Cours MALG & MOVEX

MALG
Sémantique des langages de programmation

Dominique Méry
Telecom Nancy, Université de Lorraine

(23 mars 2025 at 11:27 P.M.)

Année universitaire 2024-2025

Plan

Sommaire

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

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

- 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. ¡bloc
- 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 : :

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

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 : :

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

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 : :

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

System Modelling

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

Triptych of Software/System Engineering

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

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

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

Triptych of Software/System Engineering

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

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

- 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.

Context

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.

Implicite versus explicite

- Ecrire $\mathit{101} = \mathit{5}$ 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 : $base(2, 10, 101) = 1.2^2 + 0.2 + 1.2^0 = 5_{10}$

```
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

Example : description of static behaviour

- 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 ?

Des sémantiques pour des langages de programmation

- 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 étas 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.

Vérification du contrat

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

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

begin
$0 : P_0(x_0, x)$
`instruction`$_0$
$\ldots$
$i : P_i(x_0, x)$
$\ldots$
`instruction`$_{f-1}$
$f : P_f(x_0, x)$
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 $\ell \longrightarrow \ell'$ qui vont être traduites avec une sémantique wp.
- $x_0 \xrightarrow{\mathsf{P}} x_f$ exprime la relation de calcul de x_0 à x_f sous la forme d'une sémantique opérationnelle ou dénotationnelle.

Correction partielle

(I) $\forall x_0, x_f \in \mathsf{D}.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

(II) $\forall x_0 \in \mathsf{D}.\mathsf{pre}(x_0) \Rightarrow (\forall x_f \in \mathsf{D}.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f))$

(III) $\forall x_0 \in \mathsf{D}.\mathsf{pre}(x_0) \Rightarrow wlp(P)(\mathsf{post}(x_0, x_f))$

Principes de l'utilisation des wps

Correction partielle

(I) $\forall x_0, x_f \in \mathsf{D}.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

(II) $\forall x_0 \in \mathsf{D}.\mathsf{pre}(x_0) \Rightarrow (\forall x_f \in \mathsf{D}.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f))$

(III) $\forall x_0 \in \mathsf{D}.\mathsf{pre}(x_0) \Rightarrow wlp(P)(\mathsf{post}(x_0, x_f))$

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

Sémantique Opérationnelle Structurelle et Sémantique Natutelle

- 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 : $(S, s) \xrightarrow[\text{nat}]{} s'$

Cadre pour la sémantique opérationnelle

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

Règles de définition selon la syntaxe

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

Règles de définition selon la syntaxe

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

Fonctions sémantiques associées aux instructions

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

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

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

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

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 $(S,s) \underset{\text{nat}}{\longrightarrow} s'$, alors $(S,s) \overset{\star}{\underset{\text{sos}}{\longrightarrow}} s'$.
- Montrons que si $(S,s) \overset{\star}{\underset{\text{sos}}{\longrightarrow}} s'$., alors $(S,s) \underset{\text{nat}}{\longrightarrow} s'$.

Motivations

- 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.

Langage de programmation

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 exeple 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.

Mise en œuvre

- 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 œvre d'un langage est une activité pragmatique.

Interprétation versus compilation

- 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.

Spécifications formelles des langages

- 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.

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

P est un programme
$\mathcal{M}$ est une fonction de valuation.
D est une dénotation ou une valeur sémantique de dénotation.

Fonction de valuation

- 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.

Langage des nombres binaires

- Syntaxe abstraite :
  - Domaines syntaxiques : $\begin{array}{l} B \in Nombre{-}binaire \\ D \in Chiffre{-}binaire \end{array}$
  - Règles syntaxiques : $\begin{array}{l} B ::= BD|D \\ D ::= 0|1 \end{array}$

Sens des sentences terminales

- Sous-arbre : $\begin{array}{c} D \\ | \\ 0 \end{array}$
- Sens : $\mathcal{D}(\begin{array}{c} D \\ | \\ 0 \end{array}) = zero$
- Notation : $\mathcal{D}[\![0]\!] = zero$

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

