SEMANTIQUE D'UN LANGAGE ADAPTE A LA CONSTRUCTION DEDUCTIVE DES PROGRAMMES

Par
Pierre LESCANNE

Centre de Recherche en Informatique
de Nancy

78-R-015

I- DESCRIPTION DU LANGAGE SNOOPY ET DU LANGAGE SNOOPISSIMO

SNOOPY est un langage de description d'énoncés algorithm utilisé dans l'enseignement de la programmation et l'analyse des programmes.

Un énoncé est un ensemble de définitions de suites le pl souvent finies. Ces définitions sont soit simples c'est-à-dire sous forme d'une équation, soit conditionnelles, soit itératives simples (la longueur de la suite est connue à priori), soit itératives avec arrêt, elles font alors appel à des modules qui sont des ensembles de nouvelles définitions. Certaines définition ont des effets annexes notamment l'écriture de résultats intermédiaires. Ecrire ces définitions fait appel à une méthodologie appelée "méthode de programmation déductive" décrite par C. Pair (PAI 77) (voir aussi (BEL 77) qui est un cours destiné aux étudiants). Une fois ces définitions écrites, l'ordonnancement de celles-ci permet de transformer cet ensemble de définitions en un algorithme séquentiel dont on peut déduire un programme (PAI 77)(HUC 77). Ces caractéristiques font de SNOOPY un langage proche parent de LUCID.(ASH 77).

Une sémantique de ce langage a déjà été proposée par J.P. Finance (FIN 77) ; elle associe à un énoncé SNOOPY un ensembl de formules. Dans la suite nous allons présenter une méthode de définition de la sémantique qui associe à un énoncé, un ensemble de fonctions mathématiques présentées par des équations récursive Pour ne pas entrer dans des détails techniques, nous allons nous restreindre à un sous-langage que nous appellerons SNOOPISSIMO, qui ne contient que deux variétés de définitions : les définition simples et les définitions itératives avec arrêt. De plus, nous n'accepterons que les types entiers et booléens. Nous ne considér

pas les définitions avec effet annexe, nous supposerons que le module d'initialisation d'une définition itérative ne contient que les définitions explicites, celle qui servent à donner la valeur des premiers éléments des suites.

En fait, pour donner la sémantique d'un énoncé, il est inutile de tenir compte de l'ordonnancement qui n'est qu'une aide pour le passage à un programme,
nous ne considèrerons dans la suite que des énoncés en SNOOPISSIM sans ordonnancement.

1.1. Présentation informelle de SNOOPISSIMO

Nous allons donner tout de suite une présentation informe de SNOOPISSIMO. Un énoncé est un module, c'est-à-dire un "ensembl de définitions, ce module a toujours pour nom PRINCIPAL.

Les objets définis par un énoncé SNOOPISSIMO sont des constantes, des suites à un indice et plus généralement des suite multi-indicées (cas des itérations imbriquées). Il est commode de considérer uniquement des suites en assimilant une constante à une suite à zéro indice.

Les définitions simples sont de la forme x= exp où x est un identificateur de suite et exp est une expression construi à partir des identificateurs de suites, des opérations sur les suites, elles-mêmes définies à partir des opérations sur les éléments de ces suites et à partir de deux opérateurs. a) et % qui construisent respectivement la suite décalée d'un indice et une suite avec un indice de moins.

Exemple : som = @ som + @ score

arrêt = som eq - 1

m = max (% som, @ m)

Les définitions itératives sont de la forme

$x_1,\ldots,x_k$: INIT ; jqa arret repeter MOD

où INIT est un module qui ne contient que des définitions explicites arrêt est un identificateur de suite de booléens et MOD est un module. Crosso modo, $x_1,\ldots x_k$ sont des suites initialisées dans le module INIT appelé module d'initialisation s'arrêtant quand la suite arrêt prend la valeur vrai pour la première fois et dont la définition itérative se trouve dans le module MOD, appelé module d'itération. Ce qu'on cherche à fournir c'est le dernier élément des suites ainsi définies.

