



- ① Ingénierie des logiciels et des systèmes
- ② Introduction à la sémantique des langages de programmation
- ③ Sémantique opérationnelle
- ④ Sémantique dénotationnelle
- ⑤ Equivalence des deux sémantiques
- ⑥ Transformateurs de prédicats
  - Introduction
  - Définition et propriétés
- ⑦ Logique de Hoare
- ⑧ Utilisation des transformateurs de prédicats
- ⑨ Conclusion

- 1 Ingénierie des logiciels et des systèmes
- 2 Introduction à la sémantique des langages de programmation
- 3 Sémantique opérationnelle
- 4 Sémantique dénotationnelle
- 5 Equivalence des deux sémantiques
- 6 Transformateurs de prédicats
  - Introduction
  - Définition et propriétés
- 7 Logique de Hoare
- 8 Utilisation des transformateurs de prédicats
- 9 Conclusion



Un programme  $P$  *remplit* un contrat ( $pre, post$ ) :

- ▶  $P$  transforme une variable  $x$  à partir d'une valeur initiale  $x_0$  et produisant une valeur finale  $x_f$  :  $x_0 \xrightarrow{P} x_f$
- ▶  $x_0$  satisfait  $pre$  :  $pre(x_0)$  and  $x_f$  satisfait  $post$  :  $post(x_0, x_f)$
- ▶  $pre(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow post(x_0, x_f)$

requires  $pre(x_0)$

ensures  $post(x_0, x_f)$

variables  $X$

begin

$0 : P_0(x_0, x)$

instruction<sub>0</sub>

...

$i : P_i(x_0, x)$

...

instruction <sub>$f-1$</sub>

$f : P_f(x_0, x)$

end

▶  $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$

▶  $P_f(x_0, x) \Rightarrow post(x_0, x)$

▶ conditions de vérification pour toutes les paires  $\ell \longrightarrow \ell'$

Un programme  $P$  *remplit* un contrat (pre,post) :

- ▶  $P$  transforme une variable  $x$  à partir d'une valeur initiale  $x_0$  et produisant une valeur finale  $x_f$  :  $x_0 \xrightarrow{P} x_f$
- ▶  $x_0$  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  $\text{pre}(x_0)$

ensures  $\text{post}(x_0, x_f)$

variables  $X$

begin

$0 : P_0(x_0, x)$

instruction<sub>0</sub>

...

$i : P_i(x_0, x)$

...

instruction <sub>$f-1$</sub>

$f : P_f(x_0, x)$

end

▶  $\text{pre}(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$

▶  $\text{pre}(x_0) \wedge P_f(x_0, x) \Rightarrow \text{post}(x_0, x)$

▶ Pour toute paire d'étiquettes  $\ell, \ell'$  telle que  $\ell \longrightarrow \ell'$ , on vérifie que, pour toutes valeurs

$x, x' \in \text{MEMORY}$

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

Un programme  $P$  *remplit* un contrat (pre,post) :

- ▶  $P$  transforme une variable  $x$  à partir d'une valeur initiale  $x_0$  et produisant une valeur finale  $x_f$  :  $x_0 \xrightarrow{P} x_f$
- ▶  $x_0$  satisfait pre :  $\text{pre}(x_0)$  and  $x_f$  satisfait post :  $\text{post}(x_0, x_f)$
- ▶  $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$  et  $\mathbb{D}$  est le domaine RTE de  $X$

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

- ▶  $\text{pre}(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- ▶  $\text{pre}(x_0) \wedge P_f(x_0, x) \Rightarrow \text{post}(x_0, x)$
- ▶ Pour toute paire d'étiquettes  $\ell, \ell'$  telle que  $\ell \longrightarrow \ell'$ , on vérifie que, pour toutes valeurs  $x, x' \in \text{MEMORY}$ 

$$\left( \begin{array}{l} \left( \text{pre}(x_0) \wedge P_\ell(x_0, x) \right) \\ \wedge \text{cond}_{\ell, \ell'}(x) \wedge x' = f_{\ell, \ell'}(x) \end{array} \right) \Rightarrow P_{\ell'}(x_0, x')$$
- ▶ Pour toute paire d'étiquettes  $m, n$  telle que  $m \longrightarrow n$ , on vérifie que,  $\forall x, x' \in \text{MEMORY} : \text{pre}(x_0) \wedge P_m(x_0, x) \Rightarrow \text{DOM}(m, n)(x)$

- ▶  $\forall x_0, x \in \text{VALS}. \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x) \Rightarrow A(x)$
  - ▶  $\forall x \in \text{VALS}. (\exists x_0. x_0 \in \text{VALS} \wedge \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, x)) \Rightarrow A(x).$
  - ▶  $\text{REACHABLE}(M) = \{u | u \in \text{VALS} \wedge (\exists x_0. x_0 \in \text{VALS} \wedge \text{Init}(x_0) \wedge \text{NEXT}^*(x_0, u))\}$  est l'ensemble des états accessibles à partir des états initiaux.
  - ▶ Model Checking : on doit montrer l'inclusion  $\text{REACHABLE}(M) \subseteq \{u | u \in \text{VALS} \wedge A(u)\}.$
  - ▶ Preuves : définir un invariant  $I(\ell, v) \equiv \bigvee_{\ell \in \text{LOCATIONS}} \left( \bigvee_{v \in \text{MEMORY}} P_\ell(v) \right)$  avec la famille d'annotations  $\{P_\ell(v) : \ell \in \text{LOCATIONS}\}$  et démontrer les conditions de vérification.
  - ▶ Analyse automatique :
    - Mécaniser la vérification des conditions de vérification
    - Calculer  $\text{REACHABLE}(M)$
    - Calculer une valeur approchée de  $\text{REACHABLE}(M)$
- $$(\mathcal{P}(\text{VALS}), \subseteq) \xleftrightarrow[\alpha]{\gamma} (D, \sqsubseteq)$$
- $$\alpha(\text{REACHABLE}(M)) \sqsubseteq A \text{ ssi } \text{REACHABLE}(M) \subseteq \gamma(A)$$
- Si  $\gamma(A) \subseteq \{u | u \in \text{VALS} \wedge A(u)\}$ , alors
- $$\text{REACHABLE}(M) \subseteq \{u | u \in \text{VALS} \wedge A(u)\}$$



- ▶ Mécaniser la vérification des conditions de vérification
- ▶ Calculer  $\text{REACHABLE}(M)$  comme un point-fixe.
- ▶ Calculer une valeur approchée de  $\text{REACHABLE}(M)$

$$\begin{array}{c} (\mathcal{P}(\text{VALS}), \sqsubseteq) \xrightleftharpoons[\alpha]{\gamma} (D, \sqsubseteq) \\ \alpha(\text{REACHABLE}(M)) \sqsubseteq A \text{ ssi } \text{REACHABLE}(M) \sqsubseteq \gamma(A) \end{array}$$

