RAIRO. INFORMATIQUE THÉORIQUE

P. ENJALBERT

Systèmes de déduction pour les arbres et les schémas de programme (II)

RAIRO. Informatique théorique, tome 15, n° 1 (1981), p. 3-21

<http://www.numdam.org/item?id=ITA_1981__15_1_3_0>

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 15, n° 1, 1981, p. 3 à 21)

SYSTÈMES DE DÉDUCTION POUR LES ARBRES ET LES SCHÉMAS DE PROGRAMME (II) (*)

par P. ENJALBERT (1)

Communiqué par M. NIVAT

Résumé. — *Dans le précédent numéro nous avons établi un système de preuve de l'exactitude partielle des arbres de programmes de Cousineau. Nous en déduisons maintenant un système de déduction pour les « schémas de programmes structurés », immédiatement étendu à des programmes avec instructions de saut : les systèmes d'équations d'actions (réguliers) EXEL. Nous donnons également une sémantique et un système de preuve pour des programmes avec assertions intermédiaires, et décrivons un calcul de « weakest preconditions » dans $L\omega_1\omega$ (logique avec des conjonctions et disjonctions dénombrables).*

Abstract. — *In the previous number we established a proof system for partial correctness of Cousineau's program trees. A deductive system for "structured program schemes" is now deduced from it; this new system can be immediately extended to programs including jump instructions: EXEL (regular) systems of actions equations. We also briefly give a semantic and a proof system for programs with intermediate assertions, and describe a calculus of weakest preconditions in $L\omega_1\omega$ (logic with denumerable conjunctions and disjunctions).*

2. UN SYSTÈME DE DÉDUCTION POUR LES SCHÉMAS DE PROGRAMMES STRUCTURÉS (2)

2.1. Définition

Soit L un langage du 1er ordre, $V = \{v_1, \ldots, v_k\}$ un ensemble de variables, et les ensembles $Te(L, V)$, $A(L, V)$ et S comme au I. L'ensemble des schémas de programme structurés (S.P.S.) sur L et V est l'ensemble ainsi défini :

(i) $\forall s \in S$, s est un S.P.S.;

(ii) si E est un S.P.S. $v_i \leftarrow t \bullet E$ est un S.P.S.;

(iii) si E_1, E_2 sont des S.P.S., $r \in Te(L, V)$, $\subset r \rightarrow E_1 \diamond E_2 \supset$ est un S.P.S. (test);

(*) Reçu décembre 1978, révisé octobre 1979.

(1) Thomson-CSF, Orsay.

(2) Le lecteur trouvera en fin de la première partie deux appendices : « éléments de théorie des modèles » et « table des principales notions et notations » ainsi que la bibliographie.

(iv) si E_1, E_2 sont des S.P.S. et aucun α_j ne figure dans E_1, $E_1 \bullet E_2$ est un S.P.S. (concaténation);

(v) si E est un S.P.S., $\{E\}$ est un S.P.S. (itération);

(vi) tout S.P.S. est obtenu en appliquant un nombre fini de fois les règles (i)-(v).

Nous appelons $\mathscr{S}(L, V)$ cet ensemble :

A tout S.P.S. E sur L et V est associé un A.P. sur L et V, $\overline{E}$, de la manière suivante :

(i) $\forall s \in S, \overline{s} = s$;

(ii) $\overline{(v_i \leftarrow t) \bullet E} = (v_i \leftarrow t)(\overline{E})$;

(iii) $\overline{\subset r \rightarrow E_1 \diamond E_2 \supset} = r(\overline{E}_1, \overline{E}_2)$;

(iv) $\overline{E_1 \bullet E_2} = \overline{E}_1 \bullet \overline{E}_2$;

(v) $\overline{\{E\}} = \overline{E}^*$.

Cette fonction :

$$\mathscr{S}(L, V) \rightarrow A(L, V),$$

$$E \rightarrow \overline{E},$$

définit une sémantique des schémas de programmes structurés.

Exemple 7 : L'arbre de l'exemple 1 pourra être décrit par le S.P.S. :

$$E_1 = (X1 \leftarrow 0 \bullet Z \leftarrow 0 \bullet \omega_0) \bullet \{ \subset X1 = X \rightarrow \omega_1 \diamond X1 \leftarrow X1 + 1 \bullet Z \leftarrow Z + Y \bullet \omega_0 \supset \},$$

conformément à la décomposition de l'exemple 2, ou par :

$$E_2 = X1 \leftarrow 0 \bullet Z \leftarrow 0 \bullet \{ \subset X1 = X \rightarrow \omega_1 \diamond X1 \leftarrow X1 + 1 \bullet Z \leftarrow Z + Y \bullet \omega_0 \supset \}.$$

Ces définitions sont à peu de chose près celles de Cousineau [9, 10] : Leur signification intuitive ne pose pas de problème si l'on saisit que chaque symbole ω_i fait sortir de i itérations ($\{\ldots\}$) emboîtées. La seule différence notable avec Cousineau concerne les symboles α_j; nous l'expliciterons à propos de la « règle pour la substitution » en 2.7.

Finalement, rappelons que les S.P.S. permettent de décrire « de manière naturelle » tous les organigrammes [21].

2.2. Le système de déduction S

Soient L_1 et L_2 deux langages du 1er ordre. Nous considérons les triplets $[P]E[T]$ comme au I si ce n'est que E est ici un S.P.S. et non un arbre. D'autre part, T désigne toujours une théorie du 1er ordre écrite dans L_2.

