ÉCHAPPEMENTS ET CONTINUATIONS EN SEMANTIQUE DÉNOTATIONNELLE

P. SALLÉ

INTRODUCTION

On se propose de définir une dénotation des instructions de saut à l'aide de la notion d'échappement et de comparer cette façon de procéder avec la technique désormais classique des continuations. La dénotation utilisant l'échappement permet :

- de définir fonctionnellement les instructions de sauts généralisées du type "ESCAPE" dans un langage à structure de bloc ayant des procédures récursives,
- de définir également de manière fonctionnelle le champ ou portée d'une étiquette lorsque celui-ci est dynamique, c'est-à-dire dépend de l'historique des définitions de cette étiquette si celle-ci est située dans une procédure récursive.

Cette dénotation :

- s'évalue selon une règle de calcul unique, l'appel par valeur, contrairement à la sémantique par continuations qui nécessite un changement de stratégie au niveau de la composition de fonction [SET 1],
- est directement traductible en λ -calcul à l'aide de ζ -règles [DUR 2], [DUR 3].

Cette dernière propriété permet d'envisager la traduction automatique de programmes comportant des instructions impératives dans un langage de réduction du type " λ -Red Langage" [BAC 1], pour une exécution efficiente sur une machine du type "réduction machine" [MAG 1], comme cela a déjà été envisagé par les langages purement applicatifs [KOS 1].

* Laboratoire "Langages et Systèmes Informatiques"
UNIVERSITE PAUL SABATIER
118, route de Narbonne
31077 - TOULOUSE CEDEX

Rappelons que tout programme peut être considéré comme une application partielle d'un espace de données dans un espace de résultats. La connaissance de cette application permet de définir précisément ce que calcule un programme dont la syntaxe est supposée correcte. La sémantique dénotationnelle est l'étude de la construction de cette application en associant une règle sémantique à chaque règle de production syntaxique du langage. Une fois l'application déterminée, le calcul de la valeur de cette application sur un élément de l'espace des données se fait alors selon les techniques classiques d'approximation définies par SCOTT et STRACHEY [SCO 1]. Ce calcul correspond à l'évaluation du programme sur les données correspondantes.

Mais si la sémantique d'un programme est relativement simple dans le cas d'un langage sans instruction de saut il n'en est pas de même lorsqu'il faut prendre en compte un transfert non séquentiel du contrôle. En effet dans le premier cas chaque instruction sera considérée comme une fonction de transition entre deux états de la mémoire et la sémantique d'une séquence d'instructions comme la composition naturelle des fonctions sémantiques de chacune des instructions. Pour prendre en compte la notion de saut il est nécessaire d'étendre le domaine et le codomaine de la fonction sémantique pour lui adjoindre un argument supplémentaire. Cet argument peut être une continuation [STR 1], [STR 2], [TEN 1], [SET 1], qu'il présente pour une instruction donnée la fonction de transition associée au reste du programme. Il peut être également un argument non fonctionnel appelé 'étiquette d'échappement' par analogie avec la structure de contrôle très puissante appelée échappement dans certains interprètes de lambda langages [GRE 1], [REY 1], [DUR 1]. Cette méthode est légèrement plus restrictive que la première et ne permet pas de rendre compte du phénomène de coroutine. Nous établissons une preuve de l'équivalence des deux procédés dans les domaines où ils sont comparables. Nous étendons ainsi un résultat semblable [JON 1] établi pour des programmes décrits en VDL dans le cas d'instructions de saut simple type 'GOTO FORTRAN' dans un langage sans procédures et sans limitations du champ d'une étiquette.

1. UN LANGAGE SIMPLE DE PROGRAMMATION : LSP

1.1. Syntaxe

Nous décrirons une version légèrement simplifiée du LSP de [ROB 2] à partir :

- du vocabulaire terminal
 $\{\text{DEBUT, FIN, LOCAL, SI, ALORS, SINON, IS, TANTQUE, FAIRE, FAIT, PROC, id, cliq, :, :=, ;}\}$

- du vocabulaire auxiliaire

$\{B, E, Y, L, S, M\}$ où S est l'axiome

- de la grammaire

$$\begin{aligned}
 S &\rightarrow \text{DEBUT } Y \text{ FIN} \\
 &\rightarrow \text{PROC } id \ Y \text{ FIN} \\
 Y &\rightarrow \text{DEBUT } Y \text{ FIN} \\
 &\rightarrow L \\
 &\rightarrow M \\
 &\rightarrow \text{etiq: } M \\
 L &\rightarrow Y; L \\
 &\rightarrow Y; L \\
 M &\rightarrow \text{TANTQUE } B \text{ FAIRE } Y \text{ FAIT} \\
 &\rightarrow \text{SI } B \text{ ALORS } Y \text{ SINON } Y \text{ IS} \\
 &\rightarrow id := E \\
 &\rightarrow \text{GOTO etiq} \\
 &\rightarrow id \\
 &\rightarrow \text{LOCAL etiq}
 \end{aligned}$$

Dans cette grammaire nous ne décrirons pas les productions associées à B expressions booléennes et E expressions arithmétiques dans la mesure où seules les instructions de contrôle nous intéressent.

