



Cours MVSI Modélisation et Vérifiaction des Systèmes Informatiques

Vérification mécanisée de contrats (I)

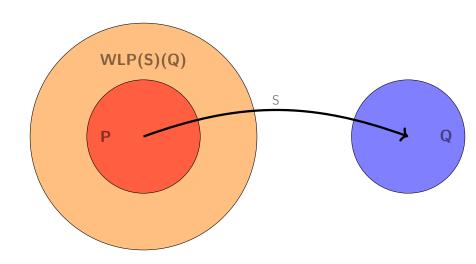
Dominique Méry Telecom Nancy, Université de Lorraine (30 septembre 2025 at 10:14 A.M.)

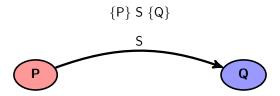
Année universitaire 2025-2026

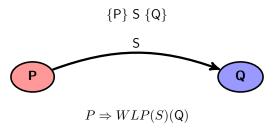
- Programs as Predicate **Transformers**
- 2 Mechanizing the contract checking
- 3 Transforming predicates Hoare Logic for PC Examples in ACSL Définition et propriétés du calcul wp
- 4 Using predicate transformers for checking contracts

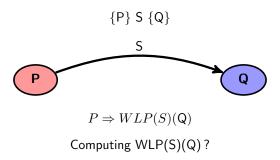
Sommaire

- 1 Programs as Predicate Transformers
- 2 Mechanizing the contract checking
- 3 Transforming predicates Hoare Logic for PC Examples in ACSL Définition et propriétés du calcul wp
- 4 Using predicate transformers for checking contracts









Method for verifying partial correctness and RTE

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

- ▶ P transforms a variable x from an iinitial value x_0 and produces a final value $x_f: x_0 \stackrel{\mathsf{P}}{\longrightarrow} x_f$
- $ightharpoonup x_0$ satisfies pre : $\operatorname{pre}(x_0)$ and x_f satisfies post : $\operatorname{post}(x_0,x_f)$
- ightharpoonup is the domain of x for RTE (No Run Time Errors) .

i

Method for verifying partial correctness and RTE

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

- ▶ P transforms a variable x from an iinitial value x_0 and produces a final value $x_f: x_0 \xrightarrow{\mathsf{P}} x_f$
- $ightharpoonup x_0$ satisfies pre : $\operatorname{pre}(x_0)$ and x_f satisfies post : $\operatorname{post}(x_0,x_f)$
- $ightharpoonup \mathbb{D}$ is the domain of x for RTE (No Run Time Errors) .

Method for verifying partial correctness and RTE

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

- ▶ P transforms a variable x from an iinitial value x_0 and produces a final value $x_f: x_0 \xrightarrow{\mathsf{P}} x_f$
- $ightharpoonup x_0$ satisfies pre : $\operatorname{pre}(x_0)$ and x_f satisfies post : $\operatorname{post}(x_0,x_f)$
- $\blacktriangleright \ \mathbb{D}$ is the domain of x for RTE (No Run Time Errors) .

 $\begin{array}{c} \text{variables } \mathsf{x} : \mathbb{D} \\ \text{requires } pre(x_0) \\ \text{ensures } post(x_0, x_f) \\ \hline \begin{bmatrix} \mathsf{begin} \\ 0 : P_0(x_0, x) \\ \mathsf{S} \\ f : P_f(x_0, x) \\ \mathsf{end} \end{bmatrix} \end{array}$

- $ightharpoonup pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- $ightharpoonup pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$
- For any pair ℓ, ℓ' such that $\ell \longrightarrow \ell'$, we verify that for any values $x, x' \in \text{MEMORY}$

$$\left(\begin{array}{c} P_{\ell}(x_0,x)) \\ \wedge cond_{\ell,\ell'}(x) \wedge x' = f_{\ell,\ell'}(x) \end{array}\right),$$

$$\Rightarrow P_{\ell'}(x_0,x')$$

For any pair m, n such that $m \longrightarrow n$, we verify that $\forall x, x' \in \text{MEMORY}:$ $pre(x_0) \land P_m(x_0, x) \Rightarrow \textbf{DOM}(m, n)(x)$

Checking verificatioon conditions

- $ightharpoonup pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$
- For any pair ℓ,ℓ' such that $\ell \longrightarrow \ell'$, we verify that for any values $x,x' \in \mathrm{MEMORY}$

For any pair m, n such that $m \longrightarrow n$, we verify that $\forall x, x' \in \text{MEMORY} : pre(x_0) \land P_m(x_0, x) \Rightarrow \textbf{DOM}(m, n)(x)$

Example **DOM**(m,n)(x)

$$DOM(\ell_0, \ell_1)(u) = u \in \underline{minint..maxint} \land 5 \in minint..maxint \land u + 5 \in$$