1.2. Un exemple de construction d'énoncé

Nous allons utiliser la méthode de programmation déductive pour construire un énoncé très simple.

Problème du score maximum : Le résultat de ce problème est le maximum, sur l'ensemble des équipes, des scores de chaque équipe, obtenus comme la somme des scores individuels. Les scores individuels sont donnés équipe par équipe en une suite; la séparation entre deux équipes est un score -1 ; la fin des données est marquée par un score -2.

Construction de l'énoncé : Un énoncé se présente en quatre colonnes:

- une colonne (la troisième) contient les définitions explicites et les définitions itératives c'est l'énoncé SNOOPY proprement dit
- une colonne (la première) contient les définitions implicites en langue naturelle de chaque identificateur de suite, c'est l'outil essentiel du programmeur ; quand il rencontre un identificateur non encore défini, il en donne une définition implicite, puis ensuite il en donne une définition explicite.

- une colonne (la quatrième) contient les définitions implicites des modules en langue naturelle.
- une colonne (la deuxième) contient des entiers qui constituent l'ordonnancement des définitions pour pouvoir en déduire un programme.

La première chose à faire est de donner une définition implicite du résultat puis une définition explicite de ce résultat (figure 1).

PRINCIPAL			
res(entier):maximum des scores des équipes m(entier):suite des maximum des scores des équipes		res=%m m:INIMAX ; jqa arretmax repeter MAX	

figure 1 : définition du résultat

La définition de m a introduit deux modules INIMAX dont il faut donner l'énoncé (figure 2).

PRINCIPAL			
res(entier):maximum des scores des équipes m(entier):suite des maximum des scores des équipes		res=%m m: INIMAX ; jqa arretmax repeter MAX	MAX : calcule le maximum des scores des équipes déjà envisagées INIMAX : initialise m à 0 et arrêt à faux
MAX			
som(entier):suite des scores d'une équipe i.e.cumul des scores individuels score(entier):suite des scores des individus de l'équipe		m=max(% som, ∂ m) som,score:INISOM;jqa arret repeter SOMM arretmax=% score eq-2	SOMME: lit les scores et calcule la somme par cumul INISOM:initialise la somme et l'arrêt
INIMAX			
		m=0 arretmax=faux	

Figure 2 : énoncé incomplet

La définition de som et de score a introduit deux modules dont il faut donner l'énoncé (figure 3). La figure 4 donne l'énoncé dans lequel on a fait figurer l'ordonnancement des définitions pour pouvoir en déduire un programme.

PRINCIPAL			
res(entier):maximum des scores des équipes m(entier):suite des scores des maximums des équipes		res=%m m:INIMAX;jqa arretmax repeter MAX	MAX:calcul le maximum des scores,déjà envisagés INIMAX initialise m à 0 et arret à faux
MAX			
som(entier):suite des scores d'une équipe i.e. cumul des scores individuels score(entier):suite des scores des individus de l'équipe		m=max(%som,@m) som,score:INISOM,jqa arret repeter SOMME arretmax=%score eq -2	SOMME:lit les scores et calcule la somme par cumul INISOM:initialise som et arret
INIMAX			
		m=0 arretmax = faux	
SOMME			
		som=@som+@score score=donnée score arret=(score eq-1 ou score eq-2)	
INISOM			
		score=donnée som = 0 arret=(score eq -1 ou score eq-2-	

figure 3 : énoncé complet

	PRINCIPAL
2	res=%m
1	m:INIMAX ; jqa arretmax repeter

	MAX
3	m=max(% som, m)
1	som, score : INISOM ; jqa arrêt
	repeter SOMME
2	arretmax = % score eq -2

	INI MAX
1	m=0
2	arretmax=faux

	SOMME
1	som=@ som + @ score
2	score = donnée
3	arrêt= (score eq -1 ou score eq -2)

