c, x, y)$ 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.

definition

The machine CM refines the machine AM , if any event c of CM refines an event a of AM :

$$\forall c.c \in E(\text{CM}) \Rightarrow \exists a.a \in E(\text{AM}) \wedge e \text{ refines } a.$$

- 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

- ## 9 Conclusion

Example of a clock

- A machine M_1 models hours or a machine M_1 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
```

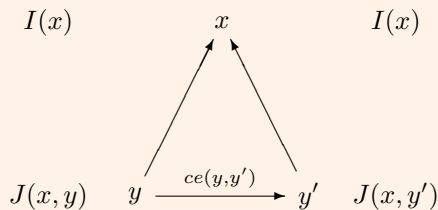
Current Summary

- ## 9 Conclusion

Refinement of a model by another one (I)



Refinement of a model by another one (II)



(REF1) : refinement of initial conditions

$$\text{INITC}(y) \Rightarrow \exists x * (\text{INIT}(x) \wedge \mathbf{J}(x, y)) :$$

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.

(REF2) : refinement of events

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

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

(REF3) : refinement of stuttering steps

$$I(x) \wedge J(x, y) \wedge ce(y, y') \Rightarrow J(x, y') :$$

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*.

(REF4) : Refinement does not introduce more blocking states

$$I(x) \wedge J(x,y) \wedge (G_1(x) \vee \dots \vee G_n(x)) \Rightarrow H_1(y) \vee \dots \vee H_k(y) :$$

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.

(REF5) : Well-definedness of variant

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

(REF6) : Well behaviour of new events

$$I(x) \wedge J(x, y) \wedge ce(y, y') \Rightarrow V(y') < V(y) :$$

New events should not block forever abstract ones.

(REF7) : Feasibility of refined events

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

Current Summary

- 1 Correctness by Construction
- 2 The refinement of models
- 3 Example of the clock
- 4 Summary of the refinement
- 5 Example of the factorial function refined into an algorithm
- 6 Review of Event-B
- 7 Intermezzo on the Event B modelling notation
- 8 Transformations of Event-B models

- ## 9 Conclusion

The factorial model

CONTEXT

fonctions

CONSTANTS *factorial, n*

AXIOMS

$$\begin{aligned} & n \in \mathbb{N} \wedge factorial \in \mathbb{N} \leftrightarrow \mathbb{N} \wedge 0 \mapsto 1 \in factorial \wedge \\ & \forall (i, fn). (i \mapsto fn \in factorial \Rightarrow i + 1 \mapsto (i + 1) * fn \in factorial) \wedge \\ & \forall f. \left(\begin{array}{l} f \in \mathbb{N} \leftrightarrow \mathbb{N} \wedge \\ 0 \mapsto 1 \in f \wedge \\ \forall (n, fn). (n \mapsto fn \in f \Rightarrow n + 1 \mapsto (n + 1) \times fn \in f) \\ \Rightarrow \\ factorial \subseteq f \end{array} \right) \end{aligned}$$

END

The factorial pre/post specification

MACHINE *B – prepostok* SEES *A – functions*

VARIABLES *r ok*

INVARIANTS

$@inv1\ ok \in \text{BOOL}$

$@inv2\ r \in \mathbb{Z}$

$@inv3\ ok = \text{TRUE} \Rightarrow r = \text{factorial}(n)$

EVENTS

EVENT *INITIALISATION*

then

$@act1\ r : \in \mathbb{Z}$

$@act2\ ok := \text{FALSE}$

end

EVENT *final*

where

$@grd1\ ok = \text{FALSE}$

then

$@act1\ r := \text{factorial}(n)$

$@act2\ ok := \text{TRUE}$

end

end

The factorial context

CONTEXT $A - \text{functions}$

CONSTANTS $\text{factorial } n$

AXIOMS

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

@axm2 $\text{factorial} \in \mathbb{N} \rightarrow \mathbb{N}$

@axm3 $0 \mapsto 1 \in \text{factorial}$

@axm4 $\forall a. \forall b. a \in \mathbb{N} \wedge b \in \mathbb{N} \wedge a \mapsto b \in \text{factorial}$

$\Rightarrow a + 1 \mapsto (a + 1) * b \in \text{factorial}$

@axm5 $\forall f. f \in \mathbb{N} \rightarrow \mathbb{N}$

$\wedge 0 \mapsto 1 \in f$

$\wedge (\forall a, b. a \in \mathbb{N} \wedge b \in \mathbb{N} \wedge a \mapsto b \in f)$

$\Rightarrow a + 1 \mapsto (a + 1) * b \in f)$

$\Rightarrow \text{factorial} \subseteq f$

theorem @th1 $\text{factorial} \in \mathbb{N} \rightarrow \mathbb{N}$

theorem @th2 $\text{factorial}(0) = 1$

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

@axm6 $n > 3$

end

The factorial model

MACHINE

specification

SEES *fonctions*

VARIABLES

resultat

INVARIANT

$resultat \in \mathbb{N}$

THEOREMS

$factorial \in \mathbb{N} \rightarrow \mathbb{N}$;

$factorial(0) = 1$;

$\forall n.(n \in \mathbb{N} \Rightarrow factorial(n+1) = (n+1) \times factorial(n))$

INITIALISATION

$resultat := \mathbb{N}$

EVENTS

$computing1 = \text{BEGIN } resultat := factorial(n) \text{ END}$

END

Refining *specification* by *computation*

MACHINE *computation*

REFINES *specification*

SEES *fonctions*

VARIABLES *resultat, fac, x*

INVARIANTS

inv1 : $fac \in \mathbb{N} \leftrightarrow \mathbb{N}$

inv2 : $dom(fac) \subseteq 0 .. n$

inv4 : $dom(fac) \neq \emptyset$

inv5 : $\forall i. i \in dom(fac) \Rightarrow fac(i) = factorial(i)$

inv3 : $x \in dom(fac)$

inv6 : $dom(fac) = 0 .. x$

EVENTS

EVENT INITIALISATION

BEGIN

act1 : $resultat := \mathbb{N}$

act2 : $fac := \{0 \mapsto 1\}$

act3 : $x := 0$

END

EVENT computing2 **REFINES** **EVENT** computing1

WHEN

grd1 : $n \in dom(fac)$

THEN

act1 : $resultat := fac(n)$

END

END

Refining *specification* by *computation*

MACHINE *computation*

REFINES *specification*

SEES *fonctions*

VARIABLES *resultat, fac, x*

INVARIANTS

inv1 : $fac \in \mathbb{N} \leftrightarrow \mathbb{N}$

inv2 : $dom(fac) \subseteq 0..n$

inv4 : $dom(fac) \neq \emptyset$

inv5 : $\forall i. i \in dom(fac) \Rightarrow fac(i) = factorial(i)$

inv3 : $x \in dom(fac)$

inv6 : $dom(fac) = 0..x$

EVENTS

EVENT *event2*

WHEN

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

END

Refining *computation* by *algorithm*

EVENTS

EVENT INITIALISATION

BEGIN

