RAIRO. INFORMATIQUE THÉORIQUE

P. ENJALBERT

Systèmes de déduction pour les arbres et les schémas de programmes (I)

RAIRO. Informatique théorique, tome 14, n° 3 (1980), p. 247-278

<http://www.numdam.org/item?id=ITA_1980__14_3_247_0>

© AFCET, 1980, tous droits réservés.

L'accès aux archives de la revue « RAIRO. Informatique théorique » implique l'accord avec les conditions générales d'utilisation (http://www.numdam.org/conditions). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

NUMDAM

Article numérisé dans le cadre du programme
Numérisation de documents anciens mathématiques
http://www.numdam.org/

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 14, n° 3, 1980, p. 247 à 278)

SYSTÈMES DE DÉDUCTION POUR LES ARBRES ET LES SCHÉMAS DE PROGRAMMES (I) (*)

par P. ENJALBERT (1)

Communiqué par M. NIVAT

RÉSUMÉ. — *Le but de cet article est de fonder un système de preuve de l'exactitude partielle des programmes sur une sémantique algébrique : la théorie des arbres à feuilles indicées de Cousineau. Dans le présent numéro on introduit le concept fondamental : le système de déduction A pour les arbres de programmes. On prouve la consistance et la complètude de A; puis, comme cas particulier d'application de ce système très général, on en dérive une version améliorée des habituelles « règles de Hoare » pour programmes équivalents à des organigrammes.*

Dans le prochain numéro de la Revue, on en déduira un système de preuve pour les schémas de programmes et on étudiera quelques extensions.

ABSTRACT. — *The aim of this paper is to found some proof system for partial correctness of programs on an algebraic semantic: Cousineau's theory of trees with indexed leaves. In the present number we introduce our basic concept: the deductive system A for program trees. A is proved to be sound and complete; then, as a special application case of this most general system, an improved version of the usual "Hoare rudes" is derived, for programs equivalent to flowcharts.*

In the next number of this Journal, a deductive system for program schemes will be deduced from it, and a few extensions will be studied.

INTRODUCTION

L'objet de cet article est de lier les idées de la « logique de Hoare » à une sémantique algébrique : la théorie des arbres à feuilles indicées de Cousineau [9, 10].

En effet, ainsi que l'ont souligné Hoare et Lauer dans [19], la seule approche déductive, la seule définition axiomatique est insuffisante pour étudier un langage :

Il faut nécessairement faire le lien avec une sémantique qui rende compte d'une manière ou d'une autre des suites de calculs effectivement réalisées. Cette question est au cœur de diverses études menées depuis Cook [6] sur la complétude des systèmes de preuve par assertions. Dans la même lignée,

(*) Reçu décembre 1978, révisé octobre 1979.

(1) Thomson-CSF, L.C.R., Domaine de Corbeville, 91401 Orsay.

R.A.I.R.O. Informatique théorique/Theoretical Informatics: 0399-0540/1980/247/$ 5.00
© AFCET-Bordas-Dunod

plusieurs formalismes ont été proposés afin de traiter d'un même mouvement la définition, la sémantique et les propriétés logiques des programmes. Citons la logique algorithmique [3], la logique dynamique [16], et les travaux de De Bakker [4] sur les « weakest preconditions ». Pour notre part, ce que nous construisons est en fait une extension de la théorie de Cousineau permettant de traiter les propriétés logiques des programmes liées à l'exactitude partielle ou totale. Ce sont les bases de ce formalisme — en ce qui concerne l'exactitude partielle essentiellement — que nous exposons ici.

La notion de base de cette théorie est celle d'Arbre de programme (A.P.) : représentation synthétique sous forme d'arbre de l'ensemble des suites de calculs (tests, affectations, exits) possibles dans l'exécution d'un programme. Cousineau a forgé des outils pour manipuler commodément ce type d'objets mathématiques, donnant ainsi une formalisation dans le langage des arbres des notions qui sont à la base du surlangage EXEL [11, 22, 25].

Nous commençons donc par définir (section I) un système de déduction pour les arbres de programme, le système A, permettant de prouver des expressions, d'allure classique, $[P]\, a\, [T]$ — mais où a est un arbre de programme, P et T étant respectivement les spécifications d'entrée et de sortie écrites à l'aide de formules mathématiques.

Notre théorème fondamental fonde le système A comme système de preuve, complet, de l'exactitude partielle des programmes représentés sous forme d'arbres. Le premier intérêt est de court-circuiter définitivement tout recours à des suites de calculs, à des « Computational models » comme en [6] ou [19]. Tout se traite ensuite sans sortir du cadre unifié des arbres à feuilles indicées. En second lieu, A est un système très général, qui devrait permettre d'envisager différents modes de définition des programmes.

A l'étape actuelle toutefois nous n'avons examiné que la classe des programmes correspondant aux arbres reconnaissables (ayant un nombre fini de sous arbres). Cette classe correspond exactement aux organigrammes, ses membres peuvent être représentés par ce que nous appelons les schémas de programme structurés (S.P.S.).

Nous donnons dans la section II [2] un système de déduction S pour les S.P.S. (c'est-à-dire aussi bien les programmes EXEL sans appel d'actions). Les axiomes et règles de déduction de S apparaissent en fait comme déduites (dérivées) de A, ce qui prouve d'emblée la consistance de S. Par contre, comme les systèmes similaires, S est incomplet; notre théorème d'incomplètude de S lie ce phénomène très général à un problème de non compacité des « logiques de

[2] A paraître dans le prochain numéro de la R.A.I.R.O.

Hoare ». Les résultats usuels de complétude pour certains cas particuliers [6, 17] se transposent bien sûr à notre formalisme.

En addition, fortement liée au formalisme des arbres, nous obtenons l'importante règle pour la substitution dans les S.P.S., qui assure la modularité des preuves et permet de traiter sans difficulté des programmes avec instruction de saut.

Les bases de notre formalisme ainsi posées, nous avons jugé utile de mentionner plus succinctement trois extensions (section III) :

1° En conséquence immédiate de la règle pour la substitution, nous avons écrit un système de déduction pour les « schémas de programmes » (selon [9]) ou systèmes d'équations d'actions EXEL [11]. C'est ce système de déduction qui constitue une solution fort simple au problème de la preuve de programmes avec instructions de saut (*voir* par exemple [2, 7]).

2° Un système de preuve pour les programmes avec assertions intermédiaires, qui donne une sémantique à ce type de programmes (sans appels de procédures) ce qui manquait à notre avis, par exemple à [20]. Ce système est à la base d'un générateur de conditions de vérifications (au sens de [20]) pour les programmes EXEL en cours d'implémentation au Laboratoire Central de Recherche Thomson-CSF.

Dans ce formalisme également, Cousineau et nous-mêmes avons commencé une étude des liens entre transformation et preuve de programmes. Nous mentionnons ici un théorème [13] selon lequel une certaine équivalence (l'équivalence syntaxique [9]) préserve la provavilité des programmes.

3° Un calcul de « weakest preconditions » (w.p.) dans $L\omega_1\,\omega$, c'est-à-dire dans un type de langages autorisant des conjonctions et disjonctions dénombrables de formules. En gros, la w.p. associée au programme E et une assertion de sortie Q est mise sous la forme

$$\Phi = \bigwedge_{i \in I} \Phi_i \qquad [\text{Card}(I) = \text{Card}(N)],$$

les Φ_i étant obtenues par un procédé constructif récurrent claqué sur la structure de E.

En définitive, nous espérons en particulier convaincre le lecteur d'un certain nombre d'avantages offerts par le formalisme des arbres aux dites « logiques de Hoare » : une plus grande « transparence » du formalisme et de nouvelles possibilités (instructions de saut; assertions intermédiaires, w.p. avec des formules infinies...).

Dernière remarque : Nous avons pris le parti d'exprimer au maximum les diverses notions de programme et de preuve dans la terminologie et avec les

outils de la théorie des Modèles, qui nous paraissent bien connus et bien maîtrisés. Le lecteur peu averti pourra se référer à un Appendice 1 en fin du présent article. D'autre part, une table des principales notions et notations introduites figure en Appendice 2; nous espérons que la lecture en sera d'autant facilitée.

1. UN SYSTÈME DE DÉDUCTION POUR LES ARBRES DE PROGRAMMES

Préliminaires

Soit L un langage du 1er ordre avec égalité (au sens de la logique mathématique), dénombrable, finitaire ([3]), et $V = \{v_1, \ldots, v_k\}$ un ensemble fini de variables (tout au long de l'article, k désigne le nombre de variables).

Pour tout terme t de L, Var (t) est l'ensemble des variables ayant une occurence dans t. Pour toute formule f de L, Lib (f) est l'ensemble des variables libres de f. Enfin, For (L) [resp. : For (L, V)] est l'ensemble des formules de L. [resp. : formules f de L telles que Lib $(f) \subseteqq V$].

Si f est une formule [resp. : un terme] de L, $v_i \in$ lib (f) [resp. : $v_i \in$ Var (f)] et t est un terme de L :

$[f]_t^{v_i}$ désigne le résultat de la substitution de t à toutes les occurrences (libres si f est une formule) de v_i dans f. Si f est une formule, on suppose qu'aucune variable de f liée n'a d'occurrence dans t; pour qu'il en soit ainsi, on renomme éventuellement certaines variables liées de f.

On note également :

$$[\langle f_1, \ldots f_p \rangle]_t^{v_i} = \langle [f_1]_t^{v_i}, \ldots, [f_p]_t^{v_i} \rangle ;$$

— si $V = \langle v_1, \ldots, v_k \rangle$, $F = \langle t_1, \ldots, t_k \rangle$, $[f]_F^V$ est le résultat de la substitution simultanée de t_1 à v_1, ..., t_k à v_k;

Nous aurons par la suite à utiliser le lemme trivial suivant :

LEMME 1.1. — *Pour toute formule* $P(v_1, \ldots, v_k)$ $(= P(\bar{v}))$ *d'un langage* L, *notons*

$$\exists_t^i P \equiv \exists \bar{y} \Big(\bigwedge_{j \neq i} v_j = y_j \wedge w_i = t(\bar{y}) \wedge P(\bar{y}) \Big).$$

Alors, pour toute théorie T *et toute* $Q(\bar{v})$ *de* L :

$$\mathrm{T} \vdash P \to [Q]_t^{v_i} \quad \Leftrightarrow \quad \mathrm{T} \vdash \exists_t^i P \to Q.$$

([3]) *Voir* Appendice 1.

