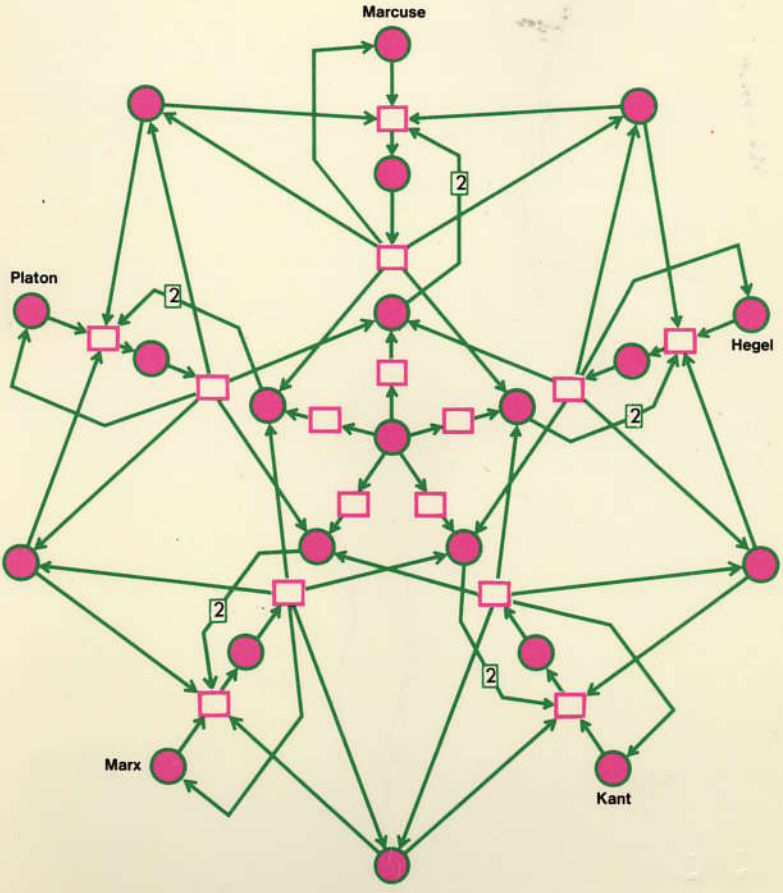
parallélisme, communication et synchronisation

édité par
J.P. VERJUS - G. ROUCAIROL



Editions du CNRS

Parallélisme, Communication et Synchronisation

Edité par

J.P. VERJUS

Professeur à l'Université de Rennes
Directeur de l'IRISA (Laboratoire Associé CNRS n° 227
et Centre INRIA de Rennes)

G. ROUCAIROL

Directeur de la Recherche en Architecture et Logiciel
BULL (Louveciennes)

ÉDITIONS DU CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
15, quai Anatole France - 75700 PARIS
1985

Principe des Méthodes de Preuve de Propriétés d'Invariance et de Fatalité des Programmes Parallèles

P. Cousot, R. Cousot

1. - INTRODUCTION

Les programmes, en particulier parallèles, doivent faire l'objet d'analyses détaillées avant d'être considérés comme valides. Nous nous intéressons à des analyses qualitatives qui consistent à comparer les comportements possibles d'un programme parallèle à une spécification des comportements souhaitables. Dans ce rapport, nous ne traitons que des méthodes de preuve. Dans un but simplificateur, nous supposons résolus les problèmes de description d'un programme parallèle, de ses comportements possibles et de sa spécification. Nous utiliserons donc des modèles simples mais très généraux des programmes (représentés par une relation de transition sur des états), de leurs comportements (définis par des ensembles de traces complètes) et de leur spécification (à l'aide de propriétés d'invariance ou de fatalité). Reste notre problème d'intérêt à savoir l'étude des méthodes de preuve permettant de démontrer qu'un programme parallèle se comporte conformément à une spécification. Dans ce rapport, nous nous limiterons au cas des méthodes de Floyd (dite des assertions inva-

riantes) et de Burstall (dites des assertions intermittentes) et à quelques-unes de leurs généralisations.