```
act1 : resultat ∈ ℕ  
act2 : fac := {0 ↦ 1}  
act3 : cfac := 0  
act4 : vfac := 1  
act5 : x := 0
```

END

EVENT computing3 **REFINES** EVENT computing2

WHEN grd2 : cfac = n

THEN act1 : resultat := vfac

END

EVENT event3 **REFINES** EVENT event2

WHEN grd1 : cfac ≠ n

THEN

```
act1 : vfac := (cfac + 1) * vfac  
act2 : cfac := cfac + 1  
act3 : fac(cfac + 1) := (cfac + 1) * fac(cfac)  
act4 : x := x + 1
```

END

END

Refining *computation* by *algorithm*

MACHINE *algorithm* **REFINES** *computation*

SEES *fonctions*

VARIABLES *resultat, vfac, cfac, fac, x*

INVARIANTS

$$inv1 : vfac \in \mathbb{N}$$
$$inv2 : cfac \in \mathbb{N}$$
$$inv3 : cfac \leq n$$
$$inv4 : cfac \geq 0$$
$$inv6 : cfac \in dom(fac)$$
$$inv5 : vfac = fac(cfac)$$
$$inv7 : cfac + 1 \notin dom(fac)$$
$$inv8 : dom(fac) = 0 .. cfac$$
$$inv9 : x = cfac$$

Refining *algorithm* by *simplealgorithm*