Sens des sentences non-terminales

- 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 : $\begin{array}{l} \mathcal{B}[\![D]\!] = \mathcal{D}[\![D]\!] \\ \mathcal{B}[\![BD]\!] = (\mathcal{B}[\![B]\!]\ fois\ deux)\ plus\ \mathcal{D}[\![D]\!] \end{array}$

Définition de la sémantique

Syntaxe abstraite

$B \in Nombre{-}binaire$

$D \in Chiffre{-}binaire B ::= BD|D$

$D ::= 0|1$

Algèbres sémantiques

Nombres naturels :

Domaine Nat $= \mathbb{N}$

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

plus, fois : Nat $\times$ Nat $\rightarrow$ Nat

Fonctions de valuation

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

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

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

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

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

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

- **Syntaxe**
  τ ::= $\texttt{bool}$ | $\texttt{nat}$ % types de données
  θ ::= $\texttt{exp}[\tau]$ | $\texttt{comm}$ % types des textes de commandes
- tout nom de type τ dénote un ensemble non vide $[\![\tau]\!]$ de valeurs possibles :
  - $[\![\texttt{bool}]\!] = \{true, false\}$
  - $[\![\texttt{nat}]\!] = \{0, 1, 2, 3 \ldots\}$
- $[\![\texttt{exp}[\tau]]\!] = \texttt{States} \longrightarrow [\![\tau]\!]$
  $[\![\texttt{comm}]\!] = \texttt{States} \not\rightarrow \texttt{States}$ (fonction partielle)
  où $\texttt{States}$ est l'ensemble des états,
  par exemple $\texttt{States} = Variable \longrightarrow Nat$

Equations sémantiques

- $[\![\bullet]\!]_{\texttt{exp}[\tau]} : \texttt{exp}[\tau] \longrightarrow [\![\texttt{exp}(\tau)]\!]$
- $[\![\bullet]\!]_{\texttt{comm}} : \texttt{comm} \longrightarrow [\![\texttt{comm}]\!]$
- $[\![\bullet]\!]_{\texttt{exp}[\tau]}$ est noté $[\![\bullet]\!]$
- $[\![\bullet]\!]_{\texttt{comm}}$ est noté $[\![\bullet]\!]$

Instruction d'affectation

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

Instruction conditionnelle

Pour tout état $s \in \texttt{States}$,

Cond+ : $[\![\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}]\!](s) = [\![S_1]\!](s) \ si \ \ [\![B]\!](s) = TRUE$

Cond- : $[\![\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}]\!](s) = [\![S_2]\!](s) \ si \ \ [\![B]\!](s) = FALSE$

Instruction conditionnelle

Pour tout état $s \in \texttt{States}$,
$[\![S_1 ; S_2]\!](s) = [\![S_2]\!]([\![S_1]\!](s)) = [\![S_2]\!] \circ [\![S_1]\!](s)$

Application à des démonstrations de propriétés sur les commandes :

- $\texttt{C} \equiv \texttt{C}'$ si, et seulement si, $[\![\texttt{C}]\!] = [\![\texttt{C}']\!]$
- $\texttt{for } 0 \texttt{ do C} \equiv \texttt{SKIP}$
- $(\texttt{C}; \texttt{C}); \texttt{C} \equiv \texttt{C}; (\texttt{C}; \texttt{C})$
- $[\![\texttt{for } N \texttt{ do } C]\!] = [\![C]\!]^{[\![N]\!]}$

Observation

$\texttt{while B do C} \equiv \texttt{if B then C; while B do C}$
traduite sémantiquement comme suit :
$[\![\texttt{while B do C}]\!] \equiv [\![\texttt{if B then C; while B do C}]\!]$

Pour tout $s \in \texttt{States}$, $W(f)(s) = \texttt{ si } [\![B]\!](s) \textit{ alors } f([\![\texttt{C}]\!](s)) \textit{ sinon } s$:

- $W \in (States \nrightarrow States) \longrightarrow (States \nrightarrow States)$
- $(States \nrightarrow States, \sqsubseteq)$ est une structure partielement 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 $graphe(\perp) = \varnothing$.
  - Soit $s \in \texttt{States}$:
    $W_{i+1}(s) = \texttt{if } [\![B]\!](s) \texttt{ then } W_i([\![C]\!](s)) \texttt{ else } s \texttt{ end}$
- Soit $s \in \texttt{States}$:
  $\mu W(s) = \texttt{if } [\![B]\!](s) \texttt{ then } \mu W([\![C]\!](s)) \texttt{ else } s \texttt{ end}$

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 ;

Exemple de ACSL/Frama-c

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

Exemple de ACSL/Frama-c

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

$wp(v = e(v))(P(v)) = P(v)[v \mapsto e(v)]$ vérifie le triplet

$$\left[\begin{array}{l} \{P(v)[v \mapsto e(v)]\} \\ v := e(v); \\ \{P(v)\} \end{array}\right]$$

Intuition

- 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 : $STATES = \mathrm{X} \rightarrow \mathbb{Z}$ 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 :

Intuition

- 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 : $STATES = \mathsf{X} \rightarrow \mathbb{Z}$ 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 : $STATES = \mathsf{X} \rightarrow \mathbb{Z}$ 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 :
    $$x_0 \xrightarrow{\mathsf{P}} x_f$$

Intuition

- 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 : $STATES = \mathsf{X} \rightarrow \mathbb{Z}$ 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 :
  $$x_0 \xrightarrow{\mathsf{P}} x_f$$
- 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 : $x_0 \xrightarrow{\mathsf{P}} x_f$
  - x_0 satisfait pre : $\mathsf{pre}(x_0)$
  - x_f satisfait post : $\mathsf{post}(x_0, x_f)$
  - $\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

Vérification du contrat

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

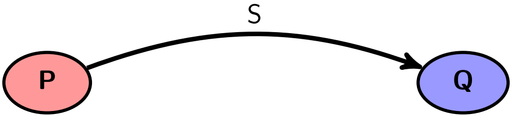
- P transforme une variable x à partir d'une valeur initiale x_0 et produisant une valeur finale x_f : $x_0 \xrightarrow{\mathsf{P}} x_f$
- x_0 satisfait pre : pre(x_0)
- x_f satisfait post : post(x_0, x_f)
- pre$(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow$ post(x_0, x_f)