 $minint..maxint \text{ where } \begin{vmatrix} \ell_0 : F_{\ell_0}(u) \\ u := u+5; \end{vmatrix}$

$$\begin{array}{l} \ell_0: P_{\ell_0}(u); \\ {\bf u} := {\bf u} {+} {\bf 5}; \\ \ell_1: P_{\ell_0}(u); \end{array}$$

Intuition

- ► A program P *produces* results or outputs from inputs according to a (operational or denotational) semantics
 - STATES is the set of states of P : STATES = x → Z where x designate variables of P.
 - s_0 et s_f two states of STATES : $\mathcal{D}(P)(s_0) = s_f$ means that P is executed from the memory state s_0 and produces a final state s_f .
 - For any current state s of P, $s(\mathbf{x}) = x$ for expressing the value of \mathbf{x} in state s :

Intuition

- A program P produces results or outputs from inputs according to a (operational or denotational) semantics
 - STATES is the set of states of P : STATES = x → Z where x designate variables of P.
 - s_0 et s_f two states of STATES : $\mathcal{D}(P)(s_0) = s_f$ means that P is executed from the memory state s_0 and produces a final state s_f .
 - For any current state s of P, $s(\mathbf{x}) = x$ for expressing the value of \mathbf{x} in state s :

$$s_0(x) = x_0, \ s_f(x) = x_f, \ s'(x) = x'$$

Intuition

- A program P produces results or outputs from inputs according to a (operational or denotational) semantics
 - STATES is the set of states of P : STATES = x → Z where x designate variables of P.
 - s_0 et s_f two states of STATES : $\mathcal{D}(P)(s_0) = s_f$ means that P is executed from the memory state s_0 and produces a final state s_f .
 - For any current state s of P, $s(\mathbf{x}) = x$ for expressing the value of \mathbf{x} in state s :

$$s_0(x) = x_0, \ s_f(x) = x_f, \ s'(x) = x'$$

• $\mathcal{D}(P)(s_0) = s_f$ defines the computation nrelation over the set of states :

$$x_0 \stackrel{\mathsf{P}}{\longrightarrow} x_f$$

- A program P produces results or outputs from inputs according to a (operational or denotational) semantics
 - STATES is the set of states of P : STATES = x → Z where x designate variables of P.
 - s_0 et s_f two states of STATES : $\mathcal{D}(P)(s_0) = s_f$ means that P is executed from the memory state s_0 and produces a final state s_f .
 - For any current state s of P, $s(\mathbf{x}) = x$ for expressing the value of \mathbf{x} in state s :

$$s_0(x) = x_0, \ s_f(x) = x_f, \ s'(x) = x'$$

• $\mathcal{D}(P)(s_0) = s_f$ defines the computation nrelation over the set of states :

$$x_0 \stackrel{\mathsf{P}}{\longrightarrow} x_f$$

- ► A program P *satisfies* the contract (x,pre,post) :
 - P transforms a variable x from a value x_0 and produces a value x_f :

$$x_0 \stackrel{\mathsf{P}}{\longrightarrow} x_f$$

- x₀ satisfies pre : pre(x₀)
- x_f satisfies post : post (x_0, x_f)
- $\operatorname{pre}(x_0) \wedge x_0 \stackrel{\mathsf{P}}{\longrightarrow} x_f \Rightarrow \operatorname{post}(x_0, x_f)$

Checking a contract

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

- ▶ P transforms a variable x from an iinitial value x_0 and produces a final value $x_f: x_0 \xrightarrow{P} x_f$
- $ightharpoonup x_0$ satisfies pre : $\operatorname{pre}(x_0)$ and x_f satisfies post : $\operatorname{post}(x_0,x_f)$

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

- ▶ P transforms a variable x from an iinitial value x_0 and produces a final value $x_f: x_0 \xrightarrow{P} x_f$
- $ightharpoonup x_0$ satisfies pre : $\operatorname{pre}(x_0)$ and x_f satisfies post : $\operatorname{post}(x_0,x_f)$

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

- ▶ P transforms a variable x from an iinitial value x_0 and produces a final value $x_f: x_0 \xrightarrow{\mathsf{P}} x_f$
- $ightharpoonup x_0$ satisfies pre : $\operatorname{pre}(x_0)$ and x_f satisfies post : $\operatorname{post}(x_0,x_f)$

```
variables \mathbf{x}:\mathbb{D} requires pre(x_0) ensures post(x_0,x_f) begin 0:P_0(x_0,x) S f:P_f(x_0,x) end
```

- For any pair ℓ, ℓ' such that $\ell \longrightarrow \ell'$, we verify that for any values $x, x' \in \text{MEMORY}$ $\left(\begin{array}{c} P_{\ell}(x_0, x) \\ \end{array} \right)$