Rappelons que pour tout ensemble d'assertions $T=(Q, R)$, tout $k<\omega$ et $F\in \mathrm{For}(L_2)$:

$$T(k, F)=((Q_0, \ldots, Q_k, F, Q_{k+1}, \ldots), R),$$

le système de déduction S comprend :

– *les axiomes :*

$$S_0 : \quad \mathrm{T}\underset{S}{\Vdash}[P]S[T] \qquad \text{si} \quad \mathrm{T}\vdash P\to T_s;$$

– *les règles de déduction :*

$$S_1 : \quad \frac{\mathrm{T}\underset{S}{\Vdash}[P]E[T]}{\mathrm{T}\underset{S}{\Vdash}[P_t^{v_i}]v_i\leftarrow t\bullet E[T]},$$

$$S_2 : \quad \frac{\mathrm{T}\underset{S}{\Vdash}\{P\wedge r\}E_1[T]\ \mathrm{T}\underset{S}{\Vdash}[P\wedge\neg r]E_2[T]}{\mathrm{T}\underset{S}{\Vdash}[P]\subset r\to E_1\diamond E_2\supset[T]},$$

$$S_3 : \quad \frac{\mathrm{T}\underset{S}{\Vdash}[P]E_1[T]\ \mathrm{T}\underset{S}{\Vdash}[Q_0]E_2[T')}{\mathrm{T}\underset{S}{\Vdash}[P]E_1\bullet E_2[T]},$$

Sous la condition : $\forall s\in S\setminus\{\omega_0\}$:

$$T_s=T'_s$$

$$S_4 : \quad \frac{\mathrm{T}\underset{S}{\Vdash}[P]E[T(0, P)]}{\mathrm{T}\underset{S}{\Vdash}[P]\{E\}[T]},$$

$$S_5 : \quad \frac{\mathrm{T}\underset{S}{\Vdash}[P]E[T]}{\mathrm{T}\underset{S}{\Vdash}[P']E[T']},$$

$$\text{si} \quad T\vdash P'\to P.$$

Il s'agit donc d'un système de déduction d'allure classique, la règle S_4 constituant une extension de la règle pour la boucle While de la même manière que l'itération $\{\ldots\}$ est une forme très générale d'itération étendant en particulier ladite boucle.

DÉFINITION : L'ensemble des conséquences de T par S est le plus petit ensemble contenant les triplets $[P]E[T]$ tels que $\mathrm{T}\underset{S}{\Vdash}[P]E[T]$ soit un axiome et clos par les règles de déduction.

Un triplet $[P]E[T]$ est donc conséquence de T ssi il existe une suite finie $([P_\alpha]\ E_\alpha\ [T_\alpha])_{\alpha \leq l}$ telle que $P=P_l$, $E=E_l$, $T=T_l$ et que pour chaque $\alpha \leq l$, $T \Vdash_S [P_\alpha] E_\alpha [T_\alpha]$ est soit un axiome, soit une conséquence par les règles de déduction de termes précédents. Une telle suite est une *preuve* de $T \Vdash_S [P]E[T]$; l est la *longueur* de la preuve.

REMARQUE : La présentation de notre système de preuve met en évidence et isole le système de déduction pour les énoncés de L_2 (la théorie T) qui est nécessairement inclus dans tout système de preuve de programmes. Il fournit des preuves modulo T, c'est-à-dire relativement à un ensemble de propriétés des structures considérées que l'on énonce dans T.

2.3. Consistance de S

Grâce au théorème fondamental pour A, la consistance de S se ramène à la consistance *par rapport* à A pour la fonction sémantique : $E \to \overline{E}$:

THÉORÈME 2.1 : *S est consistant par rapport à A :*

$$\forall \text{S.P.S. } E, \qquad \forall P, T \text{ et T},$$
$$T \Vdash_S [P]E[T] \Rightarrow T \Vdash_A [P]\overline{E}[T].$$

La démonstration est évidente, par récurrence sur la longueur de la preuve de $[P]E[T]$ en utilisant les règles dérivées $A_{51}, A_{52}, A_7, A_5, A_{21}$, correspondant aux règles S_1 à S_5.

2.4. Précondition la plus faible pour un S.P.S. sans itération

Pour tout E sans itération (sans symboles $\{\ \}$), on définit une fonction $\tilde{E} : \tau(L_2) \to \text{For}(L_2)$ de la manière suivante :

$$\tilde{s}(T) = T_s \qquad (s \in S),$$
$$(v_i \leftarrow t \tilde{\bullet} E)(T) = [\tilde{E}(T)]^{v_i}_t,$$
$$\subset r \to \tilde{E}_1 \diamond E_2 \supset (T) = r \to \tilde{E}_1(T) \wedge \neg r \to \tilde{E}_2(T),$$
$$E_1 \tilde{\bullet} E_2(T) = \tilde{E}_1((T - Q_0)(0, \tilde{E}_2(T)),$$
$$[T - Q_0 = (Q', R') \text{ avec } R' = R, \text{ et } \forall i < \omega\ Q'_i = Q_{i+1}].$$

Par induction sur la complexité de E, et en utilisant le lemme 1.4 pour la concaténation nous avons immédiatement :

LEMME 2.2 :

$$\tilde{E}(T)=\tilde{\tilde{E}}(T)$$

et de même :

LEMME 2.3 : *Pour E sans itération, et tous T, P,* T :

$$\mathrm{T} \underset{S}{\Vdash} [\tilde{E}(T)]\, E[T].$$

Alors :

PROPOSITION 2.4 : *Si E est sans itération :*

$$T \underset{S}{\Vdash} [P]\, E[T] \quad \Leftrightarrow \quad \mathrm{T} \vdash P \to \tilde{E}(T).$$

Dnas un sens ($\Rightarrow$), c'est une conséquence immédiate du résultat correspondant pour les arbres finis (prop. 1.5) et du théorème de consistance de S; dans l'autre ($\Leftarrow$) du lemme 2.3 et de la règle S_5.

Nous retrouvons ainsi un résultat présent chez plusieurs auteurs [14, 24] dans un cadre plus général incluant notamment une notion de « postcondition la plus forte » (à ce sujet *voir* [6]). Bien qu'il paraisse possible de définir une telle notion dans notre formalisme (il faudrait alors parler d'ensemble de postconditions associées aux différents éléments terminaux), la structure arborescente sur laquelle nous travaillons s'adapte plus naturellement à la notion de précondition. Par contre la méthode employée ici est *entièrement constructive*, nous y reviendrons en 3.

