

Sémantique élémentaire des langages: evaluation paresseuse

Didier Buchs

Université de Genève

26 mars 2018

1/18



Evaluation paresseuse (syn. évaluation par besoin)

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

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

Règles paresseuses :

- affectation
- et indirectement l'appel de fonction

Règles immédiates :

- condition dans if then else
- condition dans while

Il faut étendre les substitutions aux expressions.

(lors du passage de paramètre)

if then else
while

rep
(passage de paramètre)

$$V := Q \quad S \vdash V := e \Rightarrow S / V = n$$

eval

2/18

Evaluation paresseuse d'une instruction : affectation

Definition (Sémantique d'évaluation : Règle affectation)

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

$$Rlaffection : \frac{e \xrightarrow{S} e'}{S \vdash v := e \Rightarrow, S/[v = e']}$$

replace les variables
de e par les valeurs
commes de S

Avec

- $e \xrightarrow{S} e'$ 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 .

⚠️ valable pour langage
fonctionnels purs !

3/18

Evaluation paresseuse d'une instruction : réduction

Definition (Expression reduction)

$$\frac{e_1 \xrightarrow{S} e'_1, \dots, e_n \xrightarrow{S} e'_n}{f(e_1, \dots, e_n) \xrightarrow{S} f(e'_1, \dots, e'_n)}$$

$$\frac{\nu \notin Dom(S)}{\nu \xrightarrow{S} \nu}$$

$$n \xrightarrow{S} n$$

$$\frac{\nu \xrightarrow{S/[\nu=e]} e}{e \xrightarrow{S} e'', e' \xrightarrow{S} e'''}$$

$$e + e' \xrightarrow{S} e'' + e'''$$

idem autres opérations

Evaluation paresseuse d'une instruction : if then else

Definition (Sémantique d'évaluation : Règles IFTHENELSE)

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

$$RIFTHEN : \frac{S \vdash r, S \vdash p \Rightarrow_B S'}{S \vdash \text{if } r \text{ then } p \text{ else } p' \Rightarrow_I S'}$$
$$RIFELSE : \frac{S \not\vdash r, S \vdash p' \Rightarrow_B S'}{S \vdash \text{if } r \text{ then } p \text{ else } p' \Rightarrow_I S'}$$

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 $<$)

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

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

Cette règle reste 'eager'.

Idem pour les autres relations, la non satisfaction se note

$S \not\vdash e < e'$

Evaluation paresseuse d'un programme

Definition (Sémantique d'évaluation d'un programme)

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

$$\frac{S \vdash p \Rightarrow_B S'', S'' \Rightarrow S'}{S \vdash p \Rightarrow_P S'}$$

Definition (Sémantique d'évaluation)

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

$$RSubst : \frac{\begin{array}{c} S \Rightarrow S', e \xrightarrow{\quad} n \\ \hline S/[v = e] \Rightarrow S'/[v = n] \end{array}}{RSubst}$$

evalut des
expressions
des substitutions

$$S = \{ x = 1+2, y = 3*4, z = 1 \}$$

$$S' = S / [z = 1]$$

$$S' = \{ x = 1+2, y = 3*4 \} \overline{\begin{array}{l} 3 \Rightarrow 3 \\ 4 \Rightarrow 4 \end{array}} \quad \overline{\begin{array}{l} 1+2 \Rightarrow 3 \\ \varepsilon \Rightarrow \varepsilon \end{array}} \quad \overline{\begin{array}{l} 2 \Rightarrow 2 \\ \varepsilon \Rightarrow \varepsilon \end{array}}$$

$$\overline{3*4 \Rightarrow 12} \quad \overline{\varepsilon(x=1+2) \Rightarrow \varepsilon/x=3}$$

$$\overline{1 \Rightarrow 1}$$

$$S'' \setminus (y = 3*4)$$

$$\overbrace{S'' \setminus (y = 12)}$$

$$\overline{S' / [z = 1] \Rightarrow \{ x = 3, y = 12, z = 1 \}}$$

Propriété de la sémantique 'lazy'

- La relation \Rightarrow est déterministe.
- 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 :

?

8/18

```
    | x := 0
    | if false then y := x / 0 else g := x
for f (a, b)
    if false then y := a else y := b
    return y
f(x / 0, x)
```

The diagram illustrates a function $f(a, b)$ with two definitions. The first definition is eager and returns a value immediately. The second definition is lazy and returns a thunk. The eager path leads to an exception, while the lazy path leads to a correct result.

Résumé : comment définir une sémantique

Syntaxe

- 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
- Définir 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

(Syntaxe abstraite)

Interpr.

Sémantique computationnelle
= small step
? →
? ==>
big step
semantics
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
$bin = digitbin digit$	ex : 1010101
$digit = 0 1$	

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

$$(> 110) + 1 \implies 11$$

Exercice :

exp
bin
digit

$$\frac{\boxed{0} \in \text{digit}}{b \in \text{bin}} \quad \frac{d \in \text{digit}}{\boxed{bd} \in \text{bin}}$$

$$\frac{e_1, e_2 \in \text{exp}}{\boxed{e_1 + e_2} \in \text{exp}}$$

$$\frac{e \in \text{exp}}{\boxed{< e} \in \text{exp}}$$

$$\frac{\boxed{1} \in \text{digit}}{\boxed{de} \in \text{digit}} \quad \frac{\boxed{d} \in \text{bin}}{e \in \text{exp}}$$

