
Cours MALG & MOVEX

Vérification mécanisée de contrats (I)

Dominique Méry
Telecom Nancy, Université de Lorraine

Année universitaire 2023-2024

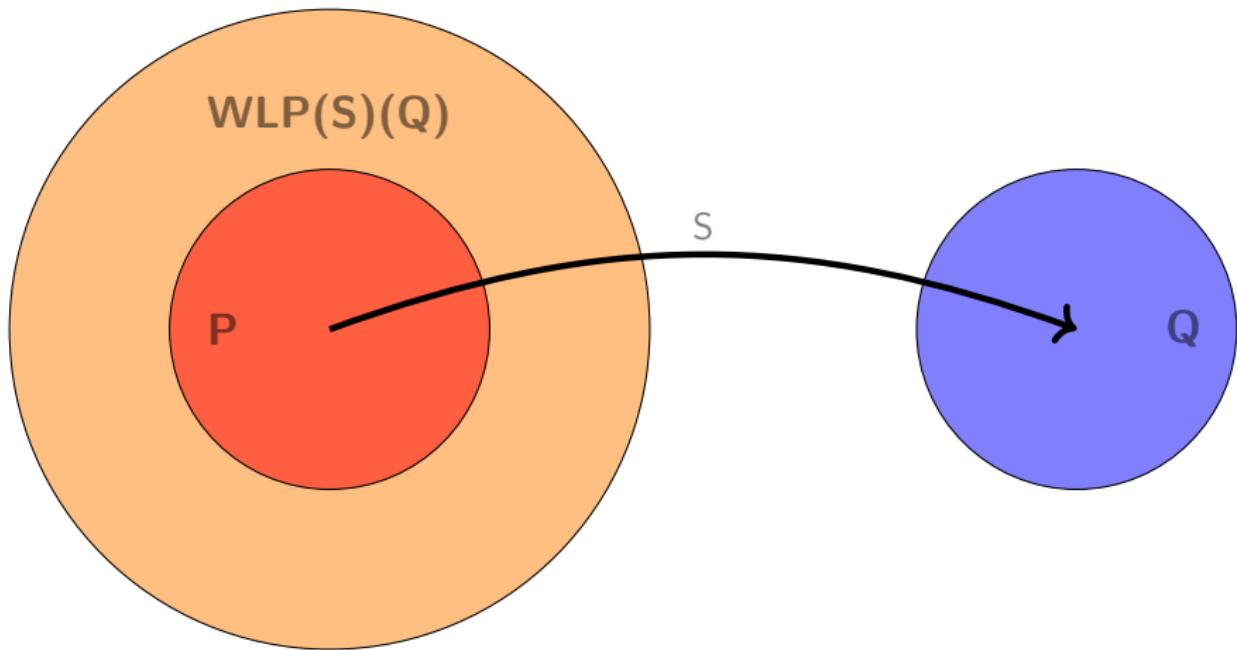
- ① Programs as Predicate
Transformers
- ② Mechanizing the contract
checking
- ③ Transforming predicates
Hoare Logic for PC
Examples in ACSL
Définition et propriétés du
calcul wp
- ④ Using predicate transformers for
checking contracts

- 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

- 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

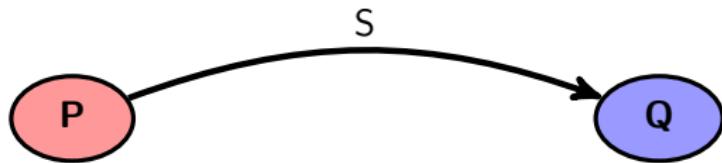
- ① Programs as Predicate Transformers
- ② Mechanizing the contract checking
- ③ Transforming predicates
 - Hoare Logic for PC
 - Examples in ACSL
 - Définition et propriétés du calcul wp
- ④ Using predicate transformers for checking contracts