```
MACHINE simplealgorithm REFINES algorithm
SEES fonctions
VARIABLES resultat, vfac, cfac
THEOREMS thm1 : vfac = factorial(cfac)
EVENTS
EVENT INITIALISATION
  BEGIN
    act1 : resultat ∈ ℕ
    act3 : cfac := 0
    act4 : vfac := 1
  END
EVENT computing4 REFINES EVENT computing3
  WHEN
    grd2 : cfac = n
  THEN
    act1 : resultat := vfac
  END
EVENT event4 REFINES EVENT event3
  WHEN
    grd1 : cfac ≠ n
  THEN
    act1 : vfac := (cfac + 1) * vfac
    act2 : cfac := cfac + 1
  END
END
```

Current Summary

- ## 9 Conclusion

Simple Form of an Event

- An event of the **simple** form is denoted by :

```

< event_name > ≡
WHEN
    < condition >
THEN
    < action >
END

```

where

- $\langle event_name \rangle$ is an identifier
- $\langle condition \rangle$ is the firing condition of the event
- $\langle action \rangle$ is a generalized substitution (parallel “assignment”)

Non-deterministic Form of an Event

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

```
< event_name >  $\hat{=}$   
  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 : $\begin{matrix} T \\ \dots \\ U \end{matrix}$

$$\text{INVARIANT} \wedge \text{GUARD} \implies \text{ACTION establishes INVARIANT}$$

Invariant Preservation Verification (1)

- Given an event of the simple form :

```

EVENT EVENT ≡
  WHEN
     $G(x)$ 
  THEN
     $x := E(x)$ 
  END

```

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

$$I(x) \wedge G(x) \implies I(E(x))$$

Invariant Preservation Verification (2)

- Given an event of the simple form :

EVENT EVENT \triangleq
WHEN
 $G(x)$
THEN
 $x : |P(x, x')$
END

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

$$I(x) \wedge G(x) \wedge P(x, x') \implies I(x')$$

Invariant Preservation Verification (3)

- Given an event of the simple form :

EVENT EVENT \triangleq
WHEN
 $G(x)$
THEN
 $x \in S(x)$
END

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

$$I(x) \wedge G(x) \wedge x' \in S(x) \implies I(x')$$

Invariant Preservation Verification (4)

- Given an event of the non-deterministic form :

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

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

$$I(x) \wedge G(x, v) \implies I(E(x, v))$$

Correct Refinement Verification (1)

- Given an **abstract** and a corresponding **concrete** event

```
EVENT ea ≐  
  WHEN  
    G(x)  
  THEN  
    x := E(x)  
  END
```

```
EVENT ec ≐  
  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  $\hat{=}$   
  ANY v WHERE  
    G(x, v)  
  THEN  
    x := E(x, v)  
  END
```

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

$$\begin{aligned} & I(x) \wedge J(x, y) \wedge H(y, w) \\ \Rightarrow & \exists v \cdot (G(x, v) \wedge J(E(x, v), F(y, w))) \end{aligned}$$

Correct Refinement Verification (3)

- Given a NEW event

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

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

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

Current Summary

- ## ⑦ Intermezzo on the Event B modelling notation

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

- 1 Correctness by Construction
- 2 The refinement of models
- 3 Example of the clock
- 4 Summary of the refinement
- 5 Example of the factorial function refined into an algorithm
- 6 Review of Event-B
- 7 Intermezzo on the Event B modelling notation
- 8 Transformations of Event-B models

- ## 9 Conclusion

```
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

```
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

Applying the rule for the while

```
EVENT computing4 REFINES EVENT computing3
  WHEN
     $grd2 : cfac = n$ 
  THEN
     $act1 : resultat := vfac$ 
  END
```

```
EVENT event4 REFINES EVENT event3
  WHEN
     $grd1 : cfac \neq n$ 
  THEN
     $act1 : vfac := (cfac + 1) * vfac$ 
     $act2 : cfac := cfac + 1$ 
  END
END
```

```
EVENT computing4 EVENT event4
  WHILE  $cfac \neq n$  DO
     $act1 : vfac := (cfac + 1) * vfac$ 
     $act2 : cfac := cfac + 1$ 
  END;
END
```

Applying the INITIALISATION rule

EVENT INITIALISATION

BEGIN

act1 : *resultat* : $\in \mathbb{N}$

act3 : *cfac* := 0

act4 : *vfac* := 1

END

init

resultat : $\in \mathbb{N}$;

cfac := 0;

vfac := 1;