1.1. Arbres de programme

Nous définissons les ensembles suivants :

L'ensemble des tests sur L et V :

$$\mathrm{Te}(L, V) = \{ r / r \in \mathrm{For}(L, V), \text{ sans quantificateur} \}.$$

L'ensemble des affectations sur L et V :

$$\mathrm{Af}(L, V) = \{ x \leftarrow t / x \in V ;\ t \text{ terme de } L \text{ t.q. } \mathrm{Var}(t) \subseteqq V \}.$$

Remarque : Afin de ne pas alourdir l'exposé par des détails inessentiels à la méthode exposée, nous ne traitons pas le cas d'affectations du genre : $t_1 \leftarrow t_2$ ou t_1 et t_2 sont des termes (par exemple $A[I] \leftarrow t$ ou A est un « tableau »).

Ce que nous faisons dans cet article peut s'étendre aux affectations de ce type au seul prix d'une plus grande complexité de certaines formules.

D'autre part on se donne deux ensembles

$$\{ \omega_i : i < \omega \} \qquad \text{et} \qquad \{ \alpha_j : j < \omega \},$$

dont les éléments sont appelés Symboles Terminaux [ω désigne le 1er ordinal non fini : $\omega = \mathrm{Card}(\mathbb{N})$] soit Ω un nouveau symbole. On posera :

$$S = \{ \omega_i : i < \omega \} \cup \{ \alpha_j : j < \omega \}.$$

Finalement $W(L, V)$ est l'alphabet

$$W(L, V) = \mathrm{Te}(L, V) \cup \mathrm{Af}(L, V) \cup S \cup \{ \Omega \}.$$

On le munit de la graduation g qui vaut 2 sur les tests, 1 sur les affectations et 0 sur $S \cup \{ \Omega \}$.

Arbre de programme

DÉFINITION 1 : Avec les notations de [8], un arbre de programme (A.P.) sur L et V est un élément du magma libre $M^{\infty}(W(L, V))$.

DÉFINITION 2 : Nous utiliserons abondamment le formalisme de Doner [15]. Un *domaine d'arbre* sera une partie D de $\{1, 2\}^*$ vérifiant :

(i) $\forall m \in D, \forall m' \in \{1, 2\}^*, m' < m \Rightarrow m' \in D$;

(ii) $\forall m \in D, \forall i, j \in \{1, 2\}, (i < j \wedge mj \in D) \Rightarrow mi \in D$.

Un A.P. sur L et V est alors une application

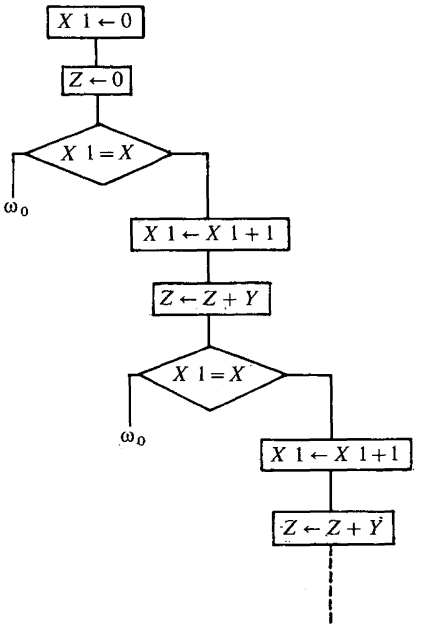
$$a : \quad D \to W(L, V),$$

qui respecte la graduation g sur $W(L, V)$. D est le *domaine de a* :

$$D = \text{dom}\ (a).$$

Nous appellerons $A(L, V)$ l'ensemble des arbres de programme sur L et V.

Exemple 1 : L'arbre a_1 est un arbre de programme calculant $Z = X \times Y$ par additions successives



1.2. Relations et opérations fondamentales sur les arbres de programme

G. Cousineau a étudié ce type de structures algébriques [9, 10]. Nous évoquons ici, avec de très légères modifications, les définitions essentielles.

Ordre : $A(L, V)$ est muni de l'ordre syntaxique : le plus petit ordre stable par les opérations de magma et vérifiant : $\forall a \in A(L, V)$, $\Omega < a$. Ou encore

$$a < b \quad \Leftrightarrow \quad \text{Dom}\ (a) \subseteq \text{Dom}\ (b) \wedge \forall m \in \text{Dom}\ (a)\ a(m) \neq \Omega \quad \Rightarrow \quad b(m) = a(m).$$

Toute partie dirigée de $A(L, V)$ admet un Sup.

Sous-arbre, occurrence

Si $a \in A(L, V)$, $m \in \text{dom}\ (a)$, le sous-arbre de racine m de a est a/m défini par : $(a/m)(m') = a(mm')$. Une occurrence d'un arbre b dans a est un $m \in \{1, 2\}^*$ tel que $a/m = b$.

La Substitution d'un arbre b à toute occurrence d'un symbole terminal $s \in S$ est notée $a[s \setminus b]$; et la substitution en parallèle de $b_1, \ldots, b_e$ à $s_1, \ldots, s_e$: $a[s_1 \setminus b_1, \ldots, s_e \setminus b_e]$.

Concaténation de deux arbres a et b :

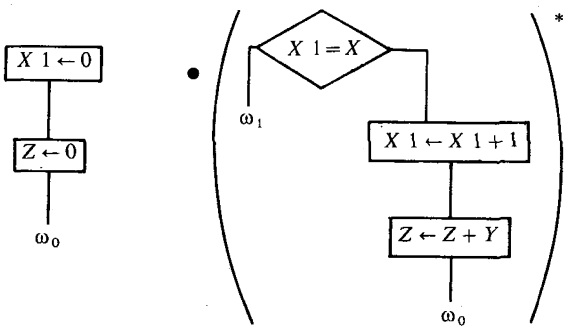
$$a.b = a[\omega_0 \setminus b] \qquad \text{et} \qquad a^n = a.a. \ldots a (n \text{ fois}).$$

Opération $\downarrow$: $\downarrow(a) = a[\omega_0 \setminus \Omega;\ \omega_i \setminus \omega_{i-1} : i > 0\}$.

Opération $*$ (itération) :

$$a^* = \operatorname*{Sup}_{n<\omega} \downarrow(a^n).$$

Exemple 2 : L'écriture



décrit l'arbre a_1 de l'exemple 1.

1.3. Les fonctions Cond et F

Pour tout A.P. a et tout sous-arbre b d'occurrence m dans a, on définit récursivement sur la longueur de m, $|m|$

$\neq$ *une formule de* L : $\mathrm{Cond}_a(m)$;

– *une suite de* k *termes de* L : $F_a(m)$,

de la manière suivante :

– si $m = \varepsilon$ $\qquad \mathrm{Cond}_a(m) = \mathrm{Vrai}$

$$F_a(m) = \langle v_1, \ldots, v_k \rangle$$

– si $a = (v_i \leftarrow t)(a')$, alors $m = 1\ m'$

$$\mathrm{Cond}_a(m) = [\mathrm{Cond}_{a'}(m')]_t^{v_i}$$

$$F_a(m) = [F_{a'}(m')]_t^{v_i}$$

– si $a = r\,(a_1, a_2)$ $(r \in Te(L, V))$
si $m = 1\,m'$: $\mathrm{Cond}_a(m) = r \wedge \mathrm{Cond}\,(a_1, m')$

$$F_a(m) = F_{a_1}(m')$$

si $m = 2\,m'$: $\mathrm{Cond}_a(m) = \neg\, r \wedge \mathrm{Cond}\,(a_2, m')$

$$F_a(m) = F_{a_2}(m')$$

LEMME 1.2 : *Si a et b sont deux A.P., $b < a$, $m \in \mathrm{dom}\,(b)$:*

$$\mathrm{Cond}_a(m) = \mathrm{Cond}_b(m)$$

$$F_a(m) = F_b(m)$$

Démonstration triviale par induction sur $|m|$

Signification intuitive de Cond et F

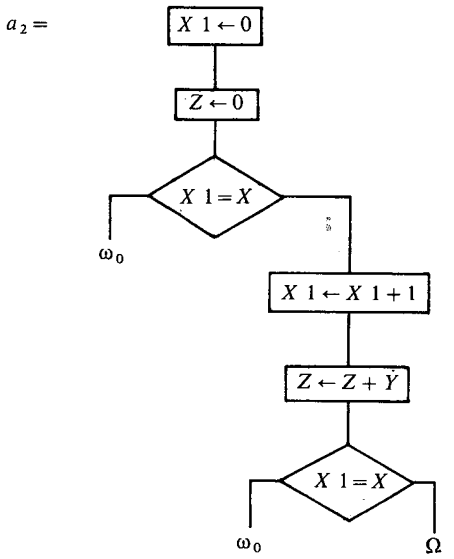
Un A.P. a représente synthétiquement l'ensemble des chemins possibles dans l'exécution d'un programme. m étant une occurrence d'un sous-arbre b :

$\mathrm{Cond}_a(m)$ est la condition sur $v_1, \ldots, v_k$ à l'entrée du programme pour que le calcul parvienne en m.

$F_a(m)$ est la (suite de k termes $\langle t_1, \ldots, t_k \rangle$ représentant la) fonction alors calculée.

Ceci peut être formalisé en utilisant la notion de calcul [9]; le lecteur sera sans doute plus convaincu par l'exemple suivant.

Exemple 3 : Reprenons l'arbre a_1 de l'exemple 1 et soit $a_2 < a_1$:



Le sous-arbre réduit à ω_0 a pour occurrences 111 et 112111; avec $V=\langle X, X1, Y, Z\rangle$:

$$\mathrm{Cond}_{a_1}(111)\equiv(X=0), \qquad F_{a_1}(111)\equiv\langle X, 0, Y, 0\rangle,$$
$$\mathrm{Cond}_{a_1}(112112)\equiv(0\neq X \wedge 0+1\neq X),$$
$$F_{a_1}(112111)\equiv\langle X, 0+1, Y, 0+Y\rangle.$$

De même, Ω a pour occurrence 112112, et

$$\mathrm{Cond}_{a_1}(112112)\equiv(0\neq X\wedge 0+1\neq X),$$
$$F_{a_2}(112112)\equiv F_{a_1}(112111)=\langle X, 0+1, Y, 0+Y\rangle.$$

1.4. Assertions

Soit $(\mathrm{For}(L))^{(\omega)}$ l'ensemble des suites d'éléments de $\mathrm{For}(L)$ dont un nombre fini est différent de Vrai.

