# Modelling Software-based Systems

## Proof Obligation Generation for Refinements and Application to Correctness-by-Construction of Sequential Algorithms

Telecom Nancy (IL et LE)

Dominique Méry
Telecom Nancy,Université de Lorraine

23 novembre 2026

dominique.mery@loria.fr

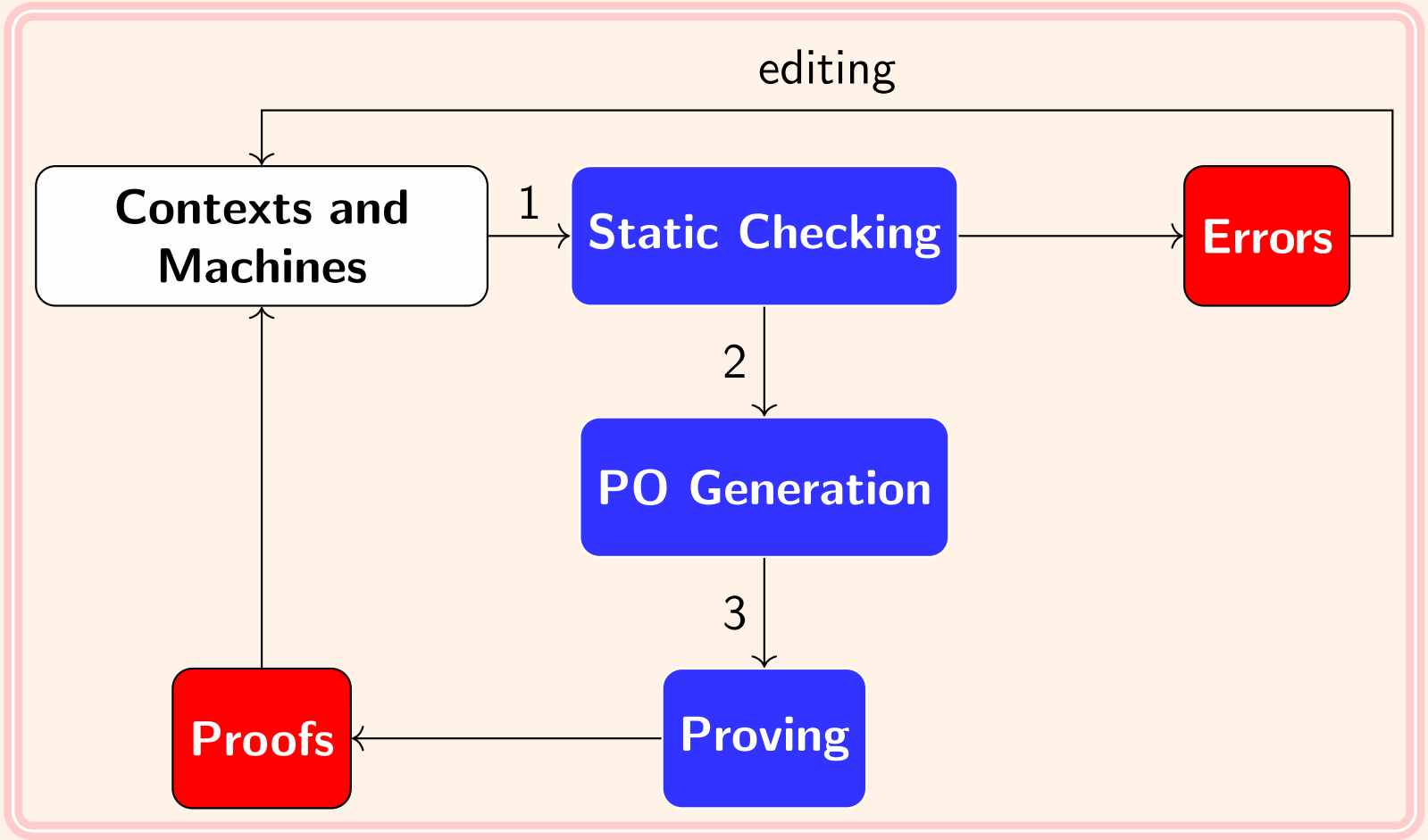
# Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event B models
5. Summary on the refinement

