Modelling Software-based Systems

Proof Obligation Generation for Machines and Verification Machines

Telecom Nancy (IL et LE)

Dominique Méry
Telecom Nancy,Université de Lorraine

23 novembre 2026

dominique.mery@loria.fr

Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
   - PO thm/THM (context)
   - PO th/THM (machine)
   - PO evt/inv/INV
   - PO evt/act/FIS
4. Checking contracts using Event-B on the FIDE Rodin
   - Method for proving invariance properties
   - Verification of contract
   - Definition of machine for checking contract
   - Soundness of the technique
5. Example
6. Conclusion

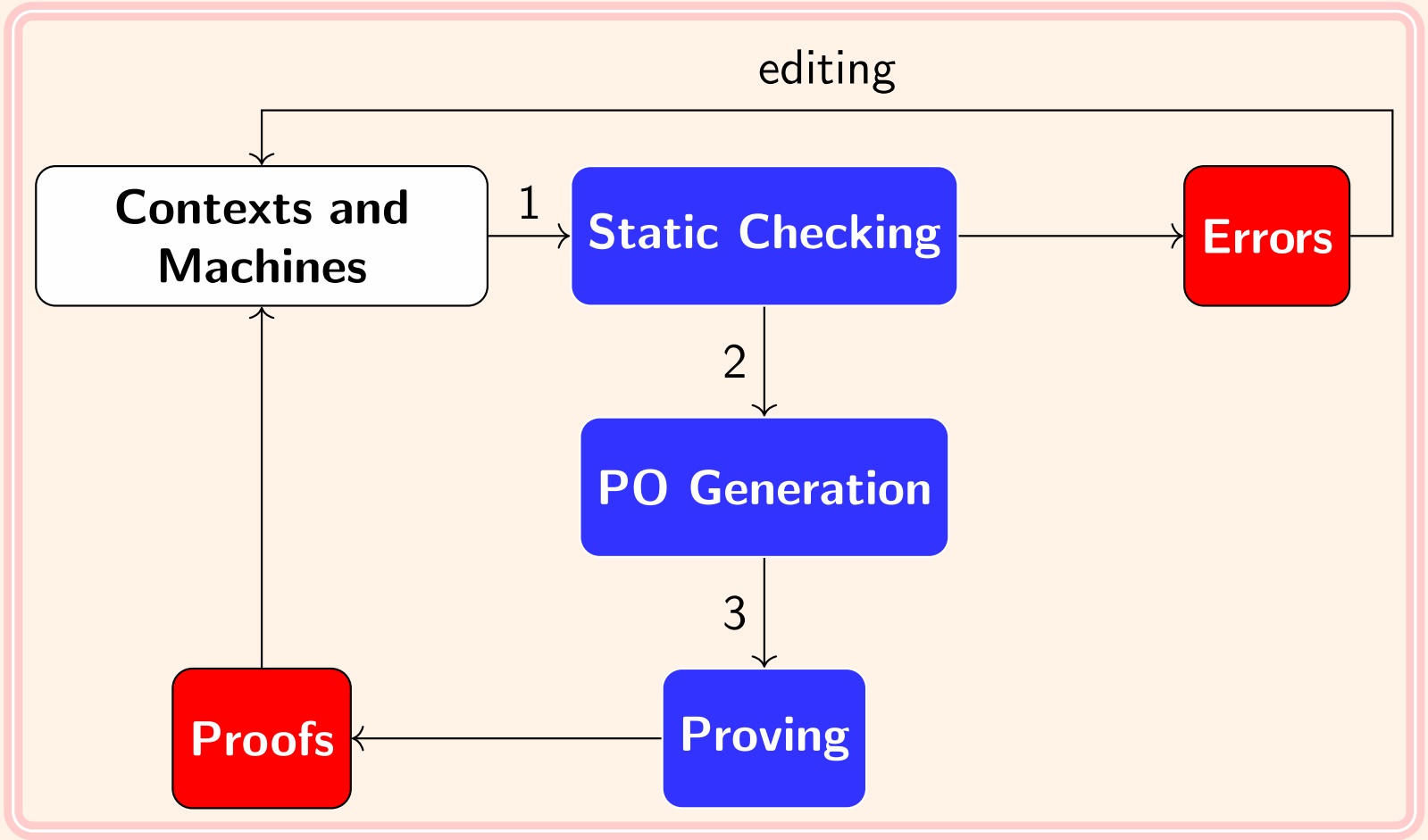
Current Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
   - PO thm/THM (context)
   - PO th/THM (machine)
   - PO evt/inv/INV
   - PO evt/act/FIS
4. Checking contracts using Event- B on the FIDE Rodin
   - Method for proving invariance properties
   - Verification of contract
   - Definition of machine for checking contract
   - Soundness of the technique
5. Example
6. Conclusion

Current Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
   - PO thm/THM (context)
   - PO th/THM (machine)
   - PO evt/inv/INV
   - PO evt/act/FIS
4. Checking contracts using Event- B on the FIDE Rodin
   - Method for proving invariance properties
   - Verification of contract
   - Definition of machine for checking contract
   - Soundness of the technique
5. Example
6. Conclusion

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 $\mathsf{wd}(e)$ is a logical property expressing the well definition of e.
- $\mathsf{wd}(1 = 2) \triangleq \mathsf{wd}(1) \wedge \mathsf{wd}(2)$
- $\mathsf{wd}(a/b) \triangleq b \neq 0 \wedge \mathsf{wd}(a) \wedge \mathsf{wd}(b)$
- $\mathsf{wd}(f(g)) \triangleq g \in dom(f) \wedge f \in A \nrightarrow B$

Current Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
   - PO thm/THM (context)
   - PO th/THM (machine)
   - PO evt/inv/INV
   - PO evt/act/FIS
4. Checking contracts using Event- B on the FIDE Rodin
   - Method for proving invariance properties
   - Verification of contract
   - Definition of machine for checking contract
   - Soundness of the technique
5. Example
6. Conclusion

Current Sub Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
   - PO thm/THM (context)
   - PO th/THM (machine)
   - PO evt/inv/INV
   - PO evt/act/FIS
4. Checking contracts using Event- B on the FIDE Rodin
5. Example
6. Conclusion

PO th/THM

CONTEXTS
c
EXTENDS
ac
SETS
s
CONSTANTS
c
AXIOMS
$Ax(s,c)$
THEOREMS
$th_1 : P_1(s,c)$
$\ldots$
$th_n : P_n(s,c)$
$th : P(s,c)$
$\ldots$
END

s	*seen sets*
c	*seen constants*
$Ax(s,c)$	*seen axioms*
$Th(s,c)$	*previous proved theorems*
	$PTh(s,c) = \{P_i(s,c) \mid i\ 1..n\}$
$P(s,c)$	*property over s and c*

PO th/THM

$Ax(s,c), Th(s,c) \vdash P(s,c)$

Current Sub Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
   - PO thm/THM (context)
   - PO th/THM (machine)
   - PO evt/inv/INV
   - PO evt/act/FIS
4. Checking contracts using Event- B on the FIDE Rodin
5. Example
6. Conclusion

post

PO th/THM (machine)

MACHINE
m
...
VARIABLES
u
INVARIANTS
$I(s, c, u)$
THEOREMS
$Q(s, c, u)$
$th : P(s, c, u)$
...
END

s	*seen sets*
c	*seen constants*
u	*variables*
$Ax(s, c)$	*seen axioms*
$Th(s, c)$	*seen theorems*
$I(s, c, u)$	*invariants*
$Q(s, c, u)$	*theorems*
$P(s, c, u)$	*property over s,c and u*

PO th/THM

$$Ax(s, c), Th(s, c), I(s, c, u) \vdash P(s, c, u)$$

Current Sub Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
   - PO thm/THM (context)
   - PO th/THM (machine)
   - PO evt/inv/INV
   - PO evt/act/FIS
4. Checking contracts using Event- B on the FIDE Rodin
5. Example
6. Conclusion

PO evt/inv/INV

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

$BA(\text{evt}) \ \widehat{=}$
$\exists x. \left(\begin{array}{l} \wedge\ G(x, s, c, u) \\ \wedge\ BAP(x, s, c, u, u') \end{array} \right)$
$GRD(\text{evt}) \ \widehat{=}\ G(x, s, c, u)$
$ACT(\text{evt}) \ \widehat{=}\ BAP(x, s, c, u, u')$

s	*seen sets*
c	*seen constants*
u	*variables*
$Ax(s, c)$	*seen axioms*
$Th(s, c)$	*seen theorems*
$I(s, c, u)$	*invariants*
$Q(s, c, u)$	*theorems*
evt	*event name*
x	*event parameter*
$G(x, s, c, u)$	*event guard*
$BAP(x, s, c, u, u')$	*event before-after predicate*
$inv : inv(s, c, u')$	*specific modified invariant*

PO evt/inv/INV

$$Ax(s, c), Th(s, c), I(s, c, u), G(x, s, c, u), BAP(x, s, c, u, u') \vdash inv(s, c, u')$$

PO Q/THM $Ax(s, c), Th(s, c), I(s, c, u) \vdash Q(s, c, u)$

Current Sub Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
   - PO thm/THM (context)
   - PO th/THM (machine)
   - PO evt/inv/INV
   - PO evt/act/FIS
4. Checking contracts using Event- B on the FIDE Rodin
5. Example
6. Conclusion

PO evt/act/FIS

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

$BA(\text{evt}) \mathrel{\widehat{=}}$
$\left(\begin{array}{l} \wedge\ G(x, s, c, u) \\ \wedge\ BAP(x, s, c, u, u') \end{array} \right)$
$GRD(\text{evt}) \mathrel{\widehat{=}} G(x, s, c, u)$
$ACT(\text{evt}) \mathrel{\widehat{=}}$
$BAP(x, s, c, u, u')$

s	*seen sets*
c	*seen constants*
u	*variables*
$Ax(s, c)$	*seen axioms*
$Th(s, c)$	*seen theorems*
$I(s, c, u)$	*invariants*
$Q(s, c, u)$	*theorems*
evt	*event name*
x	*event parameter*
$G(x, s, c, u)$	*event guard*
$BAP(x, s, c, u, u')$	*event before-after predicate*