Deriving an algorithm

```
precondition    :  $n \in \mathbb{N}$ 
```

postcondition : $result = factorial(n)$

local variables : $vfac, cfac \in \mathbb{N}$

$$cfac := 0; vfac := 1; result := \mathbb{N};$$
while $cfac \neq n$ **do**

Invariant : $vfac = fac(cfac)$

$$vfac := (cfac + 1) * vfac; cfac := cfac + 1;$$

•

```

result := vfac;

```

Current Summary

9 Conclusion

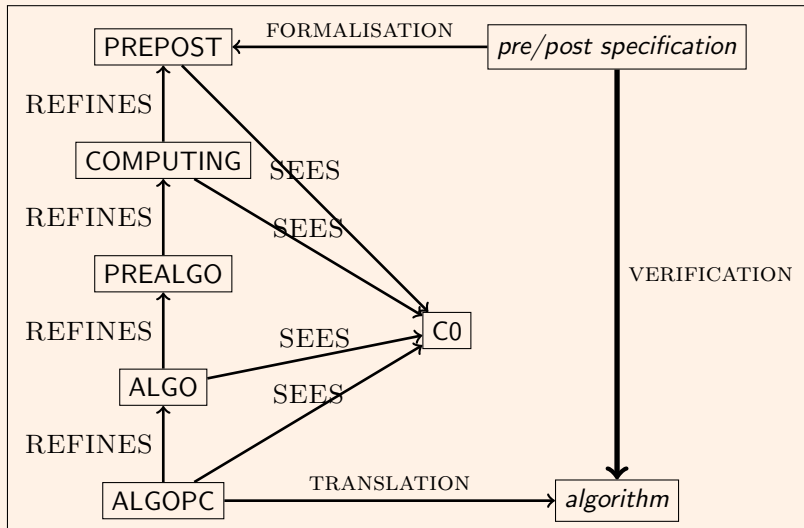
Conclusion

- 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

- ## 9 Conclusion

The Iterative Pattern



CONTEXT $C0$ **SETS** U **CONSTANTS** $x, v, d0, f, D$ **AXIOMS** $axm1 : x \in \mathbb{N}$ $axm25 : D \subseteq U$ $axm24 : f \in D \rightarrow D$ $axm23 : d0 \in D$ $axm2 : v \in \mathbb{N} \rightarrow D$ $axm3 : v(0) = d0$ $axm4 : \forall n \cdot n \in \mathbb{N} \Rightarrow v(n+1) = f(v(n))$ $th1 : Q(d0, d) \equiv (d = v(x))$

- the sequence v expresses the post-condition $Q(d0, d)$ with the precondition $P(d0)$.
- $Q(d0, d)$ is equivalent to $d = v(x)$.
- The theorem $th1$ should be proved in the context $C0$. he

General PREPOST Machine

```

MACHINE PREPOST
SEES  $C0$ 
VARIABLES
   $r$ 
INVARIANTS
   $inv1 : r \in D$ 
EVENTS
INITIALISATION
  BEGIN
     $act1 : r \in D$ 
  END
EVENT computing
  BEGIN
     $act1 : r := v(x)$ 
  END
END

```

- The theorem *th1* is validating the definition of the result *r* to compute.
- The event computing is expressing the *contract* of the given problem.
- it by a very simple problem that is the computation of the function n^2 using the addition operator.

EVENT INITIALISATION

BEGIN

$$act1 : r : \in D$$
$$act3 : vv := \{0 \mapsto d0\}$$
$$act5 : k := 0$$

END

INITIALISATION is initializing the variables with respect to the initial values of the sequences of the context.

First Refinement COMPUTING : Inductive Computation

```

EVENT computing
REFINES computing,
WHEN
     $grd1 : x \in dom(vv)$ 
THEN
     $act1 : r := vv(x)$ 
END
END

```

computing is imply observing that the result is computed *simulating* the sequence vv .