Checking a contract using the solver Z3

$$x0 \ge 0 \land x = x_0 \Rightarrow x = x0$$

$$ightharpoonup x = x0+2 \Rightarrow x = x0+2$$

conditions de vérification
$$0 \longrightarrow 1$$
:
 $x = x0 \land x' = x+2 \Rightarrow x' = x0+2$

$$(x0 >= 0, x == x0, x! = x0)$$

$$(x == x0+2, x! = x0+2)$$

$$(x == x0, xp == x+2, xp! = x0+2)$$

Listing 1 – z3 en Python

```
from numbers import Real
from z3 import *
x = Real('x')
xp = Real('x')'
x0 = Real('x')'
s = Solver()
s = add(x0 >= 0, x == x0, x != x0)
print(s.check())
s.add(x = x0+2, x != x0+2)
print(s.check())
s.add(x = x0, xp == x + 2, xp != x0+2)
print(s.check())
```

 $\qquad \forall x_0, x_f.\mathsf{pre}(x_0) \land x_0 \overset{\mathsf{P}}{\longrightarrow} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

- $\qquad \forall x_0, x_f.\mathsf{pre}(x_0) \land x_0 \overset{\mathsf{P}}{\longrightarrow} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\blacktriangleright \ \forall x_0, x. \mathsf{pre}(x_0) \Rightarrow (x_0 \overset{\mathsf{P}}{\longrightarrow} x \Rightarrow \mathsf{post}(x_0, x))$

- $\qquad \forall x_0, x_f.\mathsf{pre}(x_0) \land x_0 \overset{\mathsf{P}}{\longrightarrow} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\blacktriangleright \ \forall x_0, x. \mathsf{pre}(x_0) \Rightarrow (x_0 \overset{\mathsf{P}}{\longrightarrow} x \Rightarrow \mathsf{post}(x_0, x))$

- $\qquad \forall x_0, x_f.\mathsf{pre}(x_0) \land x_0 \overset{\mathsf{P}}{\longrightarrow} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

- $\blacktriangleright \ \forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x.x_0 \overset{\mathsf{P}}{\longrightarrow} x \Rightarrow \mathsf{post}(x_0,x)$

- $\forall x_0, x_f.\mathsf{pre}(x_0) \land x_0 \stackrel{\mathsf{P}}{\longrightarrow} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

- $\qquad \forall x_0, x_f.\mathsf{pre}(x_0) \land x_0 \overset{\mathsf{P}}{\longrightarrow} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0, x. \mathsf{pre}(x_0) \Rightarrow (x_0 \overset{\mathsf{P}}{\longrightarrow} x \Rightarrow \mathsf{post}(x_0, x))$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x.x_0 \overset{\mathsf{P}}{\longrightarrow} x \Rightarrow \mathsf{post}(x_0,x)$
- $ightharpoonup \forall x_0.\mathsf{pre}(x_0) \Rightarrow \{P\}\mathsf{post}(x_0,x)$

$$\{S\}P(x) \stackrel{def}{=} \forall x_f.x \stackrel{\mathsf{S}}{\longrightarrow} x_f \Rightarrow \mathsf{P}(x_f)$$



- $\forall x_0, x_f.\mathsf{pre}(x_0) \land x_0 \overset{\mathsf{P}}{\longrightarrow} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0, x.\operatorname{pre}(x_0) \Rightarrow (x_0 \stackrel{\mathsf{P}}{\longrightarrow} x \Rightarrow \operatorname{post}(x_0, x))$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x.x_0 \overset{\mathsf{P}}{\longrightarrow} x \Rightarrow \mathsf{post}(x_0,x)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \{P\}\mathsf{post}(x_0,x)$

$$\{S\}P(x) \stackrel{def}{=} \forall x_f.x \stackrel{\mathsf{S}}{\longrightarrow} x_f \Rightarrow \mathsf{P}(x_f)$$



- ightharpoonup WLP(S)(P(x)) is another notation for $\{S\}P(x)$.
- $\blacktriangleright \ \{ \text{while} \ b(x) \ \text{do} \ S \ \text{end} \} P(x) = \{ w \} (P(x))$

- ▶ WLP(S)(P(x)) is another notation for $\{S\}P(x)$.
- $\blacktriangleright \text{ {while } } b(x) \text{ do } S \text{ end} \} P(x) = \{w\} (P(x))$
- {if b(x) then S; w else skip }P(x) =

- ▶ WLP(S)(P(x)) is another notation for $\{S\}P(x)$.
- $\bullet \text{ {while }} b(x) \text{ do } S \text{ end} \\ P(x) = \{w\}(P(x))$
- {if b(x) then S; w else skip }P(x) =
- $\blacktriangleright \ b(x) \wedge \{S;w\}P(x) \vee \ \text{not} \ b(x) \ \wedge \ \{skip\}P(x) =$

