Sémantique élémentaire des langages: evaluation paresseuse

Didier Buchs

Université de Genève

9 avril 2019

Evaluation paresseuse (syn. évaluation par besoin)

Une expression n'a pas de raison d'être évalué immédiatement (eager semantics vs lazy semantics).
Ceci augmente l'espace de définition des programmes et permet de décrire les mécanismes d'entrée-sortie fonctionellement (stream).

$$if\ x > 0\ then\ r := 3/x\ else\ r := 0$$

n'est pas equivalent a :

$$func\ f(b, x, y)$$
$$\{switch\ b\ do$$
$$true : return\ x$$
$$false : return\ y\}$$

$$r := f(x > 0, 3/x, 0)$$

Evaluation paresseuse : principe

Doit être évalué si une décision doit être prise, sinon rien n'est calculé.

Règles paresseuses :

- affectation
- et indirectement l'appel de fonction (ressemble à l'affectation)

Règles immédiates :

- condition dans if then else
- condition dans while

Comment : Il faut étendre les substitutions aux expressions.

Evaluation paresseuse d'une instruction : affectation

Definition (Sémantique d'évaluation : Règle affectation)

$e \in ExprVar_V$ et $v \in V$, $S \in Subs$

$$Rlaffectation : \frac{e \Longrightarrow_S e'}{S \vdash v := e \Longrightarrow_I S/[v = e']}$$

Avec

- $e \Longrightarrow_S e'$ simplifie les expressions en fonction de la substitution S.
- Nous avons $Dom(S) \cap Dom(e') = \emptyset$.
- Indispensable pour garder trace de l'état correct des variables.
- Ne fonctionne pas avec des effets de bord .

Evaluation paresseuse d'une instruction : réduction

Definition (Expression reduction)

$$\frac{e_1 \Longrightarrow_S e'_1, \ldots, e_n \Longrightarrow_S e'_n}{f(e_1, \ldots, e_n) \Longrightarrow_S f(e'_1, \ldots, e'_n)}$$

$$\frac{v \notin Dom(S)}{v \Longrightarrow_S v} \qquad \frac{}{v \Longrightarrow_{S/[v=e]} e}$$

$$\frac{}{n \Longrightarrow_S n} \qquad \frac{e \Longrightarrow_S e'', e' \Longrightarrow_S e'''}{e + e' \Longrightarrow_S e'' + e'''}$$

idem autres opérations

Evaluation paresseuse d'une instruction : if then else

Definition (Sémantique d'évaluation : Règles IFTHENELSE)

$p, p' \in Bloc_V$ et $r \in Rel_V$, $S, S' \in Subs$

$$RIFTHEN : \frac{S \vdash r, S \vdash p \Longrightarrow_B S'}{S \vdash \textit{if } r \textit{ then } p \textit{ else } p' \Longrightarrow_I S'}$$

$$RIFELSE : \frac{S \not\vdash r, S \vdash p' \Longrightarrow_B S'}{S \vdash \textit{if } r \textit{ then } p \textit{ else } p' \Longrightarrow_I S'}$$

En fait rien à changer, c'est la règle de calcul de la relation qui doit imposer le choix de la branche à évaluer.

Satisfaction d'une relation

Definition (Sémantique d'évaluation : Règle $<$)

$e, e' \in ExprVar_V$, $n, n' \in \mathbb{N}$, $S \in Subs$

$$R<: \frac{S \vdash e \Longrightarrow n, S \vdash e' \Longrightarrow n', n <_{\mathbb{N}} n'}{S \vdash e < e'}$$

Cette règle reste 'eager'.

Idem pour les autres relations, la non satisfaction se note $S \nvdash e < e'$

Evaluation paresseuse d'un programme

Definition (Sémantique d'évaluation d'un programme)

$p \in Bloc_V$, $S, S', S'' \in Subs$

$$\frac{S \vdash p \Longrightarrow_B S'', S'' \Longrightarrow S'}{S \vdash p \Longrightarrow_P S'}$$

Definition (Sémantique d'évaluation)

$i \in Instr_V$ et $p \in Bloc_V$, $S, S', S'' \in Subs$

$$\text{R Subst vide} : \frac{}{\epsilon \Longrightarrow \epsilon}$$

$$RSubst : \frac{S \Longrightarrow S', e \Longrightarrow n}{S/[v = e] \Longrightarrow S'/[v = n]}$$

Propriété de la sémantique 'lazy'

- La relation $\Rightarrow$ est deterministe.
- La relation $\Rightarrow$ 'lazy' produit un résultat que la relation 'eager' produit.
- La relation $\Rightarrow$ 'lazy' produit plus de résultats que la relation 'eager' ne produit (fonction partielle).

Démonstration :
?