PO evt/act/FIS

$Ax(s, c), Th(s, c), I(s, c, u), G(x, s, c, u), \vdash \exists u'.BAP(x, s, c, u, u')$

Current Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
   - PO thm/THM (context)
   - PO th/THM (machine)
   - PO evt/inv/INV
   - PO evt/act/FIS
4. Checking contracts using Event- B on the FIDE Rodin
   - Method for proving invariance properties
   - Verification of contract
   - Definition of machine for checking contract
   - Soundness of the technique
5. Example
6. Conclusion

Current Sub Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
4. Checking contracts using Event- B on the FIDE Rodin
   - Method for proving invariance properties
   - Verification of contract
   - Definition of machine for checking contract
   - Soundness of the technique
5. Example
6. Conclusion

Contract verification : what the technique involves

A program P *fulfils* a contract (pre,post) :

- P transforms a variable v starting from an initial value v_0 and producing a final value v_f : $v_0 \xrightarrow{P} v_f$
- v_0 satisfies pre : pre(v_0) and v_f satisfies post : post(v_0, v_f)
- pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables v

begin
$0 : P_0(v_0, v)$
$\texttt{instruction}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(v_0, v)$
end

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- verification conditions for all pairs $\ell \longrightarrow \ell'$

Proving invariance and safety properties

Design of a method for proving invariance and safety properties for a programming language

- We consider a classical programming language denoted PROGRAMS
- and we assume that this programming language supports assignment, conditional statements, bounded iteration, unbounded iteration, simple or structured variables such as arrays, and the definition of constants.
- Let us consider a program P in PROGRAMS ; this program comprises
  - globally named variables v,
  - globally named constants pc,
  - types associated with the globally named variables VALUES and identified ẁith a set of possible values for the variables,
  - instructions following an order defined by the syntax of the programming language.

Hypotheses

- We define a set of control points $\textsc{Locations}$
- for each program or algorithm P. $\textsc{Locations}$ is a finite set of values and a hidden variable denoted pc traverses this set according to the sequence.
- the space of possible values $\textsc{Values}$ is a Cartesian product of the form $\textsc{Locations} \times \textsc{Memory}$
- the variables x of the system are decomposed into two independent entities $x = (pc, v)$ subject to the conditions $pc \in \textsc{Locations}$ and $v \in \textsc{Memory}$.

$$x = (pc, v) \wedge pc \in \textsc{Locations} \wedge v \in \textsc{Memory} \tag{1}$$

We consider an annotated program P ; we are given a relational model $\mathcal{MP} = (Th(s, c), x, \textsc{Values}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ where

- $Th(s, c)$ is a theory defining the sets, constants and static properties of this program
- x is a list of flexible variables, and x comprises a control part and a memory part.
- $\textsc{Locations} \times \textsc{Memory}$ is a set of possible values for x.
- $\{r_0, \ldots, r_n\}$ is a finite set of relations linking the values before x and the values after x', and conforming to the succession relation $\longrightarrow$ between control points.
- $\textsc{Init}(x)$ defines the set of initial values of (pc_0, v) and $x = (pc_0, v)$ with $pre(v)$ which characterises the initial values of v at the initial point.

Assumptions regarding the calculation

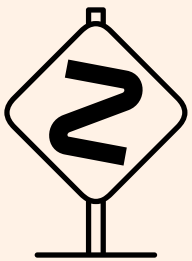
We assume that there exists a graph over the set of control values defining the flow relation, and we denote this structure $(\text{LOCATIONS}, \longrightarrow)$.

Definition

$\ell_1 \longrightarrow \ell_2 \stackrel{\Delta}{=} pc = \ell_1 \wedge pc' = \ell_2$

Definition

(Annotation of a control point)
Let $(\text{LOCATIONS}, \longrightarrow)$ be a structure and $\ell \in \text{LOCATIONS}$ a label ℓ. An annotation of a control point ℓ is a predicate $P_\ell(v)$ (assertion) or $P_\ell(v_0, v)$ (relational).



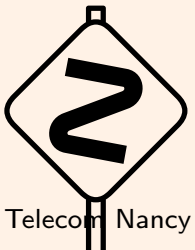
$P_\ell(v_0, v)$ expresses a relation between the initial value of V, denoted v_0, and v, the current value of V at point ℓ, and therefore $P_\ell(v_0, v) \Rightarrow pre(v_0)$ specifies that v_0 is an initial value.

Relationship between annotation and invariant

- The labels ℓ belong to $\textsc{Locations}$: $\ell \in \textsc{Locations}$.
- The variables v belong to $\textsc{Memory}$: $v \in \textsc{Memory}$.
- $pre(v_0)$ specifies the initial values of v.
- Whenever control is at ℓ, v satisfies $P_\ell(v)$: $pc = \ell \Rightarrow P_\ell(v_0, v)$.
- For any state (ℓ, v) of the program, the following property is true but must $\hat{\text{be}}$ proved :

$$J(\ell_0, v_0, pc, v) \triangleq \left[\begin{array}{l} \wedge\ pc \in \textsc{Locations} \\ \wedge\ v \in \textsc{Memory} \\ \ldots \\ \wedge\ pc = \ell \Rightarrow P_\ell(v_0, v) \\ \ldots \end{array}\right.$$

- $J(\ell_0, v_0, pc, v)$ is an invariant constructed $\grave{\text{from}}$ the annotations generated, but we must show that this invariant enables us to verify the three conditions of the induction principle.



ℓ_0 denotes the label marking the start of the algorithm and ℓ_f is the end of the program. We may simply use 0 and f.

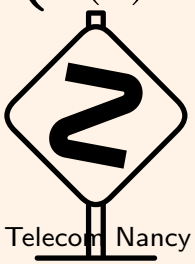
Checking to do

$$x = (pc, v) \text{ et } J(\ell_0, v_0, pc, v) \triangleq \left[\begin{array}{l} \wedge\ pc \in \textsc{Locations} \\ \wedge\ v \in \textsc{Memory} \\ \ldots \\ \wedge\ pc = \ell \Rightarrow P_\ell(v_0, v) \\ \ldots \end{array}\right.$$

Let $(Th(s,c), x, \textsc{Values}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ a relational model for a program. A property $\mathrm{A}(x_0, x)$ is a safety property for P if $\forall x_0, x \in \textsc{Locations} \times \textsc{Memory}. Init(x_0) \wedge x_0 \xrightarrow[\textsc{Next}]{} x \Rightarrow A(x)$.We know that this property implies that there exists a state property $\mathrm{I}(x_0, x)$ such that the three properties hold, but we apply this verification to J :

$\forall x_0, x, x' \in \textsc{Locations} \times \textsc{Memory}$:

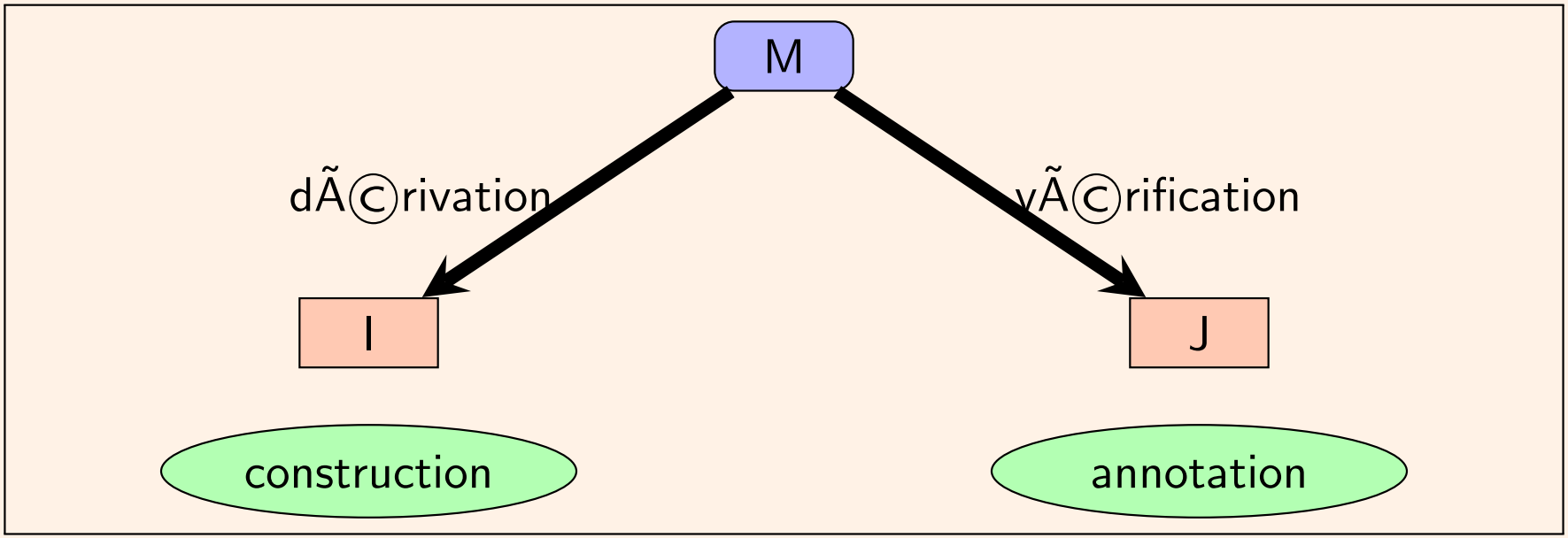
$$\begin{cases} (1)\ \ \textsc{Init}(x_0) \Rightarrow \mathrm{J}(x_0, x_0) \\ (2)\ \ \mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x) \\ (3)\ \ \forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x_0, x') \end{cases}$$