- WLP(S)(P(x)) is another notation for $\{S\}P(x)$.
- $\blacktriangleright \text{ {while } } b(x) \text{ do } S \text{ end} \} P(x) = \{w\} (P(x))$
- {if b(x) then S; w else skip }P(x) =
- $lackbox{b}(x) \land \{S; w\} P(x) \lor \text{not } b(x) \land \{skip\} P(x) =$
- ▶ $b(x) \land \{S\}(\{w\}(P(x))) \lor \text{not } b(x) \land P(x) = \{w\}(P(x))$
- $F(\{w\})(P(x)) = \{w\}(P(x))$

Examples

- {while x > 0 do x := x-1 end} $(x = 0) = x \ge 0$
- {while x > 0 do x := x+1 end} $(x = 0) = x \ge 0$

Expressing a contract using predicate transformer

Computing WLP function

- $ightharpoonup \forall x_0.\mathsf{pre}(x_0) \Rightarrow \{P\}\mathsf{post}(x_0,x)$
- $\blacktriangleright \ \forall x_0.x = x_0 \land \mathsf{pre}(x_0) \Rightarrow \{P\}\mathsf{post}(x_0,x)$
- ► Hoare Triple : $\{pre(x_0) \land x = x_0\}P\{post(x_0, x)\}$

Axiomatisation de la Logique de Hoare

☑ Definition(Axiomes et règles d'inférence)

- Axiome d'affectation : $\{P(e/x)\}$ **X** :=**E(X)** $\{P\}$.
- Axiome du saut : $\{P\}$ **skip** $\{P\}$.
- ▶ Règle de composition : Si $\{P\}\mathbf{S}_1\{R\}$ et $\{R\}\mathbf{S}_2\{Q\}$, alors $\{P\}\mathbf{S}_1$; $\mathbf{S}_2\{Q\}$.
- ▶ Si $\{P \land B\}$ S₁ $\{Q\}$ et $\{P \land \neg B\}$ S₂ $\{Q\}$, alors $\{P\}$ if B then S₁ then S₂ fi $\{Q\}$.
- ▶ Si $\{P \land B\}$ **S** $\{P\}$, alors $\{P\}$ while **B** do **S** od $\{P \land \neg B\}$.
- ▶ Règle de renforcement/affaiblissement : Si $P' \Rightarrow P$, $\{P\}$ **S** $\{Q\}$, $Q \Rightarrow Q'$, alors $\{P'\}$ **S** $\{Q'\}$.

Exemple de preuve $\{x=1\}$ **Z** :=**X**;**X** :=**Y**;**Y** :=**Z** $\{y=1\}$

- ▶ (1) $x = 1 \Rightarrow (z = 1)[x/z]$ (propriété logique)
- (2) $\{(z=1)[x/z]\}$ **Z** :=**X** $\{z=1\}$ (axiome d'affectation)
- ▶ (3) $\{x=1\}$ **Z** :=**X** $\{z=1\}$ (Règle de renforcement/affaiblissement avec (1) et (2))
- ► (4) $z = 1 \Rightarrow (z = 1)[y/x]$ (propriété logique)
- (5) $\{(z=1)[y/x]\}$ **X** :=**Y** $\{z=1\}$ (axiome d'affectation)
- ▶ (6) $\{z=1\}$ **X** :=**Y** $\{z=1\}$ (Règle de renforcement/affaiblissement avec (4) et (5))
- ▶ (7) $z = 1 \Rightarrow (y = 1)[z/y]$ (propriété logique)
- (8) $\{(z=1)[x/z]\}$ **Y** :=**Z** $\{y=1\}$ (axiome d'affectation)
- (9) $\{z=1\}$ **Y** :=**Z** $\{y=1\}$ (Règle de renforcement/affaiblissement avec (7) et (8))
- (10) $\{x = 1\}$ **Z** :=**X**;**X** :=**Y**; $\{z = 1\}$ (Règle de composition avec 3 et 6)
- ▶ (11) $\{x = 1\}$ **Z** :=**X**;**X** :=**Y**;**Y** :=**Z** $\{y = 1\}$ (Règle de composition avec 11 et 9)

Sémantique des triplets de Hoare

.....

□ Definition

 $\{P\}\mathbf{S}\{Q\} \text{ est défini par } \forall s,t \in STATES: P(s) \land \mathcal{D}(S)(s) = t \Rightarrow Q(t)$

- © Property Correction du système axiomatique des programmes commentés
 - S'il existe une preuve construite avec les règles précédentes de $\{P\}$ **S** $\{Q\}$, alors $\{P\}$ **S** $\{Q\}$ est valide.
 - ▶ Si $\{P'\}$ **S** $\{Q'\}$ est valide et si le langage d'assertions est suffisamment expressif, alors il existe une preuve construite avec les règles précédentes de $\{P\}$ **S** $\{Q\}$.

.....