# Current Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event B models
5. Summary on the refinement

# Analysis of the Event-B Models



# Machines en Event B

**MACHINE**
$m$
**REFINES**
$am$
**SEES**
$c$
**VARIABLES**
$u$
**INVARIANTS**
$I(s, c, u)$
**THEOREMS**
$Q(s, c, u)$
**VARIANT**
$exp(s, c, u)$
**EVENTS**
$INITIALIZATION$
$\ldots$
$e$
$\ldots$
**END**

# Machines en Event B

**MACHINE**
$m$
**REFINES**
$am$
**SEES**
$c$
**VARIABLES**
$u$
**INVARIANTS**
$I(s, c, u)$
**THEOREMS**
$Q(s, c, u)$
**VARIANT**
$exp(s, c, u)$
**EVENTS**
$INITIALIZATION$
$\ldots$
$e$
$\ldots$
**END**

- $\Gamma(m)$ : environment for the machine $m$ defined by the context $c$ and it provides a list of seen axioms $Ax(s, c)$ and a list of seen theorems $Th(s, c)$ for the sets s and constants c.
- $\Gamma(m) \vdash \forall u.\textsc{Init}(s, c, u) \Rightarrow \mathrm{I}(s, c, u)$
- For each event $e$ in $E$ :
  $\Gamma(m) \vdash \forall u, u'.\mathrm{I}(s, c, u) \wedge BA(e)(u, u') \Rightarrow \mathrm{I}(u')$
- For each event $e$ in $E$ :
  $\Gamma(m) \vdash \forall u.\mathrm{I}(s, c, u) \wedge GRD(e)(s, c, u) \Rightarrow \exists u'.BA(e)(u, u')$
- $\Gamma(m) \vdash \forall u.\mathrm{I}(s, c, u) \Rightarrow \mathrm{Q}(s, c, u)$
- Generated proof obligations are derived from those conditions.

# Checking the well formation of Event-B expressions

- Event-B expressions are contexts, machines, properties, equations, set-theoretical expressions ...
- e is an Event-B expression and wd$(e)$ is a logical property expressing the well definition of e.
- wd$(1 = 2) \triangleq$ wd$(1) \wedge$ wd$(2)$
- wd$(a/b) \triangleq b \neq 0 \wedge$ wd$(a) \wedge$ wd$(b)$
- wd$(f(g)) \triangleq g \in dom(f) \wedge f \in A \pfun B$

# Current Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event B models
5. Summary on the refinement

# Current Sub Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event-B models
5. Summary on the refinement

# PO evt/grd/GRD

**EVENT** ae
**ANY** $x$ **WHERE**
$G(x, s, c, u)$
**THEN**
$u :| ABAP(x, s, c, u, u')$
**END**

**EVENT** ce
**REFINES**
$ae$
**ANY** $y$ **WHERE**
$H(y, s, c, v)$
**WITH**
$x : W(x, y, s, c, v)$
**THEN**
$v :| CBAP(y, s, c, v, v')$
**END**

| | |
|---|---|
| $s$ | *seen sets* |
| $c$ | *seen constants* |
| $u, v$ | *abstract and concrete variables* |
| $Ax(s, c)$ | *seen axioms* |
| $Th(s, c)$ | *seen theorems* |
| $I(s, c, u)$ | *abstract invariants* |
| $J(s, c, u, v)$ | *concrete invariants* |
| $Q(s, c, u)$, $R(s, c, u, v)$ | *abstract and concrete theorems* |
| ae, ce | *abstract and concrete event name* |
| x,y | *event parameters* |
| $G(x, s, c, u)$ | *abstract event guard* |
| $H(y, s, c, v)$ | *concrete event guard* |
| $ABAP(x, s, c, u, u')$ | *abstract event before-after predic* |
| $CBAP(x, s, c, u, u')$ | *concrete event before-after predic* |
| W(x,y,s,c,v) | witness predicate |

**PO** evt/grd/GRD

$Ax(s, c), Th(s, c), I(s, c, u), J(s, c, u, v), W(x, y, s, c, v), H(y, s, c, v), \vdash$
$G(x, s, c, u, u')$