Si  $A$  vérifie  $\gamma(A) \subseteq \{u \mid u \in \text{VALS} \wedge A(u)\}$ , alors  
 $\text{REACHABLE}(M) \subseteq \{u \mid u \in \text{VALS} \wedge A(u)\}$

Software/System development *ideally* proceeds in three phases according to Dines Børner : :

$$\mathcal{D}, \mathcal{S} \Rightarrow \mathcal{R}$$

- ▶ First, a phase of **domain engineering**  $\mathcal{D}$  : an analysis of the application domain leads to a description of that domain.
- ▶ Second, a phase of **requirements engineering**  $\mathcal{R}$  : an analysis of the domain description leads to a prescription of requirements to software for that domain. jbloc
- ▶ Third, a phase of **software/system design**  $\mathcal{S}$  : an analysis of the requirements prescription leads to software for that domain.

Software/System development *ideally* proceeds in three phases according to Dines Børner : :

$$\mathcal{D}, \mathcal{S} \Rightarrow \mathcal{R}$$

## Pre/Post Specification

- ▶  $\mathcal{R}$  : pre/post.
- ▶  $\mathcal{D}$  : integers, reals, ...
- ▶  $\mathcal{S}$  : algorithm, program, ...

Software/System development *ideally* proceeds in three phases according to Dines Børner :

$$\mathcal{D}, \mathcal{S} \Rightarrow \mathcal{R}$$

## Pre/Post Specification

- ▶  $\mathcal{R}$  : pre/post.
  - ▶  $\mathcal{D}$  : integers, reals, ...
  - ▶  $\mathcal{S}$  : algorithm, program, ...
- 
- ▶ Semantical relationship
  - ▶ Verification by induction principle

Software/System development *ideally* proceeds in three phases according to Dines Børner : :

$$\mathcal{D}, \mathcal{S} \Rightarrow \mathcal{R}$$

# System Modelling

- ▶  $\mathcal{R}$  : safety properties in Event-B
- ▶  $\mathcal{D}$  : theories, context in Event-B
- ▶  $\mathcal{S}$  : machines for reactive systems

Software/System development *ideally* proceeds in three phases according to Dines Børner :

$$\mathcal{D}, \mathcal{S} \Rightarrow \mathcal{R}$$

### System Modelling

- ▶  $\mathcal{R}$  : safety properties in Event-B
  - ▶  $\mathcal{D}$  : theories, context in Event-B
  - ▶  $\mathcal{S}$  : machines for reactive systems
- 
- ▶ Checking proof obligations
  - ▶ Refinement of models

Software/System development *ideally* proceeds in three phases according to Dines Børner : :

$$\mathcal{D}, \mathcal{S} \Rightarrow \mathcal{R}$$

- ▶ First, a phase of **domain engineering**  $\mathcal{D}$  : an analysis of the application domain leads to a description of that domain.
- ▶ Second, a phase of **requirements engineering**  $\mathcal{R}$  : an analysis of the domain description leads to a prescription of requirements to software for that domain.
- ▶ Third, a phase of **software/system design**  $\mathcal{S}$  : an analysis of the requirements prescription leads to software for that domain.

J. Piaget. *Logique et Connaissance scientifique*. La Pléiade, encyclopaedia.

If we refer to whom is talking, or more generally to the language users, this investigation is attributed to the **pragmatics**.

If we make an abstraction of the language users and if we analyze the expressions and their meanings only, we are in the area of the **semantics**.

Finally, if we make an abstraction of the meanings to analyze the relations between expressions, we are dealing with **syntax**.

These three elements constitute the science of the language or semiotics.



## Listing 1 – annotation

```
#include <limits.h>
/*@ requires x < INT.MAX && x >= INT.MIN ;
    requires y < INT.MAX;
        requires y >= INT.MIN ;
    requires z < INT.MAX;
        requires z >= INT.MIN ;
    requires z+x*x+2*x*(y+3) < INT.MAX;
        requires z+x*x+2*x*(y+3) >= INT.MIN ;
    requires z+x*x < INT.MAX;
        requires z+x*x >= INT.MIN ;
    requires x == y + 3 && z == (y+3)*(y+3);
@*/

int funny(int x, int y, int z) {
    //@ assert x == y + 3 && z == (y+3)*(y+3);
    z=z+x*x+2*x*(y+3);
    //@ assert z == (y+3+x)*(y+3+x);

    return 0;
}
```

## Listing 2 – calcul d'une moyenne

```
/*@  
  requires a < 0 && a > 0;  
  ensures \false;  
*/  
void afunnyfun(int a){ }
```

## Listing 3 – calcul d'une moyenne

```
#include <stdio.h>
#include <limits.h>

/*@
  requires  INT_MIN <= a && a <= INT_MAX;
  requires  INT_MIN <= b && b <= INT_MAX;
  requires  INT_MIN <= a+b && a+b <= INT_MAX;
  ensures   INT_MIN <= \result && \result <= INT_MAX;
*/

int average(int a, int b)
{
    return ((a+b)/2);
}

int main()
{
    int x, y;
    x=INT_MAX; y=0;
    //@ assert INT_MIN <= x && x <= INT_MAX;
    //@ assert INT_MIN <= y && y <= INT_MAX;
    //@ assert INT_MIN <= x+y && x+y <= INT_MAX;
    printf(" Average -- for %d- and %d- is %d\n", x, y,
           average(x, y));
    return 0;
}
```



### Implicite versus explicite

- ▶ Ecrire  $101 = 5$  peut avoir une signification

### Implicite versus explicite

- ▶ Ecrire  $101 = 5$  peut avoir une signification
- ▶ Le code du nombre  $n$  est  $101$  à gauche du symbole  $=$  et le code du nombre  $n$  est sa représentation en base 10 à droite.
- ▶  $n_{10} = 5$  et  $n_2 = 101$
- ▶ Vérification :  $base(2, 10, 101) = 1.2^2 + 0.2 + 1.2^0 = 5_{10}$

- ▶ A train moving at absolute speed  $spd1$
- ▶ A person walking in this train with relative speed  $spd2$ 
  - One may compute the absolute speed of the person
- ▶ Modelling
  - Syntax. Classical expressions
    - ▶ Type  $Speed = Float$
    - ▶  $spd1, spd2 : Speed$
    - ▶  $AbsoluteSpeed = spd1 + spd2$
  - Semantics
    - ▶ If  $spd1 = 25.6$  and  $spd2 = 24.4$  then  $AbsoluteSpeed = 50.0$
    - ▶ If  $spd1 = "val"$  and  $spd2 = 24.4$  then exception raised
  - Pragmatics
    - ▶ What if  $spd1$  is given in *mph* (miles per hour) and  $spd2$  in *km/s* (kilometers per second) ?
    - ▶ What if  $spd1$  is a relative speed ?