2.5. Compacité pour S et A

PROPOSITION 2.5 : *Si* $[P]\, E[T]$ *est conséquence d'une théorie* T, *il est conséquence d'un nombre fini de formules de* T :

$$\mathrm{T} \underset{S}{\Vdash} [P]\, E[T] \quad \Rightarrow \quad \begin{cases} \exists A_1 \ldots A_n \in \mathrm{T} \\ A_1, \ldots, A_n \underset{S}{\Vdash} [P]\, E[T]. \end{cases}$$

Démonstration : Toute preuve utilise un nombre fini d'axiomes :

$$\mathrm{T} \underset{S}{\Vdash} [P^i]\, s^i[T^i] \qquad \text{avec} \quad \mathrm{T} \vdash P^i \to T^i_{s^i}$$

Pour chacun de ces axiomes, par compacité du calcul des prédicats, $P^i \to T^i_{s^i}$ est conséquence d'un nombre fini de formules de T. La réunion de toutes ces formules est le sous-ensemble fini de T cherché.

Par contre, il n'y a pas de théorème de compacité pour A :

PROPOSITION 2.6 : *Il existe une théorie* T *et un triplet* $[P]\, a\, [T]$ (*a étant un A.P.*), *conséquence de* T *par A, mais conséquence d'aucun sous-ensemble fini de* T.

Preuve : Prenons pour $L_1 = L_2$ le langage des groupes additifs (1 symbole de fonction binaire $+$, et 1 symbole de constante 0).

Soit

$$E = y \leftarrow x \bullet \{ \subset y = 0 \rightarrow \omega_1 \diamond y \leftarrow y + x \bullet \omega_0 \supset \}.$$

Pour tout q entier $\geqq 1$, soit Tq la théorie des groupes commutatifs d'exposant $\leqq q$:

$$T_q = \{ \text{axiomes de groupe} \} \cup \{ x \neq 0 \Rightarrow nx \neq 0 : n < q \}$$

(en posant $nx = x + x + \ldots + x$, n fois) et $T = \bigcup_{q<\omega} T_q$, la théorie des groupes sans torsion.

Désignons par [Faux] n'importe quel ensemble d'assertions $[T]$ tel que $T_{\omega_0} = \text{Faux}$.

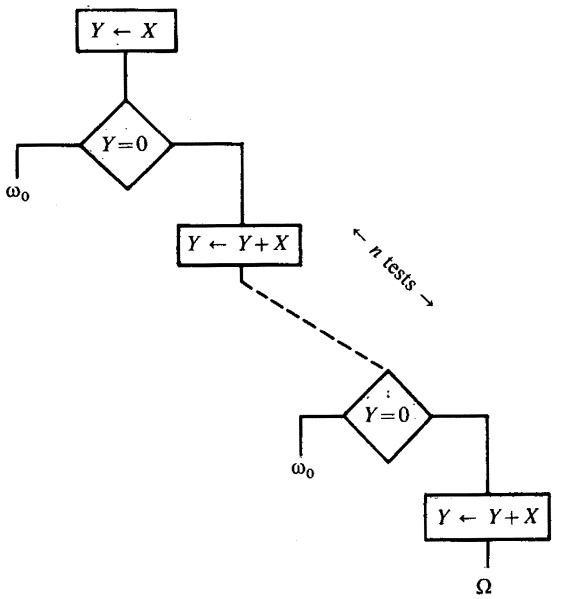
Nous aurons :

$$T \underset{A}{\Vdash} [x \neq 0]\, E\, [\text{Faux}]$$

(ce qui signifie exactement que E diverge pour toute valeur, dans tout groupe sans torsion). En effet :

$$E = \sup_{n<\omega}(a_n), \qquad \text{ou} \qquad \text{pour tout } n :$$

$a_n =$



et $\forall n<\omega\, \tilde{a}_n(\text{Faux}) = \bigwedge_{p=1}^{n} px \neq 0$,

donc

$$T \vdash x \neq 0 \to \tilde{a}_n(\text{Faux}) \Rightarrow T \underset{A}{\Vdash} [x \neq 0]\, a_n[\text{Faux}]$$

et il suffit d'appliquer la règle A_1.

Si maintenant il existait un sous-ensemble fini de T, T_f tel que

$$T_f \underset{A}{\Vdash} [x \neq 0]\, \overline{E}\, [\text{Faux}],$$

pour un certain q nous aurions encore :

$$T_q \underset{A}{\Vdash} [x \neq 0]\, \overline{E}\, [\text{Faux}],$$

donc, pour tout $n<\omega$, par la proposition 1.6 ($a_n < \overline{E}$) :

$$T_q \underset{A}{\Vdash} [x \neq 0]\, a_n[\text{Faux}] \quad \Leftrightarrow \quad T_q \vdash x \neq 0 \to \bigwedge_{p=1}^{n} px \neq 0$$

et finalement tout groupe d'exposant $\leqq q$ serait sans torsion. □

REMARQUE : Nous aurions aussi bien pu prendre pour T la théorie des groupes commutatifs sans torsion divisibles ($\forall x, \exists y, ny = x : n<\omega$). Cette théorie est complète.

Intuitivement, la proposition 2.5 dit que si $T \underset{S}{\Vdash} [P]\ E\ [T]$, alors on peut toujours extraire de T un oracle dont $[P]\, E\, [T]$ sera conséquence. Les propositions 2.5 et 2.6 réunies vont nous donner immédiatement l'incomplétude de S et de tout système similaire.