# Current Sub Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event-B models
5. Summary on the refinement

# PO evt/act/SIM

**EVENT** ae
**ANY** $x$ **WHERE**
$G(x, s, c, u)$
**THEN**
$u :| ABAP(x, s, c, u, u')$
**END**

**EVENT** ce
**REFINES**
$ae$
**ANY** $y$ **WHERE**
$H(y, s, c, v)$
**WITH**
$x : WP(x, y, s, c, v)$
$u' : WV(y, u', s, c, v)$
**THEN**
$v :| CBAP(y, s, c, v, v')$
**END**

| | |
|---|---|
| $s$ | *seen sets* |
| $c$ | *seen constants* |
| $u, v$ | *abstract and concrete variables* |
| $Ax(s, c)$ | *seen axioms* |
| $Th(s, c)$ | *seen theorems* |
| $I(s, c, u)$ | *abstract invariants* |
| $J(s, c, u, v)$ | *concrete invariants* |
| $Q(s, c, u)$, $R(s, c, u, v)$ | *abstract and concrete theorems* |
| ae, ce | *abstract and concrete event name* |
| x,y | *event parameters* |
| $G(x, s, c, u)$ | *abstract event guard* |
| $H(y, s, c, v)$ | *concrete event guard* |
| $ABAP(x, s, c, u, u')$ | *abstract event before-after predic* |
| $CBAP(x, s, c, u, u')$ | *concrete event before-after predic* |
| $WP(x, y, s, c, v)$ | witness parameter predicate |
| $WV(y, u', s, c, v)$ | witness variable predicate |

**PO** evt/act/SIM

$$\left(\begin{array}{l} Ax(s, c), Th(s, c), I(s, c, u), J(s, c, u, v) \\ WP(x, y, s, c, v), WV(y, u', s, c, v) \\ H(y, s, c, v), CBAP(y, s, c, v, v') \end{array}\right) \vdash ABAP(x, s, c, u, u')$$

# Current Sub Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of EVENT-B models
5. Summary on the refinement

# PO evt/act/SIM

**EVENT** ae
  **ANY** $x$ **WHERE**
    $G(x, s, c, u)$
  **THEN**
    $u :| BAP(x, s, c, u, u')$
  **END**
...
**VARIANT**
    $exp(s, c, u)$

| | |
|---|---|
| $s$ | *seen sets* |
| $c$ | *seen constants* |
| $u, v$ | *abstract and concrete variables* |
| $Ax(s, c)$ | *seen axioms* |
| $Th(s, c)$ | *seen theorems* |
| $I(s, c, u)$ | *abstract invariants* |
| $J(s, c, u, v)$ | *concrete invariants* |
| $Q(s, c, u)$, $R(s, c, u, v)$ | *abstract and concrete theorems* |
| evt, ce | *event name* |
| x | *event parameters* |
| $G(x, s, c, u)$ | *abstract event guard* |
| $BAP(x, s, c, u, u')$ | *event before-after predicate* |
| $exp(s, c, u)$ | *aritthmetic expression* |

**PO** evt/NAT

$$Ax(s, c), Th(s, c), I(s, c, u), J(s, c, u, v), G(x, s, c, u) \vdash exp(s, c, u) \in \mathbb{N}$$

# Current Sub Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event-B models
5. Summary on the refinement

# PO evt/act/SIM

**EVENT** ae
  **ANY** $x$ **WHERE**
    $G(x, s, c, u)$
  **THEN**
    $u :| BAP(x, s, c, u, u')$
  **END**
...
**VARIANT**
    $exp(s, c, u)$

| | |
|---|---|
| $s$ | *seen sets* |
| $c$ | *seen constants* |
| $u, v$ | *abstract and concrete variables* |
| $Ax(s, c)$ | *seen axioms* |
| $Th(s, c)$ | *seen theorems* |
| $I(s, c, u)$ | *abstract invariants* |
| $J(s, c, u, v)$ | *concrete invariants* |
| $Q(s, c, u)$, $R(s, c, u, v)$ | *abstract and concrete theorems* |
| evt, ce | *event name* |
| x | *event parameters* |
| $G(x, s, c, u)$ | *abstract event guard* |
| $BAP(x, s, c, u, u')$ | *event before-after predicate* |
| $setexp(s, c, u)$ | *set expression* |