Asserted Program $\{P\} S \{Q\}$



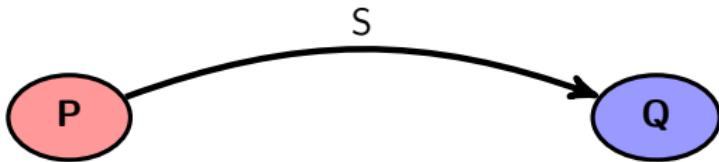
Asserted Program {P} S {Q}

$\{P\} \leq \{Q\}$



Asserted Program {P} S {Q}

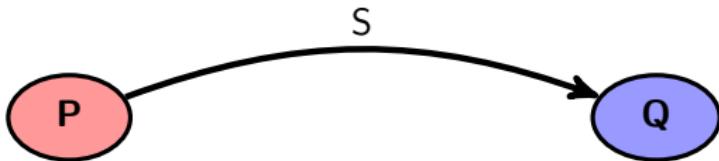
$\{P\} \leq \{Q\}$



$$P \Rightarrow WLP(S)(Q)$$

Asserted Program {P} S {Q}

$\{P\} \leq \{Q\}$



$$P \Rightarrow WLP(S)(Q)$$

Computing WLP(S)(Q) ?

- 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 $(x, \text{pre}, \text{post})$:

- ▶ P transforms a variable x from an initial value x_0 and produces a final value $x_f : x_0 \xrightarrow{P} x_f$
- ▶ x_0 satisfies pre : $\text{pre}(x_0)$ and x_f satisfies post : $\text{post}(x_0, x_f)$
- ▶ $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$
- ▶ \mathbb{D} 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 $(x, \text{pre}, \text{post})$:

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

i

```

variables 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

```

Method for verifying partial correctness and RTE

A program P satisfies a contract $(x, \text{pre}, \text{post})$:

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

i

```

variables 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

```

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

$$\left(\begin{array}{c} P_\ell(x_0, x) \\ \wedge cond_{\ell, \ell'}(x) \wedge x' = f_{\ell, \ell'}(x) \\ \Rightarrow P_{\ell'}(x_0, x') \end{array} \right),$$
 - ▶ For any pair m, n such that $m \longrightarrow n$, we verify that $\forall x, x' \in \text{MEMORY}$:
 $pre(x_0) \wedge P_m(x_0, x) \Rightarrow \text{DOM}(m, n)(x)$

Checking verification conditions

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

$$\left(\begin{array}{l} P_\ell(x_0, x) \\ \wedge cond_{\ell, \ell'}(x) \wedge x' = f_{\ell, \ell'}(x) \\ \Rightarrow P_{\ell'}(x_0, x') \end{array} \right),$$
 - ▶ For any pair m, n such that $m \rightarrow n$, we verify that
 $\forall x, x' \in \text{MEMORY} : pre(x_0) \wedge P_m(x_0, x) \Rightarrow \text{DOM}(m, n)(x)$

Example DOM(m,n)(x)

$$DOM(\ell_0, \ell_1)(u) = u \in minint..maxint \wedge 5 \in minint..maxint \wedge u+5 \in$$

minint..maxint where

```

 $\ell_0 : P_{\ell_0}(u);$ 
 $u := u + 5;$ 
 $\ell_1 : P_{\ell_0}(u);$ 

```

- ▶ A program P *produces* results or outputs from inputs according to a (operational or denotational) semantics
 - STATES is the set of states of P : $STATES = x \rightarrow \mathbb{Z}$ 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 , $s(x) = x$ for expressing the value of x in state s :

- A program P *produces* results or outputs from inputs according to a (operational or denotational) semantics

- STATES is the set of states of P : $STATES = x \rightarrow \mathbb{Z}$ 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, $s(x) = x$ for expressing the value of x in state s :

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

- A program P *produces* results or outputs from inputs according to a (operational or denotational) semantics

- STATES is the set of states of P : $STATES = x \rightarrow \mathbb{Z}$ 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 , $s(x) = x$ for expressing the value of 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 relation over the set of states :

$$x_0 \xrightarrow{P} x_f$$

- A program P *produces* results or outputs from inputs according to a (operational or denotational) semantics

- STATES is the set of states of P : $STATES = x \rightarrow \mathbb{Z}$ 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, $s(x) = x$ for expressing the value of 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 :

$$x_0 \xrightarrow{P} x_f$$

- A program P *satisfies* the contract $(x, \text{pre}, \text{post})$:

- P transforms a variable x from a value x_0 and produces a value x_f :
- $$x_0 \xrightarrow{P} x_f$$
- x_0 satisfies pre : $\text{pre}(x_0)$
 - x_f satisfies post : $\text{post}(x_0, x_f)$
 - $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$

A program P satisfies a contract $(x, \text{pre}, \text{post})$:

- ▶ P transforms a variable x from an initial value x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$
- ▶ x_0 satisfies pre : $\text{pre}(x_0)$ and x_f satisfies post : $\text{post}(x_0, x_f)$
- ▶ $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$

A program P satisfies a contract $(x, \text{pre}, \text{post})$:

- ▶ P transforms a variable x from an initial value x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$
- ▶ x_0 satisfies pre : $\text{pre}(x_0)$ and x_f satisfies post : $\text{post}(x_0, x_f)$
- ▶ $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$

```
variables x : ℂ
requires pre(x₀)
ensures post(x₀, xₙ)
begin
  0 : P₀(x₀, x)
  s
  f : Pₙ(x₀, x)
end
```

A program P satisfies a contract $(x, \text{pre}, \text{post})$:

- ▶ P transforms a variable x from an initial value x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$
 - ▶ x_0 satisfies pre : $\text{pre}(x_0)$ and x_f satisfies post : $\text{post}(x_0, x_f)$
 - ▶ $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$

```

variables 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

```

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

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

Checking a contract using the solver Z3

```

requires  $x0 \geq 0$ ;
ensures  $x_f = x0+2$ ;
variables  $X$ 

begin
   $intX = x0$ ;
   $0 : x = x0$ 
   $X = X+2$ ;
   $1 : x = x0+2$ 
end

```

- ▶ $x0 \geq 0 \wedge x = x0 \Rightarrow x = x0$
 - ▶ $x = x0 + 2 \Rightarrow x = x0 + 2$
 - ▶ conditions de vérification $0 \rightarrow 1 :$
 $x = x0 \wedge x' = x + 2 \Rightarrow x' = x0 + 2$
 - ▶ $(x0 \geq 0, x == x0, x! = x0)$
 - ▶ $(x == x0 + 2, x! = x0 + 2)$
 - ▶ $(x == x0, xp == x + 2, xp! = x0 + 2)$

Listing 1 – z3 en Python

```

from numbers import Real
from z3 import *
x = Real('x')
xp = Real('xp')
x0 = Real('x0')
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())

```

- 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

Transformation of predicates

Transformation of predicates

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

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

Transformation of predicates

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

Transformation of predicates

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

Transformation of predicates

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

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

Weakest Liberal Precondition of S for P

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

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

Weakest Liberal Precondition of S for P

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

- ▶ $\{x := e\}P(x) = P[x \mapsto e]$
- ▶ $\{\text{if } b(x) \text{ then } S1 \text{ else } S2\}P(x) = b(x) \wedge \{S1\}P(x) \vee \neg b(x) \{S2\}P(x)$

Weakest Liberal Precondition of S for P

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

Weakest Liberal Precondition of S for P

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

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

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

Examples

- ▶ $\{\text{while } x > 0 \text{ do } x := x - 1 \text{ end}\}(x = 0) = x \geq 0$
- ▶ $\{\text{while } x > 0 \text{ do } x := x + 1 \text{ end}\}(x = 0) = x \geq 0$
- ▶ $\{\text{while } x > 0 \text{ do } x := x + 1 \text{ end}\}(x \leq 0) = x \in \mathbb{Z}$

Computing WLP function

- ▶ $\forall x_0. \text{pre}(x_0) \Rightarrow \{P\} \text{post}(x_0, x)$
- ▶ $\forall x_0. x = x_0 \wedge \text{pre}(x_0) \Rightarrow \{P\} \text{post}(x_0, x)$
- ▶ Hoare Triple : $\{\text{pre}(x_0) \wedge x = x_0\} P \{\text{post}(x_0, x)\}$