2.6. Le problème de la complétude de S

Par le théorème fondamental pour A, il se ramène au problème de la complétude de S par rapport à A :

Est-il vrai que :

$$T \underset{A}{\Vdash} [P]\, \overline{E}\, [T] \quad \Rightarrow \quad T \underset{S}{\Vdash} [P]\, E\, [T] \ ?$$

Pour E sans itération, nous avons :

$$T \underset{A}{\Vdash} [P]\, \overline{E}\, [T] \Rightarrow \quad T \vdash P \to \tilde{E}(T) \quad \Rightarrow \quad T \underset{S}{\Vdash} [P]\, E\, [T].$$

Donc :

PROPOSITION 2.7 : *La restriction de S aux programmes sans itération est complète.*

Il n'en est rien si l'on considère des S.P.S. quelconques :

THÉORÈME 2.8 (incomplétude) : *Il existe une théorie* T [*complète*] *et un triplet* $[P]E[T]$ (*E étant un* S.P.S.) *tels que :*

$$T \underset{A}{\Vdash} [P]\overline{E}[T] \quad \textit{et non :} \quad T \underset{S}{\Vdash} [P]E[T].$$

Preuve : Reprenons le contre exemple de la proposition 2.6 :

$$E = y \leftarrow x \bullet \{ \subset y = 0 \rightarrow \omega_1 \diamond y \leftarrow y + x \bullet \omega_0 \supset \}$$

Nous avons :

$$T \underset{A}{\Vdash} [x \neq 0]\overline{E}[\text{Faux}].$$

Supposons que :

$$T \underset{S}{\Vdash} [x \neq 0]E[\text{Faux}].$$

Par compacité, il existe $T_f \subset T$, Fini, tel que :

$$T_f \underset{S}{\Vdash} [x \neq 0]E[\text{Faux}] \quad \Rightarrow \quad T_f \underset{A}{\Vdash} [x \neq 0]\overline{E}[\text{Faux}]$$

par consistance de *S*. Mais ceci est en contradiction avec la proposition 2.6 (Si nous désirons que T soit complète, il suffit de l'enrichir comme dans la remarque de fin du paragraphe 2.5) □.

L'incomplétude de *S* – et de systèmes de Hoare similaires – viendrait-elle de notre maladresse dans l'écriture des règles de déduction ? L'argument employé pour prouver 2.8 permet de voir qu'il n'en est rien.

Qualifions de finitaire tout système de déduction dont les règles font intervenir un nombre fini de prémisses.

Un système de déduction *finitaire, du* 1er *ordre*, sera un système de déduction Σ pour des expressions du type $T \underset{\Sigma}{\Vdash} [P]E[T]$ où *T*, *P* et T sont écrits dans un langage du 1er ordre finitaire, comprenant un nombre fini de règles toutes finitaires, et dont les axiomes sont du type :

$$T \underset{\Sigma}{\Vdash} [P]E[T] \qquad \text{si} \quad T \vdash P \rightarrow T_s$$

(cette dernière condition a pour but d'interdire l'introduction d'expressions $T \underset{\Sigma}{\Vdash} [P]\,E\,[T]$ particulières comme axiomes).

La compacité, et donc l'incomplétude de tels systèmes se prouve comme pour S. D'où :

THÉORÈME 2.9 : *Tout système de déduction finitaire, du 1er ordre, consistant, pour des triplets d'exactitude partielle* $[P]\,E\,[T]$ *est incomplet.*

Ainsi, le sous-système dérivé de A pour les arbres rationnels, et constitué des régles $A_{21}, A_{22}, A_3, A_4, A_{51}, A_{52}, A_7, A_8$ est nécessairement incomplet, comme nous l'avions annoncé en conclusion du 1.

Définir un « système de Hoare »; c'est donc tenter de résoudre le problème suivant : comment réduire la validité de tout ensemble, éventuellement *infini*, d'énoncés caractérisant l'exactitude partielle d'un programme à la validité d'un nombre fini d'entre eux (ce que l'on essaie de faire en utilisant des *invariants* de boucle).

La cause de l'incomplétude de tels systèmes réside dans l'incapacité de la logique du 1er ordre (finitaire) à réaliser cette réduction (en général). Le théorème 2.9 confirme donc et précise les conclusions de Wand [26]. Nous retrouverons cet aspect des choses au 3 à propos du calcul des « weakest preconditions ».

Un certain nombre de résultats d'incomplétude ou de complétude dans certains cas particuliers entre autres liés à la notion d'expressivité [6, 17, 24], obtenus par différents auteurs peuvent être adaptés à notre formalisme. Nous ne les reprendrons pas ici.

2.7. Règle pour la substitution

DÉFINITIONS : Nous définissons par récurrence sur la complexité de E les opérations de *substitution* et les opérations $\uparrow_k (k < \omega)$ adaptées, avec un degré de généralité moindre de Cousineau [9, 10] :

(i) si $E = s \in S$:

$$- E[\alpha_{j_1} \backslash\backslash E_1, \ldots, \alpha_{j_m} \backslash\backslash E_m] = E_p \quad \text{si} \quad s = \alpha_{j_p} \quad = E \quad \text{sinon},$$

$$- \uparrow_k (E) = \omega_{i+1} \quad \text{si} \quad s = \omega_i, \quad i \geqq k \quad = E \quad \text{sinon};$$

(ii) si $E = v_i \leftarrow t \bullet E'$:

$$- E[\alpha_1 \backslash\backslash E_1, \ldots, \alpha_m \backslash\backslash E_m] = v_i \leftarrow t \bullet E'[\alpha_1 \backslash\backslash E_1, \ldots, \alpha_m \backslash\backslash E_m],$$

$$- \uparrow_k (E) = v_i \leftarrow t \bullet \uparrow_k (E');$$

(iii) si $E = \subset r \to E' \diamond E'' \supset$:

$$- E[\alpha_1 \backslash\backslash E_1, \ldots] = \subset r \to E'[\alpha_1 \backslash\backslash E_1, \ldots] \diamond E''[\alpha_1 \backslash\backslash E_1, \ldots] \supset,$$
$$- \uparrow_k(E) = \subset r \to \uparrow_k(E') \diamond \uparrow_k(E'') \supset ;$$

(iv) si $E = E' \bullet E''$ (rappelons qu'alors aucun α_j n'a d'occurrence dans E', par définition de la concaténation) :

$$- E[\alpha_1 \backslash\backslash E_1, \ldots] = E' \bullet E''[\alpha_1 \backslash\backslash E_1, \ldots],$$
$$- \uparrow_k(E) = \uparrow_{\sup(1,k)}(E') \bullet \uparrow_k(E'');$$

(v) si $E = \{ E' \}$:

$$- E[\alpha_1 \backslash\backslash E_1, \ldots] = \{ E'[\alpha_1 \backslash\backslash \uparrow_0(E_1), \ldots] \},$$
$$- \uparrow_k(E) = \{ \uparrow_{k+1}(E') \}.$$

