
Cours MALG & MOVEX

MALG

Sémantique des langages de programmation

Dominique Méry

Telecom Nancy, Université de Lorraine

Année universitaire 2023-2024

- ① Software-based system engineering
- ② Summary on verification techniques
 - Tryptic Pattern
 - Programs as Predicate Transformers
- ③ Introduction to semantics for programming languages
- ④ Operational Semantics
- ⑤ Denotational Semantics
- ⑥ Equivalence des deux sémantiques
- ⑦ Transformateurs de prédicats
 - Introduction
 - Définition et propriétés
 - Construction du wp pour la conditionnelle
 - Construction du wp pour l'itération
- ⑧ Logique de Hoare
- ⑨ Epilogue

- 1 Software-based system engineering
- 2 Summary on verification techniques
 - Tryptic Pattern
 - Programs as Predicate Transformers
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats
 - Introduction
 - Définition et propriétés
 - Construction du wp pour la conditionnelle
 - Construction du wp pour l'itération
- 8 Logique de Hoare
- 9 Epilogue

- ① Software-based system engineering
- ② Summary on verification techniques
 - Tryptic Pattern
 - Programs as Predicate Transformers
- ③ Introduction to semantics for programming languages
- ④ Operational Semantics
- ⑤ Denotational Semantics
- ⑥ Equivalence des deux sémantiques
- ⑦ Transformateurs de prédicats
 - Introduction
 - Définition et propriétés
 - Construction du wp pour la conditionnelle
 - Construction du wp pour l'itération
- ⑧ Logique de Hoare
- ⑨ Epilogue

Summary

- 1 Software-based system engineering
- 2 Summary on verification techniques
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats
- 8 Logique de Hoare
- 9 Epilogue

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 : :

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

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

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

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**.

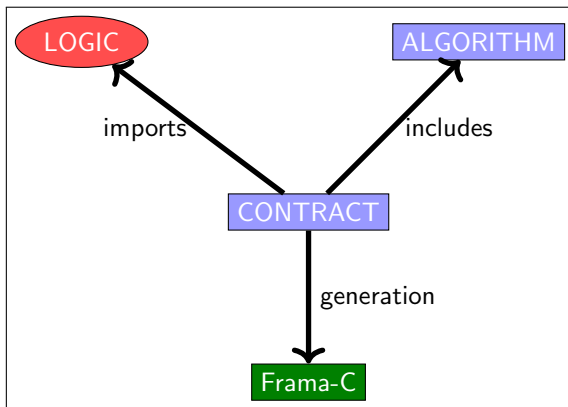
If we make an abstraction of the language users and if we analyze the expressions and their meanings only, we are in the area of the **semantics**.

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

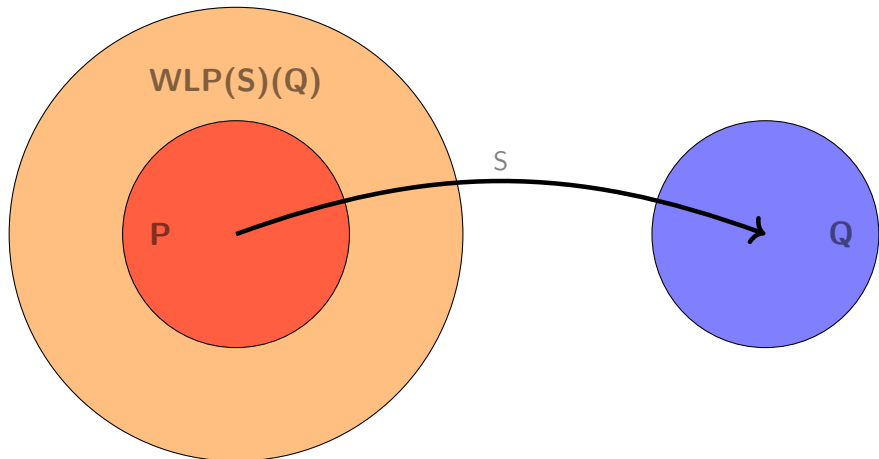
These three elements constitute the science of the language or semiotics.