□ Definition

Un langage d'assertions est la donnée d'un ensemble de prédicats et d'opérateurs de composition comme la disjonction et la conjonction ; il est muni d'une relation d'ordre partielle appelée implication. On le notera $(\operatorname{PRED}, \Rightarrow, \mathbf{false}, \mathbf{true}, \wedge, \vee) : (\operatorname{PRED}, \Rightarrow, \mathbf{false}, \mathbf{true}, \wedge, \vee)$ est un treillis complet

Introduction de wlp

- ▶ {*P*}**S**{*Q*}
- $\forall s, t \in STATES : P(s) \land \mathcal{D}(S)(s) = t \Rightarrow Q(t)$
- $\forall s \in STATES : P(s) \Rightarrow (\forall t \in STATES : \mathcal{D}(S)(s) = t \Rightarrow Q(t))$

.....

Définition de wlp

$$wlp(S)(Q) \stackrel{def}{=} (\forall t \in STATES : \mathcal{D}(S)(s) = t \Rightarrow Q(t))$$

$$wlp(S)(Q) \equiv \overline{(\exists t \in STATES : \mathcal{D}(S)(s) = t \land \overline{Q}(t))}$$

Lien entre wp et wlp

- ▶ $loop(S) \equiv (\exists t \in STATES : \mathcal{D}(S)(s) = \overline{t} \text{ (ensemble des états qui ne permettent pas à S de terminer)}$
- $\blacktriangleright wp(S)(Q) \equiv wlp(S)(Q) \wedge \overline{loop(S)}$

Définition de wlp

□ Definition

$$WLP(S)(P) = \nu \lambda X.((B \wedge wlp(BS)(X)) \vee (\neg B \wedge P))$$

- © Property
 - ▶ Si $P \Rightarrow Q$, then $wlp(S)(P) \Rightarrow wlp(S)(Q)$.

Axiomatisation de la Logique de Hoare

□ Definitiontriplets de Hoare

$$\{P\}\mathbf{S}\{Q\} \stackrel{def}{=} P \Rightarrow wlp(S)(Q)$$

Axiomatisation de la Logique de Hoare

.....

□ Definitiontriplets de Hoare

$$\{P\}\mathbf{S}\{Q\} \stackrel{def}{=} P \Rightarrow wlp(S)(Q)$$

□ Definition(Axiomes et règles d'inférence)

- Axiome d'affectation : $\{P(e/x)\}X := E(X)\{P\}$.
- Axiome du saut : $\{P\}$ **skip** $\{P\}$.
- ▶ Règle de composition : Si $\{P\}\mathbf{S}_1\{R\}$ et $\{R\}\mathbf{S}_2\{Q\}$, alors $\{P\}\mathbf{S}_1$; $\mathbf{S}_2\{Q\}$.
- ▶ Si $\{P \land B\}$ S₁ $\{Q\}$ et $\{P \land \neg B\}$ S₂ $\{Q\}$, alors $\{P\}$ if B then S₁ then S₂ fi $\{Q\}$.
- ▶ Si $\{P \land B\}$ **S** $\{P\}$, alors $\{P\}$ while **B** do **S** od $\{P \land \neg B\}$.
- Règle de renforcement/affaiblissement : Si $P' \Rightarrow P$, $\{P\}$ **S** $\{Q\}$, $Q \Rightarrow Q'$, alors $\{P'\}$ **S** $\{Q'\}$.

- $ightharpoonup \{P\} \mathbf{S}\{Q\}$
- $\forall s \in STATES.P(s) \Rightarrow wlp(S)(Q)(s)$
- $\blacktriangleright \forall s \in STATES.P(s) \Rightarrow (\forall t \in STATES : \mathcal{D}(S)(s) = t \Rightarrow Q(t))$
- $\forall s, t \in STATES.P(s) \land \mathcal{D}(S)(s) = t \Rightarrow Q(t)$
- ▶ Correction : Si on a construit une preuve de $\{P\}$ **S** $\{Q\}$ avec les règles de la logique de Hoare, alors $P \Rightarrow wlp(S)(Q)$
- ▶ Complétude sémantique : Si $P \Rightarrow wlp(S)(Q)$, alors on peut construire une preuve de $\{P\}\mathbf{S}\{Q\}$ avec les règles de la logique de Hoare si on peut exprimer wlp(S)(P) dans le langae d'assertions.

```
Listing 2 – difference of two numbers
```

```
#include <limits.h>
/*@ requires a-b >= INT_MIN && a-b <= INT_MAX;
    assigns \nothing;
    ensures \result == (a - b);
*/
static int difference(int a, int b) {
    return a-b;
}</pre>
```