**PO** evt/NAT $Ax(s, c), Th(s, c), I(s, c, u), J(s, c, u, v), G(x, s, c, u) \vdash$
$finite(setexp(s, c, u))$

# Current Sub Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event-B models
5. Summary on the refinement

# PO evt/VAR

**EVENT** ae
  **ANY** $x$ **WHERE**
    $G(x, s, c, u)$
  **THEN**
    $u :\mid BAP(x, s, c, u, u')$
  **END**
...
**VARIANT**
    $exp(s, c, u)$

| | |
|---|---|
| $s$ | *seen sets* |
| $c$ | *seen constants* |
| $u, v$ | *abstract and concrete variables* |
| $Ax(s, c)$ | *seen axioms* |
| $Th(s, c)$ | *seen theorems* |
| $I(s, c, u)$ | *abstract invariants* |
| $J(s, c, u, v)$ | *concrete invariants* |
| $Q(s, c, u)$, $R(s, c, u, v)$ | *abstract and concrete theorems* |
| evt, ce | *event name* |
| x | *event parameters* |
| $G(x, s, c, u)$ | *abstract event guard* |
| $BAP(x, s, c, u, u')$ | *event before-after predicate* |
| $exp(s, c, u)$ | *aritthmetic expression* |

**PO** evt/VAR

$$Ax(s, c), Th(s, c), I(s, c, u), J(s, c, u, v), G(x, s, c, u), BAP(x, s, c, u, u') \vdash exp(s, c, u') < exp(s, c, u)$$

# Current Sub Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event-B models
5. Summary on the refinement

# PO evt/VAR

**EVENT** ae
  **ANY** $x$ **WHERE**
    $G(x, s, c, u)$
  **THEN**
    $u :\mid BAP(x, s, c, u, u')$
  **END**
...
**VARIANT**
    $setexp(s, c, u)$

| | |
|---|---|
| $s$ | *seen sets* |
| $c$ | *seen constants* |
| $u, v$ | *abstract and concrete variables* |
| $Ax(s, c)$ | *seen axioms* |
| $Th(s, c)$ | *seen theorems* |
| $I(s, c, u)$ | *abstract invariants* |
| $J(s, c, u, v)$ | *concrete invariants* |
| $Q(s, c, u)$, $R(s, c, u, v)$ | *abstract and concrete theorems* |
| evt, ce | *event name* |
| x | *event parameters* |
| $G(x, s, c, u)$ | *abstract event guard* |
| $BAP(x, s, c, u, u')$ | *event before-after predicate* |
| $setexp(s, c, u)$ | *set-theoretic expression* |

**PO** evt/VAR

$$Ax(s, c), Th(s, c), I(s, c, u), J(s, c, u, v), G(x, s, c, u), BAP(x, s, c, u, u') \vdash setexp(s, c, u') \subset setexp(s, c, u)$$

# Current Sub Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event-B models
5. Summary on the refinement

# PO evt/x/WFIS

**EVENT** ae
**ANY** $x$ **WHERE**
$G(x, s, c, u)$
**THEN**
$u :| ABAP(x, s, c, u, u')$
**END**

**EVENT** ce
**REFINES**
$ae$
**ANY** $y$ **WHERE**
$H(y, s, c, v)$
**WITH**
$x : WP(x, y, s, c, v)$
$u' : WV(y, u', s, c, v)$
**THEN**
$v :| CBAP(y, s, c, v, v')$
**END**

| | |
|---|---|
| $s$ | *seen sets* |
| $c$ | *seen constants* |
| $u, v$ | *abstract and concrete variables* |
| $Ax(s, c)$ | *seen axioms* |
| $Th(s, c)$ | *seen theorems* |
| $I(s, c, u)$ | *abstract invariants* |
| $J(s, c, u, v)$ | *concrete invariants* |
| $Q(s, c, u)$, $R(s, c, u, v)$ | *abstract and concrete theorems* |
| ae, ce | *abstract and concrete event name* |
| x,y | *event parameters* |
| $G(x, s, c, u)$ | *abstract event guard* |
| $H(y, s, c, v)$ | *concrete event guard* |
| $ABAP(x, s, c, u, u')$ | *abstract event before-after predic* |
| $CBAP(x, s, c, u, u')$ | *concrete event before-after predic* |
| $WP(x, y, s, c, v)$ | witness parameter predicate |
| $WV(y, u', s, c, v)$ | witness variable predicate |

**PO** evt/x/WFIS

$Ax(s, c), Th(s, c), I(s, c, u), J(s, c, u, v), H(y, s, c, v) \vdash$
$\exists x.WP(x, y, s, c, v)$