First Refinement COMPUTING : Inductive Computation

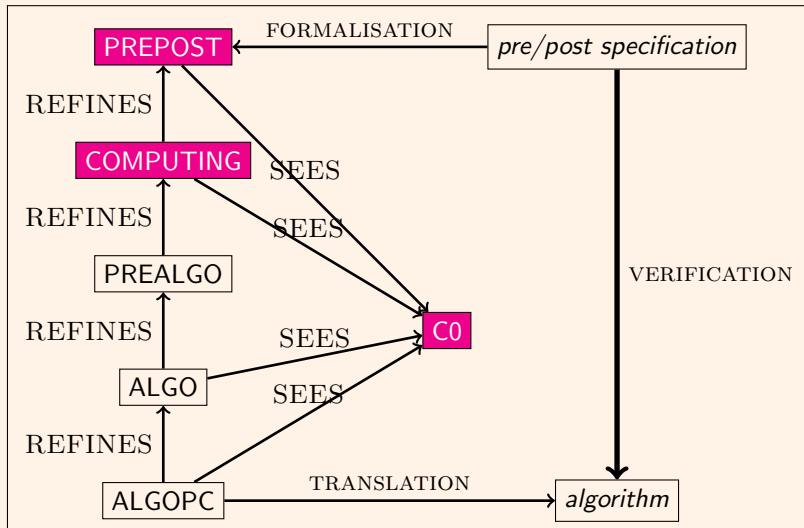
```

EVENT step
WHEN
     $grd1 : x \notin dom(vv)$ 
THEN
     $act2 : vv(k+1) := f(vv(k))$ 
     $act4 : k := k + 1$ 
END

```

step is *simulating* the computation of the values of the sequence vv as a model computation.

The Iterative Pattern



Completing the machines

- PREALGO : adding new variables for pointing out the necessary values to store cvv
- ALGO : hiding the model variables storing the unnecessary values of sequence vv
- ALGOPC ; adding control variable c

The Iterative Pattern



Listing 1 – Function derived from pattern for the sequence v

```
type(D)  f(int x)
{int    r, k, cv, or, ok, ocv;
  r=0;k=0;cv=0;or=0;ok=k;ocv=cv;
  while (k<x)
  {
    ok=k;ocv=cv;
    k=ok+1;
    cv=f(ocv);
  }
  r=cv;  return(r);}
```

Comments

- The produced algorithm can be now checked using another proof environment as for instance Frama-C.
- The inductive property of the invariant is clearly verified and is easily derived from the Event-B machines.
- The verification is not required, since the system is correct by construction but it is a checking of the process itself
- the project called ITERATIVE-PATTERN ;
- the project is the pattern itself
- The invariants of the Event-B models can be reused in the verification using Frama-C, for instance, and the verification of the resulting algorithm is a confirmation of the translation.

Listing 2 – Function derived from pattern power3

```
#include <limits.h>
/*@ requires 0 <= x;
    requires x*x*x <= INT_MAX ;
    ensures \result == x*x*x;
*/
int power3(int x)
{
    int r, ocz, cz, cv, cu, ocv, cw, ocw, ct, oct, ocu, k, ok;
    cz=0; cv=0; cw=1; ct=3; cu=0; ocw=cw; ocz=cz;
    oct=ct; ocv=cv; ocu=cu; k=0; ok=k;
    /*@ loop invariant cz == k*k*k;
        @ loop invariant cu == k;
        @ loop invariant cv+ct==3*(cu+1)*(cu+1);
        @ loop invariant cz+cv+cw==3*(cu+1)*(cu+1)*(cu+1);
        @ loop invariant cv== 3*cu*cu;
        @ loop invariant cw == 3*cu+1;
        @ loop invariant k <= x;
        @ loop assigns ct, oct, cu, ocu, cz, ocz, k, cv, cw, r, ok;
        @ loop assigns ocv, ocw; */
    while (k<x)
    {
        ocz=cz; ok=k; ocv=cv; ocw=cw; oct=ct; ocu=cu;
        cz=ocz+ocz+ocz;
        cv=ocv+oct;
        ct=oct+6;
        cw=ocw+3;
        cu=ocu+1;
        k=ok+1;
    }
    r=cz; return (r);
}
```


Summary for proof obligations

| Name | Total | Automatic | Interactive |
|--------------|-------|-----------|-------------|
| ex-induction | 40 | 36 | 4 |
| C0 | 2 | 0 | 2 |
| PREPOST | 4 | 4 | 0 |
| COMPUTING | 16 | 14 | 2 |
| PREALGO | 9 | 9 | 0 |
| ALGO | 6 | 6 | 0 |
| ALGOPC | 3 | 3 | 0 |

Summary

- The loop invariant is inductive but Frama-C does not prove it completely.
- Not the case with the RODIN platform which is able to discharge the whole set of proof obligations.
- However, the Event-B model is using auxiliary knowledge over sequences used for defining the computing process.
- The most difficult theorem is to prove that $\forall n \in \mathbb{N} : z_n = n * n * n$.

The Iterative Pattern



Current Summary

- ## 9 Conclusion

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