1.2. Domaine de calcul, sémantique de l'affectation et des expressions arithmétiques et booléennes

Nous considérerons les cinq domaines distincts suivants :

- le domaine des Identificateurs Id
- le domaine des valeurs V (vrai, faux $\in V$)
- le domaine des expressions arithmétiques Exp
- le domaine des expressions booléennes $Bool$
- le domaine des étiquettes Etq

Nous définissons l'application $m \in Id \rightarrow V$ qui à tout identificateur fait correspondre une valeur et que nous appellerons état de la mémoire.

Notations

m A valeur de l'identificateur A

$M = Id \rightarrow V$ domaine des états mémoire

Sémantique des expressions

- arithmétiques

Elle sera définie par l'application E qui associe une valeur à toute expression en fonction d'un état mémoire

$$E \in (Exp \rightarrow \mathcal{H}) \rightarrow V \quad E \llbracket e \rrbracket m = v$$

- booléennes

De la même manière B associe à une expression booléenne la valeur vrai ou la valeur faux en fonction d'un état mémoire.

$$B \in (Bool \rightarrow \mathcal{H}) \rightarrow \{vrai, faux\} \quad B \llbracket b \rrbracket m = v$$

Sémantique de l'affectation

Soit $A := e$ ou $A \in Id$ et $e \in Exp$

La sémantique de cette affectation $S \llbracket A := e \rrbracket$ est une application de $\mathcal{H} \rightarrow \mathcal{H}$ qui à un état de mémoire m fait correspondre un état de mémoire modifié m'

$S \llbracket A := e \rrbracket m = m'$ tel que
 $m'z =$ si $z = A$ alors $E \llbracket e \rrbracket m$ sinon mz
 ce qui en adoptant les notations du λ -calcul donne

$$m' = \lambda z. \text{ si } z = A \text{ alors } E \llbracket e \rrbracket m \text{ sinon } mz$$

1.3. Sémantique naïve des structures de contrôle de LSP

Dans un langage à structure de blocs il est nécessaire du point de vue sémantique de préciser quelle est la portée des définitions internes à un bloc. Ainsi dans le domaine des structures de contrôle nous nous intéresserons au caractère de globalité des étiquettes définies dans un bloc où dans une procédure. On peut résumer les choix établis pour LSP de la manière suivante :

a) Toute étiquette e définie dans un bloc **DEBUT FIN**

- est définie dans tout bloc interne à ce bloc ne contenant pas de définition de e
- est définie dans tout bloc externe à ce bloc ne contenant pas de définition de e , si e n'est pas déclarée **LOCAL** e
- n'est pas définie à l'extérieur de la procédure contenant ce bloc

b) Toute étiquette définie dans le corps d'un **TANTQUE** est locale à ce **TANTQUE**

c) Toute étiquette définie dans les alternatives d'un **SI ALORS SINON** est locale à ces alternatives.

d) Tout GOTO vers une étiquette e non définie dans une procédure sera considérée comme un "échappement" vers cette étiquette c'est-à-dire qu'il engendre

- un retour impératif à la procédure appelante
- un GOTO de la même nature vers l'étiquette e dans cette procédure appelante.

2. SEMANTIQUE S de LSP EXPRIMEE A L'AIDE D'ECHAPPEMENTS

Cette sémantique de LSP est obtenue en considérant que toute instruction peut être représentée par une application à deux arguments. Le premier est l'état de la mémoire et le second un argument non fonctionnel appelé échappement qui est apparenté à l'opérateur J. de LANDIN [LAN 1] et à la fonction ESCAPE de REYNOLDS [REY 1]. L'échappement est également une fonction de base des langages VLISP [GRE 1] et TLISP [DUR 1].

2.1. Les domaines sémantiques

Nous introduisons le domaine des échappements qui contient une infinité dénombrable d'étiquettes

$$Ech = \{ \emptyset, e_1, e_2, \dots, e_n, \dots \}$$

où $\emptyset$ représente l'étiquette vide qui sera considérée comme l'étiquette d'échappement implicite de toute instruction non étiquetée.

