Leçon 930 : Sémantique des langages de programmation. Exemples.

Julie Parreaux

2018 - 2019

Références pour la leçon

[1] Nielson et Nielson, *Semantics with application.*

Développements de la leçon

Équivalence petit pas et grand pas

Complétude de la logique de Hoare

Plan de la leçon

Motivation

Défense

Qu'est qu'un programme ? Un programme est une suite d'instructions qui à partir d'une entrée vérifiant une précondition donne une sortie vérifiant une postcondition. Ces deux conditions donnent alors la spécification de notre programme.

On s'intéresse alors à plusieurs problème. Le premier qui est le plus naturel est de savoir si un programme est correct, c'est-à-dire si celui-ci vérifie sa spécification (quand on lui donne une entrée vérifiant la précondition nous renvoie une sortie qui vérifie la postcondition). Le suivant est de savoir si un programme est équivalent à un autre : si on donne la même entrée aux deux programmes alors on obtient le même résultat. Le troisième problème et le dernier que nous abordons est la preuve de la correction d'une transformation de programmes qui intervient notamment lors de la compilation. On cherche alors à automatiser ces preuves en formalisant et en axiomatisant la sémantique d'un programme (en lui donnant un certain sens).

La sémantique d'un langage de programmation c'est un modèle mathématique permettant de raisonner sur le comportement attendu des programmes de ce langages. Une sémantique peut prendre des formes mathématiques variées et nous en présentants quelques unes ici.

Ce qu'en dit le jury

L'objectif est de formaliser ce qu'est un programme : introduction des sémantiques opérationnelle et dénotationnelle, dans le but de pouvoir faire des preuves de programmes, des preuves d'équivalence, des preuves de correction de traduction.

Ces notions sont typiquement introduites sur un langage de programmation (impératif) jouet. On peut tout à fait se limiter à un langage qui ne nécessite pas l'introduction des CPOs et des théorèmes de point fixe généraux. En revanche, on s'attend ici à ce que les liens entre sémantique opérationnelle et dénotationnelle soient étudiés (toujours dans le cas d'un langage jouet). Il est aussi important que la leçon présente des exemples d'utilisation des notions introduites, comme des preuves d'équivalence de programmes ou des preuves de correction de programmes.

1 Le langage IMP [1, p.7]

On travaille sur un langage jouet impératif mettant en place la notion de mémoire via les affectation et la boucle while.

1.1 Syntaxe du langage IMP

— *Définition* : Syntaxe et grammaire du langage IMP

— *Exemple* : Factorielle et Fibonacci (arbre de syntaxe)

— Parler de syntaxe concrète vs la syntaxe abstraite

Quelques remarques : le parenthésage de notre séquence d'instruction n'est pas fondamental car ils sont tous équivalents car la séquence d'instruction est associative. Contre-exemple quand ce n'est pas le cas ?

1.2 Sémantique dénotationnelle des expressions

Les sémantiques, ou plus exactement les fonctions sémantiques que l'on définies ici, sont très générique, mais à l'aide des expressions légales on les spécialise comme on le souhaite.

- *Idée* : Les sémantiques dénotationnelles modélisent le comportement par une fonction mathématique.
- *Définition* : Sémantique dénotationnelle des expressions
  - Sémantique des nombres : $\mathcal{N} : NUM \to \mathbb{Z}$ (on identifie les nombreux aux entiers).
  - Sémantique des variables (amène la question de la mémoire) : $STATE : Var \to \mathbb{Z}$
  - Sémantique des expressions arithmétiques $\mathcal{A}$ et booléennes $\mathcal{B}$ (indépendance à la syntaxe + expressivité de nos booléens : on peut exprimer toutes les fonctions booléennes) .
- *Remarque* : On suppose *State* totale (toutes les variables sont définies et on les initialise à 0 ; même si elle pourrait être partielle , fait l'impasse sur les cas d'erreurs) + représentation avec liste finie.
- *Exemple* : $s = [x \mapsto 3]$; $\mathcal{A}[\![x+1]\!]_s = 4$ et $\mathcal{B}[\![\neg(x=1)]\!]_s = tt$.
- *Remarque* : totalité des fonctions $\mathcal{A}$ et $\mathcal{B}$ + *contre-exemple* : ajout d'une opération binaire de division.