Exemple 8 :

$$E = \{ \subset r \to a \bullet \omega_0 \diamond \alpha_1 \supset \},$$
$$E_1 = \{ \subset t \to b \bullet \omega_0 \diamond \omega_1 \supset \}$$

(a, b sont des affectations et r, des tests) :

$$\uparrow_0(E_1) = \{ \subset t \to b \bullet \omega_0 \diamond \omega_2 \supset \},$$
$$F = E[\alpha_1 \backslash\backslash E_1] = \{ \subset r \to a \bullet \omega_0 \diamond \{ \subset t \to b \bullet \omega_0 \diamond \omega_2 \supset \} \supset \}.$$

REMARQUE : Il y a une différence notable avec les définitions de Cousineau qu'il convient d'expliciter. Les définitions données de l'opération de substitution ou d'arbre associé à un S.P.S. impliquent que toute occurrence d'un α_j fasse *sortir* du programme (quelle que soit la profondeur de cette occurrence, c'est-à-dire le nombre d'itérations emboîtées dans lesquelles elle se situe).

Ainsi, dans l'exemple 8 :

$$E = \{ \subset r \to a \bullet \omega_0 \diamond \alpha_1 \supset \}.$$

– pour Cousineau, lorsqu'on rencontre α_1 on exécute un S.P.S. associé à α_1 – E_1 dans l'exemple 8 – et on *revient* en tête de l'itération;

– pour nous : on *sort* de la boucle E pour exécuter E_1.

Cette divergence d'interprétation n'est cependant pas importante, et se réduit un simple changement de notation dans la mesure où nous ne sortons pas de ce que Cousineau appelle la classe des programmes réguliers.

Pour lui, E s'écrirait :

$$E' = \{ \subset r \to \alpha \bullet \omega_0 \diamond \uparrow(\alpha_1) \supset \}.$$

◦ La *règle pour la substitution* s'écrit :

$$S_6 : \quad \frac{\mathrm{T} \underset{S}{\Vdash} [P]\, E[T],\ \forall p = 1, \ldots, m,\ \mathrm{T} \underset{S}{\Vdash} [R_{j_p}]\, E_p[T']}{\mathrm{T} \underset{S}{\Vdash} [P]\, E[\alpha_{j_1} \backslash\backslash E_1, \ldots, \alpha_{j_m} \backslash\backslash E_m][T']}.$$

Sous la *condition de cohérence* :

$\forall \alpha_j$ ayant une occurrence dans E :

$$\alpha_j \in \{\alpha_{j_1}, \ldots, \alpha_{j_m}\} \quad \Rightarrow \quad R'_j = R_j.$$

Elle se dérive des règles S_1-S_5 et des axiomes S_0. Donnons les grandes lignes de la démonstration.

LEMME 2.10 :

$$\forall k < \omega, \quad \forall \Phi \in \mathrm{For}(L_2),$$
$$\mathrm{T} \underset{S}{\Vdash} [P]\, E[T] \quad \Rightarrow \quad \mathrm{T} \underset{S}{\Vdash} [P] \uparrow_k (E)[\mathrm{T}(k, \Phi)].$$

La démonstration du lemme se fait par induction sur la longueur de la preuve de $\mathrm{T} \underset{S}{\Vdash} [P]\, E[T]$.

★ Examinons les cas où la dernière règle appliquée est S_3 ou S_4. Si c'est S_3 :

$$\frac{\mathrm{T} \underset{S}{\Vdash} [P]\, E_1[T']\, \mathrm{T} \underset{S}{\Vdash} [Q'_0]\, E_2[T]}{\mathrm{T} \underset{S}{\Vdash} [P]\, E_1 \bullet E_2[T]},$$

avec

$$\forall s \in S \backslash \{\omega_0\}, \quad T'_s = T_s.$$

Deux cas à examiner, selon que l'on veut prouver le lemme pour $k \neq 0$ ou $k = 0$.

1er *cas* : $k \neq 0$

Par hypothèse d'induction :

$$\forall \Phi \in \mathrm{For}(L_2), \quad \mathrm{T} \underset{S}{\Vdash} [P] \uparrow_k (E_1)[T'(k, \Phi)] \qquad \text{et} \qquad \mathrm{T} \underset{S}{\Vdash} [P] \uparrow_k (E_2)[T(k, \Phi)]$$

et il suffit d'appliquer S_3.

2e *cas* : $k = 0$

Par hypothèse d'induction, nous aurons :
d'une part :

$$\forall \Phi' \in \mathrm{For}(L_2), \quad \mathrm{T} \underset{S}{\Vdash} [P] \uparrow_1 (E_1)[T'(1, \Phi')],$$

de l'autre :

$$\forall \Phi \in \mathrm{For}(L_2), \quad \mathrm{T} \underset{S}{\vdash} [Q_0'] \uparrow_0 (E_2)[T(0, \Phi)].$$

On prend $\Phi' = \Phi_0$; la condition de cohérence est alors vérifiée — par S_3 :

$$\mathrm{T} \underset{S}{\vdash} [P] \uparrow_1 (E_1) \bullet \uparrow_0 (E_2)[T(0, \Phi)].$$

Or :

$$\uparrow_0 (E_1 \bullet E_2) = \uparrow_1 (E_1) \bullet \uparrow_0 (E_2).$$

★ Si la dernière règle appliquée est S_4 :

$$\frac{\mathrm{T} \underset{S}{\vdash} [P] E[T(0, p)]}{\mathrm{T} \underset{S}{\vdash} [P]\{E\}[T]}.$$

Par hypothèse d'induction :

$$\forall k < \omega, \quad \forall \Phi \in \mathrm{For}(L_2),$$
$$\mathrm{T} \underset{S}{\vdash} [P] \uparrow_{k+1} (E)[T(0, P)(k+1, \Phi)].$$