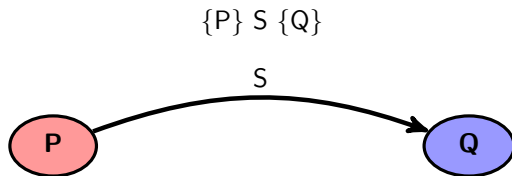
- 1 Software-based system engineering
- 2 Summary on verification techniques
 - Tryptic Pattern
 - Programs as Predicate Transformers
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats
- 8 Logique de Hoare

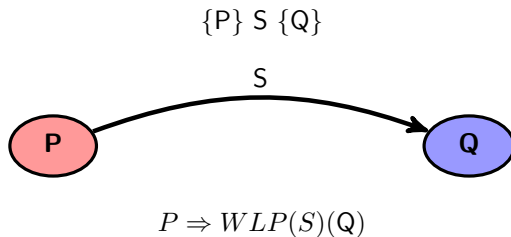
- 1 Software-based system engineering
- 2 Summary on verification techniques
 - Tryptic Pattern
 - Programs as Predicate Transformers
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats
 - Introduction
 - Définition et propriétés
 - Construction du wp pour la conditionnelle
 - Construction du wp pour l'itération
- 8 Logique de Hoare
- 9 Epilogue

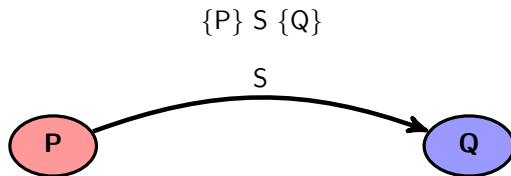


- 1 Software-based system engineering
- 2 Summary on verification techniques
 - Tryptic Pattern
 - Programs as Predicate Transformers
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats
 - Introduction
 - Définition et propriétés
 - Construction du wp pour la conditionnelle
 - Construction du wp pour l'itération
- 8 Logique de Hoare
- 9 Epilogue









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

Computing $WLP(S)(Q)$?

Summary

- 1 Software-based system engineering
- 2 Summary on verification techniques
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats
- 8 Logique de Hoare
- 9 Epilogue

Implicite versus explicite

- ▶ Writing $101 = 5$ may have a meaning ...

Implicite versus explicite

- ▶ Writing $101 = 5$ may have a meaning ...
- ▶ 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 : $base(2, 10, 101) = 1.2^2 + 0.2 + 1.2^0 = 5_{10}$

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

- ▶ average est une fonction utilisée dans des parties très profondes du code comme la recherche dichotomique.
- ▶ analyse de l'addition et de la division.
- ▶ anticiper les calculs

- ▶ 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 ?

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

- ▶ $\text{pre}(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- ▶ $P_f(x_0, x) \Rightarrow \text{post}(x_0, x)$
- ▶ conditions de vérification pour toutes les paires $\ell \longrightarrow \ell'$ qui vont être traduites avec une sémantique wp.
- ▶ $x_0 \xrightarrow{P} x_f$ exprime la relation de calcul de x_0 à x_f sous la forme d'une sémantique opérationnelle ou dénotationnelle.

Correction partielle

- (I) $\forall x_0, x_f \in D. \text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$
- (II) $\forall x_0 \in D. \text{pre}(x_0) \Rightarrow (\forall x_f \in D. x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f))$
- (III) $\forall x_0 \in D. \text{pre}(x_0) \Rightarrow \text{wlp}(P)(\text{post}(x_0, x_f))$

Correction partielle

- (I) $\forall x_0, x_f \in D. \text{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f)$
- (II) $\forall x_0 \in D. \text{pre}(x_0) \Rightarrow (\forall x_f \in D. x_0 \xrightarrow{P} x_f \Rightarrow \text{post}(x_0, x_f))$
- (III) $\forall x_0 \in D. \text{pre}(x_0) \Rightarrow wlp(P)(\text{post}(x_0, x_f))$

Techniques équivalentes de vérification

- ▶ méthode des assertions inductives de Floyd-Hoare (annotation et vérification).
- ▶ méthode du calcul wp (calcul de la précondition associée à un programme)
- ▶ définition équivalente de sémantiques des langages de programmation :
 - opérationnelle
 - dénotationnelle
 - axiomatique

Summary

- 1 Software-based system engineering
- 2 Summary on verification techniques
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics**
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats
- 8 Logique de Hoare
- 9 Epilogue

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

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

Summary

- 1 Software-based system engineering
- 2 Summary on verification techniques
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics**
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats
- 8 Logique de Hoare
- 9 Epilogue

- ▶ 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 ou l'implémentation.
- ▶ Préservation de la sémantique d'un langage de programmation dans un langage de plus bas niveau par compilation : correction du compilateur.

