# Modelling Software-based Systems

## Lecture 5 Checking contracts with Event-B

Master Informatique

Dominique Méry
Telecom Nancy,Université de Lorraine

24 octobre 2024
dominique.mery@loria.fr

# General Summary

1. Programming by contract
2. Verification
3. Floyd to Hoare

# Current Summary

# Verifying program correctness

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

- P transforms a variable v from initial values $v_0$ and produces a final value $v_f$ : $v_0 \xrightarrow{P} v_f$
- $v_0$ satisfies pre : $\text{pre}(v_0)$ and $v_f$ satisfies post : $\text{post}(v_0, v_f)$
- $\text{pre}(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow \text{post}(v_0, v_f)$
- $\mathbb{D}$ est le domaine RTE de V

# Verifying program correctness

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

- P transforms a variable v from initial values $v_0$ and produces a final value $v_f$ : $v_0 \xrightarrow{\text{P}} v_f$
- $v_0$ satisfies pre : pre$(v_0)$ and $v_f$ satisfies post : post$(v_0, v_f)$
- pre$(v_0) \wedge v_0 \xrightarrow{\text{P}} v_f \Rightarrow$ post$(v_0, v_f)$
- $\mathbb{D}$ est le domaine RTE de V

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

$$\left[\begin{array}{l}
\text{begin} \\
0 : P_0(v_0, v) \\
\texttt{instruction}_0 \\
\ldots \\
i : P_i(v_0, v) \\
\ldots \\
\texttt{instruction}_{f-1} \\
f : P_f(v_0, v) \\
\text{end}
\end{array}\right.$$

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- For any pair of labels$\ell, \ell'$ such that $\ell \longrightarrow \ell'$, one verifies that, pour any values $v, v' \in \textsc{Memory}$
$$\left( \begin{array}{l} \left( \begin{array}{l} pre(v_0) \wedge P_\ell(v_0, v)) \\ \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$

# Contracts - Verification Conditions

```
contract P
variables v
requires pre(v0)
ensures post(v0, vf)
    begin
    0 : P0(v0, v)
    S0
    ...
    i : Pi(v0, v)
    ...
    Sf-1
    f : Pf(v0, v)
    end
```

# Contracts - Verification Conditions

```
contract P
variables v
requires pre(v_0)
ensures post(v_0, v_f)
    begin
    0 : P_0(v_0, v)
    S_0
    ...
    i : P_i(v_0, v)
    ...
    S_{f-1}
    f : P_f(v_0, v)
    end
```

Verification conditions are listed as follows :

- (initialisation)
  $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- (finalisation)
  $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- (induction)
  For each labels pair $\ell, \ell'$
  such that $\ell \longrightarrow \ell'$, one checks that,
  for any value $v, v' \in \textsc{Memory}$
  $$\left( \begin{array}{l} \left( \begin{array}{l} pre(v_0) \wedge P_\ell(v_0, v)) \\ \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$

Three kinds of verification conditions should be checked and we justify the method in the full version..

# From PAP to Rodin . . .

# From PAP to Rodin ...

MACHINE $M$
SEES $C0$
VARIABLES
$v, pc$
INVARIANTS
typing : $v \in D$
control : $pc \in L$
...
at$\ell$ : $pc = \ell \Rightarrow P_\ell(v0, v)$
...
$th1 : pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
$th2 : pre(v_0) \wedge P_f(v_0, v)$
$\Rightarrow post(v_0, v)$
...
END
...
END

# From PAP to Rodin . . .

MACHINE $M$
SEES $C0$
VARIABLES
$v, pc$
INVARIANTS
typing : $v \in D$
control : $pc \in L$
. . .
at$\ell$ : $pc = \ell \Rightarrow P_\ell(v0, v)$
. . .
$th1 : pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
$th2 : pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
. . .
END
. . .
END

MACHINE $M$
EVENTS
INITIALISATION
BEGIN
$(pc, v) :| \left( \begin{array}{l} pc' = l0 \wedge v' = v0 \\ \wedge pre(v0) \end{array} \right)$
END
. . .
$e(\ell, \ell')$
WHEN
$pc = \ell$
$cond_{\ell,\ell'}(v)$
THEN
$pc := \ell'$
$v := f_{\ell,\ell'}(v)$
END
. . .
END

# Technical problems for students

## (Induction Principle (I))

A property $S(z0,z)$ is a safety for an annotated program P if, and only if, there exists a property $I(z0,z)$ satisfying :

1. $\forall z0, z \in \mathsf{L} \times \mathsf{D}.init(z0) \wedge z = z0 \Rightarrow I(z0,z)$
2. $\forall z0, z, z' \in \mathsf{L} \times \mathsf{D}.init(z0) \wedge I(z0,z) \wedge (z \underset{P}{\longrightarrow} z') \Rightarrow I(z0,z')$
3. $\forall z0, z \in \mathsf{L} \times \mathsf{D}.init(z0) \wedge I(z0,z) \Rightarrow S(z0,z)$

## (Induction Principle (II))

A property $S(\ell 0, x0, \ell, x)$ is a safety property for an annotated program P if, and only if, there exists a property $I(\ell 0, x0, \ell, x)$ satisfying :