Un ensemble d'assertions de sortie sur L est un couple

$$T=(Q, R),$$

avec

$$Q=(Q_i)_{i<\omega}, \qquad \in(\mathrm{For}(L))^{(\omega)},$$
$$R=(R_j)_{j<\omega}, \qquad \in(\mathrm{For}(L))^{(\omega)}.$$

Appelons $\tau(L)$ l'ensemble de ces « doubles suites ».

Autre notation : pour $T\in\tau(L)$ et $s\in S$, notons

$$T_s=Q_i \quad \text{si} \quad s=\omega_i,$$
$$T_s=R_j \quad \text{si} \quad s=\alpha_j.$$

Alors : $T=(T_s)_{s\in S}$.

Moralité : T doit être considéré comme un ensemble de formules associées chacune à un *symbole terminal* (même formule T_s sous toutes les occurrences de s).

1.5. Précondition la plus faible (pour les arbres finis)

A partir de maintenant nous considèrerons deux langages du 1er ordre $L_1\subseteqq L_2$. L_1 servira à écrire les arbres de programme et L_2 les spécifications.

A tout arbre de programme a sur L_1 et V nous associons une application

$$\tilde{a}: \quad \tau(L_2)\to\mathrm{For}(L_2),$$

définie recursivement par :

1° si dom $(a)=\varepsilon$ et $a=\omega_i$, $\tilde{a}(T)=Q_i$

$a=\alpha_j$, $\tilde{a}(T)=R_j$

$a=\Omega$, $\tilde{a}(T)=\text{Vrai}$

2° si $a=(v_i \leftarrow t)(a')$:

$$\tilde{a}(T)=[\tilde{a}'(T)]_t^{v_i};$$

3° si $a=r(a_1, a_2)$:

$$\tilde{a}(T)=(r \rightarrow \tilde{a}_1(T)) \wedge (\neg r \rightarrow \tilde{a}_2(T)).$$

Nous appelons $\tilde{a}(T)$ la précondition la plus faible (p. p. f.) relative à l'arbre (fini) a et aux assertions T. Cette *construction* de « weakest preconditions » (w. p.) est due à Dijkstra [14], et reprise en particulier par de Bakker [4]; la *notion* de w. p. est souvent présente sous une forme plus ou moins explicite dans les travaux sur la complétude des « systèmes de Hoare » et bien sûr dans les logiques algorithmique (A.L. [3]) ou dynamique (D.L. [16]). Nous y reviendrons en 1.7 pour justifier notre construction et en 3 pour esquisser une extension et, ce faisant, une comparaison avec les travaux de De Bakker, *AL* ou *DL*. Relevons ici la commodité avec laquelle la notion de p. p. f. s'exprime dans le formalisme des arbres.

Exemple 4: Reprenons l'arbre a_2 de l'exemple 3. Posons $L_1=\langle +; 0, 1\rangle$ et $L_2=\langle +, \times, 0, 1\rangle$. Soit T telle que $Q_0 \equiv (Z=X \times Y)$:

$$\tilde{a}_2(T) \equiv 0=X \rightarrow 0=X \times Y$$
$$\wedge\ 0 \neq X \rightarrow (0+1=X \rightarrow 0+Y=X+Y \wedge 0+1 \neq X \rightarrow \text{Vrai}).$$

On peut vérifier que

$$\mathbb{N} \models \tilde{a}_2(T) \leftrightarrow [(0=X \rightarrow 0=X \times Y) \wedge (1=X \rightarrow Y=X \times Y)].$$

Le lemme suivant est essentiel pour la suite. Il lie les trois fonctions $\sim$, Cond, et F entre elles :

LEMME 1.3 : *Pour tout arbre fini* $a \in A(L_1, V)$, *tout* $s \in S$ *et toute occurrence* m *de* s *dans* a, *tout* $T \in \tau(L_2)$:

$$\vdash (\tilde{a}(T) \wedge \text{Cond}_a(m)) \rightarrow [T_s]_{F_a(m)}^V.$$

Démonstration par récurrence sur la taille de a, $t(a)$:

— si $t(a)=0$, alors $a=s$ $(\in S)$ et le lemme se réduit au théorème tautologique

$$\vdash (T_s \wedge \text{Vrai}) \rightarrow [T_s]_V^V;$$

— si $t(a)=n+1$, et le lemme est prouvé pour $t(a)\leqq n$, deux cas possibles :
1er *cas* : $a=(v_i \leftarrow t)(a')$, alors $m=1\,m'$ et :

$$\tilde{a}(T)=[\tilde{a}'(T)]_t^{v_i}, \qquad F_a(m)=[F_{a'}(m')]_t^{v_i},$$
$$\mathrm{Cond}_a(m)=[\mathrm{Cond}_{a'}(m')]_t^{v_i}.$$

Mais, par hypothèse de récurrence (H. R.) :

$$\vdash [\tilde{a}'(T) \wedge \mathrm{Cond}_{a'}(m')] \rightarrow [T_s]^V_{F_{a'}(m')}$$
$$\Rightarrow \quad \vdash ([\tilde{a}'(T)]_t^{v_i} \quad [\mathrm{Cond}_a(m')]_t^{v_i} \rightarrow [[T_s]^V_{F_{a'}(m')}]_t^{v_i}$$
$$\Leftrightarrow \quad \vdash (\tilde{a}(T) \wedge \mathrm{Cond}_a(m)) \rightarrow [T_s]^V_{F_a(m)}.$$

2e *cas* : $a=r\,(a_1, a_2)$, et par exemple $m=1\,m_1$. Alors :

$$\tilde{a}(T)=(r \rightarrow \tilde{a}_1(T) \wedge \neg\, r \rightarrow \tilde{a}_2(T)),$$
$$\mathrm{Cond}_a(m)=r \wedge \mathrm{Cond}_{a_1}(m_1), \qquad F_a(m)=F_{a_1}(m_1).$$

Par H.R.

$$\vdash (\tilde{a}_1(T) \wedge \mathrm{Cond}_{a_1}(m_1)) \rightarrow [T_s]^V_{F_{a_1}(m_1)}.$$

Et on vérifie que :

$$\vdash (\tilde{a}(T) \wedge \mathrm{Cond}_a(m)) \leftrightarrow (r \wedge \tilde{a}_1(T) \wedge \mathrm{Cond}_{a_1}(m_1)). \quad \square$$

DÉFINITIONS :

$$\star \quad T-Q_0=T' \quad \Leftrightarrow \quad (R'=R \text{ et } \forall i<\omega\, Q'_i=Q_{i+1}),$$
$$\star \quad \forall k<\omega, \quad \forall F \in \mathrm{For}(L_2),$$
$$T(k, F)=((Q_0, Q_1, \ldots, Q_{k-1}, F, Q_k, \ldots), R).$$

Par la même technique on démontre :

LEMME 1.4 : $\vdash (a_1 \tilde{;} a_2)(T)=\tilde{a}_1((T-Q_0)(0, \tilde{a}_2(T))$.

1.6. Le système de déduction A

Nous sommes maintenant en mesure de donner la principale définition de ce chapitre.

Nous considérons les triplets (P, a, T), que nous notons : $[P]\,a\,[T]$ où : $a \in A$ (L_1, V), $P \in \mathrm{For}\,(L_2)$, et $T \in \tau(L_2)$; et d'autre part les théories T du 1er ordre écrites dans L_2.

Le système formel A est constitué de :

— *les axiomes :*

$$A_0 : \quad \mathrm{T} \underset{A}{\Vdash} [P]\, a\, [T]$$

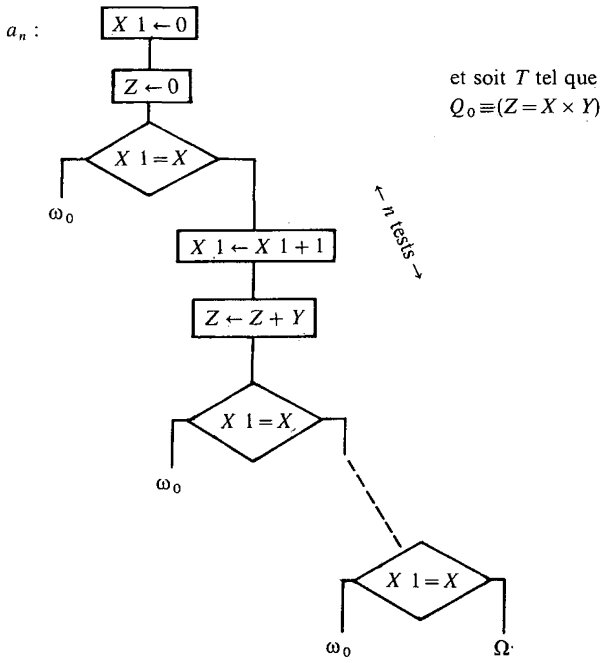
si a est fini et

$$\mathrm{T} \vdash P \to \tilde{a}(T);$$

— *la règle « de la borne supérieure » (dénombrable)* : si $(a_d)_{d \in D}$ est une famille dirigée (dénombrable) d'arbres de programmes

$$A_1 : \quad \frac{\forall\, d \in D,\ \mathrm{T} \underset{A}{\Vdash} [P]\, a_d\, [T]}{\mathrm{T} \underset{A}{\Vdash} [P] \operatorname{Sup}_{d \in D} a_d\, [T]}.$$

Exemple de preuve (5) : Poursuivant les exemples précédents, pour tout n appelons a_n $(a_n < a)$ l'arbre :



et soit T tel que $Q_0 \equiv (Z = X \times Y)$

et soit T tel que $Q_0 \equiv (Z = X \times Y)$.