Une notation pour des instructions comme par exemple C,
PASCAL, SHELL,

- ▶ Syntaxe : Structure et forme des sentences
- ▶ Sémantique : Association d'un sens aux sentences comme par exemple des nombres, des fonctions, des actions d'une machine, ...
- ▶ Pragmatique : Utilisation effective du langage comme par exemple les domaines d'applications, les performances, ...

Des éléments spécifiques pour chaque programme d'une machine.

- ▶ 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

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

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

$D ::= 0 \mid 1$

Domaines sémantiques

Nombres naturels :

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

Opérations zero, un deux, trois, ... : Nat

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

Fonctions de valuation

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

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

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

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

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

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

► Syntaxe

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

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

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

- $\llbracket \text{bool} \rrbracket = \{\text{true}, \text{false}\} = \mathbb{B}$
- $\llbracket \text{nat} \rrbracket = \{0, 1, 2, 3 \dots\} = \mathbb{N}$
- $\llbracket \text{int} \rrbracket = \{\dots -3, -2, -1, 0, 1, 2, 3 \dots\} = \mathbb{Z}$

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

Equations sémantiques

- ▶ $\llbracket \bullet \rrbracket_{\text{exp}[\tau]} : \text{exp}[\tau] \longrightarrow \llbracket \text{exp}(\tau) \rrbracket$
- ▶ $\llbracket \bullet \rrbracket_{\text{comm}} : \text{comm} \longrightarrow \llbracket \text{comm} \rrbracket$
- ▶ $\llbracket \bullet \rrbracket_{\text{exp}[\tau]}$ est noté $\llbracket \bullet \rrbracket$
- ▶ $\llbracket \bullet \rrbracket_{\text{comm}}$ est noté $\llbracket \bullet \rrbracket$

Instruction d'affectation

Pour tout état $s \in \text{States}$, $\llbracket I := E \rrbracket(s) = s[I \mapsto \llbracket E \rrbracket(s)]$

Instruction conditionnelle

Pour tout état $s \in \text{States}$,

Cond+ : $\llbracket \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi} \rrbracket(s) = \llbracket S_1 \rrbracket(s) \text{ si } \llbracket B \rrbracket(s) = \text{TRUE}$

Cond- : $\llbracket \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi} \rrbracket(s) = \llbracket S_2 \rrbracket(s) \text{ si } \llbracket B \rrbracket(s) = \text{FALSE}$

Instruction conditionnelle

Pour tout état $s \in \text{States}$,

$$\llbracket S_1; S_2 \rrbracket(s) = \llbracket S_2 \rrbracket(\llbracket S_1 \rrbracket(s)) = \llbracket S_2 \rrbracket \circ \llbracket S_1 \rrbracket(s)$$

Application à des démonstrations de propriétés sur les commandes :

- ▶ $C \equiv C'$ si, et seulement si, $\llbracket C \rrbracket = \llbracket C' \rrbracket$
- ▶ $\text{for } 0 \text{ do } C \equiv \text{SKIP}$
- ▶ $(C; C); C \equiv C; (C; C)$
- ▶ $\llbracket \text{for } N \text{ do } C \rrbracket = \llbracket C \rrbracket^{\llbracket N \rrbracket}$

Observation

`while B do C` \equiv `if B then C; while B do C`

traduite sémantiquement comme suit :

$\llbracket \text{while } B \text{ do } C \rrbracket \equiv \llbracket \text{if } B \text{ then } C; \text{while } B \text{ do } C \rrbracket$

Pour tout $s \in \text{States}$, $W(f)(s) =$ si $\llbracket B \rrbracket(s)$ alors $f(\llbracket C \rrbracket(s))$ sinon s :