Il est aisé de vérifier que :

$$T(0, P)(k+1, \Phi) = T(k, \Phi)(0, P).$$

D'où, par S_4 :

$$\mathrm{T} \underset{S}{\vdash} [P]\{\uparrow_{k+1} (E)\}[T(k, \Phi)].$$

Mais

$$\{\uparrow_{k+1} (E)\} = \uparrow_k (\{E\}). \quad \square$$

On peut alors prouver la validité de la règle S_6 pour la substitution par induction sur la longueur de la preuve de la 1re prémisse. Le seul cas un peu délicat est celui où la dernière règle appliquée est S_4 :

$$\frac{\mathrm{T} \underset{S}{\vdash} [P] E'[T(0, P)]}{\mathrm{T} \underset{S}{\vdash} [P]\{E'\}[T]} \qquad \text{et} \qquad E = \{E'\}.$$

Du deuxième groupe de prémisses nous pouvons tirer, en utilisant le lemme 2.10 ci-dessus :

$$\forall p=1, \ldots, m, \qquad \mathrm{T} \underset{S}{\Vdash} [R_{j_p}] \uparrow_0 (E_p)[T'(0, P)].$$

Par hypothèse d'induction donc :

$$\mathrm{T} \underset{S}{\Vdash} [P]\, E'[\alpha_2 \backslash\backslash \uparrow_0 (E_1), \ldots, \alpha_m \backslash\backslash \uparrow_0 (E_m)][T'(0, P)]$$

et par S_4, par définition de la substitution dans $\{E'\}$:

$$\mathrm{T} \underset{S}{\Vdash} [P]\{E'\}[\alpha_1 \backslash\backslash E_1, \ldots, \alpha_m \backslash\backslash E_m][T']. \quad \square$$

Exemple 9 : Reprenons l'exemple 8 et posons :

$$[Q_0]=[T],$$

où $T_{\omega_0}=Q_0$, et $T_s\ qq$ pour $s\neq\omega_0$;

$$[Q_0, R_1]=[T'],$$

où $T'_{\omega_0}=Q_0$, $T'_{\alpha_1}=R_1$, et $T'_s\ qq$ pour $s\neq\omega_0$ ou α_1.

Pour prouver :

$$\mathrm{T} \underset{S}{\Vdash} [P]\{\subset r \to a \bullet \omega_0 \diamond \{\subset t \to b \bullet \omega_0 \diamond \omega_2 \supset\} \supset\}[Q_0],$$

il suffit de prouver séparément :

$$\mathrm{T} \underset{S}{\Vdash} [P]\{\subset r \to a \bullet \omega_0 \diamond \alpha_1 \supset\}[Q_0, R_1]$$

et :

$$\mathrm{T} \underset{S}{\Vdash} [R_1]\{\subset t \to b \bullet \omega_0 \diamond \omega_1 \supset\}[Q_0].$$

La règle S_6 permet donc d'effectuer des preuves de façon « modulaire ». Nous en évoquons en 3 une application importante à la preuve des programmes avec instructions de saut.

2.8. Autres règles dérivées

Nous reprenons les notations du 1 pour $T \wedge T'$, $T \vee T'$, $T \to T'$, $\mathrm{T} \vdash T$.

Les règles suivantes se dérivent dans S :

$$S_7 : \frac{T \underset{S}{\Vdash} [P]\, E[T]}{T \underset{S}{\Vdash} [P]\, E[T']} \quad \text{si} \quad T \vdash T \to T',$$

$$S_8 : \frac{T \underset{S}{\Vdash} [P]\, E[T]\; T \underset{S}{\Vdash} [P']\, E[T]}{T \underset{S}{\Vdash} [P \vee P']\, E[T]},$$

$$S_9 : \frac{T \underset{S}{\Vdash} [P]\, E[T]\; T \underset{S}{\Vdash} [P]\, E[T']}{T \underset{S}{\Vdash} [P]\, E[T \wedge T']}.$$

La règle S_7 se démontre par induction sur la longueur de la preuve de la premisse; S_8 et S_9 par induction sur la somme des longueurs des prémisses.

Par exemple, pour S_9 :

On commence par éliminer les dernières applications de S_5 (déduction). Il y a donc deux formules P_1 et P_2 telles que :

$$T \vdash P \to P_1, \qquad T \vdash P \to P_2,$$

et :

$$T \underset{S}{\Vdash} [P_1]\, E[T], \tag{1}$$

$$T \underset{S}{\Vdash} [P_2]\, E[T'] \tag{2}$$

et les dernières règles appliquées pour prouver (1) et (2) sont parmi $S_1, \ldots, S_4$, à moins que ce ne soient des axiomes. On remarque ensuite que cette dernière règle est certainement la même et ne dépend que de la forme de E. Terminons la démonstration dans le cas où c'est S_4 :

$E = \{E_1\}$ et :

$$T \underset{S}{\Vdash} [P_1]\, E_1[T(0, P_1)]\; T \underset{S}{\Vdash} [P_2]\, E_1[T'(0, P_2)]$$

Par S_5 :

$$T \underset{S}{\Vdash} [P_1 \wedge P_2]\, E_1[T(0, P_1)],$$

$$T \underset{S}{\Vdash} [P_1 \wedge P_2]\, E_1[T'(0, P_2)].$$

Par hypothèse d'induction :

$$T \underset{S}{\Vdash} [P_1 \wedge P_2]\, E_1[T(0, P_2) \wedge T'(0, P_2)].$$

Or :

$$T(0, P_1) \wedge T'(0, P_2) = T \wedge T'(0, P_1 \wedge P_2).$$

On applique donc S_4 :

$$\mathrm{T} \underset{S}{\Vdash} [P_1 \wedge P_2] \{ E_1 \} [T \wedge T']$$

puis S_5 en utilisant le fait que :

$$\mathrm{T} \vdash P \to P_1 \quad \text{et} \quad \mathrm{T} \vdash P \to P_2 \Rightarrow \mathrm{T} \vdash P \to P_1 \wedge P_2.$$

3. TROIS EXTENSIONS

Nous mentionnons plus succintement pour conclure, trois extensions du formalisme décrit ci-dessus.