Nous avons déjà calculé $\tilde{a}_2(T)$ et vérifié que

$$\mathbb{N} \models \tilde{a}_2(T) \leftrightarrow [(X = 0 \to X \times Y = 0) \wedge (X = 1 \to X \times Y = Y)],$$

ce qui peut encore s'écrire, en appelant Th (ℕ) la théorie de ℕ dans L_2 (ensemble des énoncés de L_2 vrais dans ℕ) :

$$\mathrm{Th}(\mathbb{N}) \vdash \tilde{a}_2(T) \leftrightarrow [(X=0 \rightarrow \ldots].$$

De même nous aurons, pour n quelconque et en posant

$$\bar{n} = 1+1+\ldots+1 \ (n \text{ fois}), \qquad n\,Y = Y+Y+\ldots+Y \ (n \text{ fois}) :$$

$$\mathrm{Th}(\mathbb{N}) \vdash \tilde{a}_n(T) \leftrightarrow [(X=0 \rightarrow X \times Y = 0) \wedge \ldots \wedge (X=\bar{n} \rightarrow X \times Y = n\,Y)].$$

Il en résulte que pour tout n :

$$\mathrm{Th}(\mathbb{N}) \underset{A}{\Vdash} [\mathrm{Vrai}]\, a_n[T],$$

est un axiome, et la déduction

$$\frac{\forall n < \omega, \quad \mathrm{Th}(\mathbb{N}) \underset{A}{\Vdash} [\mathrm{vrai}]\, a_n[T]}{\mathrm{Th}(\mathbb{N}) \underset{A}{\Vdash} [\mathrm{vrai}]\, a[T]},$$

est valide. □

DÉFINITION : L'ensemble des *conséquences de T dans A* est le plus petit ensemble :

– contenant les triplets $[P]\,a[T]$ tels que $T \underset{A}{\Vdash} [P]\,a[T]$ est un axiome;

– clos par application de la règle A_1.

Donc : $[P]\,a[T]$ est conséquence de $T \Leftrightarrow T \underset{A}{\Vdash} [P]\,a[T]$.

De cette définition découle la :

Règle d'induction : Soit Π (a, T, P, T) une propriété quelconque. Pour démontrer

$$\mathrm{T} \underset{A}{\Vdash} [P]\,a[T] \quad \Rightarrow \quad \Pi(a, \mathrm{T}, P, T),$$

il suffit de prouver :

1° si a est fini et $\mathrm{T} \underset{A}{\Vdash} [P]\,a[T]$ est un axiome, alors : $\Pi(a, \mathrm{T}, P, T)$ est vraie;

2° si $a = \mathrm{Sup}(a_d)$ et $\forall d \in D$ $\Pi(a_d, \mathrm{T}, P, T)$ est vraie, alors : $\Pi(a, \mathrm{T}, P, T)$ est vraie.

1.7. Arbres finis

La première proposition que nous démontrons indique que pour les arbres *finis*, le système A se réduit aux seuls axiomes. Elle légitime du même coup l'appellation de « Précondition la plus faible » pour $\tilde{a}(T)$:

PROPOSITION 1.5 : *Si a est fini*

$$T \underset{A}{\Vdash} [P]a[T] \quad \Leftrightarrow \quad T \vdash P \to \tilde{a}[T].$$

Preuve : Dans un sens ($\Leftarrow$), c'est une simple reconnaissance des axiomes A_0. Dans l'autre, nous appliquons la règle d'induction :

1° si $T \underset{A}{\Vdash} [P]a[T]$ est un axiome, alors précisément $T \vdash P \to \tilde{a}(T)$;

2° Supposons : $a = \underset{d \in D}{\mathrm{Sup}}(a_d)$, et $\forall d \in D$, $T \vdash P \to \tilde{a}_d(T)$. Puisque a est fini, il existe certainement un $d_0 \in D$ tel que $a = d_0$; donc $T \vdash P \to \tilde{a}(T)$. □

1.8. Le théorème fondamental

Le théorème que nous énonçons maintenant est essentiel puisque c'est lui qui légitime notre théorie :

THEORÈME FONDAMENTAL 1.6 : *Pour tous* T, P, a, T :

(A) $T \underset{A}{\Vdash} [P]a[T]$

$\Leftrightarrow$

(B) $\forall s \in S$, $\forall m$ occurrence de s dans a :

$$T \vdash (P \wedge \mathrm{Cond}_a(m)) \to [T_s]^V_{F_a(m)}.$$

Démonstration :

— *Dans le sens* $(A) \Rightarrow (B)$, appliquons la règle d'induction :

1° si $T \underset{A}{\Vdash} [P]a[A]$ est un axiome, alors a est fini et $T \vdash P \to \tilde{a}(T)$. Il suffit donc d'appliquer le lemme 1.3 :

$$\forall s, \quad \forall m, \qquad \vdash (\tilde{a}(T) \wedge \mathrm{Cond}_a(m)) \to [T_s]^V_{F_a(m)};$$

2° Supposons que $a = \underset{d \in D}{\mathrm{Sup}}(a_d)$ et que B est vraie pour tout a_d, $d \in D$. Soit $s \in S$ et m une occurrence de s dans a; il existe certainement un $d_0 \in D$ tel que

$$m \in \mathrm{Dom}(a_{d_0}) \qquad \text{et} \qquad a_{d_0}(m) = a(m) = s,$$

B étant vraie pour a_{d_0} :

$$T \vdash (P \wedge \mathrm{Cond}_{a_{d_0}}(m)) \to [T_s]^V_{F_{a_{d_0}}(m)}.$$

Mais $a_{d_0} < a$ et $m \in \mathrm{Dom}(a_{d_0})$; d'où, par le lemme 1.2 :

$$\mathrm{Cond}_{a_{d_0}}(m) = \mathrm{Cond}_a(m) \qquad \text{et} \qquad F_{a_{d_0}}(m) = F_a(m).$$

– Pour démontrer le théorème dans le sens $(B) \Rightarrow (A)$, nous distinguerons deux cas selon que a est fini ou non.

1er *cas* : a est fini. Nous supposons (B) vrai et montrons (A) par récurrence sur la taille de a, $t(a)$.

– Si $t(a) = 0$:

soit $a = \Omega$, $\tilde{a}(T) =$ vrai et $T \underset{A}{\Vdash} [P]\, a\, [T]$ est donc un axiome;

soit $a = s \in S$; alors $m = \varepsilon$, $\mathrm{Cond}_a(m) =$ Vrai, $F_a(m) = V$ et l'hypothèse (B) s'écrit :

$$T \vdash P \to T_s;$$

soit :

$$T \vdash P \to \tilde{a}(T).$$

$T \vdash P \to \tilde{a}(T)$ est donc encore un axiome.

– Si $t(a) = n+1$, (A) étant vraie dès que (B) est vraie et que $t(a) \leqq n$.

Nous avons deux possibilités selon que $a(\varepsilon)$ est une affectation ou un test. Traitons par exemple le cas où $a = (v_i \leftarrow t)(a')$ l'autre étant tout à fait similaire.

Soit $s \in S$ et m une occurrence de s dans a; $m = 1\, m'$, m' étant une occurrence de s dans a'. Nous avons:

$$\begin{aligned} T \vdash (P \wedge \mathrm{Cond}_a(m)) \to [T_s]^V_{F_a(m)} \quad & \text{par } B \\ \Leftrightarrow \quad & T \vdash (P \wedge [\mathrm{Cond}_{a'}(m')]^{v_i}_t) \to [[T_s]^V_{F_{a'}(m')}]^{v_i}_t \\ \Leftrightarrow \quad & T \vdash P \to [\mathrm{Cond}_{a'}(m') \to [T_s]^V_{F_{a'}(m')}]^{v_i}_t \\ \Leftrightarrow \quad & T \vdash (\exists^i_t P) \to (\mathrm{Cond}_{a'}(m') \to [T_s]^V_{F_{a'}(m')}), \end{aligned}$$

par le lemme 1.1.

Ceci est vrai quels que soient $s \in S$ et l'occurrence m de s dans a, donc quelle que soit l'occurrence m' de s dans a'. Par hypothèse de récurrence, nous aurons donc

$$\mathrm{T} \underset{A}{\Vdash} [\exists_t^i P]\, a'[T]$$

$$\Rightarrow \quad \mathrm{T} \vdash (\exists_t^i P) \to \tilde{a}'(T) \quad \text{par la proposition 1.5}$$

$$\Leftrightarrow \quad \mathrm{T} \vdash P \to [\tilde{a}'(T)]_t^{v_i} \qquad \text{(lemme 1.1)}$$

$$\Leftrightarrow \quad \mathrm{T} \vdash P \to \tilde{a}(T)$$

$$\Leftrightarrow \quad \mathrm{T} \underset{A}{\Vdash} [P]\, a[T] \qquad \text{(proposition 1.5).}$$

2e cas : a n'est pas fini.

DÉFINITION : Pour tout arbre a et tout entier $n \geqq 0$, $a \mid n$ est l'arbre défini par

$$\mathrm{Dom}(a \mid n) = \{ m \in \mathrm{Dom}(a) / |m| \leqq n \},$$

$$a \mid n(m) = a(m) \quad \text{si} \quad |m| < n,$$

$$a \mid n(m) = \Omega \quad \text{si} \quad |m| = n$$

(coupe de l'arbre a à la profondeur n).

Trivialement, pour tout a : $a = \underset{n<\omega}{\mathrm{Sup}}\,(a \mid n)$.

Supposons (B) vrai. Pour tout n entier, $s \in S$, m occurrence de s dans $a \mid n$, nous aurons donc (m est encore une occurrence de s dans a) :

$$\mathrm{T} \vdash (P \wedge \mathrm{Cond}_a(m)) \to [T_s]^V_{F_a(m)}$$

$$\Leftrightarrow \quad \mathrm{T} \vdash (P \wedge \mathrm{Cond}_{a \mid n}(m)) \to [T_s]^V_{F_{a \mid n}(m)} \quad \text{(lemme 1.2),}$$

d'où, $a \mid n$ étant fini, d'après ce que nous venons de démontrer

$$\forall n < \omega, \quad \mathrm{T} \underset{A}{\Vdash} [P]\, a \mid n[T]$$

et il ne reste plus qu'à appliquer la règle A_1 pour obtenir

$$\mathrm{T} \underset{A}{\Vdash} [P]\, a[T]. \quad \square$$

Notons tout de suite le corollaire suivant :

PROPOSITION 1.7 : *Si* $a < b$ *et* $\mathrm{T} \underset{A}{\Vdash} [P]\, b\, [T]$, *alors* $T \underset{A}{\Vdash} [P]\, a[T]$.