Quelques notions : la notion de variable libre toutes les variables que l'on considères sont libres et de substitution : on souhaite décrire certaines propriétés sur nos sémantiques future : ces notions permettent de formaliser deux concepts : la portée d'une variable (lorsqu'on a des fonctions ou des blocs) et la dépendance syntaxique (est-on atteint par la substitution)). Ces notions ne sont pas nécessaires car on suppose que toutes les variables sont libres et on n'utilise pas la notion de substitution.

2 Sémantiques opérationnelles

Idée : Les sémantiques opérationnelles modélisent le comportement par un système de transitions dont les règles $\langle S, s\rangle \to s'$ modélisent le fait "l'exécution de S sur l'état s termine dans l'état s'".

2.1 Sémantique opérationnelle à grand pas (sémantique naturelle) [1, p.20]

Définition de la sémantique
- *Définition* : règles inductives
  - *Définition* : d'arbre de dérivation (Notion de terminaison ?)
  - *Exemple* : factorielle

Fonction sémantique
- *Proposition* : sémantique déterministe ($\Rightarrow$ induction sur la hauteur de l'arbre ; $\Leftarrow$ évident)
- *Application* : IMP est déterministe (Induction sur la structure des instructions)
- *Contre-exemple* : instruction choose et IMP est non-déterministe
- *Définition* : fonction sémantique (fonction car déterministe) (on s'en sert pour montrer l'équivalence de programme)

Propriétés de notre sémantique
- *Définition* : Équivalence de langage

— *Exemple* while b do S est sémantiquement équivalent à if b then $(S;$ while b do $S)$ else skip (utile dans le DEV 1) et $S_1;(S_2;S_3)$ et $(S_1;S_2);S_3$ sont sémantiquement équivalents

— *Contre-exemple* : $x := 3; x := x \oplus 1$ n'est pas équivalent à $x := x \oplus 1; x := 3$

Extension du langage repeat until ou for
Preuve de programme [1, p.206]

Limite : La sémantique à grand pas ne permet pas d'exprimer les interruptions de programmes du à la division par 0 par exemple.

2.2 Sémantique opérationnelle à petit pas (sémantique structurelle) [1, p.33]

La sémantique à petit pas ne traite de la terminaison que par la fonction sémantique qui n'est pas toujours facile à manipuler. La sémantique à petit pas vient combler ce manque.
Idée : La sémantique à petits pas est plus fine que la précédente : elle se concentre sur les étapes de l'exécution (affectations et tests).

Définition de la sémantique — *Définition* : règles inductives

— *Définition* : séquence de dérivation Notion de terminaison avec des séquences finies ou non ?

— *Exemple* : factorielle

— Structure horizontale qui représente l'évolution temporelle de notre calcul et une structure verticale qui représente les sous calcul que nous devons effectuer.

Fonction sémantique — *Proposition* : Langage déterministe sous la sémantique à petits pas

— *Application* : IMP est déterministe (induction sur k, longueur de la séquence)

— *Définition* : fonction sémantique on s'en sert pour montrer l'équivalence de programme

Propriétés de notre sémantique — *Proposition* : Propriétés de comportement de la sémantique à grand pas sur la séquence (décomposition du nombre de chemin et indépendance de l'exécution sur les actions suivantes) (utiles pour le DEV 1)

— *Contre-exemple* : $< S_1;S_2,s > \Rightarrow^* < S_2,s' >$ n'implique pas nécessairement $< S_1,s > \Rightarrow^* s'$ (on en dérive un principe d'induction sur le nombre de pas dans notre séquence).

— *Définition* : Équivalence de langage

— *Exemples* while b do S est sémantiquement équivalent à if b then $(S;$ while b do $S)$ else skip ; $S_1;(S_2;S_3)$ et $(S_1;S_2);S_3$ sont sémantiquement équivalent

— *Contre-exemple* : $S_1;S_2$ n'est pas équivalent à $S_2;S_1$.