- ▶ La sémantique décrit le sens des objets définis par la syntaxe.
- ▶ La sémantique permet d'éviter l'ambiguïté des éléments d'un langage.
- ▶ Exemples
  - L'objet  $123$  désigne le nombre 123 en base dix.
  - L'objet  $x+12+8$  désigne la somme des valeurs de la variable  $x$  et des deux nombres écrits en base dix  $12$  et  $8$ .
- ▶ Styles de sémantique
  - Sémantique Opérationnelle : la sémantique du programme est décrite par une relation de transition qui décrit les différents états du programme et la relation de transition est définie par des *opérations* ou des *actions*.
  - Sémantique Dénotationnelle : la sémantique du programme est une fonction *calculant* le résultat à partir de la donnée.
  - Sémantique Axiomatique : le programme est caractérisé par des axiomes et des règles d'inférences comme par exemple la logique de HOARE.



- 1 Ingénierie des logiciels et des systèmes
- 2 Introduction à la sémantique des langages de programmation
- 3 Sémantique opérationnelle**
- 4 Sémantique dénotationnelle
- 5 Equivalence des deux sémantiques
- 6 Transformateurs de prédicats
  - Introduction
  - Définition et propriétés
- 7 Logique de Hoare
- 8 Utilisation des transformateurs de prédicats
- 9 Conclusion

## ► Sémantique à petits pas (small steps) :

- Définition d'une relation notée  $\xrightarrow{\text{sos}}$  sur l'ensemble des configurations de la forme  $(S, s)$  où  $S$  est une instruction ou un programme ou une instruction et  $s$  est un état ou de la forme  $s$  où  $s$  est un état.
- Transitions de type 1 :  $(S, s) \xrightarrow{\text{sos}} (S', s')$
- Transitions de type 2 :  $(S, s) \xrightarrow{\text{sos}} s'$

## ► Sémantique naturelle ou à grands pas (*big step*) :

- Définition d'une relation notée  $\xrightarrow{\text{nat}}$  sur l'ensemble des configurations de la forme  $(S, s)$  où  $S$  est une instruction ou un programme ou une instruction et  $s$  est un état ou de la forme  $s$  où  $s$  est un état.
- Transitions uniquement de ce type :  $(S, s) \xrightarrow{\text{nat}} s'$

- ▶ Un état  $s$  est un élément de  $STATES = V \rightarrow \mathbb{Z}$  et  $STATES$  est l'ensemble des états.
- ▶  $\mathcal{E}$  est une fonction associant à toute expression arithmétique une fonction permettant de donner la valeur de cette expression en un état donné :  $\mathcal{E} \in EXPR \rightarrow (STATES \rightarrow \mathbb{Z})$  :
  - $\mathcal{E}(x)(s) = s(x)$  où  $x \in V$  et  $s \in STATES$ .
  - $\mathcal{E}(constant)(s) = constant$ .
  - $\mathcal{E}(e1 \text{ op } e2)(s) = \mathcal{E}(e1)(s) \text{ op } \mathcal{E}(e2)(s)$ .
- ▶  $\mathcal{B}$  est une fonction associant à toute expression booléenne une fonction permettant de donner la valeur de cette expression en un état donné :  $\mathcal{B} \in BEXPR \rightarrow (STATES \rightarrow BOOL)$  :
  - $\mathcal{B}(ff)(s) = FALSE$
  - $\mathcal{B}(tt)(s) = TRUE$
  - $\mathcal{B}(e1 \text{ relop } e2)(s) = \mathcal{E}(e1)(s) \text{ relop } \mathcal{E}(e2)(s)$ .
  - $\mathcal{B}(b1 \text{ bop } b2)(s) = \mathcal{B}(b1)(s) \text{ bop } \mathcal{B}(b2)(s)$ .

## Règles de définition selon la syntaxe

- ▶ Si  $\mathcal{E}(e)(s)$  est la valeur de l'expression  $e$  en  $s$ , alors
$$(x := e, s) \xrightarrow{\text{SOS}} s[x \mapsto \mathcal{E}(e)(s)]$$
- ▶  $(\text{skip}, s) \xrightarrow{\text{SOS}} s$
- ▶ Si  $(S_1, s) \xrightarrow{\text{SOS}} (S'_1, s')$ , alors  $(S_1; S_2, s) \xrightarrow{\text{SOS}} (S'_1; S_2, s')$ .
- ▶ Si  $(S_1, s) \xrightarrow{\text{SOS}} s'$ , alors  $(S_1; S_2, s) \xrightarrow{\text{SOS}} (S_2, s')$ .
- ▶ Si  $\mathcal{B}(b)(s) = \text{TRUE}$ , alors  $(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, s) \xrightarrow{\text{SOS}} (S_1, s)$ .
- ▶ Si  $\mathcal{B}(b)(s) = \text{FALSE}$ , alors  $(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, s) \xrightarrow{\text{SOS}} (S_2, s)$ .
- ▶  $(\text{while } b \text{ do } S \text{ od}, s) \xrightarrow{\text{SOS}}$   
 $(\text{if } b \text{ then } S; \text{while } b \text{ do } S \text{ od else skip fi}, s)$

## Règles de définition selon la syntaxe

- ▶ Si  $\mathcal{E}(e)(s)$  est la valeur de l'expression  $e$  en  $s$ , alors
$$(x := e, s) \xrightarrow[\text{nat}]{} s[x \mapsto \mathcal{E}(e)(s)]$$
- ▶  $(\text{skip}, s) \xrightarrow[\text{nat}]{} s$
- ▶ Si  $(S_1, s) \xrightarrow[\text{nat}]{} s'$  et  $(S_2, s') \xrightarrow[\text{nat}]{} s''$ , alors  $(S_1; S_2, s) \xrightarrow[\text{nat}]{} s''$ .
- ▶ Si  $(S_1, s) \xrightarrow[\text{nat}]{} s'$  et  $\mathcal{B}(b)(s) = \text{TRUE}$ , alors
$$(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, s) \xrightarrow[\text{nat}]{} s'.$$
- ▶ Si  $(S_2, s) \xrightarrow[\text{nat}]{} s'$  et  $\mathcal{B}(b)(s) = \text{FALSE}$ , alors
$$(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, s) \xrightarrow[\text{nat}]{} s'.$$
- ▶ Si  $(S, s) \xrightarrow[\text{nat}]{} s'$  et  $(\text{while } b \text{ do } S \text{ od}, s') \xrightarrow[\text{nat}]{} s''$  et  $\mathcal{B}(b)(s) = \text{TRUE}$ , alors  $(\text{while } b \text{ do } S \text{ od}, s) \xrightarrow[\text{nat}]{} s''$ .
- ▶ Si  $\mathcal{B}(b)(s) = \text{FALSE}$ , alors  $(\text{while } b \text{ do } S \text{ od}, s) \xrightarrow[\text{nat}]{} s$ .