- ▶ $W \in (\text{States} \rightarrow \text{States}) \rightarrow (\text{States} \rightarrow \text{States})$
- ▶ $(\text{States} \rightarrow \text{States}, \sqsubseteq)$ est une structure partiellement ordonnée inductive.
- ▶ W est continue pour cette structure.
- ▶ Le plus petit point fixe de W est noté μW :
 - $\mu W = \bigvee_{i \in \mathbb{N}} W_i$.
 - $W_0 = \perp$ où $\text{graphe}(\perp) = \emptyset$.
 - Soit $s \in \text{States}$:
 $W_{i+1}(s) = \text{if } \llbracket B \rrbracket(s) \text{ then } W_i(\llbracket C \rrbracket(s)) \text{ else } s \text{ end}$
- ▶ Soit $s \in \text{States}$:
 $\mu W(s) = \text{if } \llbracket B \rrbracket(s) \text{ then } \mu W(\llbracket C \rrbracket(s)) \text{ else } s \text{ end}$

Denotational Semantics for iteration

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

Pour tout $s \in \text{States}$, $W(f)(s) = \text{si } \llbracket B \rrbracket(s) \text{ alors } f(\llbracket C \rrbracket(s)) \text{ sinon } s :$

Pour tout état $s \in \text{States}$,

$\llbracket \text{while } B \text{ do } C \rrbracket(s) = \mu f. W(f)(s)$

- ▶ $\llbracket \text{while } B \text{ do } C \rrbracket \equiv \llbracket \text{if } B \text{ then } C; \text{while } B \text{ do } C \rrbracket$ is a property from the definition of $\llbracket \text{while } B \text{ do } C \rrbracket$.
- ▶ For any statement S , $\llbracket S \rrbracket$ is a partial function from States to States.

Summary

- 1 Software-based system engineering
- 2 Summary on verification techniques
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats
- 8 Logique de Hoare
- 9 Epilogue

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 Software-based system engineering
- ② Summary on verification techniques
- ③ Introduction to semantics for programming languages
- ④ Operational Semantics
- ⑤ Denotational Semantics
- ⑥ Equivalence des deux sémantiques
- ⑦ Transformateurs de prédicats
 - Introduction
 - Définition et propriétés
 - Construction du wp pour la conditionnelle
 - Construction du wp pour l'itération

- 1 Software-based system engineering
- 2 Summary on verification techniques
 - Tryptic Pattern
 - Programs as Predicate Transformers
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats**
 - Introduction
 - Définition et propriétés
 - Construction du wp pour la conditionnelle
 - Construction du wp pour l'itération
- 8 Logique de Hoare
- 9 Epilogue

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

- 1 Software-based system engineering
- 2 Summary on verification techniques
 - Tryptic Pattern
 - Programs as Predicate Transformers
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats**
 - Introduction
 - Définition et propriétés
 - Construction du wp pour la conditionnelle
 - Construction du wp pour l'itération
- 8 Logique de Hoare
- 9 Epilogue

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 :

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

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

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 | $wp(S)(P)$ |
| $X := E(X, D)$ | $P[e(x, d)/x]$ |
| 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 $wp(S)(P) \vee wp(S)(Q) = wp(S)(P \vee Q)$

- 1 Software-based system engineering
- 2 Summary on verification techniques
 - Tryptic Pattern
 - Programs as Predicate Transformers
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats**
 - Introduction
 - Définition et propriétés
 - Construction du wp pour la conditionnelle
 - Construction du wp pour l'itération
- 8 Logique de Hoare
- 9 Epilogue

$$\blacktriangleright S \stackrel{def}{=} \left[\begin{array}{l} \text{IF } B_1 \longrightarrow S_1 \\ \square B_2 \longrightarrow S_2 \\ \dots \\ \square B_n \longrightarrow S_n \\ \text{FI} \end{array} \right]$$

$$\blacktriangleright wp(S)(P) = \left[\begin{array}{l} (B_1 \vee \dots \vee B_n) \\ \wedge (B_1 \Rightarrow wp(S_1)(P)) \\ \dots \\ \wedge (B_n \Rightarrow wp(S_n)(P)) \end{array} \right]$$