- 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

- ✉ Definition(Axiomes et règles d'inférence)
 - ▶ Axiome d'affectation : $\{P(e/x)\} \mathbf{X} := E(\mathbf{X}) \{P\}$.
 - ▶ Axiome du saut : $\{P\} \mathbf{skip} \{P\}$.
 - ▶ 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 $\{P \wedge B\} \mathbf{S}_1 \{Q\}$ et $\{P \wedge \neg B\} \mathbf{S}_2 \{Q\}$, alors $\{P\} \mathbf{if } B \mathbf{ then } \mathbf{S}_1 \mathbf{ then } \mathbf{S}_2 \mathbf{ fi} \{Q\}$.
 - ▶ Si $\{P \wedge B\} \mathbf{S} \{P\}$, alors $\{P\} \mathbf{while } B \mathbf{ do } \mathbf{S} \mathbf{ od} \{P \wedge \neg B\}$.
 - ▶ Règle de renforcement/affaiblissement : Si $P' \Rightarrow P$, $\{P\} \mathbf{S} \{Q\}$, $Q \Rightarrow Q'$, alors $\{P'\} \mathbf{S} \{Q'\}$.

.....
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)

Definition

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

☺ PropertyCorrection 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 prédictats 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 $(\text{PRED}, \Rightarrow, \text{false}, \text{true}, \wedge, \vee)$: $(\text{PRED}, \Rightarrow, \text{false}, \text{true}, \wedge, \vee)$ est un treillis complet.

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

Définition de wlp

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

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

Lien entre wp et wlp

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

☒ Definition

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

☺ Property

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

.....
☒ Definition triplets de Hoare

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

.....

☒ Definition triplets de Hoare

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

☒ Definition(Axiomes et règles d'inférence)

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

- ▶ $\{P\}\mathbf{S}\{Q\}$
- ▶ $\forall s \in STATES. P(s) \Rightarrow wlp(S)(Q)(s)$
- ▶ $\forall s \in STATES. P(s) \Rightarrow (\forall t \in STATES : \mathcal{D}(S)(s) = t \Rightarrow Q(t))$
- ▶ $\forall s, t \in STATES. P(s) \wedge \mathcal{D}(S)(s) = t \Rightarrow Q(t))$
- ▶ Correction : Si on a construit une preuve de $\{P\}\mathbf{S}\{Q\}$ 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 langage d'assertions.

- 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

Example 1

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;
}
```

- ▶ INT_MIN (resp. INT_MAX) is the smallest codable integer (resp. greatest codable integer).
- ▶ $a - b \geq INT_MIN \wedge a - b \leq INT_MAX \wedge a = a_0 \wedge b = b_0 \Rightarrow [\result = a - b](\result = (a - b))$

Example 2

Listing 3 – incrément de nombre

```
/*@ 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;
}
```

requires $x_0 \geq 0$;
ensures $x_f = x_0 + 2$;
variables x

```
begin
  intx = x0;
  0 : x = x0
  x := x+2;
  1 : x = x0+2
end
```

Conditions de vérification $0 \rightarrow 1$:

- ▶ $x = x_0 \wedge x' = x+2 \Rightarrow x' = x_0+2$
- ▶ $x = x_0 \Rightarrow (x' = x+2 \Rightarrow x' = x_0+2)$
- ▶ $x = x_0 \Rightarrow (x+2 = x_0+2)$
- ▶ $wp(x := x+2)(x = x_0+2) = (x+2 = x_0+2)$
- ▶ $x = x_0 \wedge x_0 \geq 0 \Rightarrow wp(x := x+2)(x = x_0+2)$
- ▶ $x = x_0 \wedge x_0 \geq 0 \Rightarrow x+2 = x_0+2$
- ▶ $x = x_0 \wedge x_0 \geq 0 \Rightarrow x_0+2 = x_0+2$

- ▶ $x_0 \geq 0 \wedge x = x_0 \Rightarrow x = x_0$
- ▶ $x = x_0+2 \Rightarrow x = x_0+2$
- ▶ $x = x_0 \Rightarrow wp(x := x+2)(x = x_0+2)$



calcul de $wp(X := X+2)(x = x_0+2)$

Listing 4 – incrément de nombre

```
/*@ 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

```
/*@ 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;
}
```