```
requires pre(x0)
ensures post(x0, xf)
variables X
    begin
    0 : P0(x0, x)
    instruction0
    ...
    i : Pi(x0, x)
    ...
    instruction(f-1)
    f : Pf(x0, 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 $\ell \longrightarrow \ell'$

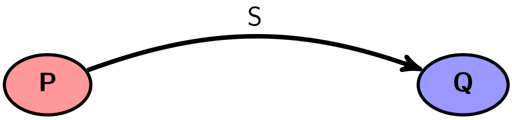
Asserted Program $\{P\}$ S $\{Q\}$

$\{P\}$ S $\{Q\}$



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

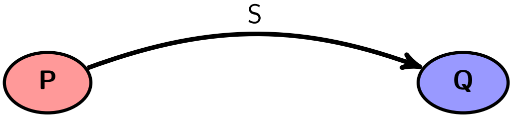
$$\{P\}\ S\ \{Q\}$$



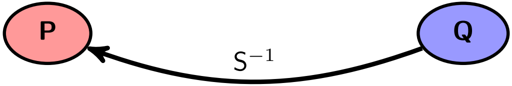
$$SP(S)(P) \Rightarrow Q$$

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

$\{P\}$ S $\{Q\}$

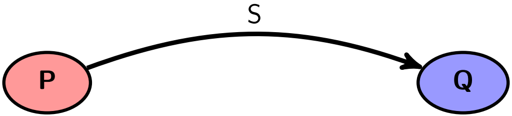


$SP(S)(P) \Rightarrow Q$

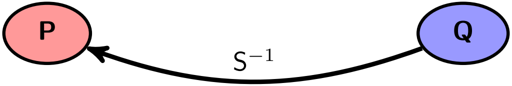


Asserted Program {P} S {Q}

$$\{P\}\ S\ \{Q\}$$



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



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

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 :