Extension du langage repeat until ou for
Preuve de programme [1, p.206]

2.3 Équivalence des deux sémantiques

On a défini deux sémantiques opérationnelles permettant de décrire le sens d'un programme, on voudrait maintenant savoir si l'une d'entre elles est plus expressive. En réalité, les deux sémantiques que nous avons définies sont équivalentes : elles expriment la même chose.

— *Théorème fondamental* : Équivalence petits pas et grands pas $\forall S, \mathcal{S}_{NS}[\![S]\!] = \mathcal{S}_{SOS}[\![S]\!]$ DEV

— *Corollaire* : L'exécution d'une instruction à partir de n'importe quel état termine dans une sémantique si et seulement si elle termine dans l'autre. Le théorème fondamental exprime cette propriétés.

3 Sémantique dénotationnelle [1, p.91]

— *Idée* : On définie la sémantique dénotationnelle pour les instructions du langage IMP (modélisation par les fonctions compositionnelles). On utilise alors la théorie du point fixe afin de définir la sémantique du while. Elle nous donne le résultat sans forcément expliquer comment on calcul en décrivant une relation entre les entrées et les sorties.

— *Remarque* : On commence par définir la sémantique pour l'ensemble des instructions. Puis, on montre que cette sémantique est bien définie en détaillant la théorie du point fixe.

3.1 Sémantique dénotationnelle

Définition de la sémantique — *Définition* : Règles : introduction de la fonction F pour le while on essaye (et on réussit) à faire passer l'intuition de pourquoi on en a besoin (car elle est uniquement compositionnelle, il nous faut donc des sous-terme strict ce que la boucle while ne peut nous garantir) et de la complexité de répondre à cette question (le nombre de choix possibles pour une instruction données car les contraintes sont uniquement sur les instructions qui terminent) ; on met le type de Fix qui se lit le plus petit points fixes.

— *Exemple* : instruction simple avec deux points fixes possible (plusieurs fonctions points fixes possibles)

Propriété de la sémantique — *Définition* : Équivalence de langage

— *Exemple* : S et S; skip sont sémantiquement équivalents

Exemples de transformation de programme

3.2 Théorie du point fixe

Permet de montrer que $FixF$ est bien définie et donc que notre sémantique est bien définie

— *Définitions* : ordre partiel (exemple : $State \hookrightarrow State$) et chaîne

— *Exemple* : $g_n s = \begin{cases} undef & \text{si } x > n \\ s[x \mapsto 1] & \text{si } 0 \leq x \leq n \\ s & sinon \end{cases}$

— *Définition* : CCPO

— *Proposition* : $\perp$ est la borne inférieure de $State \hookrightarrow State$

— *Définition* : Fonctions monotones et continues

— *Proposition* : $F(g)$ est continue

— *Théorème* : Kleene (point fixe)

— *Conséquence* : Bonne définition de la sémantique

3.3 Équivalence des sémantiques

— *Théorème* : Équivalence avec la sémantique opérationnelle à petits pas

— *Corollaire* : Équivalence avec la sémantique opérationnelle à grands pas

4 Sémantique axiomatique (Hoare partiel) [1, p.213]

Afin de faciliter la preuve automatiser, nous allons définir une logique particulière qui va représenter la sémantique.

— *Idée* : Cette sémantique facilite la preuve de programmes et son automatisation. On se base sur les systèmes de déduction en logique que nous sommes capable de bien manipuler. (correction partielle + terminaison = correction totale)

— *Correction partielle* : on donne des garanties que si le programme termine.

4.1 Triplet de Hoare

On définit la syntaxe et la sémantique de cette logique. Dans le Nielson [1], on définie la logique de Hoare comme une sur-logique : on met n'importe quelle logique dans ces prédicat. La définition syntaxe, sémantique, déduction est alors un peu bancale : il faut au moins en avoir conscience.

— *Définition* : Syntaxe des prédicats

— *Définition* : Sémantique des prédicats

— *Définition* : triplet de Hoare

— *Exemple* : triplet pour la factorielle

— *Définition* : validité On la définie avec la sémantique à grands pas mais on pourrait le faire avec n'importe quelle autre (par l'équivalence).