Nous essayons tout d'abord d'exprimer l'essence d'une méthode de preuve à l'aide de principes d'induction. Quand c'est fait, nous en démontrons la correction et la complétude sémantique. Nous cherchons ensuite à établir des comparaisons, en particulier à démontrer l'équivalence forte c'est-à-dire qu'une preuve par une méthode peut se réécrire en une preuve par une autre méthode. Nous nous efforçons également de généraliser ces principes d'induction, en particulier au cas de comportements possibles correspondant à des ensembles de traces non clos. C'est le cas des hypothèses d'exécution faiblement équitables pour les programmes parallèles que nous prendrons en exemple dans ce rapport. Finalement, nous étudions comment adapter une méthode de preuve à un langage particulier en partant d'une sémantique opérationnelle de ce langage et d'un principe d'induction formalisant la méthode de preuve.

2. - SYSTEMES DE TRANSITION : MODELES DES PROGRAMMES PARALLELES

Nous considérerons, de manière simple mais générale, qu'un porgramme parallèle est un système de transition (S,t,ε) où S est un ensemble non vide d'états, $t \in (SxS \to \{tt,ff\})$ une relation (non-déterministe) de transition entre un état et ses successeurs possibles et $\varepsilon \in (S \to \{tt,ff\})$ caractérise les états initiaux. Par commodité t et ε sont considérés comme des fonctions à valeurs de vérité (tt vrai, ff faux).

Il sera quelquefois plus aisé de considérer que la relation de transition t est partitionnée : $t = \bigvee_k t_k$ où chaque t_i correspond à un processus du programme parallèle. De même, l'ensemble S des états pourra être défini par un recouvrement $S_\ell = \bigcup S_\ell$.

Exemple 1

Au programme parallèle suivant :

$[\![$ 11: X:=false 12: $\|$ 21: while X do skip od 22: $]\!]$

(où X est une variable partagée à valeurs de vérité et 11, 12, 21, 22 des étiquettes désignant des points du programme)

nous associons le système de transition (S,t,ε) tel que :

$S = \{11,12\} \times \{21,22\} \times \{tt,ff\}$

$t = t_1 \vee t_2$

$t_1(<c_1,c_2,x>,<c'_1,c'_2,x'>) = [c_1=11 \wedge c'_1=12 \wedge c'_2=c_2 \wedge \neg x']$

$t_2(<c_1,c_2,x>,<c'_1,c'_2,x'>) = [c'_1=c_1 \wedge c_2=21 \wedge$
$((x \wedge c'_2=21) \vee (\neg x \wedge c'_2=22)) \wedge x'=x]$

$\varepsilon(<c_1,c_2,x>) = [c_1=11 \wedge c_2=21]$

où c_i, i=1,2 représente le compteur ordinal du processus i et x la valeur de la variable X .

□

3. - ENSEMBLE DE TRACES COMPLETES : MODELE DES COMPORTEMENTS POSSIBLES D'UN PROGRAMME PARALLELE

Nous représentons les comportements possibles d'un programme parallèle par un ensemble non vide θ de traces complètes non vides. Une trace complète p commence par un état initial ($\varepsilon(p_0)$ = tt). C'est une suite finie (auquel cas Dom(p)=n=$\{n-1,\ldots,0\}$ où $n \geq 1$ est la longueur de p) ou infinie (auquel cas Dom(p)= ω est l'ensemble des entiers naturels) d'états, liés par la relation de transition t ($\forall i,(i+1) \in$ Dom(p), $t(p_i,p_{i+1})$). Nous précisons traces "complètes" ou "achevées" pour insister sur le fait qu'elles ne représentent en aucun cas une observation intermédiaire du comportement du programme, mais un comportement terminé sinon infini.

Sauf à considérer des hypothèses d'exécution particulières (comme l'exécution équitable), θ est l'ensemble T des traces complètes engendrées par (S,t,ε) (à savoir les suites d'états, commençant par un état satisfaisant ε, liés par t, finies et terminées par un état de blocage (sans successeur par t) ou bien infinies).

Exemple 1 (suite) :

Pour le programme donné en exemple 1, l'ensemble $T = \theta_a$ est défini par les expressions régulières (généralisées) suivantes :

$$\theta_a = \theta_e \mid \langle 11,21,tt\rangle^{\infty}$$

$$\theta_e = [(\langle 11,21,ff\rangle.(\langle 12,21,ff\rangle \mid \langle 11,22,ff\rangle)) \mid (\langle 11,21,tt\rangle^{+}.\langle 12,21,ff\rangle)] \,.\, \langle 12,22,ff\rangle$$

où a^{∞} est la suite infinie de a. $T = \theta_a$ correspond aux exécutions arbitraires (non équitables), où le processus 2 peut être toujours actif alors que le processus 1 est toujours prêt et jamais activé. θ_a comprend donc la trace infinie $\langle 11,21,tt\rangle^{\infty}$ où le processus 2 boucle et le processus 1 attend indéfiniment. Par contre, l'ensemble θ_e décrit les comportements possibles avec hypothèse d'exécution faiblement équitable (où un processus toujours prêt est fatalement activé) pour lesquels l'exécution du programme se termine.

