Sémantique

Fabienne Carrier & Laurent Mounier & Catherine Parent

Université Grenoble Alpes

27 novembre 2019

Plan

Le problème (1/4)

Soit le langage PATCHWORK défini par :

$V_t = \{c, t, (,), \sharp, +, *\}$

$V_n = \{M, SM, SSM, P\}$

$M \rightarrow M \sharp SM \mid SM$

$SM \rightarrow SM + SSM \mid SSM$

$SSM \rightarrow *SSM \mid P$

$P \rightarrow c \mid t \mid (M)$

Remarques :

Priorités croissantes : $\sharp$, $+$, $*$

Associtivité à gauche : $\sharp$, $+$

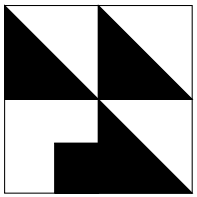
Ce langage décrit des motifs géométriques construits :

- avec des carrés primitifs c et t
- en concaténant deux motifs avec l'opérateur $\sharp$
- en superposant deux motifs avec l'opérateur $+$
- en effectuant une rotation de 90^o dans le sens trigonométrique avec l'opérateur $*$

Le problème (2/4)

Considérons les carrés primitifs :

La phrase $(c + t) \sharp (t + t)$ est le motif :



Le problème (3/4)

$M \rightarrow M \sharp SM \mid SM$
$SM \rightarrow SM + SSM \mid SSM$
$SSM \rightarrow *SSM \mid P$
$P \rightarrow c \mid t \mid (M)$

Précisons le langage : à tout motif on associe une hauteur et une largeur

La phrase $(c\ +\ t) \sharp (t\ +\ t)$ a pour hauteur 2 et largeur 2.

La phrase $c\ +\ c \sharp t$ correcte syntaxiquement doit être rejetée

Le problème (4/4)

Contraintes :

- concaténation de deux motifs de même hauteur
- superposition de de deux motifs de même largeur

L'ajoût de ces contraintes ne peut être décrit par une grammaire hors-contexte

Autre formalisme nécessaire : description rigoureuse (le plus possible), utile pour le développeur du compilateur et pour l'utilisateur du langage

Sémantique statique : introduction

Déconnecter la question de l'analyse syntaxique (reconnaissance) de celle de la spécification de contraintes contextuelles (i.e. sémantique statique)

Une fois l'analyse syntaxique réalisée, une phrase correcte peut être représentée par un arbre abstrait et des tables

Un formalisme possible : règles de semantique opérationnelle

Le langage `while` (version 1)

Syntaxe abstraite d'un langage impératif simple

P	: :=	D C
D	: :=	var x t | D ; D
C	: :=	x := E | C ; C | si E alors C sinon C | tantque E C
E	: :=	n | b | x | E + E | E * E | E et E | not E

- *D*, *C* et *E* désignent respectivement une *déclaration*, une *commande* et une *expression*.
- *x* désigne un identificateur : $x \in$ Noms
- *t* désigne un type, élément de l'ensemble Type = {Entier, Booleen, Void }.
- *n* désigne une constante entière, *b* une constante booléenne.

Spécification de la cohérence des types dans le langage `while`

Définition d'un système de transition qui permet :

- d'associer un environnement aux déclarations

  environnement = fonction partielle qui associe à certains noms un type

- d'associer un type aux commandes et aux expressions

Notations

- Environnement : Env = Noms $\longrightarrow$ Type
  On note : $\rho \in$ Env

- Dom(ρ) = $\{x \ / \ \rho(x)$ défini $\}$

- On note $< e, \rho > \xrightarrow{e} t$ le fait que l'expression e analysée dans l'environnement ρ a le type t

  Les configurations du système de transition pour les expressions sont soit dans Exp $\times$ Env, soit dans Type

  On écrit des règles de déduction permettant de passer d'une configuration à une autre