- INT_MIN (resp. INT_MAX) is the smallest codable integer (resp. greatest codable integer).
- ▶ $a0-b0 \ge INT_MIN \land a0-b0 \le INT_MAX \land a = a0 \land b = b0 \Rightarrow [\backslash result = a-b](\backslash result = (a-b))$

```
Listing 3 - incrément de nombre
/*0 requires x0 >= 0;
    assigns \ nothing;
    ensures \ result == x0+2:
  @*/
int exemple(int x0) {
  int x=x0:
  //@ assert x == x0;
 x = x + 2:
//@ assert x = x0+2;
return x:
```

Computing WLP

requires $x0 \ge 0$; ensures $x_f = x0+2$; variables x

begin

$$intx = x0;$$

 $0: x = x0$
 $x := x+2;$
 $1: x = x0+2$

Conditions de vérification $0 \longrightarrow 1$:

$$x = x0 \land x' = x+2 \Rightarrow x' = x0+2$$

$$x = x0 \Rightarrow (x' = x+2 \Rightarrow x' = x0+2)$$

$$x = x0 \Rightarrow (x+2 = x0+2)$$

$$wp(x := x+2)(x = x0+2) = (x+2 = x0+2)$$

$$\blacktriangleright$$
 $x = x0 \land x0 \ge 0 \Rightarrow wp(x := x+2)(x = x0+2)$

$$x = x0 \land x0 \ge 0 \Rightarrow x+2 = x0+2$$

$$x = x0 \land x0 \ge 0 \Rightarrow x0+2 = x0+2$$

$$\blacktriangleright x0 \ge 0 \land x = x_0 \Rightarrow x = x0$$

$$x = x0+2 \Rightarrow x = x0+2$$

$$x = x0 \Rightarrow wp(x := x+2)(x = x0+2)$$



calcul de wp(X := X+2)(x = x0+2)

```
Listing 4 – incrément de nombre
/*0 requires x0 >= 0;
    assigns \ nothing:
    ensures \ result = x0+1:
  @*/
int exemple(int x0) {
  int x=x0:
 //@ assert x == x0;
 x = x + 2:
//@ assert x==x0+2;
return x;
//@ assert \result = x0+2;
```

```
Listing 5 – incrément de nombre
/*0 requires x0 >= 0;
    assigns \ nothing;
    ensures \ result = x0:
  @*/
int exemple(int x0) {
  int x=x0:
//@ assert x = x0+1;
 x = x + 2:
//@ assert x==x0+2;
return x;
```

Opérateur WP

Soit STATES l'ensemble des états sur l'ensemble X des variables. Soit S une instruction de programme sur X. Soit A une partie de STATES. $s \in WP(S)(A)$, si la condition suivante est vérifiée :

$$\left(\begin{array}{l} \forall t \in STATES : \mathcal{D}(S)(s) = t \Rightarrow t \in A \\ \land \\ \exists t \in STATES : \mathcal{D}(S)(s) = t \end{array}\right)$$

- $WP(X := X+1)(A) = \{ s \in STATES | s[X \mapsto s(X) \oplus 1] \in A \}$
- $WP(X := Y+1)(A) = \{ s \in STATES | s[X \mapsto s(Y) \oplus 1] \in A \}$
- ▶ $WP(while \ X > 0 \ do \ X := X 1 \ od)(A) = \{s \in STATES | (s(X) \le 0) \lor (s(X) \in A \land s(X) < 0)\}$
- ▶ $WP(while \ x > 0 \ do \ x := x+1 \ od)(A) = \{s \in STATES | (s(X) \in A \land s(X) \le 0)\}$
- \blacktriangleright WP(while x > 0 do x := x+1 od)(\varnothing) = \varnothing
- $WP(while \ x > 0 \ do \ x := x+1 \ od)(STATES) = \{s \in STATES | s(X) < 0\}$

Propriétés

- ▶ WP est une fonction monotone pour l'inclusion d'ensembles de STATES.
- $\blacktriangleright WP(S)(\varnothing) = \varnothing$
- $\blacktriangleright WP(S)(A \cap B) = WP(S)(A) \cap WP(S)(B)$
- $\blacktriangleright WP(S)(A)\cup WP(S)(B) \subseteq WP(S)(A\cup B)$
- ▶ Si S est déterministe, $WP(S)(A \cup B) = WP(S)(A) \cup WP(S)(B)$
- WP est un opérateur avec le profil suivant

pour toute instruction S du langage de programmation, $WP(S) \in \mathcal{P}(STATES) \rightarrow \mathcal{P}(STATES)$

- \triangleright $(\mathcal{P}(STATES), \subseteq)$ est un treillis complet.
- $ightharpoonup (Pred, \Rightarrow)$ est une structure où
 - (1) Pred est une extension du langage d'expressions booléennes
 - (2) Pred est une intension introduite comme un langage d'assertions
 - ⇒ est l'implication
- $s \in A$ correspond une assertion P vraie en s notée P(s). Vérification mécanisée de contrats (I) (30 septembre 2025) (Dominique Méry)