# Current Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event B models
5. Summary on the refinement

# The factorial model

CONTEXT $functions$
CONSTANTS $factorial\ n0\ r0$

AXIOMS
$@req1\ n0 \in \mathbb{N}$
$@req2\ r0 \in \mathbb{Z}$
$@axm2\ factorial \in \mathbb{N} \leftrightarrow \mathbb{N}$
$@axm3\ 0 \mapsto 1 \in factorial$
$@axm4\ \forall a.\forall b\,.\, a \in \mathbb{N} \land b \in \mathbb{N} \land a \mapsto b \in factorial \Rightarrow a+1 \mapsto (a+1)*b \in factorial$
$@axm5\ \forall f\,.\, f \in \mathbb{N} \leftrightarrow \mathbb{N} \land 0 \mapsto 1 \in f \land (\forall a,b\,.\, a \in \mathbb{N} \land b \in \mathbb{N} \land a \mapsto b \in f \Rightarrow a+1 \mapsto (a+1)*b \in f) \Rightarrow factorial \subseteq f$
$theorem\ @th1\ factorial \in \mathbb{N} \to \mathbb{N}$
$theorem\ @th2\ factorial(0) = 1$
$theorem\ @th3\ \forall u\,.\, u \in \mathbb{N} \land u \neq 0 \Rightarrow factorial(u) = u * factorial(u-1)$

- The context defines the mathematical function *factorial* by axioms 2,3,4 and 5 and the properties th1, thh2 and th3 as theorems.
- *n0* is a constant defined as a natural number.

# The factorial pre/post specification

MACHINE
$prepostok$
SEES $functions$
VARIABLES $r\ ok\ n$
INVARIANTS
$@inv1\ r \in \mathbb{Z}$
$@inv2\ n \in \mathbb{Z}$
$@inv3\ ok \in BOOL$
$@inv4\ n = n0$
$@post\ ok = TRUE \Rightarrow r = factorial(n)$
EVENTS
EVENT $INITIALISATION$
then
$@act1\ r :\in \mathbb{Z}$
$@act2\ ok := FALSE$
$@act3\ n := n0$
end
EVENT $computing1$
where
$@grd1\ ok = FALSE$
then
$@act1\ ok := TRUE$
$@act2\ r := factorial(n)$
end

- Three variables are used for exprressing the end of the computation by assigning TRUE to *ok* , the mathematical value of *factorial(n)* to *r* and the variable *n* to *n0*.
- The event final/call the function *observes* the call of the *hidden* function computing the *factorial* function.
- The statement $r := factorail(n)$ expressses the relation between the mathematical value and the model variable *r*.
- Invariant *post* expresses the postcondition.

# The factorial computing : defining the state

MACHINE $computing$
REFINES $prepostok$
SEES $functions$
VARIABLES $r\ fac\ k\ n\ ok$

INVARIANTS
$@inv1\ fac \in \mathbb{N} \pfun \mathbb{N}$
$@inv2\ dom(fac) \subseteq 0..n$
$@inv3\ dom(fac) \neq \varnothing$
$@inv4\ \forall\ i\ .\ i \in dom(fac) \Rightarrow fac(i) = factorial(i)$
$@inv5\ k \in dom(fac)$
$@inv6\ dom(fac) = 0..k$
$@inv8\ ok = TRUE \Rightarrow n \in dom(fac)$
$theorem\ @th1\ n = n0$
$theorem\ @inv7\ k \in 0..n0$
VARIANT $0..n \setminus dom(fac)$

- Previous variables $n$,$r$,$ok$ are named in the same way and abstract variables are equal to concrete variables.
- Two new variables are introduced namey $k$ and $fac$.
- $fac$ is the partial copy of the function $factorial$ until $k$.
- $k$ defines the domain of $fac$.
- $fac$ is a subset of $factorial$ by $inv4$.

**The clause VARIANT is new and expresses that the new machine or view has a new event which can not always take the control over the previous abstract events as computing1 appreaing as computing2 in the new machine.**

# The factorial computing : refining the computing

EVENT $INITIALISATION$
REFINES $INITIALISATION$
then
$@act1\ r :\in \mathbb{Z}$
$@act2\ fac := \{0 \mapsto 1\}$
$@act3\ k := 0$
$@act4\ n := n0$
$@act5\ ok := FALSE$
end
EVENT $computing2$
REFINES $computing1$
where
$@grd1\ n \in dom(fac)$
$@grd2\ ok = FALSE$
then
$@act1\ r := fac(n)$
$@act2\ ok := TRUE$
end

- The two events are refined and are made more concrete by adding the two new variables.
- The event computing2 is concretizing the guard by the gaurad $n \in dom(fac)$.
- The computation is completed , when one can get the value corresponding to $factorial(n)$ which is stored in $fac$ by the event step2.
- The concrete guard of the event computing2 is stronger than the abstract guard of the event computing1.