Preuve : Soit $s \in S$ et m une occurrence de s dans a ; m est encore une occurrence de s dans b. Donc, si $T \underset{A}{\Vdash} [P]\, b\, [T]$:

$$T \vdash (P \wedge \mathrm{Cond}_b(m)) \to [T_s]^V_{F_b(m)}$$

$$\Leftrightarrow \quad T \vdash (P \wedge \mathrm{Cond}_a(m)) \to [T_s]^V_{F_b(m)} \quad \text{(lemme 1.2).}$$

D'où : $T \underset{A}{\Vdash} [P]\, a\, [T]$ par le théorème fondamental.

Signification du théorème fondamental : Consistance et complétude du système de déduction A.

Nous sommes maintenant en mesure de prouver que A est bien un système de preuve de l'exactitude partielle des programmes représentés sous forme d'arbres.

DÉFINITIONS : Un A.P. a est dit *partiellement* exact (p. e.) pour P et T dans un modèle $\mathcal{M}$ de L_2 de domaine M ssi : $\forall s \in S$, $\forall m$ occurrence de s dans a, $\forall \mu \in M^k$:

$$\mathcal{M} \models P(\mu) \wedge \mathrm{Cond}_a(m)(\mu) \quad \Rightarrow \quad \mathcal{M} \models T_s(F_a(m)(\mu)).$$

Notation : $\mathcal{M} \models [P]\, a\, [T]$.

On dira aussi que $\mathcal{M}$ *satisfait* ou *est un modèle du* triplet $[P]\, a\, [T]$.

Un A.P. a est p. e. pour P et T *modulo la théorie* T ssi

$$\forall \mathcal{M}, \qquad \mathcal{M} \models T \Rightarrow \mathcal{M} \models [P]\, a\, [T].$$

Notation : $T \models [P]\, a\, [T]$.

Intuitivement, d'après l'interprétation donnée de $\mathrm{Cond}_a(m)$ et $F_a(m)$, nous voyons qu'un A.P. a est p. e. pour P et T dans $\mathcal{M}$ ssi chaque fois que la valeur d'entrée μ des variables $v_1, \ldots, v_K$ satisfait :

1° P;

2° $\mathrm{Cond}_a(m)$ — c'est-à-dire la condition pour que le calcul suive le chemin conduisant à m;

alors, la valeur finale des variables $F_a(m)(\mu)$ satisfait l'assertion de sortie associée à s, $T_s.(a(m) = s)$.

C'est donc bien une formalisation de la notion d'exactitude partielle que nous avons définie. Notion quelque peu étendue dans la mesure où nous pouvons marquer différents exits par différents symboles $s \in S$, et leur associer différentes assertions T_s. Nous verrons que ce phénomène confère beaucoup de souplesse à notre théorie et milite donc fortement pour l'utilisation du formalisme des *arbres*.

Nous pouvons alors reformuler le théorème fondamental en utilisant la notion d'exactitude partielle introduite. En effet, par complétude du calcul des prédicats

$$\begin{aligned} \mathrm{T} \vdash (P \wedge \mathrm{Cond}_a(m)) \rightarrow [T_s]^V_{F_a(m)} \\ \Leftrightarrow \quad (\forall \mathscr{M},\ \mathscr{M} \models \mathrm{T} \Rightarrow \mathscr{M} \models (P \wedge \mathrm{Cond}_a(m)) \rightarrow [T_s]^V_{F_a(m)}) \\ \Leftrightarrow \quad (\forall \mathscr{M},\ \mathscr{M} \models \mathrm{T} \\ \Rightarrow (\forall \mu \in M^k,\ \mathscr{M} \models P(\mu) \wedge \mathrm{Cond}_a(m)(\mu) \ \Rightarrow \mathscr{M} \models T_s(F_a(m)(\mu))). \end{aligned}$$

D'où, par le théorème fondamental

$$\mathrm{T} \underset{A}{\Vdash\!\!-} [P]\, a\, [T] \ \Leftrightarrow \ (\mathscr{M} \models \mathrm{T} \Rightarrow \mathscr{M} \models [P]\, a\, [T]),$$

ou, de manière encore plus condensée :

COROLLAIRE 1.8 :

$$\mathrm{T} \underset{A}{\Vdash\!\!-} [P]\, a\, [T] \ \Leftrightarrow \ \mathrm{T} \models [P]\, a\, [T].$$

C'est en ce sens que nous pouvons qualifier le système A de *consistant* et *complet*.

Remarque complémentaire sur la notation $T \underset{A}{\Vdash\!\!-} [P]\, a\, [T]$ *et la notion d'exactitude partielle introduite*

La notion d'exactitude partielle et de complétude que l'on trouve généralement dans la littérature sont relatives à *une structure* (un modèle de L_2) *particulière*. Pour nous par contre, les démonstrations se font toujours modulo une théorie T ($\mathrm{T} \underset{A}{\Vdash\!\!-} [P]\, a\, [T]$); parallèlement, c'est la notion d'exactitude partielle modulo T ($\mathrm{T} \models [P]\, a\, [T]$) qui est fondamentale et qui sert à définir la complétude. Autrement dit, notre approche est entièrement syntaxique : la formulation de notre théorème fondamental est à cet égard exemplaire. Nous voulons dans cette remarque préciser le rapport entre ces deux démarches : relatives à une *structure* ou à une *théorie*.

Rappels :

– deux modèles $\mathscr{M}$ et $\mathscr{N}$ d'un langage L sont dits élémentairement équivalents ($\mathscr{M} \equiv \mathscr{N}$) ssi ils satisfont les mêmes énoncés de L;

– la théorie d'un modèle $\mathscr{M}$ de L ($\mathrm{Th}(\mathscr{M})$) est l'ensemble des énoncés de L satisfaits par $\mathscr{M}$;

– une classe C de modèles d'un langage L est une classe élémentaire (généralisée) ssi il existe une théorie T telle que $C = \{ \mathcal{M} / \mathcal{M} \models T \}$ clairement une telle classe est stable par équivalence élémentaire et :

$$\mathcal{M} \equiv \mathcal{N} \Leftrightarrow \mathrm{Th}(\mathcal{M}) = \mathrm{Th}(\mathcal{N}).$$

Considérons alors la classe des modèles d'un même triplet $[P]a[T]$. Nous avons vu que c'est

$$\{ \overline{\mathcal{M} / \forall s \forall m \text{ occurrence de } s \text{ dans } a,\ \mathcal{M} \models (P \wedge \overline{\mathrm{Cond}_a(m)} \rightarrow [T_s]^{V}_{F_a(m)})} \}.$$

D'où :

PROPOSITION 1.9 : *La classe des modèles d'un même triplet* $[P]a[T]$ *est élémentaire. Elle est donc close par équivalence élémentaire.*

COROLLAIRE 1.10 : *Les deux propositions suivantes sont équivalentes :*

(i) *a est p.e. pour P et T dans* $\mathcal{M}$ *;*

(ii) *a est p.e. pour P et T modulo* $\mathrm{Th}(\mathcal{M})$.

Toute preuve relative à une *structure* $\mathcal{M}$ peut donc se ramener à une preuve relative à une *théorie* ($T \underset{A}{\models} [P]a[T]$ avec $T = \mathrm{Th}(\mathcal{M})$), notion plus étendue, puisque T peut n'être pas complète; ce qui signifie : on n'a peut-être pas besoin d'une information complète sur $\mathcal{M}$ pour prouver un triplet $[P]a[T]$. Nous reviendrons sur cet aspect des choses en 2.6.

Inversement : Pour tout $\mathcal{M}$ soit

$$\mathrm{Th}'(\mathcal{M}) = \{ [P]a[T] / \mathcal{M} \models [P]a[T] \}$$

et posons

$$\mathcal{M} \approx \mathcal{N} \quad \Leftrightarrow \quad (\forall P, a, T\ \mathcal{M} \models [P]a[T] \Leftrightarrow \mathcal{N} \models [P]a[T]) \Leftrightarrow \quad \mathrm{Th}'(\mathcal{M}) = \mathrm{Th}'(\mathcal{N}),$$

clairement : $\mathcal{M} \approx \mathcal{N} \Rightarrow \mathcal{M} \equiv \mathcal{N}$ (considérer les triplets $[\text{vrai}]\omega_0[\sigma, \text{vrai} \ldots]$ pour $\mathcal{M} \models \sigma$ ou $\mathcal{N} \models \sigma$).

Ce fait, joint à la proposition 1.9 donne donc le résultat suivant noté par Wand [26] :

PROPOSITION 1.11 : *L'ensemble des triplets* $[P]a[T]$ *satisfaits dans un modèle* $\mathcal{M}$ *est caractéristique de la théorie de* $\mathcal{M}$

$$\mathrm{Th}'(\mathcal{M}) = \mathrm{Th}'(\mathcal{N}) \quad \Leftrightarrow \quad \mathrm{Th}(\mathcal{M}) = \mathrm{Th}(\mathcal{N}).$$

1.9 Règles dérivées

Nous cherchons maintenant à dériver dans A des règles de déduction correspondant aux opérations fondamentales sur les arbres (opérations de magma, d'$*$, de substitution) et les formules de $L_2(\wedge, \vee, \rightarrow)$. Nous allons ainsi obtenir l'ensemble des « règles de Hoare » ou, si l'on veut, une traduction de ces règles dans notre formalisme.

1) *Première règle pour* $\rightarrow$ (A_{21})

Elle s'énonce

$$A_{21} : \frac{T \underset{A}{\Vdash} [P]a[T]}{T \underset{A}{\Vdash} [P']a[T]},$$

sous la *condition* $T \vdash P' \rightarrow P$ et se démontre sans problème par application de la règle d'induction.

2) *Règles « de Magma »* (A_{51}, A_{52})

$$A_{51} : \frac{T \underset{A}{\Vdash} [P]a[T]}{T \underset{A}{\Vdash} [P_t^{v_i}](v_i \leftarrow t)(a)[T]},$$

$$A_{52} : \frac{T \underset{A}{\Vdash} [P \wedge r]a_1[T] \quad T \underset{A}{\Vdash} [P \wedge \neg r]a_2[T]}{T \underset{A}{\Vdash} [P]r(a_1, a_2)[T]},$$

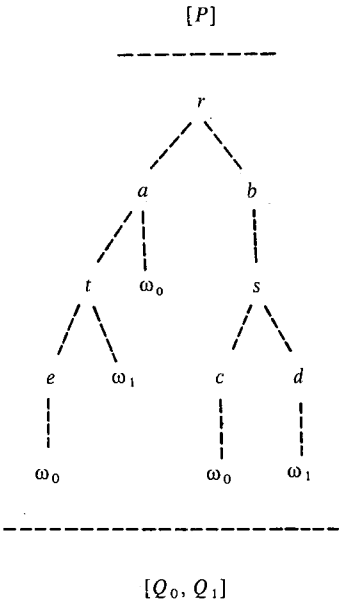
(appliquer toujours la règle d'induction).

