



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

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

# Verifying program correctness

A program  $P$  *satisfies* a  $(\text{pre}, \text{post})$  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  $\text{pre}$  :  $\text{pre}(v_0)$  and  $v_f$  satisfies  $\text{post}$  :  $\text{post}(v_0, v_f)$
- $\text{pre}(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow \text{post}(v_0, v_f)$
- $\mathbb{D}$  est le domaine RTE de  $V$

# Verifying program correctness

A program  $P$  *satisfies* a  $(\text{pre}, \text{post})$  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  $\text{pre} : \text{pre}(v_0)$  and  $v_f$  satisfies  $\text{post} : \text{post}(v_0, v_f)$
- $\text{pre}(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow \text{post}(v_0, v_f)$
- $\mathbb{D}$  est le domaine RTE de  $V$

```
requires  $\text{pre}(v_0)$ 
ensures  $\text{post}(v_0, v_f)$ 
variables  $X$ 
begin
  0 :  $P_0(v_0, v)$ 
  instruction0
  ...
  i :  $P_i(v_0, v)$ 
  ...
  instructionf-1
  f :  $P_f(v_0, v)$ 
end
```

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

# Contracts - Verification Conditions

```
contract P
variables v
requires  $pre(v_0)$ 
ensures  $post(v_0, v_f)$ 
begin
  0 :  $P_0(v_0, v)$ 
  S0
  ...
  i :  $P_i(v_0, v)$ 
  ...
  Sf-1
  f :  $P_f(v_0, v)$ 
end
```

Verification conditions are listed as follows :

```
contract P
variables v
requires  $pre(v_0)$ 
ensures  $post(v_0, v_f)$ 
begin
  0 :  $P_0(v_0, v)$ 
  S0
  ...
  i :  $P_i(v_0, v)$ 
  ...
  Sf-1
  f :  $P_f(v_0, v)$ 
end
```

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

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 :  $pc \in L$

...

atl :  $pc = \ell \Rightarrow P_\ell(v0, v)$

...

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 :  $pc \in L$

...

atl :  $pc = \ell \Rightarrow P_\ell(v0, v)$

...

th1 :  $pre(v0) \wedge v = v0 \Rightarrow P_0(v0, v)$

th2 :  $pre(v0) \wedge P_f(v0, v)$   
 $\Rightarrow post(v0, v)$

...

**END**

...

**END**

**MACHINE**  $M$

**EVENTS**

**INITIALISATION**

**BEGIN**

$(pc, v) : \mid \left( \begin{array}{l} pc' = l0 \wedge v' = v0 \\ \wedge pre(v0) \end{array} \right)$

**END**

...

$e(\ell, \ell')$

**WHEN**

$pc = \ell$

$cond_{\ell, \ell'}(v)$

**THEN**

$pc := \ell'$

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

**END**

...

**END**

## (Induction Principle (I))

A property  $S(z_0, z)$  is a safety for an annotated program  $P$  if, and only if, there exists a property  $I(z_0, z)$  satisfying :

- 1  $\forall z_0, z \in L \times D. \text{init}(z_0) \wedge z = z_0 \Rightarrow I(z_0, z)$
- 2  $\forall z_0, z, z' \in L \times D. \text{init}(z_0) \wedge I(z_0, z) \wedge (z \xrightarrow{P} z') \Rightarrow I(z_0, z')$
- 3  $\forall z_0, z \in L \times D. \text{init}(z_0) \wedge I(z_0, z) \Rightarrow S(z_0, z)$

## (Induction Principle (II))

A property  $S(\ell_0, x_0, \ell, x)$  is a safety property for an annotated program  $P$  if, and only if, there exists a property  $I(\ell_0, x_0, \ell, x)$  satisfying :

- 1  $\forall \ell_0, \in L, x_0 \in D. \ell_0 \in L_0 \wedge \text{pre}(x_0) \wedge x = x_0 \wedge \text{pc} = \ell_0 \Rightarrow J(\ell_0, x_0, \ell, x)$
- 2  $\forall \ell, \ell' \in L, x, x_0 \in D. \ell_0 \in L_0 \wedge \text{pre}(x_0) \wedge J(\ell_0, x_0, \ell, x) \wedge BA(e(\ell, \ell'),)(\ell, x, \ell', x') \Rightarrow J(\ell_0, x_0, \ell', x')$
- 3  $\forall \ell_0, \ell \in L, x_0, x \in D. \text{pre}(x_0) \wedge \ell_0 \in L_0 \wedge J(\ell_0, x_0, \ell, x) \Rightarrow S(\ell_0, x_0, \ell, x)$

## (Induction Principle (II))

A property  $S(\ell_0, x_0, \ell, x)$  is a safety property for an annotated program  $P$  if, and only if, there exists a property  $I(\ell_0, x_0, \ell, x)$  satisfying :