$$\frac{}{\boxed{> e} \in \text{exp}}$$

$$\frac{b \in \text{bin}}{\boxed{b} \in \text{exp}}$$

Syntaxe

Domaine Sémantique

Nombre bin

$$\Rightarrow_{\text{exp}} \subseteq \text{Exp} \times \text{Bin}$$

$$\overline{b \Rightarrow b}$$

$$\frac{e \Rightarrow b \quad d}{\triangleright e \Rightarrow b}$$

$$\frac{e \Rightarrow d}{\triangleright e \Rightarrow 0}$$

$$\frac{\vdash e \Rightarrow b}{\triangleleft e \Rightarrow b_0}$$

cas inducit

$$\frac{e \Rightarrow bd \quad e' \Rightarrow b'd'}{e + e' \Rightarrow b''d''}$$

$d + d' = d''$

? $b + b' = b''$

gauche plus grande

$$\frac{e \Rightarrow bd \quad e' \Rightarrow d', \quad d + d' = d''}{e + e' \Rightarrow bd''}$$

droite plus grande

$$\frac{e \Rightarrow d \quad e' \Rightarrow bd' \quad d + d' = d''}{e + e' \Rightarrow bd''}$$

1 digit
terminant avec

$$\frac{e \Rightarrow d \quad e' \Rightarrow d', \quad d'' = d+d'}{e + e' \Rightarrow d''}$$

$$\overline{0+0=0}$$

$$\overline{0+1=1}$$

$$\overline{1+0=1}$$

$$\overline{1+1=1}$$

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) :

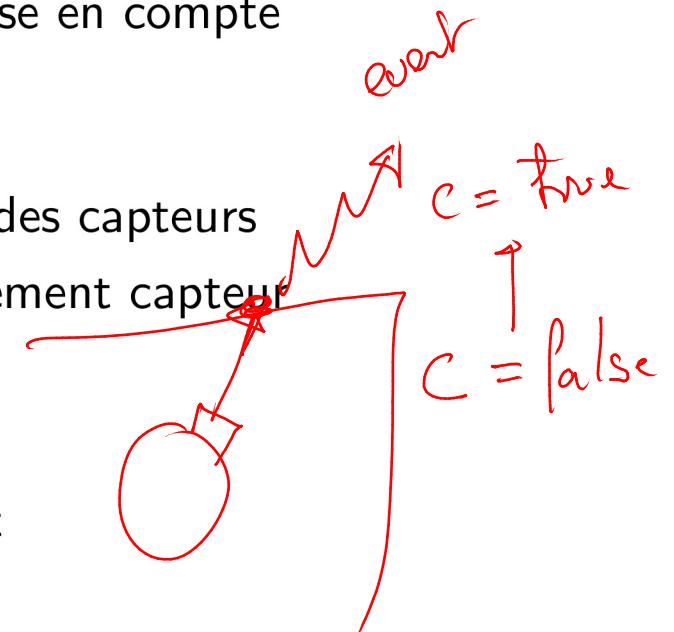
- ajouter des garde sur l'état de variables 'capteur'
- indiquer ce que fait le robot en fonction de l'état

Sémantique :

- Introduire une état supplémentaire 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 $\langle c, d, b \rangle$ et d'événements.

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

Langage avec événements : Syntaxe (Event driven) :

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

$$\frac{< c, d, f >, q \Rightarrow < c', d', b' >, q \in \text{Prog}}{< c, d, f > \xrightarrow{\text{ev}}_{\text{Prog}}^{} < c', d', b' >}$$

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'état : $\langle c, d, b \rangle$.

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

$$\frac{\begin{array}{c} \langle c, d \rangle, p \Rightarrow \langle c', d' \rangle \\ \hline \langle c, d, t \rangle, \text{if hit then } p \Rightarrow \langle c', d', f \rangle \end{array}}{\langle c, d, f \rangle, \text{if hit then } p \Rightarrow \langle c, d, f \rangle}$$

*capteur lié
à l'action
du capteur*

$$\frac{\begin{array}{c} \langle c, d \rangle, p \Rightarrow \langle c', d' \rangle \\ \hline \langle c, d, f \rangle, \text{if not hit then } p \Rightarrow \langle c', d', f \rangle \end{array}}{\langle c, d, t \rangle, \text{if not hit then } p \Rightarrow \langle c, d, f \rangle}$$

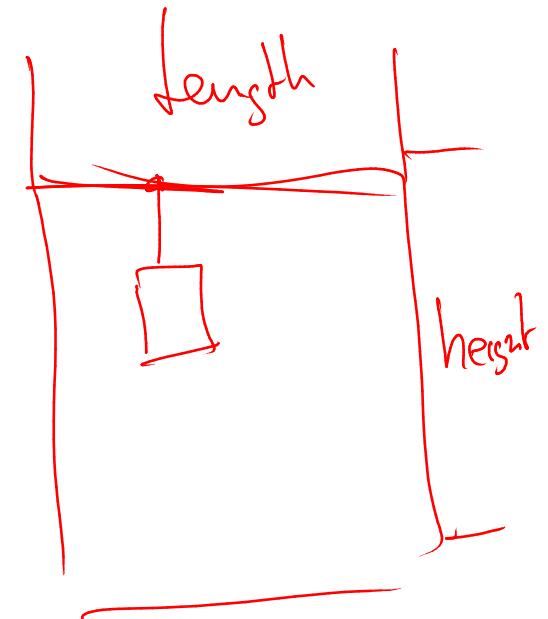
Langage avec événements : Preuve de comportement

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.

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

medizin (x) /
Medizin



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 : $e_3 \text{ if } e_1 \& e_2, e_1 \in T_1, e_2 \in T_2, e_3 \in T_3$
- 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 opérationnelles
- Langage complet
- Généralisation à n'importe quel langage

data race
conditio