— *Définition* : équivalence de programme

— *Exemple* : programme équivalent

4.2 Logique de Hoare partielle

On présente ici le système de déduction de cette logique : cette sémantique.

— *Définition* : Règles d'inférence de la logique

— *Exemple* : Preuve d'un programme simple

— *Définition* : Arbre de preuve

— *Exemple* : Preuve d'un programme simple

— *Proposition* : pour tout S et P, on a $\vdash_p \{P\}S\{\mathsf{True}\}$

— *Définition* : Prouvé sémantiquement

— *Exemple* : $S;\mathsf{skip}$ et S prouvé sémantiquement

4.3 Correction et complétude

La logique que nous avons définie ne fait pas n'importe quoi.

— *Théorème* : Correction de la logique
— *Définition* : Précondition la plus faible wlp
— *Lemmes* : propriétés sur cette précondition
— *Théorème* : Complétude de la logique DEV

5 Ouverture

— Les sémantiques offrent de nombreuses sémantiques.
  — Certification d'un compilateur
  — Vérification de la sécurité d'un programme
— D'autres sémantiques : sémantique à continuation pour la sémantique opérationnelle à petit pas.

Quelques notions importantes

Logique de Hoare

La correction de la logique de Hoare partielle La logique de Hoare partielle est également correcte. Nous allons donc montrer cette correction.

Théorème. *La logique de Hoare partielle est correcte. Pour toute formule partielle* $\{P\}\ S\ \{Q\}$*, on a :* $\vdash_p \{P\}\ S\ \{Q\}$ *implique* $\models_p \{P\}\ S\ \{Q\}$.

Démonstration. On raisonne par induction sur l'arbre de preuve de $\vdash_p \{P\}\ S\ \{Q\}$.

Cas $[ass_p]$ Soient $s, s' \in State$ tels que $\langle x := a, s\rangle \to s'$ et soit P un prédicat tel que $(P[x \mapsto \mathcal{A}[\![a]\!]])s = tt$. Montrons que $Ps' = tt$. Par $[ass_{NS}]$, on a $s' = s[x \mapsto \mathcal{A}[\![a]\!]_s]$. Comme $(P[x \mapsto \mathcal{A}[\![a]\!]])s = tt$, on a $P(s[x \mapsto \mathcal{A}[\![a]\!]_s]) = tt$. Donc $Ps' = tt$. D'où la propriété pour $[ass_p]$.

Cas $[skip_p]$ Immédiat par la règle $[skip_{NS}]$

Cas $[comp_p]$ Soient P, Q, R trois prédicats et S_1, S_2 deux instructions tels qu'on ait $\models_p \{P\}\ S_1\ \{Q\}$ et $\models_p \{Q\}\ S_2\ \{R\}$. Montrons que $\models_p \{P\}\ S_1; S_2\ \{R\}$. Soient $s, s'' \in State$ tels que $Ps = tt$ et $\langle S_1; S_2, s\rangle \to s''$. Par la règle $[comp_{NS}]$, on a l'existence de $s' \in State$ tel que $\langle S_1, s\rangle \to s'$ et $\langle S_2, s'\rangle \to s''$. Comme $\langle S_1, s\rangle \to s'$, $Ps = tt$ et $\models_p \{P\}\ S_1\ \{Q\}$, on a $Qs' = tt$. De même, on montre que $Rs'' = tt$. D'où la propriété pour $[comp_p]$.

Cas $[if_p]$ Soient P, Q deux prédicats et S_1, S_2 deux instructions tels qu'on ait $\models_p \{\mathcal{B}[\![b]\!] \wedge P\}\ S_1\ \{Q\}$ et $\models_p \{\neg\mathcal{B}[\![b]\!] \wedge P\}\ S_2\ \{Q\}$. Montrons que $\models_p \{P\}$ if b then S_1 else S_2 $\{Q\}$. Soient $s, s' \in State$ tels que $Ps = tt$ et $\langle$if b then S_1 else $S_2, s\rangle \to s'$. On distingue deux cas :
  — Si $\mathcal{B}[\![b]\!] = tt$, alors $(\mathcal{B}[\![b]\!] \wedge P)s = tt$. Par la règle $[if_{NS}]$, on a $\langle S_1, s\rangle \to s'$. Comme, $\models_p \{\mathcal{B}[\![b]\!] \wedge P\}\ S_1\ \{Q\}$, on a $Qs' = tt$.
  — Si $\mathcal{B}[\![b]\!] = ff$, on raisonne de manière analogue.