$\forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x_0, x')$ is équivalent to $\mathrm{J}(x_0, x) \wedge (\exists i \in \{0, \ldots, n\} : x\ r_i\ x') \Rightarrow \mathrm{J}(x_0, x')$

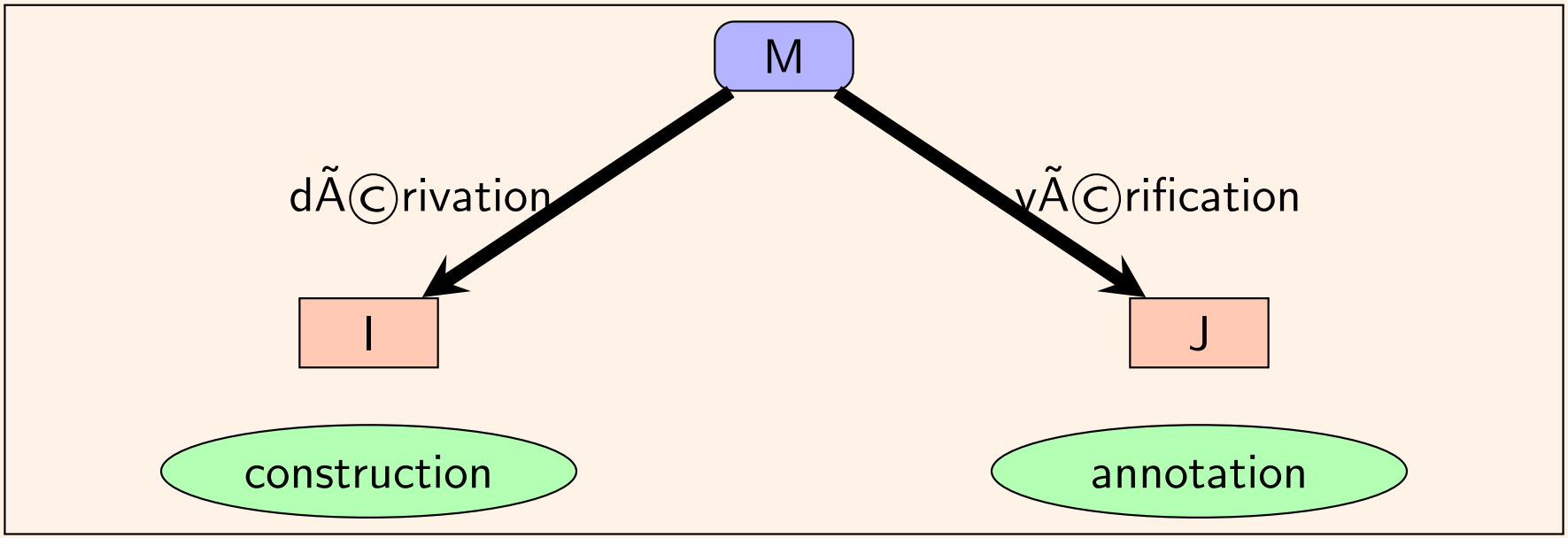
Checking to do

- Application of the correctness to the relational principle of induction : if we verify the three properties, then A is a safety property for the model in question (verification).
- If we wish to show that A is a safety property, then we must use the invariant to construct annotations for the model (derivation).



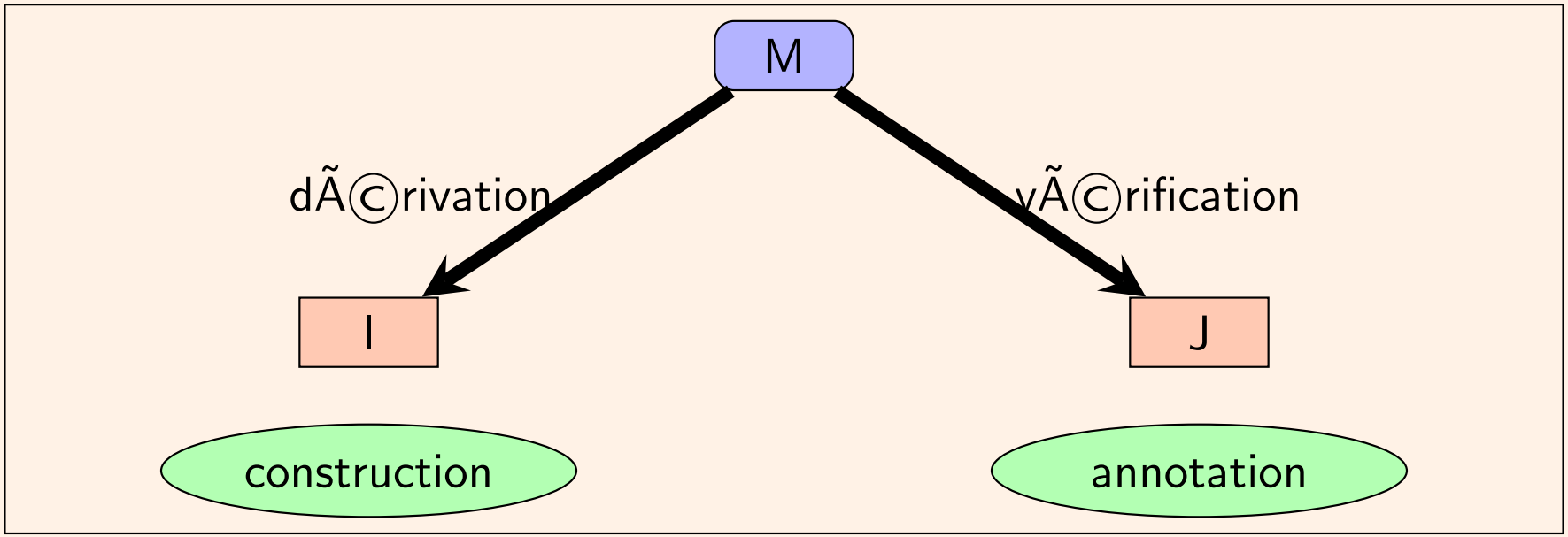
Using the relational induction principle (RI)

- $\textsc{Values} = \textsc{Locations} \times \textsc{Memory}$
- $J(pc_0, v_0, pc, v) \stackrel{\Delta}{=} \exists x_0, x \in \textsc{Values}. I(x_0, x) \wedge x = (pc, v) \wedge x_0 = (pc_0, v_0)$ (deduction)
- $I(x_0, x) \stackrel{\Delta}{=} \exists pc_0, pc \in \textsc{Locations}, v_0, v \in \textsc{Memory}. J(pc_0, v_0, pc, v) \wedge x = (pc, v) \wedge x_0 = (pc_0, v_0)$ (induction)



Using the assertional induction principle (AI)

- $\textsc{Values} = \textsc{Locations} \times \textsc{Memory}$
- $J(pc, v) \triangleq \exists x \in \textsc{Values}. I(x) \wedge x = (pc, v)$ (deduction)
- $I(x) \triangleq \exists pc \in \textsc{Locations}, v \in \textsc{Memory}. J(pc, v) \wedge x = (pc, v)$ (induction)



Alignment with programs

(1) $\forall x_0, x \in \textsc{Values}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow A(x)\ (I(x))$
(2) $\forall x_0, x \in \textsc{Values}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow R(x_0, x)\ (IR(x))$

Relations and definitions

$x = (\ell, v)$, $x_0 = (\ell_0, v_0)$, $I(x)$, $IR(x_0, x)$ and annotations $P_\ell(v)$, $RP_\ell(v_0, v)$ are linked in this way :

- $I(x) \triangleq \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v) \triangleq \exists x.(x \in \textsc{Values} \wedge x = (\ell, v) \wedge I(x))$
- $IR(x_0, x) \triangleq \exists \ell, v, v_0(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge RP_\ell(v_0, v))$
- $RP_\ell(v_0, v) \triangleq \exists x, x_0.(x, x_0 \in \textsc{Values} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge IR(x_0, x))$

The transformation is based on the transition relation defined for each pair of consecutive control labels, which is expressed very simply by the following relational form :

$$x\ r_{\ell,\ell'}\ x' \triangleq (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$$

Transition from one label to another label

- The transition from ℓ to ℓ' is possible when the condition $cond_{\ell,\ell'}(v)$ is true for V and when control is in ℓ ($pc = \ell$).
- When the transition is observed, the variables V are transformed as follows : $v' = f_{\ell,\ell'}(v)$.
- The definition of the transition does not express any assumptions relating to an execution strategy, such as fairness, for example.
- $cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v)$ is an expression in which the expressions $cond_{\ell,\ell'}(v)$ and $= f_{\ell,\ell'}(v)$ raise questions of definition :
  - $DOM(\ell,\ell')(v) \triangleq DEF(cond_{\ell,\ell'}(v))(v) \wedge DEF(f_{\ell,\ell'}(v))$
  - $DEF(E(X))(x)$ means that the expression $E(X)$ is defined for x, the current value of X.
- Some transitions may lead to bugs :
  - $DEF(X+1)(x) \triangleq x+1 \in D$, where D is the coding domain of X ; for example, $D = -2^{31}..2^{31}-1$ for a 32-bit coding scheme.
  - $DEF(T(I+1) < V)(t,x,v) \triangleq i+1 \in dom(t) \wedge v \in D \wedge t(i+1) \in D$

Relationships for program statements

$\ell : P_\ell(v_0, v)$
$V := f_{\ell,\ell'}(V)$
$\ell' : P_{\ell'}(v_0, v)$

$\ell_1 : P_{\ell_1}(v_0, v)$
WHILE $B(V)$ **DO**
  $\ell_2 : P_{\ell_2}(v_0, v)$
  $\ldots$
  $\ell_3 : P_{\ell_3}(v_0, v)$
END
$\ell_4 : P_{\ell_4}(v_0, v)$

Traduction

- $(pc = \ell \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $cond_{\ell,\ell'}(v) \stackrel{\Delta}{=} TRUE$

Traduction

$$\begin{cases} pc = \ell_1 \wedge b(v) \wedge v' = v \wedge pc' = \ell_2 \\ pc = \ell_1 \wedge \neg b(v) \wedge v' = v \wedge pc' = \ell_4 \\ pc = \ell_3 \wedge b(v) \wedge v' = v \wedge pc' = \ell_2 \\ pc = \ell_3 \wedge \neg b(v) \wedge v' = v \wedge pc' = \ell_4 \end{cases}$$

Checking the contract : what the method will be

A program P *satisfies* a contract (pre,post) :

- P transforms a variable v starting from an initial value v_0 and producing a final value v_f : $v_0 \xrightarrow{P} v_f$
- v_0 satisfait pre : pre(v_0) and v_f satisfait post : post(v_0, v_f)
- pre$(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow$ post(v_0, v_f)

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables V

begin
$0 : P_0(v_0, v)$
$\texttt{instruction}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(v_0, v)$
end

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- conditions on the transitions ℓ, ℓ' to be defined on the basis of the induction principles.