### Fonction sémantique $\mathcal{S}_{sos}$ :

- ▶  $\mathcal{S}_{sos} \in STATS \rightarrow (STATES \rightarrow STATES)$  :
- ▶  $\mathcal{S}_{sos}(S)(s) \stackrel{def}{=} \begin{cases} s' \text{ si } (S, s) \xrightarrow[\text{sos}]{*} s' \\ \text{indefinie sinon} \end{cases}$

### Fonction sémantique $\mathcal{S}_{nat}$ :

- ▶  $\mathcal{S}_{nat} \in STATS \rightarrow (STATES \rightarrow STATES)$  :
- ▶  $\mathcal{S}_{nat}(S)(s) \stackrel{def}{=} \begin{cases} s' \text{ si } (S, s) \xrightarrow[\text{nat}]{} s' \\ \text{indefinie sinon} \end{cases}$

## Equivalence pour les instructions de STATS

Pour toute instruction  $S$  de STATS, pour tout état  $s$  de STATES,  
 $\mathcal{S}_{sos}(S)(s) = \mathcal{S}_{nat}(S)(s)$

## Preuve

- ▶ Montrons que si  $(S, s) \xrightarrow{\text{nat}} s'$ , alors  $(S, s) \xrightarrow[\text{sos}]{\star} s'$ .
- ▶ Montrons que si  $(S, s) \xrightarrow[\text{sos}]{\star} s'$ , alors  $(S, s) \xrightarrow{\text{nat}} s'$ .

- 1 Ingénierie des logiciels et des systèmes
- 2 Introduction à la sémantique des langages de programmation
- 3 Sémantique opérationnelle
- 4 Sémantique dénotationnelle**
- 5 Equivalence des deux sémantiques
- 6 Transformateurs de prédicats
  - Introduction
  - Définition et propriétés
- 7 Logique de Hoare
- 8 Utilisation des transformateurs de prédicats
- 9 Conclusion



- ▶ Fondement des langages de programmation.
- ▶ Outils mathématiques permettant de raisonner sur les objets de la programmation.
- ▶ Liaison entre les programmes et les spécifications : le raffinement
- ▶ Préservation de la sémantique d'un langage de programmation dans un langage de plus bas niveau par compilation : correction du compilateur.



- ▶ Un module d'analyse syntaxique : lecture du texte fourni, vérification de la syntaxe, génération de la représentation interne.
- ▶ Un module d'évaluation : évaluation du texte fourni en donnée du texte analysé en un texte résultat ; cela définit la sémantique du langage.

La mise en œuvre d'un langage est une activité pragmatique.

- **Interprétation** : Exécution du programme  
L'interprète définit le sens par ses actions.
- **Compilation** : Transformation d'un programme écrit dans un langage L en un texte équivalent d'un langage L2 ( langage machine en général).  
Le compilateur préserve le sens par équivalence.

- ▶ Syntaxe sous BNF (Backus Naur Form)
  - Correspondance entre la BNF et l'analyseur syntaxique.
  - Générateur d'analysuer à partir de spécification du langage.
- ▶ Sémantique :
  - Opérationnelle.
  - Axiomatique
  - Dénotationnelle

- ▶ Le sens d'un programme est un objet mathématique.
- ▶ Chaque construction du langage est associée à un objet mathématique par une fonction de valuation. Le sens d'une structure est appelée une dénotation.

$$\mathcal{M}(P) = D \quad (1)$$

$P$  est un programme

$\mathcal{M}$  est une fonction de valuation.

$D$  est une dénotation ou une valeur sémantique de dénotation.

- ▶ Domaine : Structure syntaxique abstraite du langage
- ▶ Codomaine : Objets des domaines sémantiques.
- ▶ Définition structurelle : le sens d'un arbre est défini à partir du sens de ses sous-arbres.

Une fonction de valuation sémantique associe une syntaxe abstraite à des objets d'un domaine sémantique.

► Syntaxe abstraite :

- Domaines syntaxiques :  $B \in \text{Nombre-binaire}$   
 $D \in \text{Chiffre-binaire}$
- Règles syntaxiques :  $B ::= BD|D$   
 $D ::= 0|1$



► Sous-arbre :  $\begin{array}{c} D \\ | \\ 0 \end{array}$

► Sens :  $\mathcal{D}\left(\begin{array}{c} D \\ | \\ 0 \end{array}\right) = zero$

► Notation :  $\mathcal{D}[[0]] = zero$

Fonction de valuation :  $\begin{array}{l} \mathcal{D}[[0]] = zero \\ \mathcal{D}[[1]] = un \end{array}$

► Sous-arbre :

$$\begin{array}{c} B \\ | \\ D \\ | \\ 1 \end{array}$$

► Sens :  $\mathcal{B}\left(\begin{array}{c} B \\ | \\ D \\ \Delta \end{array}\right) = \mathcal{D}\left(\begin{array}{c} D \\ \Delta \end{array}\right)$

► Notation :  $\mathcal{B}[[D]] = \mathcal{D}[[D]]$

Fonction de valuation :  $\mathcal{B}[[D]] = \mathcal{D}[[D]]$   
 $\mathcal{B}[[BD]] = (\mathcal{B}[[B]] \text{ fois deux}) \text{ plus } \mathcal{D}[[D]]$

Syntaxe abstraite

$B \in \text{Nombre-binaire}$

$D \in \text{Chiffre-binaire} \quad B ::= BD \mid D$

$D ::= 0 \mid 1$

Algèbres sémantiques

*Nombres naturels :*

Domaine  $\text{Nat} = \mathbb{N}$

Opérations zero, un deux, trois, ... :  $\text{Nat}$

plus, fois :  $\text{Nat} \times \text{Nat} \rightarrow \text{Nat}$

Fonctions de valuation

$\mathcal{B} : \text{Nombre-binaire} \rightarrow \text{Nat}$

$\mathcal{B}[[D]] = \mathcal{D}[[D]]$

$\mathcal{B}[[BD]] = (\mathcal{B}[[B]] \text{ fois deux}) \text{ plus } \mathcal{D}[[D]]$