Sémantique statique pour les expressions

$$< n, \rho > \xrightarrow{e} \mathit{Entier} \qquad < b, \rho > \xrightarrow{e} \mathit{Booleen}$$

$$\frac{x \in \mathit{Dom}(\rho)}{< x, \rho > \xrightarrow{e} \rho(x)}$$

$$\frac{< E1, \rho > \xrightarrow{e} \mathit{Entier} \quad < E2, \rho > \xrightarrow{e} \mathit{Entier}}{< E1 + E2, \rho > \xrightarrow{e} \mathit{Entier}}$$

$$\frac{< E1, \rho > \xrightarrow{e} \mathit{Booleen} \quad < E2, \rho > \xrightarrow{e} \mathit{Booleen}}{< E1 \ \mathit{et} \ E2, \rho > \xrightarrow{e} \mathit{Booleen}}$$

$$\frac{< E1, \rho > \xrightarrow{e} \mathit{Entier} \quad < E2, \rho > \xrightarrow{e} \mathit{Entier}}{< E1 * E2, \rho > \xrightarrow{e} \mathit{Entier}}$$

$$\frac{< E, \rho > \xrightarrow{e} \mathit{Booleen}}{< \mathit{not} \ E, \rho > \xrightarrow{e} \mathit{Booleen}}$$

Exemple de preuve

Soit $\rho = [x \mapsto \mathit{Entier}]$

Type de $3 + x * 7$?

$$\dfrac{\dfrac{< 3, \rho > \xrightarrow{e} \mathit{Entier} \quad \dfrac{< x, \rho > \xrightarrow{e} \rho(x) = \mathit{Entier} \quad < 7, \rho > \xrightarrow{e} \mathit{Entier}}{x * 7 \xrightarrow{e} \mathit{Entier}}}{< 3 + x * 7, \rho > \xrightarrow{e} \mathit{Entier}}$$

Exemple de preuve

$$\frac{< 3, [] > \xrightarrow{e} \textit{Entier} \quad < \textit{vrai}, [] > \xrightarrow{e} \textit{Booleen}}{< 3 + \textit{vrai}, [] > \xrightarrow{e} ???}$$

Aucune règle ne s'applique : ERREUR

On ne définit que ce qui est correct

Toute phrase ne vérifiant pas les règles est rejetée

Configurations pour la sémantique statique

Configurations pour les expressions : $\mathcal{C}_E \subseteq (E \times Env) \cup Types$
Nous avons utilisé la relation : $\xrightarrow{e} \subseteq \mathcal{C}_E \times \mathcal{C}_E$

Configurations pour les commandes : $\mathcal{C}_C \subseteq (C \times Env) \cup Types$
Soit la relation : $\xrightarrow{c} \subseteq \mathcal{C}_C \times \mathcal{C}_C$

Configurations pour les déclarations : $\mathcal{C}_D \subseteq D \cup Env$
Soit la relation : $\xrightarrow{d} \subseteq \mathcal{C}_D \times \mathcal{C}_D$

Sémantique statique pour les commandes

$$\frac{x \in Dom(\rho) \quad \rho(x) = t \quad < E, \rho > \xrightarrow{e} t}{< x := E, \rho > \xrightarrow{c} Void}$$

$$\frac{< C1, \rho > \xrightarrow{c} Void \quad < C2, \rho > \xrightarrow{c} Void}{< C1; C2, \rho > \xrightarrow{c} Void}$$

$$\frac{< E, \rho > \xrightarrow{e} Booleen \quad < C1, \rho > \xrightarrow{c} Void \quad < C2, \rho > \xrightarrow{c} Void}{< si\ E\ alors\ C1\ sinon\ C2, \rho > \xrightarrow{c} Void}$$

$$\frac{< E, \rho > \xrightarrow{e} Booleen \quad < C, \rho > \xrightarrow{c} Void}{< tantque\ E\ C, \rho > \xrightarrow{c} Void}$$

Sémantique statique pour les déclarations

$$var\ x\ t \xrightarrow{\mathrm{d}} [x \longmapsto t]$$

$$\frac{D1 \xrightarrow{\mathrm{d}} \rho 1 \quad D2 \xrightarrow{\mathrm{d}} \rho 2 \quad Dom(\rho 1) \cap Dom(\rho 2) = \emptyset}{D1\ ;\ D2 \xrightarrow{\mathrm{d}} \rho 1 \cup \rho 2}$$

Sémantique statique : correction d'un programme

$$\frac{D \xrightarrow{d} \rho \quad < C, \rho > \xrightarrow{c} Void}{D\ C \longrightarrow Void}$$

Le langage `while` (version 2)

Langage étendu pour permettre :

- l'imbrication de blocs (niveaux de déclarations différents)
- la définition de procédures