The concrete action of the event computing2 is simulating the abstract event of the event computing1 (computing2/act2/SIM).

# The factorial computing : progressing for computing

$convergent$ EVENT $step2$
where
$@grd13\ n \notin dom(fac)$
$@grd14\ ok = FALSE$
then
$@act11\ fac(k+1) := (k+1) * fac(k)$
$@act1\ k := k+1$
end

- The event step2 is new and refines skip.
- The status is set to convergent and means that its occurrence decreases the value of the variant.
- It helps to converge.

The PO steps2/VAR expresses the decreasing of the variant :

$$0 \mathinner{..} n \setminus dom(fac \lhd \{ k+1 \mapsto (k+1) * fac(k) \}) \subset 0 \mathinner{..} n \setminus dom(fac)$$

# Intermediate summary for the refinement process

- The current machines expresses the computing process of the variable *fac* as well as an index *k*.
- The computation of factorial(n) does not require the storing of intermediate values.
- *fac* is a *model* variable and we are now introducing the last required value for the computation.

# The factorial pre-algorithmic machine

MACHINE $prealgo$
REFINES $computing$
SEES $functions$

VARIABLES $r\ fac\ k\ n\ ok\ ffac$

INVARIANTS
$@inv1\ ffac \in \mathbb{Z}$
$@inv2\ ffac = fac(k)$
$theorem\ @inv3\ dom(fac) = 0..k$
$theorem\ @inv4\ \forall i.\ i \in dom(fac) \Rightarrow fac(i) = factorial(i)$
$@inv5\ ok = TRUE \Rightarrow k = n$

VARIANT $n - k$

- A new variable is introduced namely *ffac* and its roe is to keep the last computed value of *factorial*.
- *inv2* is a key invariant stating the relation between *ffac* and *fac*.
- The invarant *inv5* is relating *ok* and *k*.

# The factorial pre-algorithmic machine

$Findfile:$

- The event computing3 is refining an event with a guard diffcult to evaluate $n \in \mathrm{dom}(fac)$ by strengethening the guard into the following form $k = n$.
- The action of computing3 is simple too and *ffac* contains the correct value to assign.

# The factorial pre-algorithmic machine

EVENT $step3$ REFINES $step2$
where
$@grd1\ k \neq n$
$@grd2\ ok = FALSE$
then
$@act1\ fac(k+1) := (k+1) * fac(k)$
$@act2\ k := k+1$
$@act3\ ffac := (k+1) * ffac$
end

- The same transformation of the event step2 into step3 by using the condition $k \neq n$ instead of $n \notin dom(fac)$.
- The action $ffac := (k+1) * ffac$ is added and simulates the inductive defintion.

# Intermediate summary for the refinement process

- The guards are made simpler and are easier to compute.
- The variable *ffac* is simulating the inductive step.
- The variable *fac* can be *hidden* and remains a model variable but is not implmented.

# The factorial algorithmic machine

MACHINE $algo$
REFINES $prealgo$
SEES $functions$

VARIABLES $r\ k\ n\ ffac\ ok$

INVARIANTS
$theorem\ @th1\ ffac = fac(k)$
$theorem\ @th2\ k \in 0..n$
$theorem\ @th3\ n = n0$
$theorem\ @th4\ ok = TRUE \Rightarrow k = n$
$theorem\ @th5\ ok = TRUE \Rightarrow r = factorial(n)$

- The new model is hidden the variable *fac* which is a model variable.
- Properties of variables are inherited as theorems from the previous steps.
- The theorem th5 relains the expression of the partial correctness derived from the process.