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

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

Propriétés de WP

Propriétés

- WP est une fonction monotone pour l'inclusion d'ensembles de STATES.
- $WP(S)(\varnothing) = \varnothing$
- $WP(S)(A \cap B) = WP(S)(A) \cap WP(S)(B)$
- $WP(S)(A) \cup WP(S)(B) \subseteq WP(S)(A \cup B)$
- 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,
$$WP(S) \in \mathcal{P}(STATES) \nrightarrow \mathcal{P}(STATES)$$

- $(\mathcal{P}(STATES), \subseteq)$ 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 $P(s)$.

Définition structurelle des transformateurs de prédicats

- S est une instruction de STATS.
- T est le type ou les types des variables et D est la constante ou les constantes Définie(s).
- P est un prédicat du langage Pred
- X est une variable de programme
- $E(X, D)$ (resp. $B(X, D)$) est une expression arithmétique (resp. booléenne) dépendant de X et de D.
- 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
$\mathrm{S}_1;\mathrm{S}_2$	$wp(\mathrm{S}_1)(wp(\mathrm{S}_2)(P))$
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	$\mu.(\lambda X.(B \Rightarrow wp(S)(X)) \wedge (\neg B \Rightarrow P))$

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

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 $wp(S)(P) \vee wp(S)(Q) = wp(S)(P \vee Q)$

Instruction IF

- $S \stackrel{def}{=} = \left[\begin{array}{l} \text{IF } B_1 \longrightarrow \text{S}_1 \\ \Box\ B_2 \longrightarrow \text{S}_2 \\ \ldots \\ \ \Box B_n \longrightarrow \text{S}_n \\ \text{FI} \end{array}\right]$
- $wp(S)(P) = \left[\begin{array}{l} \ \ (B_1 \vee \ldots \vee B_n) \\ \wedge(B_1 \ \Rightarrow \ wp(\text{S}_1)(P)) \\ \ldots \\ \wedge(B_n \ \Rightarrow \ wp(\text{S}_n)(P)) \end{array}\right]$

Instruction DO

- $S \stackrel{def}{=} = \left[\begin{array}{l} \text{DO } B_1 \longrightarrow \text{S}_1 \\ \Box\ B_2 \longrightarrow \text{S}_2 \\ \ldots \\ \ \Box B_n \longrightarrow \text{S}_n \\ \text{OD} \end{array}\right]$
- $BS \stackrel{def}{=} = \left[\begin{array}{l} \text{IF } B_1 \longrightarrow \text{S}_1 \\ \Box\ B_2 \longrightarrow \text{S}_2 \\ \ldots \\ \ \Box B_n \longrightarrow \text{S}_n \\ \text{FI} \end{array}\right]$
- $B \stackrel{def}{=} (B_1 \vee \ldots \vee B_n)$
- $T \stackrel{def}{=} = \left[\text{DO } B \longrightarrow \left[\begin{array}{l} \text{IF } B_1 \longrightarrow \text{S}_1 \\ \Box\ B_2 \longrightarrow \text{S}_2 \\ \ldots \\ \ \Box B_n \longrightarrow \text{S}_n \\ \text{FI} \end{array}\right] \right]$

Instruction DO

- $T \stackrel{def}{=} = \left[\text{DO } B \longrightarrow \left[\begin{array}{l} \text{IF } B_1 \longrightarrow \text{S}_1 \\ \square\ B_2 \longrightarrow \text{S}_2 \\ \ldots \\ \ \square B_n \longrightarrow \text{S}_n \\ \text{FI} \end{array} \right] \right]$
- $T \stackrel{def}{=} = [\ \text{DO } B \longrightarrow BS \text{ OD}\ ]$
- $wp(S)(P) = wp(T)(P)$
- $wp(T)(P) = \bigvee_{k \in \mathbb{N}} W_k$ où
  - Une suite d'assertions notées $W_0, \ldots, W_k \ldots$ est définie comme étant
  - $W_0(P) = \neg B \wedge P$
  - $\forall k \in \mathbb{N} : W_{k+1}(P) = W_0(P) \vee wp(BS)(W_k)$
- $wp(S)(P) = \exists k \in \mathbb{N} : W_k$

- 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

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

Itération

. . . .