P	: :=	B
B	: :=	debut D C fin
D	: :=	var x t | proc p C | proc p (y t) C | D ; D
C	: :=	x := E | C ; C | si E alors C sinon C | tantque E C
		| B | call p | call p E
E	: :=	n | b | x | E + E | E * E | E et E | not E

Exemple 1

```
debut
   var x entier
   var y entier
   x := 1; y:=7
   debut                    |
      var x booleen         | --> B2
      x := y < 10           |
   fin                      |
   x := x + y
fin
```

Exemple 2

```
debut
   var x entier
   proc p x := 3
   var y entier
   proc q (u entier)
   debut
      var x entier
      x := 5
      y := u + x
   fin
   call p
   x := 9
   call p
   call q (3)
fin
```

Extension 1 : blocs imbriqués

Un bloc est plongé dans un environnement ρ

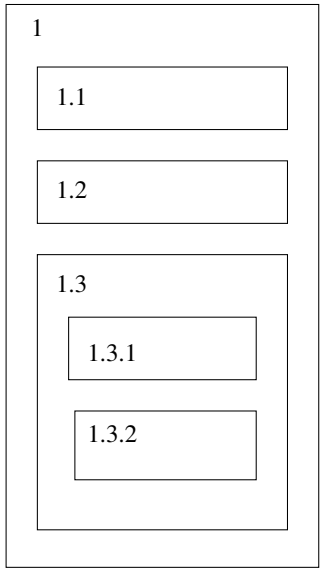
$\Longrightarrow$ correction des types d'un bloc étant donné ρ
ρ représente l'ensemble des variables déclarées dans le (ou les) environnements(s) englobant(s)

Problème : gérer la portée des noms
i.e. rattacher à chaque utilisation la bonne définition

Règle (habituelle) : utiliser la déclaration la plus imbriquée

Blocs imbriqués : solution environnement global

Identifier les environnements eux-mêmes, par exemple par une chaîne d'entiers



L'environnement devient une fonction $\rho \in (noms, \mathbb{N}^*) \rightarrow Type$
Chercher x avec le plus grand préfixe tel que $(x$, ce préfixe$) \in Dom(\rho)$

Blocs imbriqués : solution environnements locaux

Gérer un environnement local à chaque bloc qui masque l'environnement englobant dans le cas de re-définitions

Formellement

$$\begin{aligned}(\rho_1[\rho_2])(x) &= \rho_2(x) \textit{ si } x \in \textit{Dom}(\rho_2) \\ &= \rho_1(x) \textit{ sinon}\end{aligned}$$

Sémantique d'un bloc et du programme

Configurations pour les blocs : $\mathcal{C}_B \subseteq (B \times Env) \cup Types$

$$\frac{D \xrightarrow{\text{d}} \rho_l \quad < C, \rho[\rho_l] > \xrightarrow{\text{c}} Void}{< D\ ;\ C, \rho > \xrightarrow{\text{b}} Void}$$

$$\frac{< B, \emptyset > \xrightarrow{\text{b}} Void}{(P =) B \longrightarrow Void}$$

Preuve exemple 1

$<$var x entier ; var y entier ; x := 1 ; y := 7 ; B2 ; x := x + y, []$>$ $\xrightarrow{c}$? ? ?

soit ρ_0 = [x $\mapsto$ entier, y $\mapsto$ entier]

$<$x := 1 ; y := 7 ; B2 ; x := x + y, $\rho_0 >$ $\xrightarrow{c}$? ? ?
$<$x := 1, $\rho_0 >$ $\xrightarrow{c}$ Void
$<$y := 7, $\rho_0 >$ $\xrightarrow{c}$ Void
$<$B2, $\rho_0 >$ $\xrightarrow{c}$? ? ?
$<$x := x + y, $\rho_0 >$ $\xrightarrow{c}$ Void

$<$var x : booleen x := y $<$ 10, $\rho_0 >$ $\xrightarrow{c}$? ? ?
$<$ x := y $<$ 10, ρ_0[x$\mapsto$booleen]$>$ $\xrightarrow{c}$? ? ?
$<$ x := y $<$ 10, [x$\mapsto$booleen, y$\mapsto$entier] $>$ $\xrightarrow{c}$ Void

Extension 2 : procédures

Définition d'une procédure :