$\mathcal{D} : \text{Chiffre-binaire} \rightarrow \text{Nat}$

$\mathcal{D}[[0]] = \text{zero}$

$\mathcal{D}[[1]] = \text{un}$

i

## ► Syntaxe

$\tau ::= \text{bool} \mid \text{nat} \quad \% \text{ types de données}$

$\theta ::= \text{exp}[\tau] \mid \text{comm} \quad \% \text{ types des textes de commandes}$

- tout nom de type  $\tau$  dénote un ensemble non vide  $\llbracket \tau \rrbracket$  de valeurs possibles :

- $\llbracket \text{bool} \rrbracket = \{true, false\}$
- $\llbracket \text{nat} \rrbracket = \{0, 1, 2, 3 \dots\}$

- $\llbracket \text{exp}[\tau] \rrbracket = \text{States} \longrightarrow \llbracket \tau \rrbracket$   
 $\llbracket \text{comm} \rrbracket = \text{States} \rightarrow \text{States}$  (fonction partielle)  
où States est l'ensemble des états,  
par exemple  $\text{States} = \text{Variable} \longrightarrow \text{Nat}$

► Equations sémantiques :

$$\llbracket \bullet \rrbracket_{\text{exp}[\tau]} : \text{exp}[\tau] \longrightarrow \llbracket \text{exp}(\tau) \rrbracket$$

$$\llbracket \bullet \rrbracket_{\text{comm}} : \text{comm} \longrightarrow \llbracket \text{comm} \rrbracket$$

- ▶  $\llbracket C_0; C_1 \rrbracket_{\text{comm}} = \llbracket C_1 \rrbracket \circ \llbracket C_0 \rrbracket$
- ▶  $\llbracket \text{for } N \text{ do } C \rrbracket_{\text{comm}} = \llbracket C \rrbracket^{\llbracket N \rrbracket}$
- ▶ Application à des démonstrations de propriétés sur les commandes :
  - $C \equiv C'$  si, et seulement si,  $\llbracket C \rrbracket = \llbracket C' \rrbracket$
  - $\text{for } 0 \text{ do } C \equiv \text{SKIP}$
  - $(C; C); C \equiv C; (C; C)$
- ▶ Affectation :
$$\llbracket I := E \rrbracket_{\text{comm}} = \lambda s \in \text{States}. s[I \mapsto \llbracket E \rrbracket_{\text{exp}}(s)]$$

## Observation

$\text{while } B \text{ do } C \equiv_{comm} \text{if } B \text{ then } C; \text{while } B \text{ do } C$

- ▶ Vue par les commandes ou instructions
  - $C_0 = \text{diverge}$  (instruction ou comande qui ne terline jamais)
  - $\forall i \in \mathbb{N} : C_{i+1} = \text{if } B \text{ then } C; C^i$
  - $\forall i \in \mathbb{N} : C_{i+1}(s) = \text{si } \llbracket B \rrbracket(s) \text{ alors } (\llbracket C \rrbracket; C_i)(s) \text{ sinon } s$
- ▶ Vue par les fonctions sémantiques :
  - $c_0 = \emptyset$  où  $\llbracket C_0 \rrbracket = c_0$
  - $\forall i \in \mathbb{N} : c_i = \llbracket C_i \rrbracket$
  - $\forall i \in \mathbb{N} : c_{i+1}(s) = \text{si } \llbracket B \rrbracket(s) \text{ alors } (\llbracket C \rrbracket; c_i)(s) \text{ sinon } s$
  - $\forall i \in \text{Nat} : \text{graphe}(c_i) \subseteq \text{graphe}(c_{i+1})$  etc  $c_i = \llbracket C_i \rrbracket$
- ▶  $\llbracket \text{while } B \text{ do } C \rrbracket = c_\infty$  où  
 $\text{graphe}(c_\infty) = \text{graphe}(c_0) \cup \dots \cup \text{graphe}(c_i) \cup \dots$
- ▶  $\llbracket \text{while } B \text{ do } C \rrbracket = \mu W$   
où  $W(f) = \lambda s \in \text{States}. \text{si } \llbracket B \rrbracket(s) \text{ alors } f(\llbracket C \rrbracket(s)) \text{ sinon } s$



►  $w \stackrel{def}{=}$

```
while x > 0 do
  x := x-1
od
```

►  $\llbracket x := x-1 \rrbracket(s) = f(s) = s[x \mapsto x-1]$

- - $S = \{s \mid s \in \{x\} \rightarrow \mathbb{Z}\}$
  - $\forall i. i \in \mathbb{Z} \Rightarrow s_i \in S$
  - $\forall s. s \in S \Rightarrow \exists i \in \mathbb{Z}. s = s_i$

►  $w_0 = \emptyset$

►  $w_1(s) = if(s(x) > 0, w_0(f(s)), s) : w_1 = \{s_i \mapsto s_i \mid i \in \mathbb{Z} \wedge i \leq 0\}$

►  $w_2(s) = if(s(x) > 0, w_1(f(s)), s) :$   
 $w_2 = \{s_i \mapsto s_i \mid i \in \mathbb{Z} \wedge i \leq 0\} \cup \{s_j \mapsto s_0 \mid j \in 0..1\}$

► ...

►  $w_{k+1}(s) = if(s(x) > 0, w_k(f(s)), s) :$   
 $w_2 = \{s_i \mapsto s_i \mid i \in \mathbb{Z} \wedge i \leq 0\} \cup \{s_j \mapsto s_0 \mid j \in 0..k\}$

► ...

►  $w = w_0 \cup w_1 \cup \dots \cup w_k \cup \{s_i \mapsto s_i \mid i \in \mathbb{Z} \wedge i \leq 0\} \cup \{s_i \mapsto 0 \mid i \in \mathbb{Z} \wedge i > 0\}$

- 1 Ingénierie des logiciels et des systèmes
- 2 Introduction à la sémantique des langages de programmation
- 3 Sémantique opérationnelle
- 4 Sémantique dénotationnelle
- 5 Equivalence des deux sémantiques**
- 6 Transformateurs de prédicats
  - Introduction
  - Définition et propriétés
- 7 Logique de Hoare
- 8 Utilisation des transformateurs de prédicats
- 9 Conclusion

## Equivalence pour les instructions de STATS

Pour toute instruction  $S$  de STATS, pour tout état  $s$  de STATES,  
 $\mathcal{S}_{sos}(S)(s) = \mathcal{S}_{nat}(S)(s) = \mathcal{D}(S)(s)$