D'autre part nous considérerons qu'une instruction est formée de trois composantes :

- une étiquette (éventuellement $\emptyset$)
- une composante applicative qui modifie l'état de la mémoire
- une composante impérative : saut vers une étiquette.

L'absence de saut (transfert séquentiel du contrôle) sera représenté par un saut vers l'étiquette $\emptyset$. Une instruction sera représentée dans la sémantique par une application déterminée en fonction de ces trois composantes et ayant pour argument un couple formé de :

- l'état de la mémoire m
- une étiquette d'échappement e représentant 'l'échappement en cours'.

Si l'étiquette 'd'échappement en cours' correspond à l'étiquette de l'instruction considérée la composante applicative de l'instruction est appliquée à l'état de la mémoire m qu'elle modifie en un état m' . Le résultat de l'instruction

est alors un nouveau couple formé de m' et de l'étiquette e' composante impérative de l'instruction. Si l'étiquette 'd'échappement en cours' ne correspond pas à l'étiquette de l'instruction considérée la valeur calculée est le couple inchangé formé de m et e .

Notation :

- $X = Ech \times H$
- Si $\langle e, m \rangle \in X$ alors $\pi_1 \langle e, m \rangle = e$
 $\pi_2 \langle e, m \rangle = m$
- $Prog$ domaine syntaxique des programmes

Un programme est une application de $Prog \rightarrow (H \rightarrow X)$ qui à un état mémoire m associe un couple $\langle e, m' \rangle$ tel que :

- si $e \equiv \emptyset$ le programme se termine normalement sur un état mémoire m'
- si $e \neq \emptyset$ le programme a une fin anormale et l'état de la mémoire était m' au moment où s'est produite l'exécution d'un saut vers l'étiquette e non définie ou une 'erreur' auquel est associée un code d'erreur e

Plus généralement nous considérerons que $S \in Prog(X \rightarrow X)$ et que si $P \in Prog$ alors $S[P] \langle \emptyset, m \rangle = \langle e, m' \rangle$

Une procédure est également une application de $Prog \rightarrow (X \rightarrow X)$ et si $S \in PROC$ $P \in Y \text{ FIN } I$ $\langle \emptyset, m \rangle = \langle e, m' \rangle$ alors

- si $e \equiv \emptyset$ la procédure se termine par un retour normal à la procédure appelante
- si $e \neq \emptyset$ il y a saut généralisé vers la procédure appelante comme décrit en 1.3.d.

Pour prendre en compte le champ des étiquettes dans la définition de S nous considérerons qu'à tout bloc **DEBUT** Y **FIN** et à toute procédure

PROC x **d** Y **FIN** on peut associer deux ensembles :

- $EG(Y)$ = ensemble des étiquettes globales
- $EL(Y)$ = ensemble des étiquettes locales

calculés selon les règles a, b et c du paragraphe 1.3.

Procédures récursives

Pour simplifier l'exposé les procédures que nous considérons sont sans arguments et leur appel se réduit à un identificateur. Du point de vue sémantique trois étapes sont à considérer.

- La sémantique de la définition de la procédure

$$S \llbracket \text{PROC } id \text{ } \mathcal{Y} \text{ FIN } \mathcal{D} \rrbracket = \lambda x. h x$$

où h est le plus petit point fixe de

$$H \equiv \lambda x. \text{ si } \pi_1 x \in EL(\mathcal{Y}) \text{ alors } H(S \llbracket \mathcal{Y} \mathcal{D} \rrbracket x) \text{ sinon } x$$

en remarquant que toutes les étiquettes de $\mathcal{Y}$ sont locales conformément au 1.3.a. Cette définition est associée à l'identificateur id .

- La sémantique de l'appel

$$S \llbracket id \rrbracket = id$$

Nous faisons volontairement ici une confusion entre le domaine syntaxique des identificateurs et le domaine sémantique de la fonction associée à la procédure.