- pas de double définition dans un même bloc
- correction du corps de la procédure
  une procédure étant définie dans un environnement les déclarations doivent maintenant être analysées dans un environnement
  $\Longrightarrow$ modifier les configurations pour les déclarations

Utilisation d'une procédure $\mathtt{p}$:

- $\mathtt{p}$ définie dans l'environnement courant (ou un environnement englobant)
- si $\mathtt{p}$ a un paramètre, compatibilité des types du paramètre effectif et du paramètre formel
  $\Longrightarrow$ étendre la définition des types (pour les procédures)

Extension : sémantique statique des déclarations (1/2)

Type_de_Base = $\{$ Entier, Booléen, Void $\}$
Type = Type_de_Base $\cup$ $\{$ $\{$proc$\}\times$Type_de_Base $\}$
Configurations pour les déclarations : $C_D \subseteq (D \times \ Env) \cup \ Env$

$$< var\ x\ t, \rho > \xrightarrow{d} [x \mapsto t]$$

$$\frac{< C, \rho > \xrightarrow{c} Void}{< proc\ p\ C, \rho > \xrightarrow{d} [p \mapsto (Proc, Void)]}$$

$$\frac{< C, \rho[y \mapsto t] > \xrightarrow{c} Void}{< proc\ p\ (y\ t)\ C, \rho > \xrightarrow{d} [p \mapsto (Proc, t)]}$$

Extension : sémantique statique des déclarations (2/2)

$$\frac{< D1, \rho > \xrightarrow{d} \rho 1 \quad < D2, \rho > \xrightarrow{d} \rho 2 \quad Dom(\rho 1) \cap Dom(\rho 2) = \emptyset}{< D1 \ ; \ D2, \rho > \xrightarrow{d} \rho 1 \cup \rho 2}$$

ne permet pas l'utilisation d'une procédure déclarée précédemment dans le même environnement

Modifier : $< D2, \rho[\rho 1] > \xrightarrow{d} \rho 2$

Extension : sémantique statique des commandes

$$\frac{x \in Dom(\rho) \quad \rho(x) = (Proc, Void)}{< call\ x, \rho > \xrightarrow{c} Void}$$

$$\frac{x \in Dom(\rho) \quad \rho(x) = (Proc, t_1) \quad < e, \rho > \xrightarrow{e} t_2 \quad t_1 = t_2}{< call\ x\ e, \rho > \xrightarrow{c} Void}$$

$$\frac{< D, \rho > \xrightarrow{d} \rho_l \quad < C, \rho[\rho_l] > \xrightarrow{c} Void}{< D\ ;\ C, \rho > \xrightarrow{b} Void}$$

Extension : procédure récursive

Nous avons :

$$\frac{< C, \rho[y \mapsto t] > \xrightarrow{c} Void}{< proc\ p\ (y\ t)\ C, \rho > \xrightarrow{d} [p \mapsto (Proc, t)]}$$

Nouvelle règle :

$$\frac{< C, \rho[y \mapsto t, p \mapsto (proc, t)] > \xrightarrow{c} Void}{< proc\ p\ (y\ t)\ C, \rho > \xrightarrow{d} [p \mapsto (Proc, t)]}$$

Mise en œuvre

Parcours de l'arbre abstrait avec mémorisation par décoration de l'arbre et/ou construction de tables pour conserver les informations nécessaires à la suite des traitements

Chaque règle correspond à un type de noeud de l'arbre abstrait et indique quelles vérifications effectuer sur ce noeud

Gestion de l'environnement : table des symboles (globale ou locale)

La construction des environnements (parcours des déclarations) permet de stocker pour chaque nom les informations nécessaires : le type par exemple

Chaque utilisation d'un nom (`idf`) dans l'arbre abstrait est relié à son entrée dans la table des symboles

Introduction

L'objectif de la sémantique dynamique est de donner un sens à un programme

La sémantique dynamique dit ce qu'il se passe lors de l'exécution d'un programme

La sémantique dynamique fixe les règles permettant d'écrire un interpréteur ou un compilateur d'un langage

Exemple : quel est le sens d'une expression ?

Le sens d'une expression de type entier pourrait être sa valeur dans $\mathbb{Z}$

Comment trouver la valeur d'une expression ?

Trouver la valeur des sous-expressions et
appliquer l'opérateur à ces valeurs

Dans une sous-expression il peut y avoir une variable, quelle est sa valeur ?