	INI SOM
2	score = donnée
1	som = 0
3	arret = (score eq-1 ou score eq -2)

figure 4 : enoncé ordonnancé

2. OBJETS SNOOPY

Avant d'en venir à la sémantique du langage que nous étudions, nous allons parler des objets que ce langage définit. Un identificateur est associé à une "suite" d'éléments qui dépend du temps ; autrement dit, si n est un identificateur, u est associé à une fonction $\hat{u}$ de t où t est le temps ou la date, $\hat{u}(t)$ appartient à $\mathbb{N}$ ou $\mathbb{B}$, $\hat{u}(t)$ est la valeur de $\hat{u}$ à la date t. Si u intervient dans une définition du module principal, cela signifie que $\hat{u}$ est une constante, c'est-à-dire indépendant du temps, si u apparaît dans un module T_1 d'une définition itérative du module principal, $\hat{u}$ dépend du temps, t=0,1,...,n qui correspond au nombre de fois que l'on itère le module T_1 ; si u apparaît dans un module I_2 d' une définition itérative de T_1 alors $\hat{u}$ dépend de t_1 et t_2 (t_1 nombre de fois qu'est itéré le module I_1 et t_2 nombre de fois qu'est itéré le module I_2 à l'instant t_1 etc...) Ainsi dans le cas général, un identificateur u désigne une fonction $\hat{u}$ de $\mathbb{N}^k$ vers $\mathbb{B}$ ou $\mathbb{N}$.

Une manière très intuitive et terre à terre de voir les choses est la suivante : une valeur $\hat{u}(t_1,...,t_k)$ est un état de la variable au cours de sa vie ; $t_1,...t_k$ représente une date, cette date a k composants parce qu'elle est donnée avec une précision d'ordre k. Par exemple la date du 8 novembre 1971 qu'on pourrait coder (19, 71, 11, 8) est donnée avec une précision de la journée c'est-à-dire d'ordre 4 tandis que la date du 8 novembre 1971 à 11 h 32 mn 8s codée (19,71,11,8,11,32,8) a une précision de la seconde ou d'ordre 7. Une itération, c'est-à-dire une suite d'événements concernant les variables créé un nouvel ordre de grandeur dans le repérage des dates pour les variables concernées ; on peut dire que la chronologie des changements d'état de la suite s'est affinée. Ce nombre k qui est la profondeur de la définition de la suite dans l'enchaînement des modules dépend uniquement de critère syntaxiques c'est-à-dire de la position de la variable dans l'énon

3. ASPECTS SYNTAXIQUES

3.1. Profondeur des variables

Comme on vient de voir, à chaque identificateur u est associé un entier, sa profondeur $\underline{p}(u)$, qui fixe le nombre de paramètres dont dépend la suite associée û. La profondeur est obtenue grâce aux deux règles suivantes :

Régle 1 : chaque variable qui figure en partie gauche d'une définition simple (resp. d'une définition itérative) dans un module autre qu'un module d'initialisation a la profondeur de ce module (resp. a la profondeur de module augmentée de 1).

Régle 2 : La profondeur du module PRINCIPAL est zéro. La profondeur d'un module d'itération est la profondeur des variables qu'il définit, celle d'un module d'intialisation est celle du module d'itération associé diminuée de un.

Ces règles sont purement syntaxiques et nous ne détaillerons pas leur formalisation.

3.2. A propos des signes d'égalité

Dans la définition de la sémantique de SNOOPISSIMO il y a quatre signes qui symbolisent des notions d'égalité très différentes.

- le signe d'"égalité par définition" du langage SNOOPY apparaît dans les définitions explicites, immédiatement après l'identificateur de la suite que l'on définit, il est noté =
- L'opérateur, à résultat booléen, d'égalité du langage SNOOPY, il apparaît en partie gauche d'une définition simple dans une expression à résultat booléen, il est noté eg
- chaque définition a pour sémantique un certain nombre d'équations récursives, ces signes d'égalité dans les équations sont notés $\Leftarrow$

- la sémantique d'un module M est définie par une équation du type $\mathcal{S}[\![M]\!] \equiv \ldots$