- La règle de calcul au moment de l'évaluation de la procédure qui sera la suivante:

Si au cours des calculs on est amené à évaluer l'expression $id \ x$ où id représente un identificateur de procédure et x un élément de X alors il existe une définition de $\text{PROC } id \text{ } \mathcal{Y} \text{ FIN}$ et $id \ x$ s'évalue $S \llbracket \text{PROC } id \text{ } \mathcal{Y} \text{ FIN } \rrbracket x$

2.3. Règle de calcul

Le calcul de la fonction S se fait en mode d'appel par valeur comme on peut le constater sur la séquence d'instructions suivante.

$$\mathcal{Q} \equiv \text{GOTO } e ; \text{ TANTQUE } vrai \text{ FAIRE } \mathcal{Y} \text{ FAIT } ; e : \mathcal{Y}$$

En effet $S \llbracket \mathcal{Q} \rrbracket \langle \emptyset, m \rangle$ est de la forme $S_3(S_2(S_1 \langle \emptyset, m \rangle))$

$$\text{avec } S_1 \langle \emptyset, m \rangle \equiv S \llbracket \text{GOTO } e \rrbracket \langle \emptyset, m \rangle \equiv \langle e, m \rangle$$

$$S_2 \langle e, m \rangle \equiv S \llbracket \text{TANTQUE } vrai \text{ FAIRE } \mathcal{Y} \text{ FAIT } \rrbracket \langle e, m \rangle \equiv \langle e, m \rangle$$

$$S_3 \langle e, m \rangle \equiv S \llbracket e : \mathcal{Y} \rrbracket \langle e, m \rangle \equiv S \llbracket \mathcal{Y} \rrbracket \langle \emptyset, m \rangle$$

Si on considère une sémantique $\overset{1}{S}$ utilisant la technique des continuations on constate [SET 1] qu'il est nécessaire d'opérer en changement de stratégie

au niveau de la composition des instructions qui doit se faire en mode d'appel par nom. On peut le constater sur l'exemple précédent où

$$\hat{S}[\mathbb{L}](\lambda x. x) m \text{ est de la forme } \hat{S}_1(\hat{S}_2(\hat{S}_3(\lambda x. x))) m$$

avec $\hat{S}_1 \equiv \lambda c m. C_e m_1$ où $C_e \equiv \hat{S}[\mathbb{Y}]$ représente la continuation associée à l'étiquette e . $\hat{S}_1$ doit être évaluée avec la règle d'appel par nom.

Cette particularité de la sémantique S est due au fait que toute instruction $e: \mathbb{Y}$ a une sémantique de la forme $\lambda x. \text{ si } \pi_1 x \in \{e, \emptyset\} \text{ alors } \bar{S}[\mathbb{Y}] < \emptyset, \pi_2 m > \text{ sinon } x$ qui ne peut être évaluée que si l'argument est un couple de $\text{Ech} \times \mathcal{M}$. Cette propriété se retrouve dans la traduction de S en λ -calcul.

2.4. Traduction de S en λ -calcul

Le mécanisme d'échappement peut être introduit en λ -calcul à l'aide de δ -règles [DUR 2], [DUR 3]. Nous donnons un exemple :

- si $\langle e, m \rangle \equiv \lambda t. t e m$ alors $\pi_1 \equiv \lambda u. u (\lambda v w. v)$, $\pi_2 \equiv \lambda u. u (\lambda v w. w)$
- si $\{\emptyset, e_1, \dots, e_n, \dots\}$ sont des constantes alors

$$\left. \begin{array}{l} \emptyset \\ e_i \end{array} \right\} \xrightarrow{\delta} \lambda u v. u \quad \left. \begin{array}{l} \emptyset e_i \\ e_j e_i \end{array} \right\} \xrightarrow{\delta} \lambda u v. v \quad i \neq j$$

$$S[e: \mathbb{Y}] = \lambda x. (e_i (\pi_1 x)) (\bar{S}[\mathbb{Y}] x)$$

On constate aisément :

$$\begin{array}{l} S[e_i: \mathbb{Y}] < e, m \rangle \xrightarrow{\beta\delta} \bar{S}[\mathbb{Y}] < \emptyset, m \rangle \text{ pour } e \equiv \emptyset \text{ ou } e_i \\ S[e_i: \mathbb{Y}] < e_j, m \rangle \xrightarrow{\beta\delta} < e_j, m \rangle \text{ pour } i \neq j \end{array}$$

En λ -calcul quelle que soit la règle de calcul utilisée l'expression $e_i (\pi_1 x)$ ne peut être réduite que si $\pi_1 x$ est une forme normale c'est-à-dire un couple $\langle e, m \rangle$

3. EQUIVALENCE ENTRE ECHAPPEMENTS ET CONTINUATIONS

Le domaine d'application de la sémantique utilisant des échappements est plus restreint que celui de la sémantique des continuations. En effet l'échappement est inadéquat pour décrire le mécanisme des coroutines. Toutefois partout où une étiquette peut avoir une définition multiple simultanée, partout où le champ d'une étiquette est dynamique la technique des continuations doit faire intervenir la notion de pile dont le caractère fonctionnel n'est pas évident. Ceci apparaît dans la preuve de l'équivalence des deux sémantiques pour le langage LSP. Soit $\hat{S}$ la