La valeur d'une variable est stockée en mémoire

Comment représenter la mémoire et comment décrire l'accès à la valeur d'une variable ?

La sémantique dynamique va décrire la séquence des exécutions nécessaires pour trouver la valeur d'une expression

Notion d'exécution

L'état d'une exécution est constitué de :

- la partie du programme restant à exécuter
- le contenu de la mémoire

Exécution = séquence de couples (programme, état)

Etat final d'un programme : (ε, mémoire) c'est-à-dire la mémoire

Sémantique dynamique = ensemble de règles qui indiquent comment construire les séquences d'exécution

Le langage `while`

Programme décrit par sa syntaxe abstraite

P	: :=	B
B	: :=	D C
D	: :=	var x | D ; D
C	: :=	x := E | C ; C | si E alors C sinon C | tantque E C | B
E	: :=	n | vrai | faux | x | add(E, E) | and(E, E)

Les programmes sont corrects du point de vue de la sémantique statique

Mémoire et environnement

$\mathrm{Adr} = \mathbb{N}$

$\mathrm{Valeur} = \mathbb{Z} \cup \mathbb{B}$

$\mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}$

$$\mathrm{Env} = \mathrm{Noms} \longrightarrow \mathrm{Adr}$$
$$\mathrm{Mem} = \mathrm{Adr} \longrightarrow \mathrm{Valeur}$$

Notations : $\sigma \in \mathrm{Mem}$

$\eta \in \mathrm{Env}$

On se donne une fonction : $\mathrm{new\_adr} : \longrightarrow \mathrm{Adr}$

Configurations pour la sémantique dynamique

- Les déclarations permettent la construction d'un environnement η
- Les expressions sont évaluées dans un certain environnement η et la mémoire σ et délivrent une valeur
- Les commandes sont évaluées dans un certain environnement η et la mémoire σ et font évoluer la mémoire *sigma*

- pour une déclaration : $\mathcal{C}_D \subseteq D \cup Env$
- pour une expression : $\mathcal{C}_E \subseteq (E \times Env \times Mem) \cup Val$
- pour une commande : $\mathcal{C}_C \subseteq (C \times Env \times Mem) \cup Mem$

Sémantique dynamique pour les déclarations

Relation : $\xrightarrow{\mathrm{d}} \subseteq \mathcal{C}_D \times \mathcal{C}_D$

$$\frac{ad = new\_adr()}{var\ i \xrightarrow{\mathrm{d}} [i \longmapsto ad]}$$

$$\frac{d1 \xrightarrow{\mathrm{d}} \eta 1 \quad d2 \xrightarrow{\mathrm{d}} \eta 2}{d1\ ;\ d2 \xrightarrow{\mathrm{d}} \eta 1 \cup \eta 2}$$

Sémantique dynamique pour les expressions

Relation : $\xrightarrow{e} \subseteq \mathcal{C}_E \times \mathcal{C}_E$

$$< n, \eta, \sigma > \xrightarrow{e} n \qquad < x, \eta, \sigma > \xrightarrow{e} \sigma(\eta(x))$$

$$< vrai, \eta, \sigma > \xrightarrow{e} tt \qquad < faux, \eta, \sigma > \xrightarrow{e} ff$$

$$\frac{< e1, \eta, \sigma > \xrightarrow{e} v1 \quad < e2, \eta, \sigma > \xrightarrow{e} v2}{< add(e1, e2), \eta, \sigma > \xrightarrow{e} v1 + v2}$$

$$\frac{< e1, \eta, \sigma > \xrightarrow{e} b1 \quad < e2, \eta, \sigma > \xrightarrow{e} b2}{< and(e1, e2), \eta, \sigma > \xrightarrow{e} b1 \wedge b2}$$

Sémantique dynamique pour les commandes (1/2)