3.3. Les données de l'algorithme

Certaines suites qui apparaissent dans l'énoncé ne sont pas définies par une expression mais sont des données de l'algorithme; dans ce cas la définition s'écrit

$$u = \underline{\text{donnée}}_{u}$$

3.4. Enoncé SNOOPY syntaxiquement corrects

La sémantique qui suit s'applique uniquement à des énoncés SNOOPY correctement écrits, c'est-à-dire vérifiant un certain nombre de règles syntaxiques. Pour formaliser ces règles, nous avons besoin de quelques définitions :

- une définition d apparaît dans un module M noté $d \in M$ si l'une des définitions de M est d.
- un module M' apparaît dans un autre module M' noté $M' \mathcal{Q} M$ si l'une des définitions de M est itérative et fait appel au module M' autrement dit

  $M' \mathcal{Q} M \Leftrightarrow (\exists x \exists I \exists a)(x:I \text{ jqa } a \text{ repeter } M') \in M$
- une définition ou module M utilise une expression noté $d \mathcal{U} e$ ou $M \mathcal{U} e$ si

  $M \mathcal{U} e \Leftrightarrow (\exists d) \quad d \in M \wedge d \mathcal{U} e$

  $d \mathcal{U} e \Leftrightarrow d \equiv (x=e') \wedge e \mathcal{S} e'$

  $\vee (\exists I, M \exists a) \quad d \equiv (x:I; \text{jqa } a \text{ repeter } M) \wedge (I \mathcal{U} e \vee M \mathcal{U} e)$

  $e \mathcal{S} e'$ signifie que e est une sous-expression de e'.

Ces règles s'énoncent ainsi :

- un module M n'utilise l'expression a) x que si la définition simple de x apparaît dans le module M.

  $M \mathcal{U} a) x \Rightarrow (x=e) \in M$

- un module n'utilise l'expression % x que s'il contient une définition itérative de x et réciproquement si M a une définition itérative de x alors tout autre définition qui utilise x, l'utilise sous la forme % x.

$M\,\mathcal{U}\,\%\,x \Rightarrow \exists M,I,M'\ \exists a \quad (x : I \text{ jqa } a \text{ repeter } M') \in M$

$d=(x : I \text{ jqa } a \text{ repeter } M') \wedge d \in M$

$\Rightarrow \forall e\ \forall d'(d' \in M \wedge d' \neq d \wedge d\,\mathcal{U}\,e \wedge x\,\mathcal{S}\,e \wedge x \neq e) \Rightarrow \%\,x\,\mathcal{S}\,e$

- un module utilise un identificateur x si il appartient de façon itéré à un module où figure une définition de x

x M',I, e M M (x=e) M (x:I ; jqa a repe
M") M

$M\,\mathcal{U}\,x \Rightarrow \exists M',I,M''\ \exists e\ \exists a\ M\,\alpha^*\,M' \quad [(x=e) \in M' \vee (x:I;\text{jqa a repeter } M'' \in M']$

4. SEMANTIQUE

La sémantique d'un énoncé SNOOPYSSIMO est un ensemble d'équations récursives définissant les suites dont nous avons parlé au paragraphe 3. Cet ensemble est défini lui-même par induction sur la structure de l'énoncé, c'est-à-dire à partir des définitions apparaissant dans cet énoncé.

4.1. Sémantique d'un module

Très naturellement c'est la réunion au sens ensembliste des sémantiques des définitions qui la composent :

$$\mathcal{S}[\![M]\!] \equiv \{\mathcal{S}[\![d]\!] \quad / \text{ d définition appartenant à M}\}$$

4.2. Sémantique d'une définition simple dans un module d'itération ou dans le module principal

Dans ce cas d'une définition simple est associée une seule équation récursive.