Définition structurelle des transformateurs de prédicats

- S est une instruction de STATS.
- ► *T* est le type ou les types des variables et *D* est la constante ou les constantes Définie(s).
- ▶ P est un prédicat du langage Pred
- ► X est une variable de programme
- ▶ E(X, D) (resp. B(X, D)) est une expression arithmétique (resp. booléenne) dépendant de X et de D.
- ightharpoonup x est la valeur de X (X contient la valeur x).
- ullet e(x,d) (resp. b(x,d)) est l'expression arithmétique (resp. booléenne) du langage Pred associée à l'expression E(X,D) (resp. B(X,D)) du langage des expressions arithmétiques (resp. booléennes) du langage de programmation Prog
- $lackbox{b}(x,d)$ est l'expression arithmétique du langage Pred associée à l'expression E(X,D) du langage des expressions arithmétiques du langage de programmation Prog

Définition structurelle des transformateurs de prédicats

S	wp(S)(P)
X := E(X,D)	P[e(x,d)/x]
SKIP	P
$S_1; S_2$	$wp(S_1)(wp(S_2)(P))$
IF $B S_1$ ELSE S_2 FI	$(B \Rightarrow wp(S_1)(P)) \land (\neg B \Rightarrow wp(S_2)(P))$
WHILE B DO S OD	$\mu.(\lambda X.(B \Rightarrow wp(S)(X)) \land (\neg B \Rightarrow P))$

- $ploons wp(X := X+5)(x \ge 8) \stackrel{def}{=} x+5 \ge 8 \land x \ge 3$
- \blacktriangleright wp(WHILE x > 1 DO X := X+1 OD)(x = 4) = FALSE
- $ightharpoonup wp(WHILE \ x > 1 \ DO \ X := X+1 \ OD)(x=0) = x=0$

Logique de Hoare Correction Totale

□ Definitiontriplets de Hoare Correction Totale

$$[P]\mathbf{S}[Q] \stackrel{def}{=} P \Rightarrow wp(S)(Q)$$

Logique de Hoare Correction Totale

☑ Definitiontriplets de Hoare Correction Totale

$$[P]\mathbf{S}[Q] \stackrel{def}{=} P \Rightarrow wp(S)(Q)$$

☑ Definition(Axiomes et règles d'inférence)

- Axiome d'affectation : [P(e/x)]X := E(X)[P].
- Axiome du saut : [P]**skip**[P].
- ightharpoonup Règle de composition : Si $[P]\mathbf{S}_1[R]$ et $[R]\mathbf{S}_2[Q]$, alors $[P]\mathbf{S}_1$; $\mathbf{S}_2[Q]$.
- ▶ Si $[P \land B]$ S₁[Q] et $[P \land \neg B]$ S₂[Q], alors [P]if B then S₁ then S₂ fi[Q].
- ▶ Si [P(n+1)]**S**[P(n)], $P(n+1) \Rightarrow b$, $P(0) \Rightarrow \neg b$, alors $[\exists n \in \mathbb{N}.P(n)]$ while **B** do **S** od[P(0)].
- ▶ Règle de renforcement/affaiblissement : Si $P' \Rightarrow P$, $[P]\mathbf{S}[Q]$, $Q \Rightarrow Q'$, alors $[P']\mathbf{S}[Q']$.

Correction

:

Si $[P]\mathbf{S}[Q]$ est dérivé selon les règles ci-dessus, alors $P\wp(S)5Q)$.

- ▶ [P(e/x)]**X** :=**E(X)**[P] est valide : wp(X := E)(P)/x = P(e/x).
- ▶ $[\exists n \in \mathbb{N}.P(n)]$ while **B** do **S** od[P(0)]: si s est un état de P(n) alors au bout de n boucles on atteint un état s_f tel que P(0) est vrai en s_f .

Complétude

÷

Si $P\Rightarrow wp(S)(Q)$, alors il existe une preuve de $[P]\mathbf{S}[Q]$ construites avec les règles ci-dessus,

- ▶ $P \Rightarrow wp(X := E(X))(Q) : P \Rightarrow Q(e/x)$ et [Q(e/x)]**X** :=**E(X)**[Q] constituent une preuve.
- $ightharpoonup P \Rightarrow wp(while)(Q)$:
 - On construit la suite de P(n) en définissant $P(n) = W_n$.
 - On vérifie que cela vérifie la règle du while.