3.1. Schémas de programmes réguliers

Un schémas de programme – ou : système d'équations d'actions EXEL [9, 11] – régulier est un ensemble :

$$\mathscr{E} = \{ \alpha_{j_\rho} \vdash \tau_\rho \dashv / \rho = 1, \ldots, n \},$$

où : α_{j_ρ} sont les noms d'action; τ_ρ sont des S.P.S. : les *corps d'action*.

Une action peut être vue comme une procédure sans paramètre : pour exécuter $\mathscr{E}$, commencer par exécuter τ_1, puis lorsqu'on rencontre un symbole α_{j_ρ}, appeler l'action α_{j_ρ}, c'est-à-dire exécuter τ_ρ, et ainsi de suite. Formellement, la sémantique de tels programmes se définit à partir des arbres associés à chaque τ_ρ, en utilisant la règle de substitution. *Voir* à ce sujet [9, 11], avec les modifications signalées en 2.7.

◦ *Système de déduction* Σ *pour les schémas de programme réguliers*

Avec les mêmes notations que précédemment, $\mathscr{E}$ étant un schéma de programme, nous posons par définition de Σ :

$$\left.\begin{array}{c} \mathrm{T} \vdash P \to R_1, \\ \forall p = 1 \ldots m, \ \mathrm{T} \underset{S}{\Vdash} [P]\, \tau_\rho [T] \end{array}\right\} \Leftrightarrow \mathrm{T} \underset{\Sigma}{\Vdash} [P]\, \mathscr{E}\, [T].$$

Exemple 10 : L'exemple 9 peut s'écrire :

$$\mathscr{E} \begin{cases} \alpha_0 \vdash \{ \subset r \to a \bullet \omega_0 \diamond \alpha_1 \supset \} \dashv \\ \alpha_1 \vdash \{ \subset t \to b \bullet \omega_0 \diamond \omega_1 \supset \} \dashv \end{cases} \quad \text{ou} \quad \begin{cases} \alpha_0 \vdash E \dashv, \\ \alpha_1 \vdash E_1 \dashv. \end{cases}$$

Pour prouver $T \underset{\Sigma}{\Vdash} [P]\, \mathscr{E}\, [Q_0, R_0, R_1]$ on prouvera :

$$T \vdash P \to R_0, \qquad T \underset{S}{\Vdash} [R_0]\, E\, [Q_0, R_0, R_1],$$

$$T \underset{S}{\Vdash} [R_1]\, E_1\, [Q_0, R_0, R_1].$$

D'un côté la validité de Σ est une conséquence immédiate de la règle pour la substitution. De l'autre, la puissance de Σ est exactement la même que celle de S — de même que la classe des programmes définis par S.P.S. ou par schémas réguliers est le même.

Nous voyons donc que cette caractéristique des programmes réguliers EXEL — qui sont loin d'épuiser la classe des programmes EXEL sans procédure, *voir* [11] — qui est d'allier :

— la rigueur de la programmation structurée;

— la souplesse permise par ces « instructions de saut » que sont les appels d'actions,

se retrouve intégralement lorsqu'il s'agit de prouver ces programmes.

Nommément : la solution que nous donnons ici au problème de la preuve des programmes avec instructions de saut, nous paraît incomparablement plus simple que celle proposée dans les articles de notre connaissance sur le sujet (*voir* [7], et surtout [2] en notant la parenté avec notre formalisme).

3.2. Programmes et arbres avec assertions

Dans toute réalisation pratique on est amené à introduire dans le corps du programme à prouver des « assertions intermédiaires »; au moins des invariants de boucles, et éventuellement d'autres assertions qui allègent la preuve (*cf.* par exemple [20]).

Un S.P.S. avec assertions (S.P.S.A.) sur deux langages $L_1 \subset L_2$ est obtenu à partir des règles (i)-(v) pour les S.P.S. sur L_1, plus la règle additionnelle :

(vi) si E est un S.P.S.A., $A \in \mathrm{For}\,(L_2)$, $[A] \bullet E$ est un S.P.S.A.

Exemple 11 : A partir du S.P.S. E_2 de l'exemple 7 nous construisons :

$$F = X\,1 \leftarrow 0 \bullet Z \leftarrow 0 \bullet \{[Z = X\,1 \times Y] \bullet \subset X = X\,1 \to \omega_1 \diamond$$
$$X\,1 \leftarrow X\,1 + 1 \bullet Z \leftarrow Z + Y \bullet \omega_0 \supset \}.$$

Nous donnons une sémantique en terme *d'arbres* à de tels programmes. Très simplement, $L_1 \subset L_2$ étant deux langages, l'alphabet $W(L_1, L_2, V)$ est l'alphabet gradué $W(L_1, V)$ comme au 1, auquel on ajoute $\{[A]/A \in \mathrm{For}(L_2)\}$ comme termes de graduation 1. Un arbre de programme avec assertions est un

$W(L_1, L_2, V)$-arbre. La fonction : $E \to \overline{E}$ pour les S.P.S. s'étend trivialement aux S.P.S.A.

Nous avons construit un système de preuve (AA) pour les arbres avec assertions. Il y a un théorème fondamental analogue au théorème 1 qui dit en substance que $T \underset{AA}{\Vdash} [P]\ a[T]$ ssi :

(1) tout chemin entre le sommet et un symbole terminal s ou une assertion $[A]$ est exact (modulo T) relativement à P et T_s ou A;

(2) pour toute occurrence dans a d'un sous-arbre $[B]$ (b), la même condition est vraie avec B pour P et b pour a.

Par ailleurs, nous avons un système de preuve pour les S.P.S.A. qui est *consistant et complet* par rapport à AA.

Ces deux résultats définissent une *sémantique* pour les programmes avec assertions intermédiaires (sans appel de procédures avec paramètres). Ce qui a permis de fonder rigoureusement un *générateur de conditions de vérifications* (au sens de [20]) pour les programmes EXEL, actuellement implémenté au Laboratoire central de Recherche de Thomson-CSF. Un tel fondement manquait singulièrement, à notre sens, dans [20].