Checking a contract : a simple example

variables U, V
requires $u_0, v_0 \in \mathbb{N}$
ensures $u_f + v_f = u_0 + v_0$

begin
$0 : u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N}$
$U := U + 2$
$1 : u = u_0 + 2 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N}$
$V := V - 2$
$2 : u = u_0 + 2 \wedge v = v_0 - 2 \wedge u_0, v_0 \in \mathbb{N}$
end

Checking to do

$$x = (pc, v) et J(\ell_0, v_0, pc, v) \triangleq \left[\begin{array}{l} \wedge\ pc \in \textsc{Locations} \\ \wedge\ v \in \textsc{Memory} \\ \ldots \\ \wedge\ pc = \ell \Rightarrow P_\ell(v_0, v) \\ \ldots \end{array} \right.$$

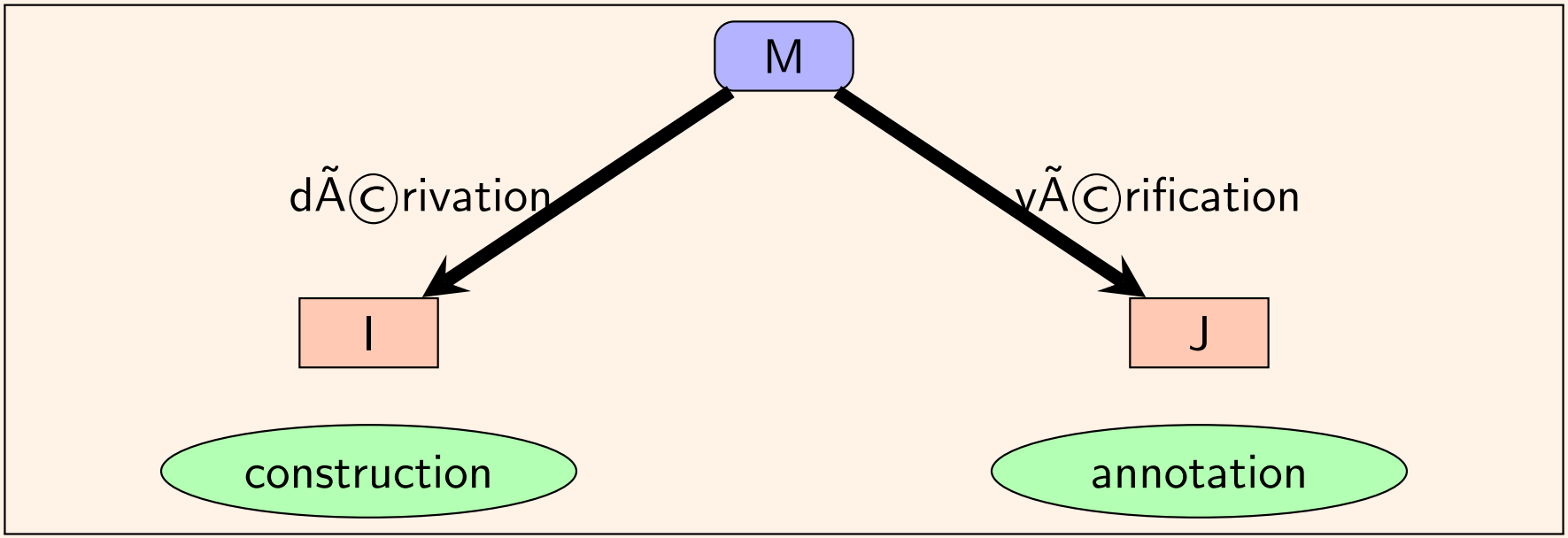
Let $(Th(s,c), x, \textsc{Values}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ be a relational model for this program. A property $\mathrm{A}(x)$ is a safety property for P if $\forall x_0, x \in \textsc{Locations} \times \textsc{Memory}. Init(x_0) \wedge x_0 \xrightarrow[\textsc{Next}]{} x \Rightarrow A(x)$. We know that this property implies that there exists a state property $\mathrm{I}(x)$ such that the three properties hold, but we apply this verification to J : $\forall x_0, x, x' \in \textsc{Locations} \times \textsc{Memory}$:

$$\left\{ \begin{array}{ll} (1) & \textsc{Init}(x_0) \Rightarrow \mathrm{J}(x_0, x_0) \\ (2) & \mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x) \\ (3) & \forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x_0, x') \end{array} \right.$$

- $\forall i \in \{0, \ldots, n\} : \mathrm{J}(x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x')$ est équivalent à $\mathrm{J}(x) \wedge (\exists i \in \{0, \ldots, n\} : x\ r_i\ x') \Rightarrow \mathrm{J}(x')$

Checking to do

- Application of the correctness to the relational induction principle : if we verify the three properties, then A is a safety property for the model in question (verification).
- If we wish to show that A is a safety property, then we must use the invariant to construct annotations for the model (derivation).



Checking to do : back to the example

- $x = (pc, u, v)$
- $J(0, u_0, v_0, pc, u, v) \triangleq$
$$\left[\begin{array}{l} \wedge\ pc \in \{0,1,2\} \\ \ \wedge\ u, v \in \mathbb{Z} \\ \wedge\ pc = 0 \Rightarrow u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \\ \wedge\ pc = 1 \Rightarrow u = u_0 + 2 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \\ \wedge\ pc = 2 \Rightarrow u = u_0 + 2 \wedge v = v_0 - 2 \wedge u_0, v_0 \in \mathbb{N} \end{array}\right.$$
- $A(0, u_0, v_0, pc, u, v) \triangleq (pc = 2 \Rightarrow u + v = u_0 + v_0 - 2 \wedge u_0, v_0 \in \mathbb{N})$

$\forall pc, u, v, pc', u', v' \in \{0,1,2\} \times \mathbb{Z}$:
$$\left\{\begin{array}{ll} (1) & \textsc{Init}(0, u_0, v_0) \Rightarrow \mathrm{J}(0, u_0, v_0, 0, u_0, v_0) \\ (2) & \mathrm{J}(0, u_0, v_0, pc, u, v) \Rightarrow \mathrm{A}(0, u_0, v_0, pc, u, v) \\ (3) & \forall i \in \{0, \ldots, n\} : \mathrm{J}(0, u_0, v_0, pc, u, v) \wedge x\ r_i\ pc', u', v' \Rightarrow \mathrm{J}(0, u_0, v_0, pc', \end{array}\right.$$

Checking the example

1. $\textsc{Init}(0, u_0, v_0,) \Rightarrow \mathrm{J}(0, u_0, v_0, 0, u_0, v_0)$:
   $pc = 0 \wedge u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \Rightarrow \mathrm{J}(0, u_0, v_0, 0, u_0, v_0)$:
2. $\mathrm{J}(0, u_0, v_0, pc, u, v) \Rightarrow \mathrm{A}(0, u_0, v_0, pc, u, v)$
   $\mathrm{J}(pc, u, v) \Rightarrow (pc = 2 \Rightarrow u + v = u_0 + v_0 - 2 \wedge u_0, v_0 \in \mathbb{N})$
3. $\forall i \in \{0, \ldots, n\} : \mathrm{J}(0, u_0, v_0, pc, u, v) \wedge x\ r_i\ pc', u', v' \Rightarrow$
   $\mathrm{J}(0, u_0, v_0, pc', u', v')$

$$\left[\begin{array}{l} r01(pc, u, v, pc', u', v') \triangleq pc = 0 \wedge u' = u + 2 \wedge pc' = 1 \wedge v' = v \\ r12(pc, u, v, pc', u', v') \triangleq pc = 1 \wedge v' = v - 2 \wedge pc' = 2 \wedge u' = u \end{array}\right.$$

   - $J(0, u_0, v_0, pc, u, v) \wedge r01(pc, u, v, pc', u', v') \Rightarrow J(pc', u', v')$
   - $J(0, u_0, v_0, pc, u, v) \wedge r12(pc, u, v, pc', u', v') \Rightarrow J(0, u_0, v_0, pc', u', v')$