- ▶ La sémantique opérationnelle est une sémantique liée à une fonction d'interprétation et de calcul du programme évalué.
- ▶ La sémantique dénotationnelle est une expression fonctionnelle du programme ;



- 1 Ingénierie des logiciels et des systèmes
- 2 Introduction à la sémantique des langages de programmation
- 3 Sémantique opérationnelle
- 4 Sémantique dénotationnelle
- 5 Equivalence des deux sémantiques
- 6 Transformateurs de prédicats**
  - Introduction
  - Définition et propriétés
- 7 Logique de Hoare
- 8 Utilisation des transformateurs de prédicats
- 9 Conclusion

- 1 Ingénierie des logiciels et des systèmes
- 2 Introduction à la sémantique des langages de programmation
- 3 Sémantique opérationnelle
- 4 Sémantique dénotationnelle
- 5 Equivalence des deux sémantiques
- 6 Transformateurs de prédicats**
  - Introduction
  - Définition et propriétés
- 7 Logique de Hoare
- 8 Utilisation des transformateurs de prédicats
- 9 Conclusion

### Listing 4 – annotation

```
int main(void){  
    int x,y,z,u;  
    x = 1;  
    /*@ assert x == 1; */  
    y = 2;  
    /*@ assert x == 1 && y == 2; */  
    z = x *y;  
    /*@ assert x == 1 && y == 2 && z == 2; */  
    u = z;  
    /*@ assert x == 1 && y == 2 && z == 2 && u == 2; */  
    return 0;  
}
```

### Listing 5 – difference de deux nombres

```
/*@  
    assigns \result;  
    ensures \result == (a - b);  
*/  
static int difference(int a, int b) {  
    return a-b;  
}
```



### Listing 6 – swap

```
/*@  
    requires \valid(a) && \valid(b);  
    assigns *a, *b;  
    ensures *a == \old(*b);  
    ensures *b == \old(*a);  
*/  
static void swap(int* a, int* b) {  
    int temp;  
    temp = (*a);  
    (*a) = (*b);  
    (*b) = temp;  
}
```

### Listing 7 – call

```
/*@  
    assigns \result;  
    ensures \result == (a - b);  
*/  
static int difference(int a, int b) {  
    return a-b;  
}  
  
/*@  
    requires \valid(a) && \valid(b);  
    assigns *a, *b;  
    ensures *a == \old(*b);  
    ensures *b == \old(*a);  
*/  
static void swap(int* a, int* b) {  
    int temp;  
    temp = (*a);  
    (*a) = (*b);  
    (*b) = temp;  
}
```

- Un programme  $P$  *produit* des résultats à partir de données en accord avec une sémantique :
- STATES est l'ensemble de tous les états de  $P$  :  $STATES = X \rightarrow \mathbb{Z}$  où  $X$  désigne les variables de  $P$ .
  - $s_0$  et  $s_f$  deux états de STATES :  $\mathcal{D}(P)(s_0) = s_f$  signifie que  $P$  est exécuté à partir d'un état  $s_0$  et produit un état  $s_f$ .
  - Pour un état  $s$  de  $P$  courant, on notera  $s(X) = x$  pour distinguer la valeur de la variable  $X$  et sa valeur courante en  $s$  :

- Un programme  $P$  *produit* des résultats à partir de données en accord avec une sémantique :
- STATES est l'ensemble de tous les états de  $P$  :  $STATES = X \rightarrow \mathbb{Z}$  où  $X$  désigne les variables de  $P$ .
  - $s_0$  et  $s_f$  deux états de STATES :  $\mathcal{D}(P)(s_0) = s_f$  signifie que  $P$  est exécuté à partir d'un état  $s_0$  et produit un état  $s_f$ .
  - Pour un état  $s$  de  $P$  courant, on notera  $s(X) = x$  pour distinguer la valeur de la variable  $X$  et sa valeur courante en  $s$  :

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

- Un programme  $P$  *produit* des résultats à partir de données en accord avec une sémantique :

- STATES est l'ensemble de tous les états de  $P$  :  $STATES = X \rightarrow \mathbb{Z}$  où  $X$  désigne les variables de  $P$ .
- $s_0$  et  $s_f$  deux états de STATES :  $\mathcal{D}(P)(s_0) = s_f$  signifie que  $P$  est exécuté à partir d'un état  $s_0$  et produit un état  $s_f$ .
- Pour un état  $s$  de  $P$  courant, on notera  $s(X) = x$  pour distinguer la valeur de la variable  $X$  et sa valeur courante en  $s$  :

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

- $\mathcal{D}(P)(s_0) = s_f$  définit la relation suivante sur l'ensemble des valeurs :

$$x_0 \xrightarrow{P} x_f$$

- Un programme  $P$  *produit* des résultats à partir de données en accord avec une sémantique :

- STATES est l'ensemble de tous les états de  $P$  :  $STATES = X \rightarrow \mathbb{Z}$  où  $X$  désigne les variables de  $P$ .
- $s_0$  et  $s_f$  deux états de STATES :  $\mathcal{D}(P)(s_0) = s_f$  signifie que  $P$  est exécuté à partir d'un état  $s_0$  et produit un état  $s_f$ .
- Pour un état  $s$  de  $P$  courant, on notera  $s(X) = x$  pour distinguer la valeur de la variable  $X$  et sa valeur courante en  $s$  :

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

- $\mathcal{D}(P)(s_0) = s_f$  définit la relation suivante sur l'ensemble des valeurs :

$$x_0 \xrightarrow{P} x_f$$

- Un programme  $P$  *remplit* un contrat (pre,post) :

- $P$  transforme une variable  $x$  à partir d'une valeur initiale  $x_0$  et produisant une valeur finale  $x_f$  :  $x_0 \xrightarrow{P} x_f$
- $x_0$  satisfait pre :  $\text{pre}(x_0)$
- $x_f$  satisfait post :  $\text{post}(x_0, x_f)$
- $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$

Un programme  $P$  *remplit* un contrat (pre,post) :

- ▶  $P$  transforme une variable  $x$  à partir d'une valeur initiale  $x_0$  et produisant une valeur finale  $x_f$  :  $x_0 \xrightarrow{P} x_f$
- ▶  $x_0$  satisfait pre :  $\text{pre}(x_0)$
- ▶  $x_f$  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  $\text{pre}(x_0)$

ensures  $\text{post}(x_0, x_f)$

variables  $X$

begin

$0 : P_0(x_0, x)$

instruction<sub>0</sub>

...