□

4. - SPECIFICATION DES COMPORTEMENTS SOUHAITABLES D'UN PROGRAMME PARALLELE PAR PROPRIETES D'INVARIANCE ET DE FATALITE

On peut également spécifier de manière générale les comportements souhaitables d'un programme parallèle pour un ensemble Σ de traces complètes. Nous nous sommes principalement intéressés jusqu'à présent aux cas d'intérêt pratique où :

- Σ est définie par une propriété invariante $\phi \in (SxS \rightarrow \{tt,ff\})$ sous condition $\varphi \in (SxS \rightarrow \{tt,ff\})$, c'est-à-dire :

$$\Sigma = \{p \in \theta : \forall i \in Dom(p) \,.\, ((\forall j<i \,.\, \varphi(p_o,p_j)) \Rightarrow \phi(p_o,p_i))\}$$

- Σ est définie par une propriété fatale $\psi \in (SxS \rightarrow \{tt,ff\})$ après $\varphi \in (SxS \rightarrow \{tt,ff\})$, c'est-à-dire :

$$\Sigma = \{p \in \theta : \exists i \in Dom(p) \,.\, (\forall j<i \,.\, \varphi(p_o,p_j) \wedge \psi(p_o,p_i))\}$$

(Pour simplifier, dans ce rapport, nous prendrons φ identiquement vraie dans les formules ci-dessus).

Exemple 1 (suite) :

- La correction partielle est une propriété d'invariance qui exprime que pour toute trace possible et tout état dans cette trace qui est un état de sortie, la spécification de sortie est satisfaite :

$$\forall p \in \partial \,.\, \forall i \in Dom(p) \,.\, \phi(p_o,p_i)$$

où, pour notre exemple, $\theta = \theta_a$ et X est faux au point de sortie, soit :

$$(<c_1,c_2,x>,<c'_1,c'_2,x'>) \,.\, [(c'_1=12, \wedge c'_2=22) \Rightarrow \neg x']$$

La non-terminaison, l'absence d'erreurs à l'exécution, l'exclusion mutuelle, l'absence de blocages globaux sont également des propriétés d'invariance.

- La terminaison est une propriété de fatalité qui exprime que pour toute trace possible, il existe un état dans cette trace qui est un état de sortie :

$$\forall p \in \theta \,.\, \exists i \in Dom(p) \,.\, \psi(p_o,p_i)$$

où pour notre exemple $\theta = \theta_e$ et

$$\psi(<c_1,c_2,x>, <c'_1,c'_2,x>) \,.\, [\, c'_1=12 \wedge c'_2=22]$$

La garantie d'entrée en section critique, de réponse à un signal, l'absence de famine individuelle sont d'autres exemples de propriétés de fatalité.
□

En pratique, l'ensemble Σ des traces souhaitables est spécifié à l'aide d'une combinaison d'ensembles de traces correspondant chacun à une propriété particulière (un cas simple est la correction totale définie comme conjonction de la correction partielle, de l'absence d'erreurs à l'exécution, de l'absence de blocages globaux et de la terminaison). Chacune de ces propriétés est une propriété d'invariance, de fatalité ou une propriété duale (dont l'étude formelle peut être évitée puisqu'elle s'obtient en appliquant le principe de dualité).

5. - PRINCIPES D'INDUCTION POUR LA PREUVE DE PROGRAMMES PARALLELES

Etant donnés les comportements possibles θ d'un programme parallèle (S,t,ε) et une spécification Σ, une preuve de correction consiste à démontrer que $\theta=\Sigma$. Dans ce rapport, nous ne nous préoccupons que du cas où Σ est définie par une propriété d'invariance ou de fatalité.