$b = b_1 \wedge b_2$	$I_4 ; P_1$
$b = b_1 \wedge \bar{b}_2$	3 ^e définition de e $C_e \equiv I_4 ; I_5 ; I_5 ; I_7 ; I_5 ; I_2$
$b = \bar{b}_1 \wedge b_2$	GOTO e
$b = \bar{b}_1 \wedge \bar{b}_2$	$I_8 ; I_5 ; I_5 ; I_7 ; I_5 ; I_2$

On peut alors remarquer que si I_5 est une séquence d'instructions pouvant référencer l'étiquette e il est nécessaire de conserver toutes les définitions de e , donc d'empiler toutes les continuations possibles associées à e .

CONCLUSION

La sémantique par échappement permet de donner une définition simple des sauts généralisés sans la nécessité d'introduire une pile de continuations. Elle permet également de prendre en compte la notion de champ dynamique d'une étiquette. Elle présente l'avantage d'être facilement traductible dans un langage de réduction du type λ -calcul et de s'évaluer selon un mode de calcul unique, l'appel par valeur.

BIBLIOGRAPHIE

- [BAC 1] BACKUS J.
Reduction Language and Variable-free Programming,
IBM Research Report RJ 1010, IBM Research Laboratory, 1972,
San José, Californie.
- [DUR 1] DURIEUX J.L.,
T-LISP le Système LISP de Toulouse
Ecole IRIA de la Recherche, Toulouse, Mars 1978.
- [DUR 2] DURIEUX J.L. , SALLE P. ,
Application de la Notion d'Echappement à la Description des Instruc
tions de Saut.
Groupe Programmation et Langages de l'AFCEt, Bulletin n° 5, 1978.
p. 21-35.
- [DUR 3] DURIEUX J.L., SALLE P.,
L'échappement comme Sémantique des Structures de contrôle.
Actes du Congrès AFCEt TTI, Nov. 78 p. 77-87.

- [GRE 1] GREUSSAY P.,
Contribution à la Définition Interprétative et à l'Implémentation
des Lambda-Languages,
Thèse d'Etat, Paris 7, 1977.
- [JON 1] JONES C.B.
Denotational Semantics of Goto : An Exit Formulation and its Relation
to Continuations,
The Vienna Development Method : The Meta Language, Ed. D. Bjornes
and C.B. Jones, Springer Verlag, n° 61, 1978 - p. 280-304.
- [KOS 1] KOSTER A.
An Algorithm for Translating LISP Programs into Reduction Language
Programs,
IV^e International Symposium Programming, Paris, France, 1980.
- [LAN 1] LANDIN J.P.,
A Correspondence Between ALGOL 60 and CHURCH'S Lambda Notations,
ACM, Vol. 8, n° 2 et 3, 1965, p. 8-101 et 158-165.
- [MAG 1] MAGO G.,
A Network of Microprocessors to Execute Reduction Languages,
International Journal of Computer and Information Sciences, 1979,
5 et 6.
- [REY 1] REYNOLDS J.C.,
Definitional Interpreters for Higher-Orders Programming Languages.
Proc 25th ACM National Conférence, 1972, p. 717-740.
- [ROB 1] ROBINET B.,
Sémantique des Structures de Contrôle,
RAIRO Informatique Théorique, Vol. 11, n° 3, 1977, p. 213-236
- [ROB 2] ROBINET B.,
De la Sémantique Dénotationnelle.
UNIVERSITE PARIS VII n° 79-3, 1979.
- [SCO 1] SCOTT D., STRACHEY C.,
Towards a Mathematical Semantics for Computer Language,
Proc. Symposium on Computer and Automata. Polytechnique Institute
or Brooklyn, 1971, p. 437-453.

- [STR 1] STRACHEY C.,
A Mathematical Semantics Wich can Deal With Full Jumps,
Seminaire IRIA, Théorie des Algorithmes, des Langages et de la
Programmation, 1973, 1973, p. 175-191
- [STR 2] STRACHEY C., WADSWORTH C.,
Continuations : A Mathematical Semantics for Handling Full
Jumps,
Oxford University, 1974.
- [TEN 1] TENNENT R.D.,
The Denotational Semantics of Programming Languages.
Comm. ACM, Vol. 19, n° 8, 1976, p. 437-452.
- [SET 1] SETHI R.
Semantics of Computer Programs : Overview of Language Definition
Methods
Ecole de Sémantique, Ecoles des Mines - Sofia Antipolis (Oct. 77)