$i : P_i(x_0, x)$

...

instruction <sub>$f-1$</sub>

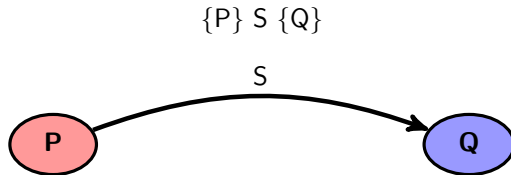
$f : P_f(x_0, x)$

end

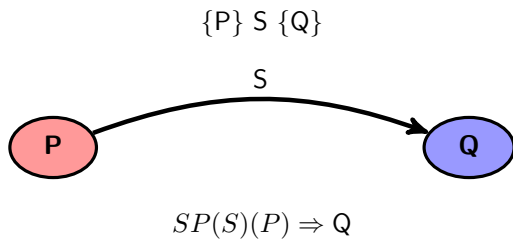
▶  $\text{pre}(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$

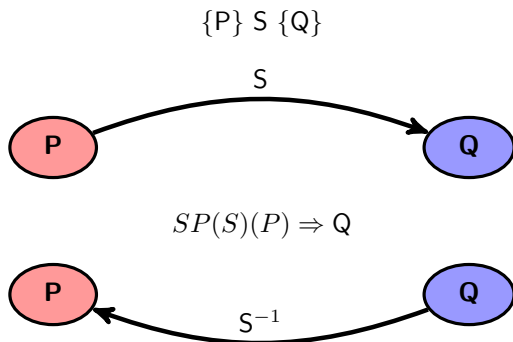
▶  $P_f(x_0, x) \Rightarrow \text{post}(x_0, x)$

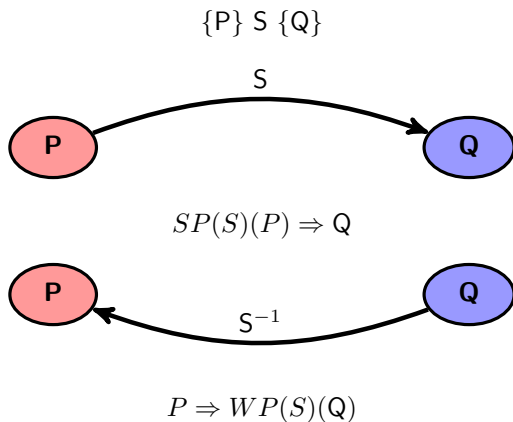
▶ conditions de vérification pour toutes les paires  $\ell \longrightarrow \ell'$











- 1 Ingénierie des logiciels et des systèmes
- 2 Introduction à la sémantique des langages de programmation
- 3 Sémantique opérationnelle
- 4 Sémantique dénotationnelle
- 5 Equivalence des deux sémantiques
- 6 Transformateurs de prédicats**
  - Introduction
  - Définition et propriétés
- 7 Logique de Hoare
- 8 Utilisation des transformateurs de prédicats
- 9 Conclusion

### 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 :

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

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

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

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



$S$  est une instruction et  $P$  et  $Q$  sont des prédicats.

- ▶ Loi du miracle exclu :  $wp(S)(FALSE) = FALSE$
- ▶ Distributivité de la conjonction :  
 $wp(S)(P) \wedge wp(S)(Q) = wp(S)(P \wedge Q)$
- ▶ Distributivité de la disjonction :  
 $wp(S)(P) \vee wp(S)(Q) \Rightarrow wp(S)(P \vee Q)$
- ▶ Si  $S$  est déterministe, alors  $wp(S)(P) \vee wp(S)(Q) = wp(S)(P \vee Q)$

# Current Summary

---

- 1 Ingénierie des logiciels et des systèmes
- 2 Introduction à la sémantique des langages de programmation
- 3 Sémantique opérationnelle
- 4 Sémantique dénotationnelle
- 5 Equivalence des deux sémantiques
- 6 Transformateurs de prédicats
  - Introduction
  - Définition et propriétés
- 7 Logique de Hoare**
- 8 Utilisation des transformateurs de prédicats
- 9 Conclusion

.....

☒ Definition(Axiomes et règles d'inférence)

- ▶ Axiome d'affectation :  $\{P(e/x)\} \mathbf{X} := \mathbf{E}(\mathbf{X}) \{P\}$ .
  - ▶ 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  $\{P\} \mathbf{if\ B\ then\ S_1\ then\ S_2\ fi} \{Q\}$ .
  - ▶ Si  $\{P \wedge B\} \mathbf{S} \{P\}$ , alors  $\{P\} \mathbf{while\ B\ do\ S\ od} \{P \wedge \neg B\}$ .
  - ▶ Règle de renforcement/affaiblissement : Si  $P' \Rightarrow P$ ,  $\{P\} \mathbf{S} \{Q\}$ ,  $Q \Rightarrow Q'$ , alors  $\{P'\} \mathbf{S} \{Q'\}$ .
- .....

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

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

.....

### ⊗ Définition

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

.....

.....

### ☺ PropertyCorrection du système axiomatique des programmes commentés

- ▶ S'il existe une preuve construite avec les règles précédentes de  $\{P\}\mathbf{S}\{Q\}$ , alors  $\{P\}\mathbf{S}\{Q\}$  est valide.
- ▶ Si  $\{P'\}\mathbf{S}\{Q'\}$  est valide et si le langage d'assertions est suffisamment expressif, alors il existe une preuve construite avec les règles précédentes de  $\{P\}\mathbf{S}\{Q\}$ .

.....

### ⊗ Définition

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  $(\text{PRED}, \Rightarrow, \mathbf{false}, \mathbf{true}, \wedge, \vee) : (\text{PRED}, \Rightarrow, \mathbf{false}, \mathbf{true}, \wedge, \vee)$  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

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

.....

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

.....

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

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

.....



.....

☒ Definition triplets de Hoare

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

.....

.....

☒ Definition (Axiomes et règles d'inférence)

- ▶ Axiome d'affectation :  $\{P(e/x)\}X := E(X)\{P\}$ .
  - ▶ Axiome du saut :  $\{P\}\text{skip}\{P\}$ .
  - ▶ Règle de composition : Si  $\{P\}S_1\{R\}$  et  $\{R\}S_2\{Q\}$ , alors  $\{P\}\text{if } B \text{ then } S_1 \text{ then } S_2 \text{ fi}\{Q\}$ .
  - ▶ Si  $\{P \wedge B\}S_1\{Q\}$  et  $\{P \wedge \neg B\}S_2\{Q\}$ , alors  $\{P\}\text{if } B \text{ then } S_1 \text{ then } S_2 \text{ fi}\{Q\}$ .
  - ▶ Si  $\{P \wedge B\}S\{P\}$ , alors  $\{P\}\text{while } B \text{ do } S \text{ od}\{P \wedge \neg B\}$ .
  - ▶ Règle de renforcement/affaiblissement : Si  $P' \Rightarrow P$ ,  $\{P\}S\{Q\}$ ,  $Q \Rightarrow Q'$ , alors  $\{P'\}S\{Q'\}$ .
- .....