Notre démarche consiste à étudier formellement les méthodes de preuve pour les présenter dans un formalisme unique réduit à quelques concepts simples, de façon à mieux les comprendre et les expliquer (indépendamment des langages de programmation particuliers pour lesquels elles ont été conçues), les comparer (ce qui n'est pas toujours facile quand elles sont présentées pour des langages et avec des formalismes discordants), les généraliser (on ne sait pas appliquer certaines méthodes bien connues pour les programmes séquentiels au cas des programmes parallèles), en découvrir de nouvelles,

etc... La principale étape de cette étude consiste à réduire les méthodes de preuve à quelques principes d'induction aussi élémentaires que faire se peut et dont la correction et la complétude est, si possible, triviale.

5.1. - Principes d'induction pour les preuves d'invariances

Prenant pour simplifier $\theta=T$, une preuve d'invariance consiste à démontrer $\Sigma=T$, c'est-à-dire :

$$\forall p \in T \;.\; \forall i \in Dom(p) \;.\; \phi(p_o, p_i)$$

Ceci est équivalent à :

(a) $(\exists I \in (SxS \rightarrow \{tt,ff\}) \;.\; \forall s,s',s'' \in S.$

(b) $[\varepsilon(s) \Longrightarrow I(s,s)$

(c) $\wedge[I(s,s') \wedge t(s',s'')] \Longrightarrow I(s,s'')$

(d) $\wedge I(s,s') \Longrightarrow \phi(s,s')])$

Autrement dit, ϕ est invariante s'il existe une relation I (ligne (a)) et sous-entendu entre un état initial s et un état courant s' quelconques) qui est invariante lignes (b) et (c)) et plus forte que ϕ (ligne (d)). I est invariante puisqu'elle est vraie quand l'état courant est un état initial (ligne (b)) et reste vraie pour tout successeur possible s" de l'état courant s' (ligne (c)).

De nombreuses méthodes pour les programmes séquentiels (Floyd-Naur, Hoare) aussi bien que pour les programmes parallèles (Ashcroft, Newton, Keller, Lamport, Owicki & Gries, Mazukiewitz, ...) se ramènent à ce principe d'induction (cf [5] pour plus de détails, en particulier pour les références précises).

Exemple 1 (suite) :

Pour démontrer la correction partielle du programme de l'exemple 1 par la méthode de Lamport (ou encore la méthode de Owicki & Gries en utilisant des compteurs ordinaux explicites au lieu de variables auxiliaires (ce qui revient au même à un homomorphisme entre systèmes de transition près, cf (5)), il faut :

- Trouver les invariants locaux :

$$J_{11}(c_2,x) = tt \qquad J_{12}(c_2,x) = \neg x$$

$$J_{21}(c_1,x) = tt \qquad J_{22}(c_1,x) = \neg x$$

- Faire une preuve séquentielle du processus 1 :

. $[c_2=21] \Rightarrow J_{11}(C_2,x)$ (condition d'entrée)

. $[J_{11}(c_2,x) \wedge t_1(\langle 11,c_2,x\rangle,\langle 12,c_2',x'\rangle)] \Longrightarrow J_{12}(c_2',x')$

(ou encore $\{J_{11}\}$X:=**false** $\{J_{12}\}$ avec la notation de Hoare).

- Faire une preuve séquentielle du processus 2 :

• $[c_1=11] \Rightarrow J_{21}(c_1,x)$

• $[J_{21}(c_1,x) \wedge t_2(\langle c_1,21,x\rangle,\langle c_1',21,x'\rangle)] \Longrightarrow J_{21}(c_1',x')$

(soit $\{I_{21} \wedge X\}$ **skip** $\{I_{21}\}$)

• $[J_{21}(c_1,x) \wedge t_2(\langle c_1,21,x\rangle,\langle c_1',22,x'\rangle)] \Rightarrow J_{22}(c_1',x')$

(soit $(J_{21} \wedge \neg X) \Rightarrow J_{22}$)

- Faire une preuve d'absence d'interférence de la preuve du processus 1 avec le processus 2, soit pour i = 1,2 :

• $J_{1i}(21,x) \wedge J_{21}(1i,x) \wedge t_2(<1i,21,x>,<1i,21,x'>) \Rightarrow J_{1i}(21,x')$

• $J_{1i}(21,x) \wedge J_{21}(1i,x) \wedge t_2(<1i,21,x>,<1i,22,x'>) \Rightarrow J_{1i}(22,x')$

- Faire une preuve d'absence d'interférence de la preuve du processus 2 avec le processus 1, soit pour i=1,2 :