Cas $[while_p]$ Soient P un prédicat et S une instruction tels qu'on ait $\models_p \{\mathcal{B}[\![b]\!] \wedge P\}\ S\ \{P\}$. Montrons que $\models_p \{P\}$ while b do S $\{\neg\mathcal{B}[\![b]\!] \wedge P\}$. Soient $s, s'' \in State$ tels que $Ps = tt$ et $\langle$while b do $S, s\rangle \to s''$. Montrons que $(\neg\mathcal{B}[\![b]\!] \wedge P)s'' = tt$. On raisonne par induction sur la dérivation de l'arbre.
  — Si $\mathcal{B}[\![b]\!]_s = ff$, alors, par la règle $[while_{NS}]$, $s'' = s$ donc $(\neg\mathcal{B}[\![b]\!] \wedge P)s'' = tt$.
  — Si $\mathcal{B}[\![b]\!]_s = tt$, alors il existe $s' \in State$ tel que $\langle S, s\rangle \to s'$ et $\langle$while b do $S, s'\rangle \to s''$. Comme $(\mathcal{B}[\![b]\!] \wedge P)s = tt$, par application de l'hypothèse $\models_p \{\mathcal{B}[\![b]\!] \wedge P\}\ S\ \{P\}$, $Ps' = tt$. On applique alors l'hypothèse d'induction sur la dérivation $\langle$while b do $S, s'\rangle \to s''$ qui donne $(\neg\mathcal{B}[\![b]\!] \wedge P)s'' = tt$.

Cas $[cons_p]$ Soient P, Q, P', Q' quatre prédicats et S une instruction tels qu'on ait $\models_p \{P'\}\ S\ \{Q'\}$ et $P \Rightarrow P'$ ainsi que $Q' \Rightarrow Q$. Montrons que $\models_p \{P\}\ S\ \{Q\}$. Soient $s, s' \in State$ tels que $Ps = tt$ et $\langle S, s\rangle \to s'$. Comme $ps = tt$ et $P \Rightarrow P'$, on a $P's = tt$. De plus, par hypothèse $Q's' = tt$ et $Q' \Rightarrow Q$, donc $Qs = tt$. Donc $\models_p \{P\}\ S\ \{Q\}$.

D'où la correction. □

Logique de Hoare La logique de Hoare partielle [1, p.221] permet de garantir des propriétés sur le programme uniquement si celui-ci termine. On définit les triplets de Hoare qui sont alors une formule dans cette logique puis les formules étendues qui nous permettent de définir les règles d'inférences.

Définition. Un triplet de Hoare est la donnée d'une précondition P, d'une instruction S et d'une postcondition Q. On le note $\{P\}\ S\ \{Q\}$.

Définition. Comme P et Q sont des formules, on définit le langage des formules étendues suivant.

— $P = P_1 \wedge P_2$ si $\forall s \in State$, Ps si et seulement si $P_1 s$ et $P_2 s$.

— $P = P_1 \vee P_2$ si $\forall s \in State$, Ps si et seulement si $P_1 s$ ou $P_2 s$.

— $P = \neg P_1$ si $\forall s \in State$, Ps si et seulement si $\neg(P_1 s)$.

— $P = P_1[x \mapsto \mathcal{A}[\![a]\!]]$ si $\forall s \in State$, Ps si et seulement si $P_1\ (s[x \mapsto \mathcal{A}[\![a]\!]_s])$.

— $P = P_1 \Rightarrow P_2$ si $\forall s \in State$, Ps implique $P_2 s$.