D'autre part, ce formalisme a permis à Cousineau et nous-mêmes de prouver le théorème suivant [13] :

THÉORÈME 3.1 : *Si E et E' sont deux S.P.S. syntaxiquement équivalents $(\overline{E} = \overline{E'})$ et si $T \underset{S}{\Vdash} [P]\ E[T]$, alors $T \underset{S}{\Vdash} [P]\ E'[T]$.*

Autrement dit : l'équivalence syntaxique préserve la prouvabilité. Remarquons que ce n'est pas le cas pour des équivalences plus faibles. Un contre-exemple peut être fourni pour l'équivalence de Paterson.

2.3. Preuves dans $L_{\omega_1 \omega}$

(Pour la définition des langages $L_{\omega_1 \omega}$, *voir* l'appendice.)

Travaillant dans $L_{\omega_1 \omega}$ nous pouvons alors étendre la notion de p. p. f. à tous les S.P.S. Il suffit d'ajouter la règle de formation :

$$\{\tilde{E}\}\ (T) = \bigwedge_{n<\omega} \tilde{E}^n(T),$$

où $\tilde{E}^n(T)$ est définie par :

$$\tilde{E}^0(T) = \text{Vrai},$$
$$\tilde{E}^{n+1}(T) = \tilde{E}(T(0, \tilde{E}^n(T)).$$

Désignons par S_1 (resp. A_1) l'extension de S (resp. A) aux langages $L_{\omega_1\omega}$. Nous aurons :

$$\mathrm{T} \underset{S_1}{\Vdash} [P]\, E[T] \quad \Leftrightarrow \quad \mathrm{T} \underset{\omega_1\omega}{\vdash} P \to \tilde{E}(T) \quad \Leftrightarrow \quad \mathrm{T} \underset{A_1}{\Vdash} [P]\, \overline{E}[T]$$

et, si T, P, T sont écrits dans un langage finitaire :

$$\mathrm{T} \underset{\omega_1\omega}{\vdash} P \to \tilde{E}(T) \quad \Leftrightarrow \quad \mathrm{T} \underset{A}{\Vdash} [P]\, \overline{E}[T],$$

$\tilde{E}(T)$ est donc la p. p. f. de E relativement à T. De manière duale on construit une formule de $L_{\omega_1\omega}$, $\hat{E}(T)$ qui est la condition d'exactitude *totale* de E relativement à T :

$$\mathrm{T} \underset{\omega_1\omega}{\vdash} P \to \hat{E}(T) \quad \Leftrightarrow \quad E \text{ est totalement exact par rapport à } P \text{ et } T.$$

L'existence de telles formules a déjà été signalée [17]; elle est assez évidente si l'on songe qu'il n'y a jamais qu'un ensemble *dénombrable* de chemins finis possibles dans l'exécution d'un programme. Mais nous les obtenons de manière constructive et calquée sur la structure de contrôle. En cela nous nous rapprochons plutôt de Dijskra [14] et surtout de De Baker (par exemple [4]). Analogie renforcée si l'on considère que, en munissant l'ensemble des formules de $L_{\omega_1\omega}$ de l'ordre

$$A \leqq B \quad \Leftrightarrow \quad \underset{\omega_1\omega}{\vdash} A \to B,$$

$\{\tilde{E}\}(T)$ est le plus grand point fixe de : $P \to \tilde{E}(P, T)$,

$\{\hat{E}\}(T)$ est le plus petit point fixe de : $P \to \hat{E}(P, T)$.

Avec cette différence essentielle toutefois que notre approche est entièrement *syntaxique*.

En second lieu, mentionnons que nous avons pu trouver ou retrouver un certain nombre de résultats de la Logique Dynamique : et en effet, ce que nous faisons, c'est exprimer les formules $[a]\, P$ et $\langle a \rangle\, P$ de la *L-D* dans $L_{\omega_1\omega}$.

Ainsi, pour nous, un langage L_2 sera L_1-expressif (L_1, L_2 du 1er ordre, *finitaires*) relativement à une structure U, ssi :

$$\forall E \in \mathscr{S}(L_1, V),$$

$\forall T$ ensemble d'assertion de sortie sur L_2, il existe une formule de L_2, Φ, telle que :

$$U \models (\Phi \leftrightarrow \tilde{E}(T)).$$

Le problème de la L_1-expressivité apparaît donc comme un problème de compactification d'une formule d'un langage infinitaire. Nous retrouvons donc là, sous une autre forme, la même question qu'à propos de la complétude de S.

CONCLUSION

Nous avons tenté de montrer dans cet article comment le surlangage EXEL et la sémantique qu'en a donnée G. Cousineau peuvent aider à la compréhension et à la manipulation de « logiques de Hoare ».

En particulier, à notre sens :

– transparence à la structure intuitive des programmes;

– simplicité et généralité du système A qui en fait une référence aisée pour l'étude de règles de déduction;

– simplicité des « règles pour la substitution » et application au problème du GOTO;

– définition d'une sémantique et d'un système de preuve pour les programmes avec assertions intermédiaires, passage obligé de toute réalisation pratique où l'on demande au programmeur de fournir les invariants de boucle;

– liens possibles avec la transformation de programme et les nombreux travaux sur ce sujet [1, 12] dans le formalisme d'EXEL.

D'autres questions sont ou devront être étudiées. C'est le cas de la preuve de programmes non réguliers, c'est-à-dire de systèmes d'actions non reductibles à des organigrammes [11], sorte d'intermédiaires entre les organigrammes et les programmes avec procédures; des liens avec la logique algorithmique ou dynamique, *via* notre construction de w. p. dans $L_{\omega_1 \omega}$ en particulier; ou encore, une situation à préciser en ce qui concerne les liens entre l'équivalence et la « prouvabilité » des programmes (*voir* [13]).

Certains problèmes, enfin, seraient à défricher. Citons : la preuve des programmes avec procédures : un formalisme « arboricole » pourrait-il être utilisé ; ou un lien possible avec l'étude des « chemins » d'un programme telle que la mène R. Parikh dans sa *Process Logic* [23]. De telles études permettraient de préciser, ou de repousser, les limites du formalisme des arbres à feuilles indicées.

N.B. : La bibliographie se trouve en fin de la première partie, dans le précédent numéro.