⊠ 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 ;

⊠ Definition(Axiomes et règles d'inférence)

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

Axiomatisation : la Logique de Hoare

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

. . . .

. . . .

☺ 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 $\{P\}\mathbf{S}\{Q\}$.

. . . .

⊠ 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 $(\textsc{Pred}, \Rightarrow, \mathbf{false}, \mathbf{true}, \wedge, \vee)$: $(\textsc{Pred}, \Rightarrow, \mathbf{false}, \mathbf{true}, \wedge, \vee)$ 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

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

....

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

....

....

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)
- $wp(S)(Q) \equiv wlp(S)(Q) \wedge \overline{loop(S)}$

....

Définition de wlp

☒ Definition

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

☺ Property

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

Axiomatisation de la Logique de Hoare

. . . .

⊠ Definitiontriplets de Hoare

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

. . . .

☒ Definitiontriplets de Hoare

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

☒ Definition(Axiomes et règles d'inférence)

- Axiome d'affectation : $\{P(e/x)\}$**X :=E(X)**$\{P\}$.
- Axiome du saut : $\{P\}$**skip**$\{P\}$.
- Règle de composition : Si $\{P\}\mathbf{S}_1\{R\}$ et $\{R\}\mathbf{S}_2\{Q\}$, alors $\{P\}$**if B then** $\mathbf{S}_1$ **then** $\mathbf{S}_2$ **fi**$\{Q\}$.
- Si $\{P \wedge B\}\mathbf{S}_1\{Q\}$ et $\{P \wedge \neg B\}\mathbf{S}_2\{Q\}$, alors $\{P\}$**if B then** $\mathbf{S}_1$ **then** $\mathbf{S}_2$ **fi**$\{Q\}$.
- Si $\{P \wedge B\}\mathbf{S}\{P\}$, alors $\{P\}$**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 $\{P'\}\mathbf{S}\{Q'\}$.

- $\{P\}\mathbf{S}\{Q\}$
- $\forall s \in STATES.P(s) \Rightarrow wlp(S)(Q)(s)$
- $\forall s \in STATES.P(s) \Rightarrow (\forall t \in STATES : \mathcal{D}(S)(s) = t \Rightarrow Q(t))$
- $\forall s, t \in STATES.P(s) \wedge \mathcal{D}(S)(s) = t \Rightarrow Q(t))$
- 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 langae d'assertions.

Logique de Hoare Correction Totale

☒ Definitiontriplets de Hoare Correction Totale

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

$\boxtimes$ Definitiontriplets de Hoare Correction Totale

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

$\boxtimes$ Definition(Axiomes et règles d'inférence)

- Axiome d'affectation : $[P(e/x)]$**X :=E(X)**$[P]$.
- Axiome du saut : $[P]$**skip**$[P]$.
- Règle de composition : Si $[P]\mathbf{S}_1[R]$ et $[R]\mathbf{S}_2[Q]$, alors $[P]$**if B then** $\mathbf{S}_1$ **then** $\mathbf{S}_2$ **fi**$[Q]$.
- Si $[P \wedge B]\mathbf{S}_1[Q]$ et $[P \wedge \neg B]\mathbf{S}_2[Q]$, alors $[P]$**if B then** $\mathbf{S}_1$ **then** $\mathbf{S}_2$ **fi**$[Q]$.
- Si $[P(n+1)]\mathbf{S}[P(n)]$, $P(n+1) \Rightarrow b$, $P(0) \Rightarrow \neg b$, alors $[\exists n \in \mathbb{N}.P(n)]$**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 $[P]$**S**$[Q]$ est dérivé selon les règles ci-dessus, alors $P\wp(S)5Q)$.

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

Complétude

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

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

Conclusion et Perspectives

- 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.