- 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

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 :

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

- ▶ $WP(X := X+1)(A) = \{s \in STATES | s[X \mapsto s(X) \oplus 1] \in A\}$
 - ▶ $WP(X := Y+1)(A) = \{s \in STATES | s[X \mapsto s(Y) \oplus 1] \in A\}$
 - ▶ $WP(\text{while } X > 0 \text{ do } X := X-1 \text{ od})(A) = \{s \in STATES | (s(X) \leq 0) \vee (s(X) \in A \wedge s(X) < 0)\}$
 - ▶ $WP(\text{while } x > 0 \text{ do } x := x+1 \text{ od})(A) = \{s \in STATES | (s(X) \in A \wedge s(X) \leq 0)\}$
 - ▶ $WP(\text{while } x > 0 \text{ do } x := x+1 \text{ od})(\emptyset) = \emptyset$
 - ▶ $WP(\text{while } x > 0 \text{ do } x := x+1 \text{ od})(STATES) = \{s \in STATES | s(X) \leq 0\}$

Propriétés

- ▶ WP est une fonction monotone pour l'inclusion d'ensembles de STATES.
 - ▶ $WP(S)(\emptyset) = \emptyset$
 - ▶ $WP(S)(A \cap B) = WP(S)(A) \cap WP(S)(B)$
 - ▶ $WP(S)(A) \cup WP(S)(B) \subseteq WP(S)(A \cup B)$
 - ▶ 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,
 $WP(S) \in \mathcal{P}(STATES) \rightarrow \mathcal{P}(STATES)$
 - ▶ $(\mathcal{P}(STATES), \subseteq)$ est un treillis complet.
 - ▶ $(Pred, \Rightarrow)$ 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
 - \Rightarrow est l'implication
 - $s \in A$ correspond une assertion P vraie en s notée $P(s)$.

- ▶ 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 prédict 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 .
- ▶ x est la valeur de X (X contient la valeur x).
- ▶ $e(x, d)$ (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
- ▶ $b(x, d)$ 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 prédictats

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	$(B \Rightarrow wp(S_1)(P)) \wedge (\neg B \Rightarrow wp(S_2)(P))$
WHILE B DO S OD	$\mu.(\lambda X.(B \Rightarrow wp(S)(X)) \wedge (\neg B \Rightarrow P))$

- ▶ $wp(X := X+5)(x \geq 8) \stackrel{def}{=} x+5 \geq 8 \wedge x \geq 3$
- ▶ $wp(\text{WHILE } x > 1 \text{ DO } X := X+1 \text{ OD})(x = 4) = \text{FALSE}$
- ▶ $wp(\text{WHILE } x > 1 \text{ DO } X := X+1 \text{ OD})(x = 0) = x = 0$

- Definition triplets de Hoare Correction Totale

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

☒ Definition triplets de Hoare Correction Totale

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

☒ Définition(Axiomes et règles d'inférence)

- ▶ Axiome d'affectation : $[P(e/x)]\mathbf{X} := \mathbf{E(X)}[P]$.
 - ▶ Axiome du saut : $[P]\mathbf{skip}[P]$.
 - ▶ 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 $[P \wedge B]\mathbf{S}_1[Q]$ et $[P \wedge \neg B]\mathbf{S}_2[Q]$, alors
 $[P]\mathbf{if}\; \mathbf{B}\;\mathbf{then}\; \mathbf{S}_1\;\mathbf{then}\; \mathbf{S}_2\;\mathbf{fi}[Q]$.
 - ▶ Si $[P(n+1)]\mathbf{S}[P(n)]$, $P(n+1) \Rightarrow b$, $P(0) \Rightarrow \neg b$, alors
 $\exists n \in \mathbb{N}. P(n)\mathbf{while}\; \mathbf{B}\;\mathbf{do}\; \mathbf{S}\;\mathbf{od}[P(0)]$.
 - ▶ 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 $P\wp(S)5Q$.