- ①  $\forall \ell_0, \ell \in L, x_0 \in D. \ell_0 \in L_0 \wedge pre(x_0) \wedge x = x_0 \wedge pc = \ell_0 \Rightarrow J(\ell_0, x_0, \ell, x)$
- ②  $\forall \ell, \ell' \in L, x, x_0 \in D. \ell_0 \in L_0 \wedge pre(x_0) \wedge J(\ell_0, x_0, \ell, x) \wedge BA(e(\ell, \ell'), \ell, x, \ell', x') \Rightarrow J(\ell_0, x_0, \ell', x')$
- ③  $\forall \ell_0, \ell \in L, x_0, x \in D. pre(x_0) \wedge \ell_0 \in L_0 \wedge J(\ell_0, x_0, \ell, x) \Rightarrow S(\ell_0, x_0, \ell, x)$

## (Induction Principle (III))

A property  $S(x_0, \ell, x)$  is a safety for an annotated program  $P$  with one entry point if, and only if, there exists a property  $I(x_0, \ell, x)$  satisfying :

- ①  $\forall x_0 \in D. pre(x_0) \wedge x = x_0 \wedge \ell = \ell_0 \Rightarrow J(x_0, \ell, x)$
- ②  $\forall \ell, \ell' \in L, x, x_0 \in D. pre(x_0) \wedge J(x_0, \ell, x) \wedge BA(e(\ell, \ell'), \ell, x, \ell', x') \Rightarrow J(x_0, \ell', x')$
- ③  $\forall \ell \in L, x_0, x \in D. pre(x_0) \wedge J(x_0, \ell, x) \Rightarrow S(x_0, \ell, x)$

(Soundness of the method)

If the initialisation  $\text{init}$ , the generalisation  $\text{gen}$  and the step induction are proved to be correct by the Rodin platform, the property  $S(x_0, \ell, x)$  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 method)

If the initialisation  $\text{init}$ , the generalisation  $\text{gen}$  and the step induction are proved to be correct by the Rodin platform, the property  $S(x_0, \ell, x)$  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

$act1 : x := x_0$

$act2 : l := l_0$

Event *el0l1*

WHEN

$grd1 : l = l_0$

$grd2 : 0 < x$

THEN

$act1 : l := l_1$

INVARIANTS

$inv1 : x \in \mathbb{N}$

$inv2 : l \in L$

$inv3 : l = l_0 \Rightarrow$

$0 \leq x \wedge x \leq x_0 \wedge x_0 \in \mathbb{N}$

$inv4 : l = l_1 \Rightarrow$

$0 < x \wedge x \leq x_0 \wedge x_0 \in \mathbb{N}$

$inv5 : l = l_2 \Rightarrow x = 0$

$requires : x_0 \in \mathbb{N} \wedge x = x_0$

$\Rightarrow x = x_0 \wedge x_0 \in \mathbb{N}$

$ensures : x = 0 \wedge x = x_0$

$\Rightarrow x = 0$

Event *el0l2*

WHEN

$grd1 : l = l_0$

$grd2 : \neg(0 < x)$

THEN

$act1 : l := l_2$

Event *el1l0*

WHEN

$grd1 : l = l_1$

THEN

$act1 : l := l_0$

$act2 : x := x - 1$

# Current Summary

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



## Summary

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

# Annotation of programs

$$\begin{array}{l} \ell : \{P_\ell(v)\} \\ \text{cond}_{\ell, \ell'}(v) \longrightarrow v := f_{\ell, \ell'}(v) \\ \ell' : \{P_{\ell'}(v)\} \end{array}$$
$$\begin{array}{l} e(\ell, \ell') \\ \textbf{WHEN} \\ \quad c = \ell \\ \quad \text{cond}_{\ell, \ell'}(v) \\ \textbf{THEN} \\ \quad c := \ell' \\ \quad v := f_{\ell, \ell'}(v) \\ \textbf{END} \end{array}$$
$$\begin{array}{l} \ell_0^1 : \{x = 0\} \\ \quad x := x + 1; \\ \ell_0^1 : \{x = 1\} \end{array}$$

- $v$  is the state memory 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 **LOCATIONS**.
- $e(\ell, \ell')$  is simulating the computation flow starting from  $\ell$  and moving to  $\ell'$ ;  $v$  is updated.

## From annotations to invariants

## INVARIANTS

$$inv_i : c \in \text{LOCATIONS}$$
$$inv_i : v \in Type$$

• • •

$$inv_k : c = \ell \Rightarrow P_\ell(v)$$
$$\text{inv}_m : c = \ell' \Rightarrow P_{\ell'}(v)$$

• • •

$$th_n : A(c, v)$$

- $Type$  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 :

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

## L

et AA an annotated algorithm with precondition **pre**(AA)(v) and postcondition **post**(AA)(v<sub>0</sub>, v). Let the context *C* and the machine *M* generated from 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* :

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

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

## Current Summary

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

# From Floyd to Hoare

# From Floyd to Hoare

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

# From Floyd to Hoare

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



# From Floyd to Hoare

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

# From Floyd to Hoare

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

# From Floyd to Hoare

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

# From Floyd to Hoare

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

# From Floyd to Hoare

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

# From Floyd to Hoare

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

# From Floyd to Hoare

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