$$\mathcal{S}[\![x = exp]\!] \equiv \hat{x}(t_1,\ldots,t_k) \Leftarrow \mathcal{E}[\![exp]\!]\ (t_1,\ldots,t_k)$$

$$\mathcal{S}[\![x = \underline{\text{donnée}}_x]\!] \equiv \hat{x}(t_1,\ldots,t_k) \Leftarrow \delta_x(t_1,\ldots,t_k)$$

où δ_x est une suite dépendant de k paramètres et où $k = \underline{p}(x)$

4.3. Sémantique d'une définition simple dans un module d'initialisation.

Dans ce cas, à une définition est associée une seule équation récursive où, dans le membre gauche, le dernier paramètre est 0, cela correspond bien à l'idée qu'une initialisation définit le premier terme d'une suite.

$$\mathcal{S}[\![x = \text{exp}]\!] \equiv \hat{x}(t_1,\ldots,t_{k-1},0) \Leftarrow \mathcal{E}[\![\text{exp}]\!] \quad (t_1,\ldots,t_{k-1})$$

où $k = \underline{p}(x)$

$$\mathcal{S}[\![x = \underline{\text{donnée}}_x]\!] \equiv \hat{x}(t_1,\ldots,t_k) \Leftarrow \delta_x \quad (t_1,\ldots,t_{k-1},0)$$

4.4. Sémantique d'une définition itérative

Dans ce cas, la sémantique est la réunion de la sémantiqu du module d'initialisation, du module d'itération et de définition récursives donnant les derniers termes des suites x_i à l'aide de suites $\omega[x_i]$

$$\mathcal{S}[\![x_1,\ldots,x_m : \text{I jqa arret } \underline{\text{repeter}} \text{ M}]\!] \equiv \mathcal{S}[\![\text{I}]\!] \cup \mathcal{S}[\![\text{M}]\!] \cup \bigcup_{i=1}^{m} \omega[x_i] \quad (t_1,\ldots,t_k) \Leftarrow \underline{\text{si}} \text{ arret}(t_1,\ldots,t_k) \underline{\text{alors}} \hat{u}(t_1,..$$

$$\underline{\text{sinon}} \ \omega[x_i] \quad (t_1,\ldots,t_k+1)$$

4.5. Définition de $\mathcal{E}$

Il faut définir $\mathcal{E}[\![\text{exp}]\!]$ suivant la configuration de exp et son contexte; en effet, suivant ce contexte c'est-à-dire essentiellement suivant qu'il s'agit d'une initialisation ou non, un identificateur aura un sous différent a exp est un identificate dans un module M d'itération, principal ou d'intialisation ; mais dans ce dernier cas, la définition de u ne figure pas dans ce module (et $k < p$).

$\mathcal{E}[\![x]\!]\ (t_1,\ldots,t_p) \equiv \hat{x}(t_1,\ldots,t_k)$

où $p = \underline{p}(M)$ et $k = \underline{p}(u)$ et $p \geq k$

b. exp est un identificateur x dans un module M d'initialisation et la définition de x figure dans ce module.

$$\mathcal{E}[\![x]\!]\ (t_1,\ldots,t_{k-1}) \equiv \hat{x}(t_1,\ldots,t_{k-1},0)$$

où $k = \underline{p}(x)$

c. exp est a) x

$$\mathcal{E}[\![\cdot\ x]\!]\ (t_1,\ldots,t_k) \equiv \hat{x}(t_1,\ldots,t_k-1)$$

d. exp est % x

$$\mathcal{E}[\![\% \ x]\!]\ (t_1,\ldots,t_{k-1}) \equiv \omega[x]\ (t_1,\ldots,t_{k-1},0) \text{ où } k=\underline{p}(x)$$

e. exp est composée

$$\mathcal{E}[\![f(exp_1,\ldots,exp_m)]\!]\ (t_1,\ldots,t_k) \equiv \hat{f}(\mathcal{E}[\![exp1]\!]\ (t_1,\ldots,t_k),\ldots,\mathcal{E}[\![exp\ m]\!]\ (t_1,\ldots,t_k))$$

ou $\hat{f}$ est l'interprêtation de f.