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

.....

⊠ Définition triplets de Hoare Correction Totale

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

.....

.....

### ☒ Definition triplets de Hoare Correction Totale

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

.....

.....

### ☒ Definition (Axiomes et règles d'inférence)

- ▶ Axiome d'affectation :  $[P(e/x)]\mathbf{X} := \mathbf{E}(\mathbf{X})[P]$ .
  - ▶ Axiome du saut :  $[P]\mathbf{skip}[P]$ .
  - ▶ Règle de composition : Si  $[P]\mathbf{S}_1[R]$  et  $[R]\mathbf{S}_2[Q]$ , alors  $[P]\mathbf{if\ B\ then\ S_1\ then\ S_2\ fi}[Q]$ .
  - ▶ Si  $[P \wedge B]\mathbf{S}_1[Q]$  et  $[P \wedge \neg B]\mathbf{S}_2[Q]$ , alors  $[P]\mathbf{if\ B\ then\ S_1\ then\ S_2\ fi}[Q]$ .
  - ▶ Si  $[P(n+1)]\mathbf{S}[P(n)]$ ,  $P(n+1) \Rightarrow b$ ,  $P(0) \Rightarrow \neg b$ , alors  $[\exists n \in \mathbb{N}. P(n)]\mathbf{while\ B\ do\ S\ od}[P(0)]$ .
  - ▶ 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  $P \wp(S) \wp Q$ .

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

- 1 Ingénierie des logiciels et des systèmes
- 2 Introduction à la sémantique des langages de programmation
- 3 Sémantique opérationnelle
- 4 Sémantique dénotationnelle
- 5 Equivalence des deux sémantiques
- 6 Transformateurs de prédicats
  - Introduction
  - Définition et propriétés
- 7 Logique de Hoare
- 8 Utilisation des transformateurs de prédicats
- 9 Conclusion

## Vérification du contrat (I)

Un programme  $P$  *remplit* un contrat  $(pre, post)$  :

- ▶  $P$  transforme une variable  $x$  à partir d'une valeur initiale  $x_0$  et produisant une valeur finale  $x_f$  :  $x_0 \xrightarrow{P} x_f$
- ▶  $x_0$  satisfait  $pre$  :  $pre(x_0)$  and  $x_f$  satisfait  $post$  :  $post(x_0, x_f)$
- ▶  $pre(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow post(x_0, x_f)$

requires  $pre(x_0)$

ensures  $post(x_0, x_f)$

variables  $X$

begin

$0 : P_0(x_0, x)$

instruction<sub>0</sub>

...

$i : P_i(x_0, x)$

...

instruction <sub>$f-1$</sub>

$f : P_f(x_0, x)$

end

▶  $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$

▶  $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$

▶ Pour toute paire d'étiquettes  $\ell, \ell'$  telle que  $\ell \longrightarrow \ell'$ , on vérifie que, pour toutes valeurs

$x, x' \in \text{MEMORY}$

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





## Vérification du contrat (II)

Un programme  $P$  *remplit* un contrat (pre,post) :

- ▶  $P$  transforme une variable  $x$  à partir d'une valeur initiale  $x_0$  et produisant une valeur finale  $x_f$  :  $x_0 \xrightarrow{P} x_f$
- ▶  $x_0$  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  $\text{pre}(x_0)$   
ensures  $\text{post}(x_0, x_f)$   
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
```

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

Un programme  $P$  *remplit* un contrat  $(\text{pre}, \text{post})$  :

- ▶  $P$  transforme une variable  $x$  à partir d'une valeur initiale  $x_0$  et produisant une valeur finale  $x_f$  :  $x_0 \xrightarrow{P} x_f$
- ▶  $x_0$  satisfait  $\text{pre}$  :  $\text{pre}(x_0)$  and  $x_f$  satisfait  $\text{post}$  :  $\text{post}(x_0, x_f)$
- ▶  $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$
- ▶  $\forall x_0. \text{pre}(x_0) \Rightarrow WLP(P)(\text{post}(x_0, x))$

Un programme  $P$  *remplit* un contrat  $(\text{pre}, \text{post})$  :

- ▶  $P$  transforme une variable  $x$  à partir d'une valeur initiale  $x_0$  et produisant une valeur finale  $x_f$  :  $x_0 \xrightarrow{P} x_f$
- ▶  $x_0$  satisfait  $\text{pre}$  :  $\text{pre}(x_0)$  and  $x_f$  satisfait  $\text{post}$  :  $\text{post}(x_0, x_f)$
- ▶  $\text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$
- ▶  $\forall x_0. \text{pre}(x_0) \Rightarrow WLP(P)(\text{post}(x_0, x))$
- ▶ WLP n'est pas calculable.
- ▶ Utilisation de la logique de HOARE comme support du calcul de WLP.

## Vérification du contrat (III)

Un programme  $P$  *remplit* un contrat (pre,post) :

- ▶  $P$  transforme une variable  $x$  à partir d'une valeur initiale  $x_0$  et produisant une valeur finale  $x_f$  :  $x_0 \xrightarrow{P} x_f$
- ▶  $x_0$  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(x0)
ensures post(x0, xf)
variables X
begin
  /·@assert P0(x0, x)·/
  T;
  /·@loop invariant I(x0, x)·/
  while B(x) do
    S
  od
  /·@assert Pf(x0, 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  $pre(x_0)$   
ensures  $post(x_0, x_f)$   
variables  $X$   
  begin  
    /·@assert  $P_0(x_0, x)$ ·/  
    S1;  
    S2;  
    /·@assert  $P_f(x_0, x)$ ·/  
  end
```

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

```
requires  $pre(x_0)$   
ensures  $post(x_0, x_f)$   
variables  $X$   
begin  
  /·@assert  $P_0(x_0, x)$ ·/  
  if  $B(x)$  do  
     $S1$   
  else  
     $S2$   
  elfi  
  /·@assert  $P_f(x_0, x)$ ·/  
end
```

$$x = x_0 \wedge \text{pre}(x_0) \Rightarrow P_0(x_0, x)$$

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

## Current Summary



- ▶ Trois notions importantes : syntaxe, sémantique et pragmatique
- ▶ La sémantique est le fondement des langages de programmation.
- ▶ La sémantique permet de donner une vue cohérente des programmes et des spécifications.
- ▶ Développement de techniques et d'outils de vérification et de validation de systèmes.