Définition. On définit la logique de Hoare pour les instructions du langage IMP à l'aide des règles d'interférence. Soit S une instruction et pour toute précondition P, P' et postcondition Q, Q', on a :

$[skip_p]$ $\quad \{P\}\ \mathsf{skip}\ \{P\}$ $\qquad$ $[ass_p]$ $\quad \{P[x \mapsto \mathcal{A}[\![a]\!]]\}\ x := a\ \{P\}$

$[comp_p]$ $\quad \dfrac{\{P\}\ S_1\ \{Q\} \qquad \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1;S_2\ \{R\}}$ $\qquad$ $[if_p]$ $\quad \dfrac{\{\mathcal{B}[\![b]\!] \wedge P\}\ S_1\ \{Q\} \qquad \{\neg\mathcal{B}[\![b]\!] \wedge P\}\ S_2\ \{Q\}}{\{P\}\ \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \{Q\}}$

$[while_p]$ $\quad \dfrac{\{\mathcal{B}[\![b]\!] \wedge P\}\ S\ \{P\}}{\{P\}\ \ \mathsf{while}\ b\ \mathsf{do}\ S\ \{\neg\mathcal{B}[\![b]\!] \wedge P\}}$ $\qquad$ $[cons_p]$ $\quad \dfrac{\{P'\}\ S\ \{Q'\}}{\{P\}\ S\ \{Q\}}$ si $P \Rightarrow P'$ et $Q' \Rightarrow Q$

Définition. On notera $\vdash_p \{P\}\ S\ \{Q\}$ s'il existe une preuve donnée par un arbre d'interférence tel que $\{P\}\ S\ \{Q\}$ soit à la racine et que toutes ses feuilles sont des axiomes.

Proposition. *Pour toute instruction S et propriété P, on a* $\vdash_p \{P\}\ S\ \ \{True\}$.

Démonstration. On applique la règle $[\text{CONS}_p]$ avec $True \Rightarrow Q$ qui est toujours vraie. □

Définition. Deux instructions S_1 et S_2 sont sémantiquement équivalente si et seulement si $\forall P, Q, \vdash_p \{P\}\ S_1\ \{Q\}$ est équivalent à $\vdash_p \{P\}\ S_2\ \{Q\}$.

Définition. (Validité pour la sémantique à grands pas) $\models_p \{P\}\ S\ \{Q\}$ si et seulement si $\forall s \in State$ tel que $Ps = tt$, si $\langle S, s\rangle \to s'$ alors $Qs' = tt$.

Les sémantiques opérationnelles

Sémantique opérationnelle à grands pas La sémantique opérationnelle à grands pas [1, p.20] est une sémantique permettant de décrire comment on calcul à l'aide de grandes étapes (dans une boucle par exemple, on ne détaille pas toutes les exécutions mais seulement le résultat si elle termine).

Définition. On définit la sémantique à grands pas des instructions du langage IMP à l'aide de règle du type $\langle S, s\rangle \to s'$ avec $s, s' \in State$.

$[skip_{NS}]$ $\langle \mathsf{skip}, s\rangle \to s$ $\quad$ $[ass_{NS}]$ $\langle x := a, s\rangle \to s\,[x \mapsto \mathcal{A}[\![a]\!]_s]$

$[comp_{NS}]$ $\dfrac{\langle S_1,s\rangle \to s' \quad \langle S_2,s'\rangle \to s''}{\langle S_1;S_2,s\rangle \to s''}$

$[if^{tt}_{NS}]$ $\dfrac{\langle S_1,s\rangle \to s'}{\langle \text{ if } b \text{ then } S_1 \text{ else } S_2,s\rangle \to s'}$ si $\mathcal{B}[\![b]\!]_s = tt$ $\quad$ $[if^{ff}_{NS}]$ $\dfrac{\langle S_2,s\rangle \to s'}{\langle \text{ if } b \text{ then } S_1 \text{ else } S_2,s\rangle \to s'}$ si $\mathcal{B}[\![b]\!]_s = ff$

$[while^{tt}_{NS}]$ $\dfrac{\langle S,s\rangle \to s' \ \langle \mathsf{while}\ b\ \mathsf{do}\ S,s'\rangle \to s''}{\langle \mathsf{while}\ b\ \mathsf{do}\ S,s\rangle \to s''}$ si $\mathcal{B}[\![b]\!]_s = tt$ $\quad$ $[while^{ff}_{NS}]$ $\langle \mathsf{while}\ b\ \mathsf{do}\ S, s\rangle \to s$ si $\mathcal{B}[\![b]\!]_s = ff$

Définition. On appelle arbre de dérivation la succession d'applications des règles et des axiomes.