4.6. Equations récursives et point fixe

La sémantique d'un énoncé est la sémantique de son module principal, c'est-à-dire d'un ensemble d'équations récursives. Ces équations récursives dépendent des valeurs des suites δ_x pour chaque identificateur x, apparaissant dans une définition x=$\underline{\text{donnée}}_x$. Pour chaque valeur d'un terme des δ_x, la solution de ces équations δ_x est le plus petit point fixe pour la relations "moins défini que".

4.7. Etude de l'exemple

Dans l'énoncé de la figure 3 on obtient

$$\mathcal{S}[\![\text{PRINCIPAL}]\!] \equiv \text{rês} \Leftarrow \omega[m](0) \cup \mathcal{S}[\![\text{INI MAX}]\!] \cup \mathcal{S}[\![\text{MAX}]\!] \cup$$

$$\omega[m](t_1) \Leftarrow \underline{\text{si}}\ \text{arrêtmax}(t_1)\ \underline{\text{alors}}\ \hat{m}(t_1)\ \underline{\text{sinon}}\ \omega[m](t_1+1)$$

où

$$\mathcal{S}[\![\text{INI MAX}]\!] \equiv \hat{m}(0) \Leftarrow 0 \cup \text{arrêtmax}(0) \Leftarrow \underline{\text{faûx}}$$

$$\mathcal{S}[\![\text{MAX}]\!] \equiv \hat{m}(t_1) \Leftarrow \text{mâx}(\omega[\text{som}](t_1,0), \hat{m}(t_{1-1})) \cup$$

$$\mathcal{S}[\![\text{INI SOM}]\!] \cup \mathcal{S}[\![\text{SOMME}]\!] \cup$$

$$\text{arrêtmax}(t_1) \Leftarrow (\omega[\text{som}](t_1,0)\ \hat{\text{e}}q\text{-}2)$$

où $\mathcal{S}[\![\text{INI SOM}]\!] \equiv \text{scôre}(t_1,0) \Leftarrow \delta_{\text{score}}(t_1,0) \cup$

$$\text{sôm}(t_1,0) \Leftarrow 0 \cup$$

$$\text{arrêt}(t_1,0) \Leftarrow [(\text{scôre}(t_1,0)\ \underline{\text{eq}}\text{-}1)\ \underline{\text{oû}}\ (\underline{\text{score}}(t_1,0)\text{eq-}2)]$$

$$\mathcal{S}[\![\text{SOMME}]\!] \equiv \text{sôm}(t_1,t_2) \Leftarrow (\text{sômt}_1,t_2\text{-}1) + \text{scôre}(t_1,t_2\text{-}1)) \cup$$

$$\text{scôre}(t_1,t_2) \Leftarrow \delta_{\text{score}}(t_1,t_2) \cup$$

$$\text{arrêt}(t_1,t_2) \Leftarrow (\text{scôre}(t_1,t_2)\hat{\text{e}}q\text{-}1\ \text{oû}(\text{scôre}(t_1,t_2)\hat{\text{e}}q\text{-}2))$$

BIBLIOGRAPHIE

[ASH 77]

E.A. ASHCROFT et W.W. WADGE

Lucid a Monprocedural language with iteration. Comm. ACM 20 (1977), p. 519-526.

[ASH 76]

E.A. ASHCORFT et W.W. WADGE

Lucid:a formal system for writing and proving programs

SIAM J. Comptg 5, (1976), pp 336-354.

[BEL 77]

F. BELLEGARDE, B. HUC, J.M. PIERREL et A. QUERE

Initiation à une construction méthodique des programmes

Centre de Recherche en Informatique de Nancy CRIN 77-E-003

[FIN 77]

J.P. FINANCE

Static and computationnal semantics of a definitional languag

IFIP Work conf. on Formal Description of Programming Concepts St Andrews (1977)

[HUC 77]

B. HUC

Mise en oeuvre de la méthode de programmation déductive.

Thèse de docteur-Ingénieur. Université de Nancy 1 (1977)

[PAI 77]

C. PAIR

La construction des programmes

Centre de Recherche en Informatique de Nancy. CRIN 77-R-019