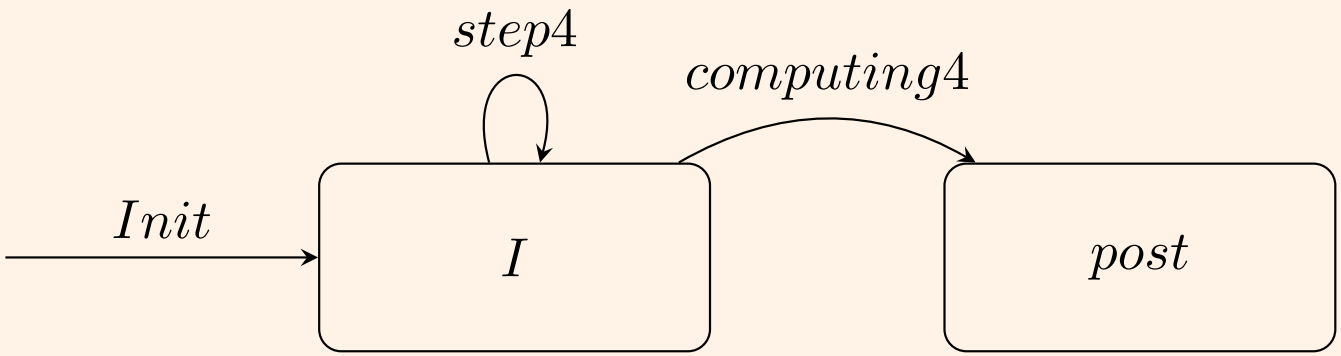
# The factorial algorithmic machine

EVENT $INITIALISATION$
then
$@act1\ r :\in \mathbb{Z}$
$@act3\ k := 0$
$@act4\ n := n0$
$@act6\ ffac := 1$
$@act7\ ok := FALSE$
end
EVENT $computing4$
REFINES $computing3$
where
$@grd1\ k = n$
$@grd2\ ok = FALSE$
then
$@act1\ r := ffac$
$@act2\ ok := TRUE$
end

EVENT $step4$ REFINES $step3$
where
$@grd1\ k \neq n$
$@grd2\ ok = FALSE$
then
$@act2\ k := k + 1$
$@act3\ ffac := (k + 1) * ffac$
end
$end$

**The events are no more containing the variable fac. Guards are simpler to transform into an algorithmic statement.**

# Summary of the refinement-based development




- The last machine is simple and has bo more problmetic expressions.
- The diagram expresses that step4 is observed until the event computing4 is observed.
- The diagram leads to an iteration.