3) *Règle pour la substitution* (A_6)

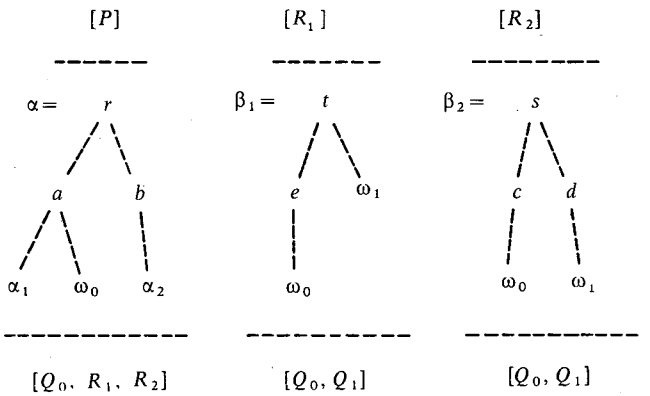
$$A_6 : \frac{T \underset{A}{\Vdash} [P]a[T], \quad \forall p = 1, \ldots, m, \quad T \underset{A}{\Vdash} [T_{s_p}]b_p[T']}{T \underset{A}{\Vdash} [P]a[s_p \setminus b_p : p = 1, \ldots, m][T']},$$

sous la condition de cohérence : $\forall s \in S \setminus \{s_1, \ldots, s_m\}$, ayant une occurrence au moins dans a : $T_s = T'_s$.

Exemple 6 : Avec des notations évidentes, et en « oubliant » les assertions de sortie « superflues », la règle A_6 dit que pour prouver :



il suffit de prouver séparément les trois arbres suivants :



pour un certain R_1 et un certain R_2 (r, s, t, sont des tests; a, b, c, d, e des affectations).

Démonstration de la règle A_6 : Nous appliquons la règle d'induction en considérant la 1re prémisse : $T \underset{A}{\vdash} [P]\,a\,[T]$.

1) S'il s'agit d'un axiome : alors a est fini et nous opérons une récurrence sur la taille de a, $t(a)$.

– $t(a)=0$: $a=s\in S$ ou Ω :

si $a=\Omega$ TRIVIAL;

si $a=s\in S$, $s\notin\{s_1, \ldots, s_m\}$ alors $T_s=T'_s$ (condition de cohérence) et

$$a[s_p\backslash b_p : p=1, \ldots, m]=s=a.\text{TRIV.};$$

si $a=s\in\{s_1, \ldots, s_m\}$, par exemple $a=s_1$.

Par la 1re prémisse : $T \underset{A}{\Vdash\!\!-} [P]\,a[T] \Rightarrow T \vdash P \to T_{s_1}$.

D'autre part : $T \underset{A}{\Vdash\!\!-} [T_{s_1}]\,b_1[T']$ fait partie des prémisses; donc, par applications de la règle A_{21} : $T \underset{A}{\Vdash\!\!-} [P]\,b_1[T']$.

Or : $a=s_1 \Rightarrow a[s_p\backslash b_p : p=1, \ldots, m]=b_1$.

– Si la règle est valide pour $t(a)\leqq n$ et si nous supposons maintenant $t(a)=n+1$.

Soit par exemple : $a=(v_i \leftarrow t)(a')$ [le cas $a=r(a_1, a_2)$ se traite de façon similaire]. Alors

$T \underset{A}{\Vdash\!\!-} [P]\,a[T]$ (1re prémisse)

$$\Leftrightarrow T \vdash P \to \tilde{a}(T) \quad \text{(proposition 1.5)}$$

$$\Leftrightarrow T \vdash P \to [\tilde{a}'(T)]_t^{v_i}$$

$$\Leftrightarrow T \vdash (\exists_t^i P) \to \tilde{a}'(T).$$

Donc, par H.R.

$$T \underset{A}{\Vdash\!\!-} [\exists_t^i P]\,a'[s_p\backslash b_p : p=1 \ldots m][T']$$

et par

$$A_{51} : \quad T \underset{A}{\Vdash\!\!-} [\exists_t^i P]_t^{v_i}(v_i \leftarrow t)(a'[s_p\backslash b_p : p=1 \ldots m])[T']$$

$$\Leftrightarrow T \underset{A}{\Vdash\!\!-} [\exists_t^i P]_t^{v_i}\,a[s_p\backslash b_p : p=1 \ldots m][T'].$$

or : $\vdash P \to [\exists_t^i P]_t^{v_i} \Leftrightarrow \vdash (\exists_t^i P) \to (\exists_t^i P)$.

Donc, en appliquant A_{21}, nous obtenons bien

$$T \underset{A}{\Vdash\!\!-} [P]\,a[s_p\backslash b_p : p=1 \ldots m][T'].$$

2) Supposons maintenant que $a = \operatorname{Sup}_{d \in D}(a_d)$, et que la règle A_6 est valide avec $T \underset{A}{\Vdash} [P]\, a_d [T]$ comme première prémisse pour tout $d \in D$.

Si la condition de cohérence est satisfaite pour a, elle l'est aussi pour tout a_d, $d \in D$. En effet, une occurrence de s dans $a_d (< a)$ est encore une occurrence de s dans a. Nous pouvons donc appliquer la règle A_6 pour chaque a_d séparément et obtenir

$$\forall d \in D, \quad T \underset{A}{\Vdash} [P]\, a_d [s_p \backslash b_p : p = 1 \ldots m] [T']$$

D'où, par A_1

$$T \underset{A}{\Vdash} [P] \operatorname{Sup}_{d \in D} (a_d [s_p \backslash b_p : p = 1 \ldots m]) [T'].$$

Le résultat final découle alors de la continuité de l'opération de substitution

$$\operatorname{Sup}_{d \in D} (a_d [s_p \backslash b_p : p = 1 \ldots m]) = (\operatorname{Sup}_{d \in D} a_d) [s_p \backslash b_p : p = 1 \ldots m]. \quad \square$$

3) *Règle pour la concaténation* (A_7)

Comme conséquence immédiate de A_6, puisque la concaténation de deux arbres est définie par

$$a_1 \cdot a_2 = a_1 [\omega_0 \backslash a_2].$$

Nous obtenons la règle

$$A_7 : \quad \frac{T \underset{A}{\Vdash} [P]\, a_1 [T], \qquad T \underset{A}{\Vdash} [Q_0]\, a_2 [T']}{T \underset{A}{\Vdash} [P]\, a_1 \cdot a_2 [T']},$$

sous la condition de cohérence : $\forall s \in S \backslash \{\omega_0\}$, s ayant une occurrence dans a, $T_s = T'_s$.

4) *Règle pour l'opération* $*$ (A_8)

$$A_8 : \quad \frac{T \underset{A}{\Vdash} [P]\, a [T(0, P)]}{T \underset{A}{\Vdash} [P]\, a^* [T]}.$$

Schémas de la démonstration :

– On démontre d'abord par induction sur $n < \omega$ et en utilisant A_7 que

$$T \underset{A}{\Vdash} [P]\, a [T(0, P)] \Rightarrow \forall n, \quad 1 \leqq n < \omega, \qquad T \underset{A}{\Vdash} [P] (a^n \mid n) [T(0, P)].$$

– Pour tout $n<\omega$, $a^n \mid n$ est fini. Posons :

$$h(n)=\text{Max}\{\, i/\omega_i \text{ a une occurrence dans } a^n \mid n \,\}.$$

Si $T'=[T(0, P)]$:

$$\forall i=1 \ldots h(n), \quad \mathrm{T} \underset{A}{\Vdash} [Q'_i]\,\omega_{i-1}[T]$$

et trivialement

$$\mathrm{T} \underset{A}{\Vdash} [Q'_0]\,\Omega[T].$$

D'où l'inférence suivante (A_6) : $\forall i=1 \ldots h(n)$:

$$\frac{\mathrm{T} \underset{A}{\Vdash} [P]\,a^n \mid n[T'], \qquad \mathrm{T} \underset{A}{\Vdash} [Q'_0]\,\Omega[T], \qquad \mathrm{T} \underset{A}{\Vdash} [Q'_i]\,\omega_{i-1}[T]}{\mathrm{T} \underset{A}{\Vdash} [P]\,(a^n \mid n)[\omega_0 \setminus \Omega;\ \omega_i \setminus \omega_{i-1} : i=1 \ldots h(n)][T]},$$

c'est-à-dire

$$\mathrm{T} \underset{A}{\Vdash} [P]\downarrow(a^n \mid n)[T] \quad \Leftrightarrow \quad \mathrm{T} \underset{A}{\Vdash} [P](\downarrow(a_n)) \mid n[T].$$

– Finalement : $a* = \underset{n<\omega}{\text{Sup}} \downarrow(a^n) = \underset{n<\omega}{\text{Sup}} (\downarrow(a_n)) \mid n.$

Il suffit donc d'appliquer A_1. □

5) *Autres règles pour les connecteurs logiques*

Notation : Pour $T, T' \in \tau(L_2, v)$, on pose :

$$-T \vee T' = T'' \quad \Leftrightarrow \quad \forall s \in S, \quad T''_s = T_s \vee T'_s$$

et de même pour $\wedge$ et $\rightarrow$

$$-\mathrm{T} \vdash T \quad \Leftrightarrow \quad \forall s \in S, \quad \mathrm{T} \vdash T_s.$$

– Une application de la règle pour la substitution (A_6), faisant intervenir les prémisses : $\mathrm{T} \underset{A}{\Vdash} [T_s]\,s[T']$ donne la règle :

$$A_{22} : \quad \frac{\mathrm{T} \underset{A}{\Vdash} [P]\,a[T]}{\mathrm{T} \underset{A}{\Vdash} [P]\,a[T']} \qquad \text{si} \quad \mathrm{T} \vdash T \rightarrow T'.$$

– En utilisant de nouveau la règle d'induction on prouve la validité des deux règles

$$A_3 : \quad \frac{T \underset{A}{\Vdash} [P]a[T], \qquad T \underset{A}{\Vdash} [P']a[T]}{T \underset{A}{\Vdash} [P \vee P']a[T]},$$

$$A_4 : \quad \frac{T \underset{A}{\Vdash} [P]a[T], \qquad T \underset{A}{\Vdash} [P]a[T']}{T \underset{A}{\Vdash} [P]a[T \wedge T']}$$

[utiliser pour A_4, le *lemme* : $\vdash \tilde{a}(T \wedge T') \leftrightarrow (\tilde{a}(T) \wedge \tilde{a}(T'))$ qui se démontre trivialement par induction sur la taille de a].