Définition. Le langage IMP est déterministe sous la sémantique à grands pas, on définit donc une fonction sémantique : $\mathcal{S}_{NS} : Stm \to (State \hookrightarrow State)$ par

$$\mathcal{S}_{NS}[\![S]\!]s = \begin{cases} s' & \text{si } \langle S, s\rangle \to s' \\ \mathsf{undef} & \text{sinon} \end{cases}$$

Limite : La sémantique à grands pas ne permet pas toujours d'avoir la terminaison de notre instruction.

Sémantique opérationnelle à grands pas La sémantique opérationnelle à petits pas [1, p.33] est une sémantique permettant de décrire comment on calcul à l'aide de grandes étapes. Elle permet de plus, grâce à ses petits itérations de décider si une instruction termine ou non.

Définition. On définit la sémantique à petits pas des instructions du langage IMP à l'aide de règle du type $\langle S, s\rangle \Rightarrow \gamma$ avec $s \in State$ et γ de la forma $\langle S', s'\rangle$ ou s' avec $s' \in State$ et $S' \in Stm$.

$[skip_{SOS}]$ $\langle \mathsf{skip}, s\rangle \Rightarrow s$ $\quad$ $[ass_{SOS}]$ $\langle x := a, s\rangle \Rightarrow s\,[x \mapsto \mathcal{A}[\![a]\!]_s]$

$[comp^1_{SOS}]$ $\dfrac{\langle S_1,s\rangle \Rightarrow \langle S_1',s'\rangle}{\langle S_1;S_2,s\rangle \Rightarrow \langle S_1';S_2,s'\rangle}$ $\quad$ $[comp^2_{SOS}]$ $\dfrac{\langle S_1,s\rangle \Rightarrow s'}{\langle S_1;S_2,s\rangle \Rightarrow \langle S_2,s'\rangle}$

$[if^{tt}_{SOS}]$ $\langle \text{ IF } b \text{ THEN } S_1 \text{ ELSE } S_2, s\rangle \Rightarrow \langle S_1, s'\rangle$ si $\mathcal{B}[\![b]\!]_s = tt$

$[if^{ff}_{SOS}]$ $\langle \text{ IF } b \text{ THEN } S_1 \text{ ELSE } S_2, s\rangle \Rightarrow \langle S_2, s'\rangle$ si $\mathcal{B}[\![b]\!]_s = ff$

$[while_{SOS}]$ $\langle \mathsf{while}\ b\ \mathsf{do}\ S, s\rangle \to \langle \text{ IF } b \text{ THEN } (S; \mathsf{while}\ b\ \mathsf{do}\ S) \text{ ELSE } \mathsf{skip}, s\rangle$

Définition. Une séquence de dérivation pour $S \in Stm$ et $s \in State$ est soit une séquence finie $\gamma_0 \Rightarrow \cdots \Rightarrow \gamma_k$ avec $\gamma_0 = \langle S, s\rangle$ et γ_k une configuration finale ; soit une séquence infinie $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \ldots$ avec $\gamma_0 = \langle S, s\rangle$.

Définition. Le langage IMP est déterministe sous la sémantique à petits pas, on définit donc une fonction sémantique : $\mathcal{S}_{SOS} : Stm \to (State \hookrightarrow State)$ par

$$\mathcal{S}_{SOSS}[\![S]\!]s = \begin{cases} s' & \text{si } \langle S, s\rangle \Rightarrow^* s' \\ \mathsf{undef} & \text{sinon} \end{cases}$$

Références

[1] H. R. Nielson and F. Nielson. *Semantics with application*. Springer, 2007.