Relation : $\xrightarrow{c} \subseteq \mathcal{C}_C \times \mathcal{C}_C$

$$\frac{< e, \eta, \sigma > \xrightarrow{e} v}{< x := e, \eta, \sigma > \xrightarrow{c} \sigma[\eta(x) \longmapsto v]}$$

$$\frac{< c1, \eta, \sigma > \xrightarrow{c} \sigma' \quad < c2, \eta, \sigma' > \xrightarrow{c} \sigma''}{< c1; c2, \eta, \sigma > \xrightarrow{c} \sigma''}$$

$$\frac{< e, \eta, \sigma > \xrightarrow{e} tt \quad < c1, \eta, \sigma > \xrightarrow{c} \sigma'}{< \textit{si } e \textit{ alors } c1 \textit{ sinon } c2, \eta, \sigma > \xrightarrow{c} \sigma'}$$

$$\frac{< e, \eta, \sigma > \xrightarrow{e} ff \quad < c2, \eta, \sigma > \xrightarrow{c} \sigma'}{< \textit{si } e \textit{ alors } c1 \textit{ sinon } c2, \eta, \sigma > \xrightarrow{c} \sigma'}$$

Sémantique dynamique pour les commandes (2/2)

$$\frac{< e, \eta, \sigma > \xrightarrow{e} tt, \quad < c, \eta, \sigma > \xrightarrow{c} \sigma', \quad < \textit{tantque } e\ c, \eta, \sigma' > \xrightarrow{c} \sigma''}{< \textit{tantque } e\ c, \eta, \sigma > \xrightarrow{c} \sigma''}$$

$$\frac{< e, \eta, \sigma > \xrightarrow{e} ff}{< \textit{tantque } e\ c, \eta, \sigma > \xrightarrow{c} \sigma}$$

$$\frac{< b, \eta, \sigma > \xrightarrow{b} \sigma}{< b, \eta, \sigma > \xrightarrow{c} \sigma}$$

Sémantique dynamique pour les blocs et le programme

relation : $\xrightarrow{b} \subseteq \mathcal{C}_B \times \mathcal{C}_B$

$$\frac{d \xrightarrow{d} \eta_l, \quad < c, \eta[\eta_l], \sigma > \xrightarrow{c} \sigma'}{< d\ ;\ c, \eta, \sigma > \xrightarrow{b} \sigma'}$$

$$\frac{d \xrightarrow{d} \eta \quad < c, \eta, \emptyset > \xrightarrow{c} \sigma}{< d\ ;\ c > \longrightarrow \sigma}$$

Sémantique dynamique des procédures : le problème

```
var x entier
procedure p (*1*) is x:=0
procedure q is call p
procedure r is
  procedure p (*2*) is x:=1
  call q
call r
ecrire(x)
```

Scénarios d'exécution :

- le programme appelle `r` qui appelle `q` qui appelle `p (*1*)` et affiche 0. On parle de liaison statique.
- le programme appelle `r` qui appelle `q` qui appelle `p (*2*)` et affiche 1. On parle de liaison dynamique.

Sémantique des procédures : questions

Quelle “valeur” associe-t-on à une procédure ?

Comment une procédure est-elle représentée lors de son exécution ?

Procédures : liaison dynamique

A une procédure on associe les instructions représentant le corps de la procédure : $\text{Env}_P = \text{Noms} \longrightarrow \text{C}$

Les déclarations de procédures permettent de produire des environnements de procédures : $< \textit{proc p is C} > \xrightarrow{\text{p}} [p \mapsto C]$

Une commande est évaluée dans un environnement de variables, une mémoire et un environnement de procédures

$$\frac{p \in Dom(envp) \quad envp(p) = C \quad < C, \eta, \sigma, envp > \xrightarrow{\text{c}} \sigma'}{< call\ p, \eta, \sigma, envp > \xrightarrow{\text{c}} \sigma'}$$

Lors de l'exécution de C les procédures utilisées sont celles de *envp* qui est l'environnement d'appel de C

Procédures : liaison statique

Les déclarations de procédures sont analysées dans l'environnement de définition et celui-ci est conservé

$< \mathit{proc}\ p\ \mathit{is}\ C, \mathit{envp} > \xrightarrow{\mathrm{p}} [p \mapsto (C, \mathit{envp})]$

L'exécution du corps d'une procédure se fera ainsi dans son environnement de définition

$$\frac{p \in \mathit{Dom}(\mathit{envp}) \quad \mathit{envp}(p) = (C, \mathit{envp}') \quad < C, \eta, \sigma, \mathit{envp}' > \xrightarrow{\mathrm{c}} \sigma'}{< \mathit{call}\ p, \eta, \sigma, \mathit{envp} > \xrightarrow{\mathrm{c}} \sigma'}$$