# Current Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event B models
5. Summary on the refinement

## Merging Rules for transforming Event B models into algorithms by J.-R. Abrial




**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

# Merging Rules for transforming Event B models

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

**WHEN**
$P$
$\neg Q$
**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

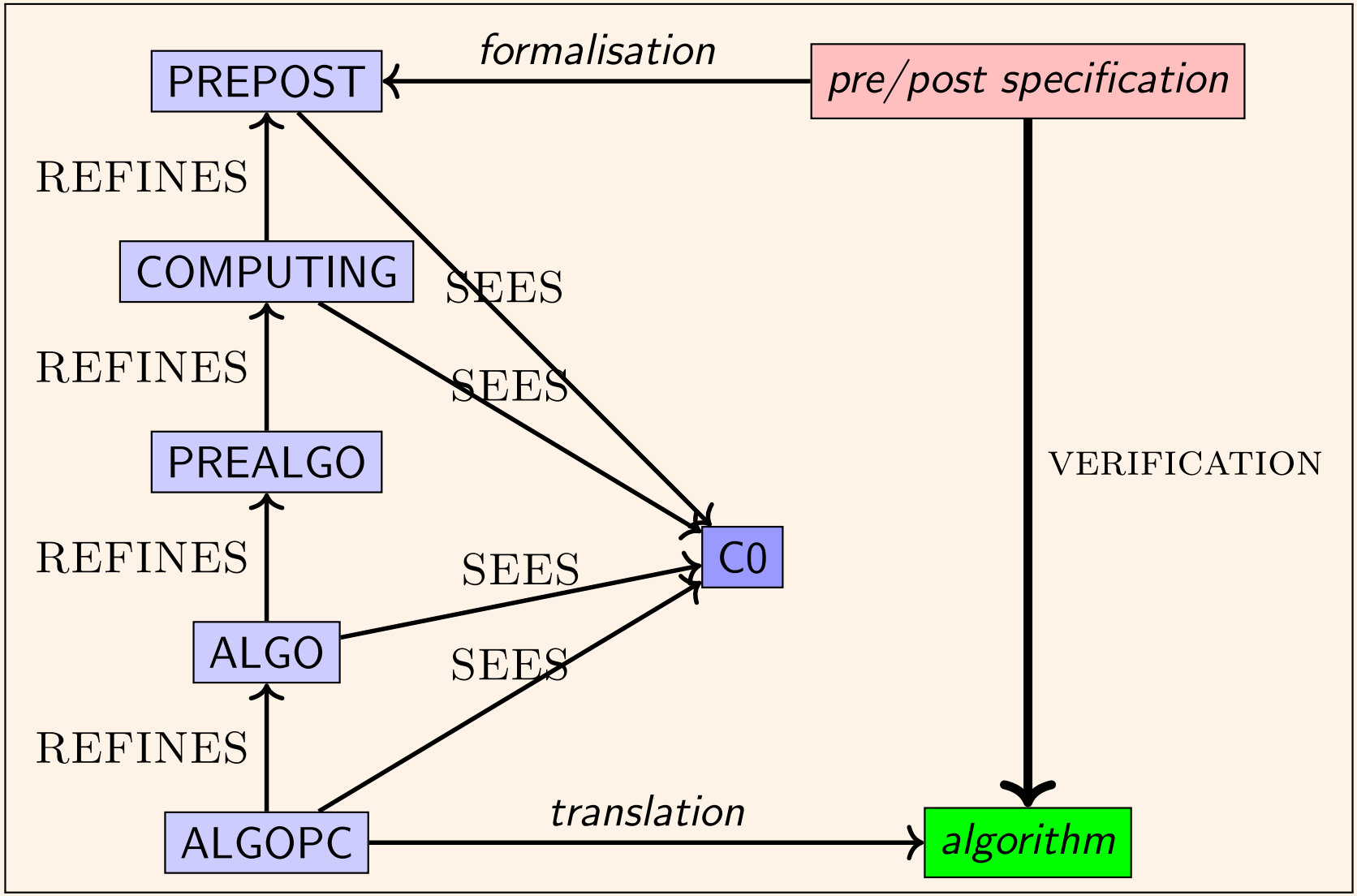
# Deriving an algorithm

VARIABLES $r, k, n, ffac, ok$
requires $n0 \in \mathbb{N}$
ensures $r = factorial(n0)$
$begin$
$k := 0; n := n0; ffac := 1; ok := FALSE;$
while $k \neq n$ do
    $k := k + 1;$
    $ffac := k * ffac;$
od;
$r := ffac;$
$ok := TRUE;$
end

# Current Section Summary


1. Overview of machines, contexts and proof obligations
2. Proof Obligations for Refinement
   - PO evt/grd/GRD
   - PO evt/act/SIM
   - PO evt/NAT
   - PO NAT
   - PO evt/VAR (arithmetic)
   - PO evt/VAR (set-theoretic)
   - PO evt/x/WFIS
3. Example of the factorial function refined into an algorithm
4. Transformations of Event B models
5. Summary on the refinement

# The Iterative Pattern (I)



# Summary on the refinement

- Correctness by construction helps to design both the algorithm and its proof
- Proofs are progressive.
- Invariants are discovered in a step by step way and are derived from the intermediate steps.

# Summary on the refinement

- Correctness by construction helps to design both the algorithm and its proof
- Proofs are progressive.
- Invariants are discovered in a step by step way and are derived from the intermediate steps.
- Inductive definitions helps to define invariants :

$$
\begin{array}{l}
u_0 \in D \\
\forall i.i \geq 0 \Rightarrow u_{i+1} = f(u_i) \\
\texttt{ff} = code(f) \\
\texttt{uu} : type(D) \\
init : \texttt{uu} := u_0 \\
next : \texttt{uu} := \texttt{ff}(\texttt{uu})
\end{array}
$$

# Summary on the refinement

- Correctness by construction helps to design both the algorithm and its proof
- Proofs are progressive.
- Invariants are discovered in a step by step way and are derived from the intermediate steps.
- Inductive definitions helps to define invariants :

$$
\begin{array}{l}
u_0 \in D \\
\forall i.i \geq 0 \Rightarrow u_{i+1} = f(u_i) \\
\texttt{ff} = code(f) \\
\texttt{uu} : type(D) \\
init : \texttt{uu} := u_0 \\
next : \texttt{uu} := \texttt{ff}(\texttt{uu})
\end{array}
$$

- Chain of refinements relates machines of the same langauge
- Input of the chain is the contract
- Output of the chain is an algorithm.