Résumé : comment définir une sémantique

- Définir une syntaxe :
  - Définir les éléments de base : variables (X), fonctions (F), ...
  - Donner les noms des domaines syntaxiques (ensembles) : $F, X, Expr_{F,X}$
  - Définition des domaines syntaxiques : inductivement ou avec domaine prédéfini des termes $T_{OP(D)}$
- Définir un domaine d'interprétation sémantique :
  - Par exemple :
    - pour les expressions : Entiers naturels ,
    - pile pour expressions dans machine abstraite
    - état de la mémoire pour les instructions
  - substitutions si il y a des variables
- Definir le contexte global d'évaluation (F et S) pour chaque catégorie syntaxique
- Définir les relations d'évaluation sémantique pour chaque catégorie syntaxique
- Construire les règles d'inférence des relations d'évaluation

Exercice

sachant qu'un langage de manipulation binaire à la syntaxe abstraite suivante :

$exp = exp + exp$	union bit par bit
$exp = bin$	nombre binaires
$exp => exp$	shift right
$exp =< exp$	shift left
$bin = bin\ digit \vert digit$	ex : 1010101
$digit = 0 \vert 1$	

Donner une sémantique par règle d'induction de telle manière que l'expression (les parenthèses lèvent les ambigutés) $(> 110) + 1$ s'évalue en 11
$(> 110) + 1 \Longrightarrow 11$

Exercice :

Langage avec événements :

Idée : modéliser le déplacement des robots avec la prise en compte de capteurs

Syntaxe (Event driven) :

- ajouter des instructions pour la prise en compte des capteurs
- indiquer ce que fait le robot en l'absence d'événement capteur

Syntaxe (State driven) :

- ajouter des garde sur l'état de variables 'capteur'
- indiquer ce que fait le robot en fonction de l'état

Sémantique :

- Introduire une état suplémentaire indiquant qu'un capteur a été touché

Preuve :

- faire une preuve de parcours en fixant une description de l'environnement en parallèle

Langage avec événements : Syntaxe (Event driven) :

- ajouter des instructions pour la prise en compte des capteurs
- indiquer ce que fait le robot en l'absence d'événement capteur

Idée : modéliser le déplacement des robots avec la prise en compte de capteurs $< c, d, b >$ et d'événements.

- c, d comme avant, coordonnée et direction
- b occurence de l'événement $\{t, f\}$.

Langage avec événements : Syntaxe (Event driven) :

$$\frac{< c, d, t >, q \Rightarrow < c', d', b' >, q \in Prog}{< c, d, f > \rightarrow^{ev}_{Prog} < c', d', b' >}$$

$$\frac{< c, d, f >, q \Rightarrow < c', d', b' >, q \in Prog}{< c, d, f > \rightarrow^{\not ev}_{Prog} < c', d', b' >}$$

Utilise la semantique du 'state driven', 'q' est le programme réactif contrôlant les mouvements du robot.

Langage avec événements : Syntaxe (State driven) :

Idée : modéliser le déplacement des robots avec la prise en compte de capteurs modifiant le booleen b de l'etat : $< c, d, b >$.

- ajouter des gardes sur l'état de variables 'capteur'
- indiquer ce que fait le robot en fonction de l'état

$$\frac{< c, d >, p \Rightarrow < c', d' >}{< c, d, t >, \mathit{if}\ \ \mathit{hit}\ \ \mathit{then}\ \ p \Rightarrow < c', d', f >}$$

$$\frac{}{< c, d, f >, \mathit{if}\ \ \mathit{hit}\ \ \mathit{then}\ \ p \Rightarrow < c, d, f >}$$

$$\frac{< c, d >, p \Rightarrow < c', d' >}{< c, d, f >, \mathit{if}\ \ \mathit{not}\ \ \mathit{hit}\ \ \mathit{then}\ \ p \Rightarrow < c', d', f >}$$

$$\frac{}{< c, d, t >, \mathit{if}\ \ \mathit{not}\ \ \mathit{hit}\ \ \mathit{then}\ \ p \Rightarrow < c, d, f >}$$

Langage avec événements : Preuve de comportement

Idée : modéliser l'environnement avec les contraintes imposées, telle que une boite fixant les 'murs' détecté par le robot.

- ajouter les murs
- lier le robot aux murs ! !
- faire évoluer le modèle et vérifier la position finale.

$$\frac{(x + \pi_x(d)) < \mathit{length} \wedge (y + \pi_y(d)) < \mathit{height}}{< x, y > \rightarrow (d, f) < x + \pi_x(d), y + \pi_y(d) >}$$

$$\frac{(x + \pi_x(d)) \geq \mathit{length} \vee (y + \pi_y(d)) \geq \mathit{height}}{< x, y > \rightarrow (d, t) < x, y >}$$

Langage avec événements : produit de composants

Idée : prendre deux composants pour en produire un nouveau selon des règles de composition.

- événements de T_1 combiné avec des événements de T_2 produisant des événements de T_3
- règles : e_3 *if* e_1 & e_2, $e_1 \in T_1$, $e_2 \in T_2$, $e_3 \in T_3$
- le nouveau système de transition est construit par la règle :

$$\frac{< c, d, b > \rightarrow_{Prog} < c', d', b', >, < x, y > \rightarrow (d, b') < x', y' >}{< c, d, b, x, y > \rightarrow < c', d', b', x', y' >}$$

Conclusion

- Induction
- Different genre de sémantiques opérationelles
- Langage complet
- Généralisation à n'importe quel langage