$J(pc,u,v) \wedge r01(pc,u,v,pc',u',v') \Rightarrow J(pc',u',v')$ (1)

$$
\left(\begin{array}{l}
\wedge\ pc \in \{0,1,2\} \\
\wedge\ u,v \in \mathbb{Z} \\
\wedge\ pc = 0 \Rightarrow u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \\
\wedge\ pc = 1 \Rightarrow u = u_0 + 2 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \\
\wedge\ pc = 2 \Rightarrow u = u_0 + 2 \wedge v = v_0 - 2 \wedge u_0, v_0 \in \mathbb{N}
\end{array}\right)
\wedge
\left(\begin{array}{l}
\wedge pc = 0 \\
\wedge u' = u + 2 \\
\wedge pc' = 1 \\
\wedge v' = v
\end{array}\right)
$$

$$
\Rightarrow
$$

$$
\left(\begin{array}{l}
\wedge\ pc' \in \{0,1,2\} \\
\wedge\ u',v' \in \mathbb{Z} \\
\wedge\ pc' = 0 \Rightarrow u' = u_0 \wedge v' = v_0 \wedge u_0, v_0 \in \mathbb{N} \\
\wedge\ pc' = 1 \Rightarrow u' = u_0 + 2 \wedge v' = v_0 \wedge u_0, v_0 \in \mathbb{N} \\
\wedge\ pc' = 2 \Rightarrow u' = u_0 + 2 \wedge v' = v_0 - 2 \wedge u_0, v_0 \in \mathbb{N}
\end{array}\right)
$$

Using the Rodin platform for checking those elements

Initialisation

The relational model $M(P)$ for the annotated program P is therefore defined as follows :
$M(P) \triangleq (Th(s,c),(pc,v),\textsc{Locations} \times$
$\textsc{Memory}, Init(\ell,v), \{r_{\ell,\ell'} | \ell,\ell' \in \textsc{Locations} \wedge \ell \longrightarrow \ell'\})$.
The definition of $Init(x)$ depends on the precondition of P :

$$Init(x) \triangleq .x = (\ell_0, v) \wedge \textbf{pre}(\mathrm{P})(v).$$

Initial conditions

The following two properties are equivalent :

- $\forall x_0 \in \textsc{Values}$: $\mathit{Init}(x_0) \Rightarrow \mathrm{J}(x_0,x_0)$
- $\forall v \in \textsc{Memory}.\textbf{pre}(\mathrm{P})(v) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0,v)$

Induction step

- Les relations r_i correspondent aux transitions satisfaisant $\ell \longrightarrow \ell'$ et on associe à chaque r_i la relation $r_{\ell,\ell'}$
  - $x\ r_{\ell,\ell'}\ x' \triangleq (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $J(x_0, x) \triangleq \exists v_0, \ell, v.(\ell \in \textsc{Locations} \wedge v_0, v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v_0, v))$
- $P_\ell(v_0, v) \triangleq \exists x_0, x.(x_0, x \in \textsc{Values} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$

Induction step

The following two properties are equivalent :

- $\forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $\forall \ell, \ell' \in \textsc{Locations} : \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

Induction step (explanation)

- $\mathrm{J}(x_0 n x) \wedge x \; r_{\ell,\ell'} \; x' \Rightarrow \mathrm{J}(x_0, x')$

Induction step (explaination)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{\Delta}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$

Induction step (explaination)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \triangleq (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \triangleq \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$

Induction step (explaination)

- $\mathrm{J}(x_0 n x) \wedge x \; r_{\ell,\ell'} \; x' \Rightarrow \mathrm{J}(x_0, x')$
- $x \; r_{\ell,\ell'} \; x' \; \triangleq (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \triangleq \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \triangleq \exists x.(x \in \textsc{Values} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$

Induction step (explaination)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \triangleq (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \triangleq \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \triangleq \exists x.(x \in \textsc{Values} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$

Induction step (explaination)

- $\mathrm{J}(x_0 n x) \wedge x \; r_{\ell,\ell'} \; x' \Rightarrow \mathrm{J}(x_0, x')$
- $x \; r_{\ell,\ell'} \; x' \; \triangleq (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \triangleq \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \triangleq \exists x.(x \in \textsc{Values} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$

Induction step (explaination)

- $\mathrm{J}(x_0 n x) \wedge x \; r_{\ell,\ell'} \; x' \Rightarrow \mathrm{J}(x_0, x')$
- $x \; r_{\ell,\ell'} \; x' \; \triangleq (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \triangleq \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \triangleq \exists x.(x \in \textsc{Values} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \;\Rightarrow\; pc = \ell' \wedge P_\ell(v_0, v')$

Induction step (explaination)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \triangleq (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \triangleq \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \triangleq \exists x.(x \in \textsc{Values} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell'$ (Tautologie)

Induction step (explaination)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \triangleq (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \triangleq \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \triangleq \exists x.(x \in \textsc{Values} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell'$ (Tautologie)
- $pc = \ell \wedge P_\ell(v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ P_{\ell'}(v_0, v')$

Induction step (explaination)

- $\mathrm{J}(x_0 n x) \wedge x \; r_{\ell,\ell'} \; x' \Rightarrow \mathrm{J}(x_0, x')$
- $x \; r_{\ell,\ell'} \; x' \; \triangleq (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \triangleq \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \triangleq \exists x.(x \in \textsc{Values} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \;\Rightarrow\; pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \;\Rightarrow\; pc = \ell'$ (Tautologie)
- $pc = \ell \wedge P_\ell(v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \;\Rightarrow\; P_{\ell'}(v_0, v')$
- $P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

Conclusion

- $J(x_0, x) \triangleq \exists \ell, v, v_0.(\ell \in \textsc{Locations} \wedge v, v_0 \in \textsc{Memory} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v))$
- $P_\ell(v_0, v) \triangleq \exists x.(x, x_0 \in \textsc{Values} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge J(x_0), x)$
- $\mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x)$
- $\exists \ell, v, v_0.(\ell \in \textsc{Locations} \wedge v, v_0 \in \textsc{Memory} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v)) \Rightarrow \mathrm{A}(x_0, x)$
- $\forall \ell, v, v_0.(\ell \in \textsc{Locations} \wedge v, v_0 \in \textsc{Memory} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v)) \Rightarrow \mathrm{A}(x_0, x)$
- $\forall \ell \in \textsc{Locations}, v, v_0 \in \textsc{Memory}.P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v)$

Conclusion

The following two properties are equivalent :

- $\mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x)$
- $\forall \ell \in \textsc{Locations}, v, v_0 \in \textsc{Memory}.P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v)$

Translation of the verification conditions

The following verification conditions are equivalent :

- $\forall x_0, x, x' \in \text{LOCATIONS} \times \text{MEMORY}$:
$$\begin{cases} (1) \quad \text{INIT}(x_0) \Rightarrow \text{J}(x_0, x_0) \\ (2) \quad \text{J}(x_0, x) \Rightarrow \text{A}(x_0, x) \\ (3) \quad \forall i \in \{0, \ldots, n\} : \text{J}(x_0, x) \wedge x \; r_i \; x' \Rightarrow \text{J}(x_0, x') \end{cases}$$
- $\forall v_0, v, v' \in \text{MEMORY}$:
$$\begin{cases} (1) \quad \textbf{pre}(\text{P})(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v) \\ (2) \quad \forall \ell \in \text{LOCATIONS}. P_\ell(v_0, v) \Rightarrow \text{A}(\ell_0, v_0, \ell, v) \\ (3) \quad \forall \ell, \ell' \in \text{LOCATIONS} : \\ \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v') \end{cases}$$

Method for verification of safety properties

A relational method for proving the correctness of safety properties

- The program is annotated.
- The annotations define an invariant to be verified in accordance with the verification conditions.
- $A(\ell, v)$ is the statement of the safety property to be verified.

Relational method for verifying safety properties

Let $\mathrm{A}(\ell_0, v_0, \ell, v)$ be a property of a program$\mathrm{P}$. Let $\{P_\ell(v_0, v) : \ell \in \textsc{Locations}\}$ be a family of annotations, i.e. a family of properties, for this program. If the following conditions are satisfied : Therefore, $\mathrm{A}(\ell_0, v_0, \ell, v)$ is a safety property for the program P.

Equivalence Floyd/Hoare

Definition

Condition de vérification
L'expression $P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$ où ℓ, ℓ' sont deux étiquettes liées par la relation $\longrightarrow$, est appelée une condition de vérification.

Floyd and Hoare

- $\forall v_0, v, v' \in \textsc{Memory}.\forall \ell, \ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$ is equivalent to $\forall \ell, \ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow \forall v' \in \textsc{Memory}.P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell,\ell'}(v))$
- $\forall v_0, v, v' \in \textsc{Memory}.\forall \ell, \ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$ est équivalent à $\forall \ell, \ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow \forall v' \in \textsc{Memory}. (\exists v \in \textsc{Memory}.P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v)) \Rightarrow P_{\ell'}(v_0, v')$

Verification condition for the assignment

We can summarise the two possible forms of the assignment as follows :

- $\forall v, v' \in \textsc{Memory}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

$\ell : P_\ell(v_0, v)$
$V := f_{\ell,\ell'}(V)$
$\ell' : P_{\ell'}(v_0, v)$

Verification condition for the assignment

We can summarise the two possible forms of the assignment as follows :

- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

$\ell : P_\ell(v_0, v)$
$V := f_{\ell,\ell'}(V)$
$\ell' : P_{\ell'}(v_0, v)$

Verification condition for the assignment

We can summarise the two possible forms of the assignment as follows :

- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

$\ell : P_\ell(v_0, v)$
$V := f_{\ell,\ell'}(V)$
$\ell' : P_{\ell'}(v_0, v)$

Verification condition for the assignment

We can summarise the two possible forms of the assignment as follows :

- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v \in \text{MEMORY}. P_\ell(v_0, v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell,\ell'}(v))$ (Hoare axiomatic system).

$\ell : P_\ell(v_0, v)$
$V := f_{\ell,\ell'}(V)$
$\ell' : P_{\ell'}(v_0, v)$

Verification condition for the assignment

We can summarise the two possible forms of the assignment as follows :

- $\forall v, v' \in \text{MEMORY}.P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}.P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}.P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v \in \text{MEMORY}.P_\ell(v_0, v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell,\ell'}(v))$ (Hoare axiomatic system).
- $\forall v \in \text{MEMORY}.\ (\exists v' \in \text{MEMORY}.P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v)) \Rightarrow P_{\ell'}(v_0, v')$
  corresponds to Floyd assignment rule.