Verification of contract (I)

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

- ▶ P transforms a variable x from an initial value x_0 and produces a final value $x_f: x_0 \xrightarrow{P} x_f$
- ightharpoonup x₀ satisfait pre : pre(x_0) and x_f satisfait post : post(x_0, x_f)
- $\qquad \qquad \mathsf{pre}(x_0) \wedge x_0 \overset{\mathsf{P}}{\longrightarrow} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

```
requires pre(x_0)
ensures post(x_0, x_f)
variables X
                  \begin{bmatrix} \text{begin} \\ 0: P_0(x_0, x) \\ \text{instruction}_0 \\ \dots \\ i: P_i(x_0, x) \\ \dots \\ \text{instruction}_{f-1} \\ f: P_f(x_0, x) \end{bmatrix}
```

- $ightharpoonup pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$
- For each pair ℓ,ℓ' such that $\ell \longrightarrow \ell'$, one checks that foa any value $x,x' \in \text{MEMORY}$

$$\begin{pmatrix}
pre(x_0) \land P_{\ell}(x_0, x)) \\
\land cond_{\ell, \ell'}(x) \land x' = f_{\ell, \ell'}(x)
\end{pmatrix},$$

$$\Rightarrow P_{\ell'}(x_0, x')$$

Verification du contract (II)

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

- \triangleright P transforms a variable x from an initial value x_0 and produces a final value $x_f: x_0 \stackrel{\mathsf{P}}{\longrightarrow} x_f$
- \triangleright x₀ satisfait pre : pre(x₀) and x_f satisfait post : post(x₀, x_f)
- $ightharpoonup \operatorname{pre}(x_0) \wedge x_0 \stackrel{\mathsf{P}}{\longrightarrow} x_f \Rightarrow \operatorname{post}(x_0, x_f)$

```
requires pre(x_0)
ensures post(x_0, x_f)
variables X
           \begin{aligned} & \text{begin} \\ & 0: P_0(x_0, x) \\ & \text{instruction}_0 \end{aligned}
           i:P_i(x_0,x)
            instruction_{f-1}
            f: P_f(x_0, x)
```

- $\blacktriangleright \forall x_f, x_0.\mathsf{pre}(x_0) \land x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow$ $post(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow (x_0 \overset{\mathsf{P}}{\longrightarrow} x_f \Rightarrow$ $post(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x_f.(x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow$ $post(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x.(x_0 \xrightarrow{\mathsf{P}} x \Rightarrow$ $post(x_0,x)$
- $\blacktriangleright \forall x_0.\mathsf{pre}(x_0) \Rightarrow$ $WLP(P)(\mathsf{post}(x_0,x))$

Partial Correctness by computing WLPs

Un programme P satisfies a contract (pre,post) :

- ▶ P transforms a variable x from an initial value x_0 and produces a final value $x_f: x_0 \xrightarrow{P} x_f$
- ightharpoonup x₀ satisfies pre : pre(x_0) and x_f satisfies post : post(x_0, x_f)
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow WLP(P)(\mathsf{post}(x_0,x))$

Un programme P satisfies a contract (pre,post) :

- ▶ P transforms a variable x from an initial value x_0 and produces a final value $x_f: x_0 \xrightarrow{P} x_f$
- ightharpoonup x₀ satisfies pre : pre(x_0) and x_f satisfies post : post(x_0, x_f)
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow WLP(P)(\mathsf{post}(x_0,x))$
- WLP is not computable . . .
- ▶ Using Hoare logic in the WLP computing as suggested by Rustan Leino. de WLP.

Verification of contract (III)

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

- ▶ P transforms a variable x from an initial value x_0 and produces a final value $x_f: x_0 \xrightarrow{P} x_f$
- ightharpoonup x₀ satisfait pre : pre(x_0) and x_f satisfait post : post(x_0, x_f)

```
requires pre(x_0)
ensures post(x_0, x_f)
variables X
        /·@assert P_0(x_0,x)·/
        /·@loop invariant I(x_0, x)·/
        while B(x) do
        /\cdot@assert P_f(x_0,x)\cdot/
        end
```

- $x = x_0 \land \operatorname{pre}(x_0) \Rightarrow P_0(x_0, x)$
- $I(x_0, x) \land B(x) \Rightarrow WLP(S)(I(x_0, x))$
- $I(x_0, x) \land \neg B(x) \Rightarrow P_f(x_0, x)$

Verification of contract (IV)

```
\begin{array}{c} \bullet & x = \\ x_0 \land \mathsf{pre}(x_0) \Rightarrow P_0(x_0, x) \end{array}
```

$$P_0(x_0, x) \Rightarrow WLP(S1; S2)(P_f(x_0, x))$$

Verification of contract (V)

```
requires pre(x_0)
ensures post(x_0, x_f)
variables X
        /\cdot @assert P_0(x_0,x)\cdot / if B(x) do
         elfi
          /\cdot@assert P_f(x_0,x)\cdot/
         end
```

```
x = x_0 \land \operatorname{pre}(x_0) \Rightarrow P_0(x_0, x)
P_0(x_0, x) \Rightarrow
B(x) \land WLP(S1)(P_f(x_0, x))
\lor
\neg B(x) \land WLP(S2)(P_f(x_0, x))
```