- ▶ $[P(e/x)]\mathbf{X} := \mathbf{E}(\mathbf{X})[P]$ est valide : $wp(X := E)(P)/x = P(e/x)$.
- ▶ $[\exists n \in \mathbb{N}. P(n)]\mathbf{while}\; \mathbf{B}\;\mathbf{do}\; \mathbf{S}\;\mathbf{od}[P(0)]$: 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 $[Q(e/x)]\mathbf{X} := \mathbf{E}(\mathbf{X})[Q]$ constituent une preuve.
- ▶ $P \Rightarrow wp(while)(Q) :$
 - On construit la suite de $P(n)$ en définissant $P(n) = W_n$.
 - On vérifie que cela vérifie la règle du while.

- 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

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 : $x_0 \xrightarrow{P} x_f$
 - ▶ x_0 satisfait pre : $\text{pre}(x_0)$ and x_f satisfait post : $\text{post}(x_0, x_f)$
 - ▶ $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$

requires $pre(x_0)$
 ensures $post(x_0, x_f)$
 variables X

```

begin
  0 :  $P_0(x_0, x)$ 
  instruction0
  ...
  i :  $P_i(x_0, x)$ 
  ...
  instructionf-1
  f :  $P_f(x_0, x)$ 
end

```

- ▶ $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
 - ▶ $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$
 - ▶ For each pair ℓ, ℓ'
 such that $\ell \rightarrow \ell'$, one checks that
 for any value $x, x' \in \text{MEMORY}$

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

Verification du contrat (II)

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 : $x_0 \xrightarrow{P} x_f$
 - ▶ x_0 satisfait pre : $\text{pre}(x_0)$ and x_f satisfait post : $\text{post}(x_0, x_f)$
 - ▶ $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$

requires $pre(x_0)$
 ensures $post(x_0, x_f)$
 variables X

```

begin
  0 :  $P_0(x_0, x)$ 
  instruction0
  ...
  i :  $P_i(x_0, x)$ 
  ...
instructionf-1
f :  $P_f(x_0, x)$ 
end

```

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

Un programme P *satisfies* a contract $(\text{pre}, \text{post})$:

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

Un programme P *satisfies* a contract $(\text{pre}, \text{post})$:

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

requires $pre(x_0)$
 ensures $post(x_0, x_f)$
 variables X

```

begin
  /·@assert  $P_0(x_0, x)$ ·/
  T;
  /·@loop invariant  $I(x_0, x)$ ·/
  while  $B(x)$  do
    S
  od
  /·@assert  $P_f(x_0, x)$ ·/
end

```

- ▶ $x = x_0 \wedge \text{pre}(x_0) \Rightarrow P_0(x_0, x)$
 - ▶ $\text{pre}(x_0) \wedge P_0(x_0, x) \Rightarrow WLP(T)(I(x_0, x))$
 - ▶ $I(x_0, x) \wedge B(x) \Rightarrow WLP(S)(I(x_0, x))$
 - ▶ $I(x_0, x) \wedge \neg B(x) \Rightarrow P_f(x_0, x)$

```
requires  $\text{pre}(x_0)$ 
ensures  $\text{post}(x_0, x_f)$ 
variables  $X$ 
begin
  /·@assert  $P_0(x_0, x)$ ·/
  S1;
  S2;
  /·@assert  $P_f(x_0, x)$ ·/
end
```

- ▶ $x = x_0 \wedge \text{pre}(x_0) \Rightarrow P_0(x_0, x)$
- ▶ $P_0(x_0, x) \Rightarrow WLP(S1; S2)(P_f(x_0, x))$

Verification of contract (V)

requires $\text{pre}(x_0)$
ensures $\text{post}(x_0, x_f)$
variables X

```
begin
  /·@assert  $P_0(x_0, x)$ ·/
  if  $B(x)$  do
    S1
  else
    S2
  elfi
  /·@assert  $P_f(x_0, x)$ ·/
end
```

$$\begin{aligned} x = x_0 \wedge \text{pre}(x_0) &\Rightarrow P_0(x_0, x) \\ P_0(x_0, x) \Rightarrow \\ &B(x) \wedge WLP(S1)(P_f(x_0, x)) \\ \vee \\ \neg B(x) \wedge WLP(S2)(P_f(x_0, x)) \end{aligned}$$