$\ell : P_\ell(v_0, v)$
$V := f_{\ell,\ell'}(V)$
$\ell' : P_{\ell'}(v_0, v)$

Verification condition for the iteration

$\ell_1 : P_{\ell_1}(v_0, v)$
WHILE $B(v)$ **DO**
  $\ell_2 : P_{\ell_2}(v_0, v)$
  $\ldots$
  $\ell_3 : P_{\ell_3}(v_0, v)$
END
$\ell_4 : P_{\ell_4}(v_0, v)$

For the iteration structure, the verification conditions are as follows :

- $P_{\ell_1}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- $P_{\ell_1}(v_0, v) \wedge \neg B(v) \Rightarrow P_{\ell_4}(v_0, v)$
- $P_{\ell_3}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- $P_{\ell_3}(v_0, v) \wedge \neg B(v) \Rightarrow P_{\ell_4}(v_0, v)$

Verification conditions for the conditional

$\ell_1 : P_{\ell_1}(v_0, v)$
IF $B(v)$ **THEN**
  $\ell_2 : P_{\ell_2}(v_0, v)$
  $\ldots$
  $\ell_3 : P_{\ell_3}(v_0, v)$
ELSE
  $m_2 : P_{\ell_2}(v_0, v)$
  $\ldots$
  $m_3 : P_{\ell_3}(v_0, v)$
FI
$\ell_4 : P_{\ell_4}(v_0, v)$

For the conditional statement, the verification cond

- $P_{\ell_1}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- $P_{\ell_3}(v_0, v) \Rightarrow P_{\ell_4}(v_0, v)$
- $P_{\ell_1}(v_0, v) \wedge \neg B(v) \Rightarrow P_{m_2}(v_0, v)$
- $P_{m_3}(v_0, v) \Rightarrow P_{\ell_4}(v_0, v)$

Program correctness properties

Let v be a state variable of P. **pre**$(\mathrm{P})(v)$ is the precondition of P for v ; it characterises the initial values of v. **post**$(\mathrm{P})(v_0, v)$ is the postcondition of P for v ; it characterises the final values of v in relation to the initial value v_0

Example

1. **pre**$(\mathrm{P})(x, y, z)$=$x, y, z \in \mathbb{N}$ et
   post$(\mathrm{P})(x_0, y_0, z_0, x, y, z)$=$z = x_0 * y_0$
2. **pre**$(\mathrm{Q})(x, y, z)$=$x, y, z \in \mathbb{N}$ et
   post$(\mathrm{Q})(x_0, y_0, z_0, x, y, z)$=$z = x_0 + y_0$

$$\forall \underline{x}, \underline{y}, \underline{r}, \underline{q}, \overline{x}, \overline{y}, \overline{r}, \overline{q}.$$
$$\textbf{pre}(\mathrm{P})(\underline{x}, \underline{y}, \underline{r}, \underline{q}) \wedge (\underline{x}, \underline{y}, \underline{r}, \underline{q}) \xrightarrow{\mathrm{P}} (\overline{x}, \overline{y}, \overline{r}, \overline{q})$$
$$\Rightarrow \textbf{post}(\mathrm{P})(\underline{x}, \underline{y}, \underline{r}, \underline{q}, \overline{x}, \overline{y}, \overline{r}, \overline{q})$$

Partial Correctness of a program

Partial correctness aims to establish that a program P is partially correct with respect to its precondition and its postcondition.

- the data specification for P **pre**(P) (v_0)
- the specification of the outputs of P **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$
- a family of property annotations $\{P_\ell(v_0, v) : \ell \in \textsc{Locations}\}$ for this program.
- a safety property defining the partial correctness $pc = \ell_f \Rightarrow$ **post**$(\mathrm{P})(v_0, v_f)$, where ℓ_f is the label marking the end of the program P

Definition

The program P is partially correct with respect to **pre**$(\mathrm{P})(\mathrm{v}_0)$ and **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$, if the property $pc = \ell_f \Rightarrow$ **post**$(\mathrm{P})(v_0, v)$ is a safety property for this program.

Partial Correctness of a program

If the following conditions are met :

- $\forall v_0, v \in \text{MEMORY} : \textbf{pre}(\text{P})(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$
- $\forall v_0, v \in \text{MEMORY} : P_{\ell_f}(v_0, v) \Rightarrow \textbf{post}(\text{P})(v_0, v)$
- $\forall \ell, \ell' \in \text{LOCATIONS} : \ell \longrightarrow \ell' : \forall v_0, v, v' \in \text{MEMORY}. (P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$,

So the program P is partially correct with respect to **pre**(P)(v_0) et **post**(P)(v_0,v).

- The partial correctness states that if the program terminates normally, then the postcondition is satisfied by the current variables.
- The semantics of the contract are therefore fairly straightforward to define :

Reformulation of computation

- $\mathsf{pre}(v_0) \wedge v_0 \xrightarrow{\mathsf{P}} v_f \Rightarrow \mathsf{post}(v_0, v_f)$

  (expression of the partial correctness)

- $pc_0 = \ell_0 \wedge \mathsf{pre}(v_0) \wedge (pc_0, v_0) \overset{\textsc{Next}^\star}{\longrightarrow} (pc, v) \wedge pc = \ell_f \Rightarrow \mathsf{post}(v_0, v_f)$

  (big-step semantics and small-step semantics equivalence)

- $pc_0 = \ell_0 \wedge \mathsf{pre}(v_0) \wedge (pc_0, v_0) \overset{\textsc{Next}^\star}{\longrightarrow} (pc, v) \Rightarrow (pc = \ell_f \Rightarrow \mathsf{post}(v_0, v_f))$

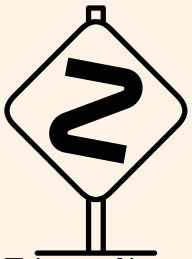
  (implication and conjunction property)

- $Init(x_0) \wedge x_0 \overset{\textsc{Next}^\star}{\longrightarrow} x \Rightarrow \mathsf{PC}(x_0, x)$

$(Init(x_0) \triangleq pc_0 = \ell_0 \wedge \mathsf{pre}(v_0)$

$x_0 \triangleq (\ell_0, v_0)$ and $x \triangleq (pc, v)$

$\mathsf{PC}(x_0 x) \triangleq x_0 = (\ell_0, v_0) \wedge x = (pc, v) \Rightarrow (pc = \ell_f \Rightarrow \mathsf{post}(v_0, v_f))$



Partial correctness is a safety property and the relational method for safety properties is applied.

Checking the contract

A program P *satsifies* a contract (pre,post) :

- P transforms a variable v from an initial value v_0 to a final value v_f : $v_0 \xrightarrow{P} v_f$
- v_0 satisfies pre : pre(v_0) and v_f satisfies post : post(v_0, v_f)
- pre$(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow$ post(v_0, v_f)

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables V
begin
$0 : P_0(v_0, v)$
$\texttt{statement}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{statement}_{f-1}$
$f : P_f(v_0, v)$
end

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- For any pair of labels ℓ, ℓ' such that $\ell \longrightarrow \ell'$, we verify that, for all values $v, v' \in \text{MEMORY}$

$$\left(\begin{array}{l} \left(\begin{array}{l} P_\ell(v_0, v)) \\ \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$

Checking RTE

- The transition to be executed is the one going from ℓ to ℓ', characterised by the condition or guard $cond_{\ell,\ell'}(v)$ on v and a transformation of the variable v, $v' = f_{\ell,\ell'}(v)$.
- An RTE condition is defined by $\mathbf{DOM}(\ell, \ell')(v)$ for the transition in question. $\mathbf{DOM}(\ell, \ell')(v)$ means that the transition $\ell \longrightarrow \ell'$ is possible and does not lead to an error.
- An error is an arithmetic overflow, a reference to an array element that does not exist, a reference to a null pointer, ...

example

1. The transition corresponds to an assignment of the form $x := x + y$ or $y := x + y$:
   $\mathbf{DOM}(x + y)(x, y) \triangleq \mathbf{DOM}(x)(x, y) \wedge \mathbf{DOM}(y)(x, y) \wedge x + y \in int$
2. The transition corresponds to an assignment of the form $x := x + 1$ ou $y := x + 1$:
   $\mathbf{DOM}(x + 1)(x, y) \triangleq \mathbf{DOM}(x)(x, y) \wedge x + 2 \in int$

Definition RTE

The absence of run time errors is intended to éstablish that a program P will not produce errors during its execution with regard to its precondition and postcondition.

- the data specification for P **pre**$(\mathrm{P})(\mathrm{v})$
- the result specification for P **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$
- a family of property annotations $\{P_\ell(v) : \ell \in \textsc{Locations}\}$ for this program.
- a safety property defining the absence of errors at execution :

$$\bigwedge_{\ell \in \textsc{Locations} - \{output\}, n \in \textsc{Locations}, \ell \longrightarrow n} (\mathbf{DOM}(\ell, n)(v))$$

Definition

The program P will not produce any errors during execution with respect to **pre**$(\mathrm{P})(\mathrm{v})$ and **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$, if the property $\bigwedge_{\ell \in \textsc{Locations} - \{output\}, n \in \textsc{Locations}, \ell \longrightarrow n} (\mathbf{DOM}(\ell, n)(v))$ is a safety property for this program.

Induction principle for RTE

RTE = Run Time Error

If the following conditions are satisfied :

- $\forall v_0, v \in \text{MEMORY} : \textbf{pre}(\text{P})(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$
- $\forall m \in \text{LOCATIONS} - \{\ell_f\}, n \in \text{LOCATIONS}, \forall v_0, v, v' \in \text{MEMORY} :$ $m \longrightarrow n : \textbf{pre}(\text{P})(v_0) \wedge P_m(v_0, v) \Rightarrow \textbf{DOM}(m, n)(v)$
- $\forall \ell, \ell' \in \text{LOCATIONS} : \ell \longrightarrow \ell' : \forall v_0, v, v' \in$ $\text{MEMORY}. (P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')),$

Therefore, the program P will not produce any run-time error with respect to **pre**(P)(v_0) and **post**(P)(v_0,v).