Rassemblons l'ensemble de ces règles de déduction :

Le système de déduction A

A_0 : *Axiomes*

$$T \underset{A}{\Vdash} [P]a[T] \qquad \text{si} \quad a \text{ est fini} \quad \text{et} \quad T \vdash P \to \tilde{a}(T).$$

A_1 : *Règle de la borne supérieure* (dénombrable)

$$\frac{\forall d \in D, \quad T \underset{A}{\Vdash} [P]a_d[T]}{T \underset{A}{\Vdash} [P] \underset{d \in D}{\mathrm{Sup}} a_d[T]}.$$

Règles dérivées :

– *Connecteurs logiques*

$$A_2 : \quad \frac{T \underset{A}{\Vdash} [P]a[T]}{T \underset{A}{\Vdash} [P']a[T]} \qquad \text{si} \qquad T \vdash P' \to P,$$

$$A_{22} : \quad \frac{T \underset{A}{\Vdash} [P]a[T]}{T \underset{A}{\Vdash} [P]a[T']} \qquad \text{si} \quad T \vdash T \to T',$$

$$A_3 : \quad \frac{T \underset{A}{\Vdash} [P]a[T], \qquad T \underset{A}{\Vdash} [P']a[T]}{T \underset{A}{\Vdash} [P \vee P']a[T]},$$

$$A_4 : \quad \frac{T \underset{A}{\Vdash} [P]a[T], \qquad T \underset{A}{\Vdash} [P]a[T']}{T \underset{A}{\Vdash} [P]a[T \wedge T']}.$$

— *Opérations de Magma*

$$A_{51} : \frac{\mathrm{T} \underset{A}{\Vdash} [P]\, a\, [T]}{\mathrm{T} \underset{A}{\Vdash} [P_t^{v_i}]\, (v_i \leftarrow t)(a)\, [T]},$$

$$A_{52} : \frac{\mathrm{T} \underset{A}{\Vdash} [P \wedge r]\, a_1\, [T], \qquad \mathrm{T} \underset{A}{\Vdash} [P \wedge \neg\, r]\, a_2\, [T]}{\mathrm{T} \underset{A}{\Vdash} [P]\, r(a_1, a_2)\, [T]}.$$

— *Substitution*

$$A_6 : \frac{\mathrm{T} \underset{A}{\Vdash} [P]\, a\, [T], \qquad \forall p = 1 \ldots m, \ \mathrm{T} \underset{A}{\Vdash} [T_{s_p}]\, b_p\, [T']}{\mathrm{T} \underset{A}{\Vdash} [P]\, a\, [s_p \diagdown b_p : p = 1 \ldots m]\, [T']},$$

sous la condition de cohérence

$$\forall s \in S \diagdown \{s_1, \ldots, s_m\}, \qquad s \text{ ayant une occurrence dans } a,\ T_s = T'_s.$$

— *Concaténation*

$$A_7 : \frac{\mathrm{T} \underset{A}{\Vdash} [P]\, a\, [T], \qquad \mathrm{T} \underset{A}{\Vdash} [Q_0]\, b\, [T']}{\mathrm{T} \underset{A}{\Vdash} [P]\, a \cdot b\, [T']},$$

sous la condition de cohérence

$$\forall s \in S - \{\omega_0\}, \qquad s \text{ ayant une occurrence dans } a : T_s = T'_s.$$

— *Opération* $*$

$$A_8 : \frac{\mathrm{T} \underset{A}{\Vdash} [P]\, a\, [T\,(0, P)]}{\mathrm{T} \underset{A}{\Vdash} [P]\, a^*\, [T]}.$$

Remarques sur le système de déduction A :

1) Avec Cousineau, appelons rationnel un arbre obtenu à partir des arbres élémentaires : ω_i et α_j, par un nombre fini d'opérations de magma et d'étoile. On montre que la classe des programmes décrits par de tels arbres est exactement la classe des programmes décrits par des organigrammes (par exemple [21]).

L'ensemble des règles de déductions habituelles — ou « règles de Hoare » — pour ce genre de programmes, apparaît donc comme *règles dérivées* de notre système A : règles A_{21}, A_{22}, A_3, A_4, A_{51}, A_{52}, A_7, A_8.

Mais A est un système *plus puissant* que la simple collection de ces règles, dans la mesure où :

1° En se limitant au cas des *arbres rationnels* : A est bien sûr toujours complet alors que, ainsi que nous verrons en II, la collection des règles $A_{21} \ldots A_8$ sus mentionnée (*ni aucun ensemble de règles « finitaire »*) ne saurait l'être.

2° A peut s'appliquer à *des classes plus larges* de programmes : par exemple les schémas de programmes simples de Cousineau [11] équivalents aux programmes avec appels de procédures sans paramètre des « μ-expressions » de De Bakker [4] ou de la « Context Free Dynamic Logic » de Harel [16] (une telle application fait actuellement l'objet d'une étude).

La raison de ce phénomène est claire : c'est la présence d'une règle « *infinitaire* » (faisant intervenir un nombre infini de prémisses), la règle A_1, qui rend A complet; et c'est la forme très générale de cette règle (pour toute partie dirigée $(a_d)_{d \in D}$) qui donne à A son extension.

3° Notons dès à présent comment la notion de *sous-arbre* et la règle de déduction associée — règle pour la substitution A_6 — permettent d'exprimer simplement la « modularité » d'une preuve. Nous y reviendrons à propos des schémas de programmes et du GO TO.

APPENDICE 1
ÉLÉMENTS DE THÉORIE DES MODÈLES [1]

1. LANGAGES

Un langage L est une collection de symboles. Ces symboles sont répartis en trois groupes : symboles de prédicats (ou de relations), symboles de fonctions, symboles de constantes

$$L = \{P_1, \ldots, P_n; F_1, \ldots, F_m; c_1, \ldots, c_q\}.$$

A chaque symbole de fonction ou de prédicat est associé un entier, son « arité » ou « nombre de places ».

L est dénombrable si l'ensemble de ces symboles est dénombrable. Un langage « avec égalité » comprend nécessairement un symbole de relation à deux places « = ».

Exemple 1 : Langage pour l'arithmétique

$$L = \{\leqq; +, \times; 0, 1\}.$$

[1] Pour un exposé plus détaillé, se reporter par exemple à [5] p. 18 sqq.

2. FORMULES

Pour formaliser un langage L, on a besoin des symboles logiques suivants :

- parenthèses : (,);
- variables : $x_1, x_2, \ldots, x_n, \ldots$;
- connecteurs : $\wedge$ (et), $\vee$ (ou), $\neg$ (non), $\rightarrow$ (implique);
- quantificateurs : $\forall$ (pour tout) $\exists$ (il existe);
- deux symboles : *Vrai* et *Faux*.

On définit alors :

- *Les termes de L*, obtenus par application un nombre fini de fois des règles :

(i) une variable, un symbole de constante sont des termes;

(ii) si F est un symbole de fonction à n places, et $t_1 \ldots t_n$ sont des termes, $F(t_1 \ldots t_n)$ est un terme.

- *Les formules atomiques*, obtenues par la règle :

Si P est un symbole de relation à n places et $t_1 \ldots t_n$ sont des termes, alors $P(t_1 \ldots t_n)$ est une formule atomique.

- *Les formules* obtenues par les règles :

(i) une formule atomique, Vrai, Faux sont des formules;

(ii) si φ et ψ sont des formules $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\neg \varphi)$, $(\varphi \rightarrow \psi)$ sont des formules;

(iii) si x est une variable, φ est une formule, $(\forall x \varphi(x))$ et $(\exists x \varphi(x))$ sont des formules.

Une occurrence d'une variable dans une formule est *liée* si elle se trouve « dans le champs » d'un quantificateur; le contraire est « *libre* ». Une formule dont toutes les variables sont liées sera appelée un *énoncé*.

3. MODÈLES

Soit L un langage et M un ensemble. A chaque symbole de prédicat P (à p places) [resp. : symbole de fonction F (à p places), symbole de constante c] faisons correspondre une relation $\overline{P}$ (à p arguments) [resp. : une fonction $\overline{F}$ (à p arguments), une constante $\overline{c}$] définie sur M. Nous obtenons ainsi une *réalisation* (ou un *modèle*) de L, $\mathcal{M}$

$$\mathcal{M} = \langle M; \overline{p}_1 \ldots \overline{p}_n; \overline{F}_1 \ldots \overline{F}_m; \overline{c}_1 \ldots \overline{c}_q \rangle,$$

M est le domaine de $\mathcal{M}$.

Exemple 2 :

$$\mathcal{N} = \langle \mathbb{N};\ +_{\mathbb{N}},\ \times_{\mathbb{N}};\ O_{\mathbb{N}},\ 1_{\mathbb{N}} \rangle, \text{ ou } +_{\mathbb{N}},\ \times_{\mathbb{N}} \ldots$$

sont l'addition, la multiplication... habituelles dans l'ensemble $\mathbb{N}$ des nombres entiers, est une réalisation du langage L de l'exemple 1.

φ étant un énoncé de L, on peut ainsi donner une signification précise (*cf.* [5]) à la phase : « φ est vraie dans $\mathcal{M}$ ». On dira encore que $\mathcal{M}$ « *satisfait* φ » ou est un « *modèle de* φ ». Ce que l'on note : $\mathcal{M} \models \varphi$. On dira que la formule φ de L est *valide* ($\models \varphi$) si elle est satisfaite par toute réalisation de L. C'est la notion de *vérité sémantique*.

Enfin, étant donné un ensemble Σ d'énoncés de L, on dit que $\mathcal{M}$ satisfait Σ (est un modèle de Σ), si M satisfait tout $\sigma \in \Sigma$.

4. THÉORIES

D'un autre côté, on a une notion de vérité syntaxique. C'est-à-dire qu'on se donne sous une forme ou sous une autre les axiomes (purement logiques) et les règles de déduction qui constituent le Calcul des Prédicats.