- 1 Software-based system engineering
- 2 Summary on verification techniques
 - Tryptic Pattern
 - Programs as Predicate Transformers
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats**
 - Introduction
 - Définition et propriétés
 - Construction du wp pour la conditionnelle
 - Construction du wp pour l'itération
- 8 Logique de Hoare
- 9 Epilogue

$$\blacktriangleright S \stackrel{def}{=} \left[\begin{array}{l} \text{DO } B_1 \longrightarrow S_1 \\ \square B_2 \longrightarrow S_2 \\ \dots \\ \square B_n \longrightarrow S_n \\ \text{OD} \end{array} \right]$$

$$\blacktriangleright BS \stackrel{def}{=} \left[\begin{array}{l} \text{IF } B_1 \longrightarrow S_1 \\ \square B_2 \longrightarrow S_2 \\ \dots \\ \square B_n \longrightarrow S_n \\ \text{FI} \end{array} \right]$$

$$\blacktriangleright B \stackrel{def}{=} (B_1 \vee \dots \vee B_n)$$

$$\blacktriangleright T \stackrel{def}{=} \left[\begin{array}{l} \text{DO } B \longrightarrow \left[\begin{array}{l} \text{IF } B_1 \longrightarrow S_1 \\ \square B_2 \longrightarrow S_2 \\ \dots \\ \square B_n \longrightarrow S_n \\ \text{FI} \end{array} \right] \end{array} \right]$$

$$\blacktriangleright T \stackrel{def}{=} \left[\text{DO } B \longrightarrow \left[\begin{array}{l} \text{IF } B_1 \longrightarrow S_1 \\ \square B_2 \longrightarrow S_2 \\ \dots \\ \square B_n \longrightarrow S_n \\ \text{FI} \end{array} \right] \right]$$

$$\blacktriangleright T \stackrel{def}{=} [\text{DO } B \longrightarrow BS \text{ OD}]$$

$$\blacktriangleright wp(S)(P) = wp(T)(P)$$

$$\blacktriangleright wp(T)(P) = \bigvee_{k \in \mathbb{N}} W_k \text{ où}$$

- Une suite d'assertions notées $W_0, \dots, W_k \dots$ est définie comme étant
- $W_0(P) = \neg B \wedge P$
- $\forall k \in \mathbb{N} : W_{k+1}(P) = W_0(P) \vee wp(BS)(W_k)$

$$\blacktriangleright wp(S)(P) = \exists k \in \mathbb{N} : W_k$$

- ▶ W est un transformateur de prédicats de Pred dans Pred défini par $W(X) = (B \wedge wp(BS)(X)) \vee (\neg B \wedge P)$
- ▶ W est monotone croissant (Si $P \Rightarrow Q$, alors $W(P) \Rightarrow W(Q)$).
- ▶ W admet un plus petit point-fixe d'après le Théorème de Tarski noté μW et défini par :
 - $F_0 = FALSE$
 - $\forall i \in \mathbb{N} : F_{i+1} = W(F_i)$

.....

☺ Théorème

$$\forall i \in \mathbb{N} : F_{i+1} = W_i$$

.....

☺ Théorème

$$\mu W = WP(S)(P)$$

.....

☒ Definition

$$WP(S)(P) = \mu\lambda X.((B \wedge wp(BS)(X)) \vee (\neg B \wedge P))$$

.....

- ▶ La suite W_k compte le nombre de boucles avant de terminer.
- ▶ La méthode de terminaison consiste à définir une borne de terminaison.
- ▶ En général, il faut une relation bien fondée telle que chaque boucle décroît strictement selon la relation bien fondée ;

Summary

- 1 Software-based system engineering
- 2 Summary on verification techniques
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats
- 8 Logique de Hoare

9 Epilogue

.....

☒ 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 $\{P\} \mathbf{S_1 ; S_2} \{Q\}$.
 - ▶ 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)

.....

⊗ Definition

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

.....

.....

☺ Property(Correction du système axiomatique des programmes commentés)

- ▶ S'il existe une preuve construite avec les règles précédentes de $\{P\}\mathbf{S}\{Q\}$, 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 $\{P\}\mathbf{S}\{Q\}$.

.....

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

-

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

.....

.....

- ▶ $loop(S) \equiv \overline{(\exists t \in STATES : \mathcal{D}(S)(s) = t)}$ (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.

.....

☒ Definition 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.

Summary

- 1 Software-based system engineering
- 2 Summary on verification techniques
- 3 Introduction to semantics for programming languages
- 4 Operational Semantics
- 5 Denotational Semantics
- 6 Equivalence des deux sémantiques
- 7 Transformateurs de prédicats
- 8 Logique de Hoare

9 Epilogue

MALG

Sémantique des langages de programmation (26 janvier 2026) (Dominique Méry)

- ▶ 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.