- First, one must verify the partial correction and then reinforce the assertions of the partial correctness with domain conditions.
- One can therefore derive a contract that also incorporates the verification of RTE.

Relational verification method for RTE

A program P *fulfils* a contract (pre,post) :

- P transforms a variable v from an initial value v_0 to a final value v_f : $v_0 \xrightarrow{P} v_f$
- v_0 satisfies pre : $pre(v_0)$ and v_f satisfies post : $post(v_0, v_f)$
- $pre(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow post(v_0, v_f)$
- $\mathbb{D}$ is the RTE domain of V

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables V
begin
$0 : P_0(v_0, v)$
$\texttt{instruction}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(v_0, v)$
end

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(x_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- For any pair of labels ℓ, ℓ' such that $\ell \longrightarrow \ell'$, we find that, for all values$v, v' \in \textsc{Memory}$
$$\left(\begin{array}{l} \left(\begin{array}{l} P_\ell(v_0, v)) \\ \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$
- $\forall m \in \textsc{Locations} - \{\ell_f\}, n \in \textsc{Locations}, \forall v_0, v, v' \in \textsc{Memory}$: $m \longrightarrow n$:
$pre(v_0) \wedge P_m(v_0, v) \Rightarrow \mathbf{DOM}(m, n)(v)$

Current Sub Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
4. Checking contracts using Event- B on the FIDE Rodin
   - Method for proving invariance properties
   - Verification of contract
   - Definition of machine for checking contract
   - Soundness of the technique
5. Example
6. Conclusion

Event B Machine for checking contract of a program

A program P *satisfies* a contract (pre,post) and P transforms a variable v.

Event B Machine for checking contract of a program

A program P *satisfies* a contract (pre,post) and P transforms a variable v.

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables V

begin
$0 : P_0(v_0, v)$
$\texttt{statement}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{statement}_{f-1}$
$f : P_f(v_0, v)$
end

Event B Machine for checking contract of a program

A program P *satisfies* a contract (pre,post) and P transforms a variable v.

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables V

begin
$0 : P_0(v_0, v)$
$\texttt{statement}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{statement}_{f-1}$
$f : P_f(v_0, v)$
end

machine M
sees C
variables pc, v
invariants
$I(pc, v)$
theorems
$pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
$pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
INITIALISATION $Init(v)$
...
e(ℓ, ℓ')
...
end

Event B Machine for checking contract of a program

A program P *satisfies* a contract (pre,post) and P transforms a variable v.

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables V

begin
$0 : P_0(v_0, v)$
$\texttt{statement}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{statement}_{f-1}$
$f : P_f(v_0, v)$
end

machine M
sees C
variables pc, v
invariants
$I(pc, v)$
theorems
$pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
$pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
INITIALISATION $Init(v)$
...
e(ℓ, ℓ')
...
end

What are the invariants, the theorems and the events, such that M is checked if, and only if, the program P fulfils a contract (pre,post)

Current Sub Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
4. Checking contracts using Event- B on the FIDE Rodin
   - Method for proving invariance properties
   - Verification of contract
   - Definition of machine for checking contract
   - Soundness of the technique
5. Example
6. Conclusion

Sketch of the construction

- Writing a context C for defining constants as v_0 and properties (axioms) as $pre(v_0$.
- Defining additional constants and predicates for expressing annotations as for instance the domain D of the problem.
- Defining invariants from annotations
- Defining theorems from conditions for checking partial correctness.
- Defining events from the graph of labels.

Defining the context

- A simple algorithm decreases a variable x initially set to a value x_0 of $\mathbb{N}$
- A set L of labels is used for annotating the algorithm.
- Two constants *maxi* and *mini* for expressing the domain of values x_0

variables x
requires $x_0 \in \mathbb{N}$
ensures $x_f = 0$
begin
$\ell_0 : \{ x = x_0 \wedge x_0 \in \mathbb{N}\}$
while $0 < x$ do
  $\ell_1 : \{0 < x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
$x := x - 1;$
$\ell_2 : \{0 \leq x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
od;
$\ell_3 : \{x = 0\}$
end

Defining the context

- A simple algorithm decreases a variable x initially set to a value x_0 of $\mathbb{N}$
- A set L of labels is used for annotating the algorithm.
- Two constants *maxi* and *mini* for expressing the domain of values x_0

variables x
requires $x_0 \in \mathbb{N}$
ensures $x_f = 0$
begin
$\ell_0 : \{ x = x_0 \wedge x_0 \in \mathbb{N}\}$
while $0 < x$ do
$\ell_1 : \{0 < x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
$x := x - 1;$
$\ell_2 : \{0 \leq x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
od;
$\ell_3 : \{x = 0\}$
end

CONTEXT CONTEXT
SETS L
CONSTANTS $x0\ l0\ l1\ l2\ l3\ mini\ maxi$
AXIOMS
$@pre\ x0 \in \mathbb{N}$
$@axm2\ partition(L, \{l0\}, \{l1\}, \{l2\}, \{l3\})$
$@axm3\ mini \in \mathbb{Z} \wedge maxi \in \mathbb{Z}$
$@axm4\ mini < 0 \wedge maxi > 0$
$@axm5\ x0 \leq maxi$
end

From annotations to invariants

$$
\begin{array}{l}
\quad inv_i : pc \in \textsc{Locations} \\
\quad inv_j : v \in Type \\
\cdots \\
\quad inv_k : pc = \ell \Rightarrow P_\ell(v) \\
\quad inv_m : pc = \ell' \Rightarrow P_{\ell'}(v) \\
\cdots \\
\quad th_n : A(pc, v)
\end{array}
$$

- $Type$ is the type of the variables v and is a set of possible values defined in the context C.
- The annotation is giving us for free the conditions satisfied by v when the control is in ℓ, (resp. in ℓ').
- $A(c, v)$ is a safety property that we are supposed to check and the case of Event-B, it is a theorem.

Defining the invariant for our example

- The counter variable is ofte denoted *pc* or *l* or *c*.
- Theorems are stating the conditions of initialistion and finalisation and the RTE conditions.

INVARIANTS

$@inv1\ x \in \mathbb{Z}$

$@inv2\ l \in L$

$@inv3\ l = @xtruc1$

$l0 \Rightarrow x = x0 \wedge x0 \in \mathbb{N}$

$@inv4\ l = l1 \Rightarrow 0 < x \wedge x \leq x0 \wedge x0 \in \mathbb{N}$

$@inv5\ l = l2 \Rightarrow 0 \leq x \wedge x \leq x0 \wedge x0 \in \mathbb{N}$

$@inv6\ l = l3 \Rightarrow x = 0$

$theorem\ @precond\ x = x0 \wedge x0 \in \mathbb{N} \Rightarrow x = x0 \wedge x0 \in \mathbb{N}$

$theorem\ @postcond\ x = 0 \Rightarrow x = 0$

$theorem\ @rte1\ x \in \mathbb{Z}$

$theorem\ @overflow\ mini \leq x \wedge x \leq maxi$

Defining events for transition steps

$$\begin{array}{l}\ell : \{P_\ell(v)\} \\ \quad cond_{\ell,\ell'}(v) \longrightarrow v := f_{\ell,\ell'}(v) \\ \ell' : \{P_{\ell'}(v)\}\end{array}$$

$$\begin{array}{l}\ell_0^1 : \{x = 0\} \\ \quad x := x + 1; \\ \ell_0^1 : \{x = 1\}\end{array}$$

$$\begin{array}{l}e(\ell, \ell') \\ \quad \textbf{WHEN} \\ \quad\quad pc = \ell \\ \quad\quad cond_{\ell,\ell'}(v) \\ \quad \textbf{THEN} \\ \quad\quad pc := \ell' \\ \quad\quad v := f_{\ell,\ell'}(v) \\ \quad \textbf{END}\end{array}$$

- v is the state meory variable or list of memory variables ; v includes the local variables and the results variables.
- pc is a new variable which is modelling the control flow and its type is LOCATIONS.
- $e(\ell, \ell')$ is simulating the computation flow starting from ℓ and moving to ℓ' ; v is updated.

Defining the events for the example

EVENT $INITIALISATION$
then
$@act1\ x := x0$
$@act2\ l := l0$
end
EVENT $el0l1$
where
$@grd1\ l = l0$
$@grd2\ 0 < x$
then
$@act1\ l := l1$
end
EVENT $el0l3$
where
$@grd1\ l = l0$
$@grd2\ \neg(\ 0 < x)$
then
$@act1\ l := l3$
end

EVENT $el1l2$
where
$@grd1\ l = l1$
then
$@act1\ l := l2$
$@act2\ x := x - 1$
end
EVENT $el2l1$
where
$@grd1\ l = l2$
$@grd2\ 0 < x$
then
$@act1\ l := l1$
end
EVENT $el2l3$
where
$@grd1\ l = l2$
$@grd2\ \neg(0 < x)$
then
$@act1\ l := l3$
end

Current Sub Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
4. Checking contracts using Event- B on the FIDE Rodin
   - Method for proving invariance properties
   - Verification of contract
   - Definition of machine for checking contract
   - Soundness of the technique
5. Example
6. Conclusion

Verification conditions to check for PC

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- PC (Partial Correctness)
  For any pair of labels ℓ, ℓ'
  such that $\ell \longrightarrow \ell'$,We check that, for all values $v, v' \in \textsc{Memory}$
  $$\left(\begin{array}{l} \left(\begin{array}{l} P_\ell(v_0, v)) \\ \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$
  i

Verification conditions to check for PC

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$ **THEOREM**
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$ **THEOREM**
- PC (Partial Correctness)
  For any event $\mathsf{e}(\ell, \ell')$ with $\ell \longrightarrow \ell'$,
  for all values $v, v' \in \textsc{Memory}$

$$\left(\begin{array}{l} \left(\begin{array}{l} P_\ell(v_0, v)) \\ \wedge BA(\mathsf{e}(\ell, \ell'))(pc, v, pc', v') \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$

Partial correctness using Event-B models

- Introduction of the definition of events between two successive labels.