1. $\forall \ell 0, \in \mathsf{L}, x0 \in \mathsf{D}.\ell 0 \in \mathsf{L0} \wedge pre(x0) \wedge x = x0 \wedge pc = \ell 0 \Rightarrow J(\ell 0, x0, \ell, x)$
2. $\forall \ell, \ell' \in \mathsf{L}, x, x0 \in \mathsf{D}.\ell 0 \in \mathsf{L0} \wedge pre(x0) \wedge J(\ell 0, x0, \ell, x) \wedge$
   $BA(e(\ell, \ell'),)(\ell, x, \ell', x') \Rightarrow J(\ell 0, x0, \ell', x')$
3. $\forall \ell 0, \ell \in \mathsf{L}, x0, x \in \mathsf{D}.pre(x0) \wedge \ell 0 \in$
   $\mathsf{L0} \wedge J(\ell 0, x0, \ell, x) \Rightarrow S(\ell 0, x0, \ell, x)$

# Technical problems for students

## (Induction Principle (II))

A property $S(\ell 0, x0, \ell, x)$ is a safety property for an annotated program P if, and only if, there exists a property $I(\ell 0, x0, \ell, x)$ satisfying :

1. $\forall \ell 0, \in \mathsf{L}, x0 \in \mathsf{D}.\ell 0 \in \mathsf{L0} \wedge pre(x0) \wedge x = x0 \wedge pc = \ell 0 \Rightarrow J(\ell 0, x0, \ell, x)$
2. $\forall \ell, \ell' \in \mathsf{L}, x, x0 \in \mathsf{D}.\ell 0 \in \mathsf{L0} \wedge pre(x0) \wedge J(\ell 0, x0, \ell, x) \wedge BA(e(\ell, \ell'), )(\ell, x, \ell', x') \Rightarrow J(\ell 0, x0, \ell', x')$
3. $\forall \ell 0, \ell \in \mathsf{L}, x0, x \in \mathsf{D}.pre(x0) \wedge \ell 0 \in \mathsf{L0} \wedge J(\ell 0, x0, \ell, x) \Rightarrow S(\ell 0, x0, \ell, x)$

## (Induction Principle (III))

A property $S(x0, \ell, x)$ is a safety for an annotated program P with one entry point if, and only if, there exists a property $I(x0, \ell, x)$ satisfying :

1. $\forall x0 \in \mathsf{D}.pre(x0) \wedge x = x0 \wedge \ell = \ell 0 \Rightarrow J(x0, \ell, x)$
2. $\forall \ell, \ell' \in \mathsf{L}, x, x0 \in \mathsf{D}.pre(x0) \wedge J(x0, \ell, x) \wedge BA(e(\ell, \ell'), )(\ell, x, \ell', x') \Rightarrow J(x0, \ell', x')$
3. $\forall \ell \in \mathsf{L}, x0, x \in \mathsf{D}.pre(x0) \wedge J(x0, \ell, x) \Rightarrow S(x0, \ell, x)$

# Soundness of the translation

## (Soundness of the method)

If the initialisation init, the generalisation gen and the step induction are proved to be correct by the Rodin platform, the property $S(x0, \ell, x)$ is a correct safety property for the program P. In particular, one can handle the partial correctness and the run time error safety properties.

# Soundness of the translation

**(Soundness of the method)**

If the initialisation init, the generalisation gen and the step induction are proved to be correct by the Rodin platform, the property $S(x0, \ell, x)$ is a correct safety property for the program P. In particular, one can handle the partial correctness and the run time error safety properties.

- Contract and verification conditions are translated into Event-B and are discharged by Rodin and its provers.
- Verification conditions are derived from Floyd's method.
- Annotation as assertion

# 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
$act1 : l := l0$
$act2 : x := x - 1$

# Current Summary

# Summary

1. Programming by contract
2. Verification
3. Floyd to Hoare

# Annotation of programs

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

$$
\begin{array}{l}
e(\ell,\ell') \\
\quad \textcolor{blue}{\text{WHEN}} \\
\qquad c = \ell \\
\qquad cond_{\ell,\ell'}(v) \\
\quad \textcolor{blue}{\text{THEN}} \\
\qquad c := \ell' \\
\qquad v := f_{\ell,\ell'}(v) \\
\quad \textcolor{blue}{\text{END}}
\end{array}
$$

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

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

# From annotations to invariants

**INVARIANTS**

$inv_i : c \in \textsc{Locations}$
$inv_j : v \in Type$
$\ldots$
$inv_k : c = \ell \Rightarrow P_\ell(v)$
$inv_m : c = \ell' \Rightarrow P_{\ell'}(v)$
$\ldots$
$th_n : A(c, v)$

- $Type$ 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 $\ell$, (resp. in $\ell'$).
- $A(c, v)$ is a safety property that we are supposed to check and the case of Event-B, it is a theorem.

# Partial correctness using Event-B models

For each pair of successive labels $\ell, \ell'$, the three statements are equivalent :

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

L

et $\mathrm{AA}$ an annotated algorithm with precondition **pre**$(\mathrm{AA})$(v) and postcondition **post**$(\mathrm{AA})$($v_0$,v). Let the context $C$ and the machine $M$ generated from $\mathrm{AA}$ using the construction given previously. We assume that $\ell_0$ is the first label and $\ell_e$ is the last label. We add the following safety properties in the machine $M$ :

- $c = \ell_0 \wedge$ **pre**$(\mathrm{AA})(v) \Rightarrow P_{\ell_0}(v)$
- $c = \ell_e \Rightarrow (P_{\ell_e}(v) \Rightarrow$ **post**$(\mathrm{AA})(v_0, v)$

If proof obligations are discharged, then the annotated algorithm $\mathrm{AA}$ is partially correct with respect to ist pre/post specification.

# Current Summary

# 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
- $[x := e]P(x) = P[x \mapsto e]$
- $[\text{if } b(x) \text{ then } S1 \text{ else } S2\ ]P(x) = b(x) \wedge [S1]P(x) \vee \text{ not } b(x)\ [S2]P(x)$

# 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 ...