Si un énoncé σ peut être démontré dans ce système de déduction, σ est un *théorème*. Ce que l'on note : $\vdash \sigma$. On a le théorème fondamental suivant qui relie les deux notions de vérité :

THÉORÈME DE COMPLÉTUDE DU CALCUL DES PRÉDICATS : *Pour tout énoncé* σ, σ *est un théorème ssi il est valide :*

$$\vdash \sigma \quad \Leftrightarrow \quad \models \sigma.$$

Mais on peut aussi se donner des axiomes supplémentaires.

Appelons *théorie* écrite dans L, tout ensemble d'énoncés de L. On dira qu'un énoncé σ de L est *conséquence de* T ssi σ peut être démontré à partir des axiomes logiques *et* de T, en appliquant les règles de déduction. On note alors : $T \vdash \sigma$.

THÉORÈME DE COMPACITÉ (SYNTAXIQUE) : *Si* $T \vdash \sigma$, *il existe un sous ensemble fini de* T, T_f *tel que*

$$T_f \vdash \sigma.$$

5. AUTRES LOGIQUES

Un langage comme décrit ci-dessus est dit :

– du 1er ordre, parce qu'il ne comporte que des « variables d'individus » (i. e. qui seront interprétés dans tout modèle comme des *éléments* du domaine), et pas de variable de relation (ou de fonction);

— finitaire parce que les formules sont de longueur finie : elles ne comportent qu'un nombre fini de conjonctions, disjonctions, et de quantificateurs.

Les langages $L\omega_1, \omega$ sont des langages du 1er ordre également, mais autorisent la conjonction ou la disjonction d'un *ensemble dénombrable* de formule, c'est-à-dire les formules du genre

$$\bigwedge_{i\in I} \varphi_i, \qquad \bigvee_{i\in I} \varphi_i, \qquad \text{Card } (I)=\omega.$$

Mais la règle pour l'introduction de quantificateurs est la même que pour un langage finitaire : pas de séquence infinie.

APPENDICE 2

TABLE DES PRINCIPALES NOTIONS ET NOTATIONS INTRODUITES

	Paragraphes	Notations
ARBRE DE PROGRAMME (A.P.)		
Définition	1.1	$a, b, c, \ldots$
Ensembles des A.P. sur le langage L et les variables de V	1.1	$A(L, V)$
Opérations :	1.2	
Substitution de a_1 à $s_1 \ldots$		$a[s_1 \setminus a_1, \ldots, s_p \setminus a_p]$
« Itération »		a^*
Opérations $\uparrow$ et $\downarrow$		$\uparrow a, \downarrow a$
Concatenation		$a.b$
Ordre	1.2	$a<b$
ASSERTIONS DE SORTIE		
	1.4	Q_i, R_j, T_s
Suite d'ass. de sortie associées aux ω_i		$Q=(Q_i)_{i<\omega}$
Suite d'ass. de sortie associées aux α_j	1.4	$R=(R_j)_{j<\omega}$
Ens. d'ass. de sortie	1.4	$T=(Q, R)$
		$s\in S\ T_s=Q_i$ si $s=\omega_i$
		$T_s=R_j$ si $s=\alpha_j$
$\{T$: écrits dans le lang. L sur les var. de $V\}$	1.4	$\tau(L, V)$
Insertion de P dans $Q=(Q_i)_{i<\omega}$ au rang k	1.5 et 2.2	$T(k, P)$
ASSERTIONS INTERMÉDIAIRES		
Définitions :		
Arbres, schémas de prog. avec ass. int.	3.2	$[A], [B] \ldots$
COND ET F		
Cond. pour parvenir au nœud $m (m\in\{1, 2\}^*)$ dans a	1.3	$\text{Cond}_a(m)$
Fonction calc. le long de ce chemin	1.3	$F_a(m)$
MODÈLES		
Définition	App. 1	$\mathcal{M}$
$\mathcal{M}$ est modèle de la formule du 1er ordre φ	App. 1	$\mathcal{M}\models\varphi$
Le triplet $[P]\,a\,[T]$ est vrai dans $\mathcal{M}$	1.8	$\mathcal{M}\models[P]a[T]$
φ est vrai dans tt mod de T	App. 1	$T\models\varphi$
$[P]\,a[T]$ est vrai dans tt mod de T	1.8	$T\models[P]a[T]$

	Paragraphes	Notations
PRÉCONDITION LA PLUS FAIBLE		
Pour un arbre fini	1.5	$\tilde{a}(T)$
Pour un S.P.S. sans itération	2.4	$\tilde{E}(T)$
Pour un S.P.S. qq, dans $L\omega_1\omega$	3.3	$\tilde{E}(T)$
SYSTÈME DE DÉDUCTION		
En logique classique	App. 1	$T \vdash \varphi$
Pour les A.P. (Système A)	1.6, 1.10	$T \underset{A}{\Vdash} [P]a[T]$
Pour les S.P.S. (Système S)	2.2	$T \underset{S}{\Vdash} [P]E[T]$
Dans $L\omega_1\omega$	3.3	$\underset{A_1}{\Vdash}, \underset{S_1}{\Vdash}$
SCHÉMAS DE PROGRAMME		
Structuré (S.P.S.)		
Définition	2.1	E
Ens. des S.P.S. sur le lang. L et les var. V	2.1	$\varphi(L, V)$
Substitution de E_1 à l'élément terminal s_1, etc.	2.7	$E[s_1 \Vert E_1, \ldots, s_p \Vert E_p]$
Opérations $\uparrow_k$	2.7	$\uparrow_k(E)$
Arbre associé à un S.P.S.	2.1	E
Syst. d'équations d'actions		
Déf., syst. de déd. Σ	3.1	
THÉORIE		
Définition	App. 1	T
Théorie de la structure $\mathcal{M}$	App. 1	$\mathrm{Th}(\mathcal{M})$

BIBLIOGRAPHIE

1. J. Arsac *La construction de programmes structurés*, Dunod, Paris, 1977.
2. M. A. Arbib et S. Alagic, *Proof Rules for « Gotos »*, Acta Informatica, vol. 11, 1979, p. 139-148.
3. L. Banakowski, A. Kreczmar, G. Mirkowska, H. Rasiowa et A. Salwicki, *An Introduction to Algorithmic Logic. Mathematical Investigations in the Theory of Programs, in* Banach Center Publications V.2 Mathematical Foundations of Computer Science, P. W. N. Polish Scientific Publishers, 1977, Warsaw, p. 7-99.
4. J. W. de Bakker, *Recursive Programs as Predicate Transformers, in* Formal descriptions of programing concepts, E. J. Neuhold, éd., North-Holland, 1978, p. 165-202.
5. Chang Keisler, *Model Theory*, N.-H. Amsterdam.
6. S. A. Cook, *Soundness and Completeness for an Axiom System for Program Verification*, J.S.I.A.M. on computing, vol. 7, 1978, p. 70-90.
7. M. Clint et C. A. R. Hoare, *Program Proving: Jumps and Functions*, Acta Informatica, vol. 1, 1972, p. 214-224.
8. B. Courcelle et M. Nivat, *The algebraic Semantics of Recursive Program Schemes, in* Proc. 7th Math. Found of Comput. Sc. Symposium, 1978, Lecture Notes in Comput. Sc., vol. 62, p. 16-30.

9. G. COUSINEAU *Les arbres à feuilles indicées : un cadre algébrique de définition des structures de contrôle*, Thèse d'État, Paris, 1977.
10. G. COUSINEAU, *An algebraic definition of control structures*, L.I.T.P. Report, 78-27 (à paraître dans Theor. Comp. Sci.).
11. G. COUSINEAU, *La programmation en EXEL*, 1re partie; Revue Technique THOMSON-CSF, vol. 10, n° 2, 1978, p. 209-234.
12. G. COUSINEAU, *La Programmation en EXEL*, 2e partie, Revue Technique THOMSON-CSF, vol. 11, n° 1, 1979, p. 13-35.
13. G. COUSINEAU et P. ENJALBERT, *Program Equivalence and Provability, in* Proc. 8th Math. Found. of Comput. Sc. Symposium, 1979, Lecture Notes in Comput. Sc., n° 74, p. 237-245.
14. E. W. DIJSKRA, *Guarded Commands, Non Determinacy and Formal Derivations of Programs*, Com. Assoc. comput. Math., vol. 18, n° 8, 1975, p. 453-457.
15. DONER, *Tree Acceptors and Some of their Applications*, J. Comput. System Sci., vol. 4, 1970, p. 406-451.
16. D. HAREL, *Dynamic Logic*, Springer Lecture Notes in Comput. Sc., vol. 68, 1979.
17. D. HAREL, A. MEYER et V. R. PRATT, *Computability and Completeness in Logics of Programs*, Proc. 9th Annual A.C.M. Symposium on Theory of Computing, 1977, p. 261-268.
18. C. A. R. HOARE, *An Axiomatic Basis for Computer Programing*, Com. Assoc. Comput. Math., vol. 12, 1969, p. 576-580.
19. C. A. R. HOARE et P. E. LAUER, *Consistent and Complementary Formal Theories of the Semantics of Programming Languages*, Acta Informatica, vol. 3, 1974, p. 135-153.
20. S. IGARASHI, R. L. LONDON et D. C. LUCKMAN, *Automatic Verification I...*, Acta Informatica, vol. 4, 1975, p. 149-181.
21. M. NIVAT, *Chartes, arbres, Programmes itératifs*, I.R.I.A.-S.E.S.O.R.I., Journées d'étude : Synthèse, Manipulation et transformation des programmes, 1978, p. 165-187.
22. L. NOLIN et G. RUGGIU, *A Formalization of EXEL*, Assoc. Comput. Math. SIGACT-SIGPLAN Symposium on the Principle of Programming Languages, Boston, 1973.
23. R. PARIKH, *Second Order Process Logic*, 19th I.E.E.E. Symposium on Found. of Computer Science, 1978.
24. V. R. PRATT, *Semantical Consideration on Floyd-Hoare Logic*, 17th I.E.E.E. Symposium on Foundation of Computer Science, 1976, p. 109-121.
25. G. RUGGIU, *De l'organigramme à la formule*, Thèse d'État, Paris, 1973.
26. M. WAND, *A new Incompleteness Result for Hoare's System*, J. Assoc. Comput Mach., vol. 25, 1978, p. 168-175.