$BA(\mathsf{e}(\ell, \ell'))(pc, v, pc', v')$

For each pair of successive labels ℓ, ℓ', the three statements are equivalent :

- $P_\ell(v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v')$
- $I(pc, v) \wedge pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge pc' = \ell' \wedge v' = f_{\ell,\ell'}(v) \Rightarrow (pc' = \ell' \Rightarrow P_{\ell'}(v'))$
- $I(pc, v) \wedge BA(e(\ell, \ell'))(pc, v, pc', v') \wedge pc = \ell \Rightarrow (pc' = \ell' \Rightarrow P_{\ell'}(v'))$

Soundness for PC

Partcial Correctness

Let AA an annotated algorithm with precondition **pre**(AA)(v) and postcondition **post**(AA)(v$_0$,v). Let the context C and the machine M generated from AA using the construction given previously. We assume that ℓ_0 is the first label and ℓ_e is the last label. We add the following properties in the machine M :

- $pc = \ell_0 \wedge$ **pre**$(\mathrm{AA})(v) \Rightarrow P_{\ell_0}(v)$ (condition of initialisation as THEOREM)
- $pc = \ell_e \Rightarrow (P_{\ell_e}(v) \Rightarrow$ **post**$(\mathrm{AA})(v_0, v)$ (condition of safety as THEOREM).

If proof obligations are discharged, then the annotated algorithm AA is partially correct with respect to its pre/post specification.

- The first added theorem is expressing the sound initialisation.
- The second added property is expressing the partial correctness.
- The discharged proof obligations are corresponding to the induction steps.

Checking invariance and safety properties

A relational method for proving the correctness of safety properties

- The contract is stated for variables v namely $v : [pre(v_0, post(v_0, v_f]$
- The program is annotated on the set of labels denoted $\textsc{Locations}$ ($\ell \in \textsc{Locations}$).
- The annotations define an invariant to be verified according to the verification conditions.
- $A(\ell, v)$ is the statement of the safety property to be verified.

A relational method for proving safety properties

Let $\mathrm{A}(\ell_0, v_0, \ell, v)$ be a property of a program P. Let $\{P_\ell(v_0, v) : \ell \in \textsc{Locations}\}$ be a family of annotations, i.e. a family of properties, for this program. If the following conditions are satisfied :
$\forall v, v' \in \textsc{Memory}$:

$$\begin{cases} (1) \quad Pre(v) \Rightarrow P_{\ell_0}(v) \\ (2) \quad \forall \ell \in \textsc{Locations}. P_\ell(v) \Rightarrow \mathrm{A}(\ell, v) \\ (3) \quad \forall \ell, \ell' \in \textsc{Locations} : \\ \ell \longrightarrow \ell' \Rightarrow P_\ell(v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v') \end{cases}$$

Therefore, $\mathrm{A}(\ell_0, v_0, \ell, v)$ is a safety property for the program P.

Checking using the verification Machine

- Adding the properties (2) $\forall \ell \in \text{LOCATIONS}.P_\ell(v) \Rightarrow \text{A}(\ell, v)$ as THEOREMS
- Szfety properties as RTE (Run Time Error), ADF (Absence of Deadlock Freedom), ...
- Expression of safety properties using the contexte or the theory plugin facilities.

From Floyd to Hoare

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x_f.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x_f.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow [P]\mathsf{post}(x_0, x_f)$
- wlp calculus is introduced

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x_f.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow [P]\mathsf{post}(x_0, x_f)$
- wlp calculus is introduced
- $[x := e]P(x) = P[x \mapsto e]$

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x_f.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow [P]\mathsf{post}(x_0, x_f)$
- wlp calculus is introduced
- $[x := e]P(x) = P[x \mapsto e]$
- [if $b(x)$ then $S1$ else $S2$ $]P(x) = b(x) \wedge [S1]P(x) \vee$ not $b(x)$ $[S2]P(x)$

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x_f.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow [P]\mathsf{post}(x_0, x_f)$
- wlp calculus is introduced
- $[x := e]P(x) = P[x \mapsto e]$
- $[\text{if } b(x) \text{ then } S1 \text{ else } S2\ ]P(x) = b(x) \wedge [S1]P(x) \vee \text{ not } b(x)\ [S2]P(x)$
- Frama-c uses the HOARE logic for defining the verification conditions as R. Leino in DAFNY.

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x_f.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow [P]\mathsf{post}(x_0, x_f)$
- wlp calculus is introduced
- $[x := e]P(x) = P[x \mapsto e]$
- $[\text{if } b(x) \text{ then } S1 \text{ else } S2\ ]P(x) = b(x) \wedge [S1]P(x) \vee \text{ not } b(x)\ [S2]P(x)$
- Frama-c uses the HOARE logic for defining the verification conditions as R. Leino in DAFNY.
- Questions of termination require the wp calculus ...

Current Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
   - PO thm/THM (context)
   - PO th/THM (machine)
   - PO evt/inv/INV
   - PO evt/act/FIS
4. Checking contracts using Event- B on the FIDE Rodin
   - Method for proving invariance properties
   - Verification of contract
   - Definition of machine for checking contract
   - Soundness of the technique
5. Example
6. Conclusion

A short example

s

contract SIMPLE
variables x
requires $x_0 \in \mathbb{N}$
ensures $x_f = 0$
begin
$\ell_0 : \{0 \leq x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
while $0 < x$ **do**
$\ell_1 : \{0 < x \wedge x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
$x := x - 1;$
od
$\ell_2 : \{x = 0\}$end

Event $Init$
THEN
$act1 : x := x0$
$act2 : l := l0$

Event $el0l1$
WHEN
$grd1 : l = l0$
$grd2 : 0 < x$
THEN
$act1 : l := l1$

INVARIANTS
$inv1 : x \in \mathbb{N}$
$inv2 : l \in L$
$inv3 : l = l0 \Rightarrow$
$0 \leq x \wedge x \leq x0 \wedge x0 \in \mathbb{N}$
$inv4 : l = l1 \Rightarrow$
$0 < x \wedge x \leq x0 \wedge x0 \in \mathbb{N}$
$inv5 : l = l2 \Rightarrow x = 0$
$requires : x0 \in \mathbb{N} \wedge x = x0$
$\Rightarrow x = x0 \wedge x0 \in \mathbb{N}$
$ensures : x = 0 \wedge x = x0$
$\Rightarrow x = 0$

Event $el0l2$
WHEN
$grd1 : l = l0$
$grd2 : \neg(0 < x)$
THEN
$act1 : l := l2$

Event $el1l0$
WHEN
$grd1 : l = l1$
THEN
$act1 : l := l0$
$act2 : x := x - 1$

mcfsi0-alg-simple

MACHINE $algorithm$ SEES CONTEXT
VARIABLES $x\ l$
INVARIANTS
$@inv1\ x \in \mathbb{Z}$
$@inv2\ l \in L$
$@inv3\ l = l0 \Rightarrow x = x0 \land x0 \in \mathbb{N}$
$@inv4\ l = l1 \Rightarrow 0 < x \land x \leq x0 \land x0 \in \mathbb{N}$
$@inv5\ l = l2 \Rightarrow 0 \leq x \land x \leq x0 \land x0 \in \mathbb{N}$
$@inv6\ l = l3 \Rightarrow x = 0$
$theorem\ @precond\ x = x0 \land x0 \in \mathbb{N} \Rightarrow x = x0 \land x0 \in \mathbb{N}$
$theorem\ @postcond\ x = 0 \Rightarrow x = 0$
$theorem\ @rte1\ x \in \mathbb{Z}$
$theorem\ @overflow\ mini \leq x \land x \leq maxi$

mcfsi0-alg-simple

EVENT $INITIALISATION$
then
$@act1\ x := x0$
$@act2\ l := l0$
end
EVENT $el0l1$
where
$@grd1\ l = l0$
$@grd2\ 0 < x$
then
$@act1\ l := l1$
end

EVENT $el0l3$
where
$@grd1\ l = l0$
$@grd2\ \neg(\ 0 < x)$
then
$@act1\ l := l3$
end

EVENT $el1l2$
where
$@grd1\ l = l1$
then
$@act1\ l := l2$
$@act2\ x := x - 1$
end
EVENT $el2l1$
where
$@grd1\ l = l2$
$@grd2\ 0 < x$
then
$@act1\ l := l1$
end
EVENT $el2l3$
where
$@grd1\ l = l2$
$@grd2\ \neg(0 < x)$
then
$@act1\ l := l3$
end

Current Section Summary

1. CM3
2. Overview of machines, contexts and proof obligations
3. Proof Obligations for Contexts and Machines
   - PO thm/THM (context)
   - PO th/THM (machine)
   - PO evt/inv/INV
   - PO evt/act/FIS
4. Checking contracts using Event- B on the FIDE Rodin
   - Method for proving invariance properties
   - Verification of contract
   - Definition of machine for checking contract
   - Soundness of the technique
5. Example
6. Conclusion

Summary on the technique for verifying safety properties

- Formalizing safety properties as state properties
- Defining the context for expressing the precondition and the postcondition if necessary.
- Using the verification machine, as an engine for cecking safety properties and especially the contract for partial correctness.

- Handling termination by using convergent events and variants.
- Version using the Floyd method contrary to Frama-c.