• $[J_{2i}(11,x) \wedge J_{11}(2i,x) \wedge t_1(<11,2i,x>,<12,2i,x'>)] \Rightarrow J_{2i}(12,x')$

- Démontrer la condition de sortie :

• $[J_{12}(22,x) \wedge J_{22}(12,x)] \Rightarrow \neg x$

Ces conditions de vérification se ramènent au principe d'induction ci-dessus en définissant l'invariant global I par :

$$I(<c_1,c_2,x>,<c_1',c_2',x'>) = [J_{c_1'}(c_2',x') \wedge J_{c_2'}(c_1',x')]$$

□

Dans (6), nous avons proposé des transformations systématiques permettant de dériver de nombreux principes d'induction équivalents au principe ci-dessus. Par exemple, la méthode "Subgoal induction" due à Morris et Wegbreit consiste à appliquer le principe d'induction ci-dessus à (S,t^{-1},σ) et ϕ^{-1} (où σ caractérise les états de sortie) au lieu de (S,t,ε) et ϕ (cf (4) pour plusde détails et une généralisation de la méthode au cas des programmes parallèles). De même en considérant un "anti-invariant" (négation de I) on obtient des méthodes de preuve par l'absurde (qui n'ont guère d'utilisé dans les preuves à la main, mais nous ont permis d'imaginer

de nouvelles méthodes d'analyse sémantique, généralisant (1) et qui sont utiles puisqu'en général la négation de l'approximation d'une propriété est différente de l'approximation de la négation de cette propriété).

5.2. - Principes d'induction pour les preuves de fatalité

5.2.1. - INDUCTION "A LA FLOYD"

Une preuve de fatalité consite à montrer

$$\forall p \in \theta.\ \exists i \in \mathrm{Dom}(p).\ \psi(p_i, p_{i+1})$$

et nous commençons par examiner le cas où θ est l'ensemble T des traces engendrées par (S, t, ε). Ceci est alors équivalent à :

(e) $\exists I \in (S \times S \to \{tt, ff\}), \prec \in \omega f((S \times S)^2 \to \{tt, ff\}).\ \forall s, s' \in S.$

(f) $\quad [\ \varepsilon(s) \Longrightarrow I(s,s)$

(g) $\quad \wedge\ I(s,s') \Longrightarrow \psi(s.s')$

(h) $\quad \vee [\ \exists s'' \in S.\ t(s',s'')$

(i) $\quad \wedge \forall s'' \in S.\ t(s',s'') \Longrightarrow (I(s,s'') \wedge$

(j) $\quad (s,s'') \prec (s,s'))]])$

Autrement dit, la preuve peut se décomposer en :

- une preuve d'invariance de I reliant les états courants s' accessibles depuis un état initial s par un chemin le long duquel ψ n'est jamais vraie (lignes (f) et (i)).
- une preuve d'absence de blocages (tout état non final a au moins un successeur, ligne (h))
- une preuve de termination au moyen d'une relation $\prec$ bien fondée, c'est-à-dire sans chaîne infinie décroissante (lignes (e) et (j)).

On aura reconnu comme cas particulier et sous forme abstraite la méthode de preuve de correction totale proposée par Floyd pour les programmes séquentiels.

Exemple 2 :

Considérons le "programme" à non-déterminisme infini :

while $X \neq 0$ **do if** $X < 0$ **then** X=? **else** X:=X-1 **fi od**

où X:=? affecte à X un entier positif quelconque. Nous lui faisons correspondre le système de transition (S,t, ε) où S est l'ensemble $\mathbb{Z}$ des entiers naturels, $t(x,x') = [(x < 0 \wedge x' \geqslant 0) \vee (x > 0 \wedge x' = x-1)]$ et $\varepsilon(x) = tt$. Ce programme se termine car $\psi(x,x') = [x' = 0]$ est fatale. On peut le démontrer directement en appliquant le principe d'induction avec $I(x,x') = tt$ et $(x_1,x'_1) \prec\!\!\!\!- (x_2,x'_2)$ si et seulement si $(x'_2 < 0 \wedge x'_1 > 0) \vee (0 \leqslant x'_1 < x'_2)$ qui est une relation bien fondée sur les entiers.
□

Dans [7], nous avons étudié de nombreux principes d'induction et montré qu'ils sont équivalents au principe ci-dessus et donc corrects et sémantiquement complets. De plus, en considérant des systèmes à transitions et états partitionnés nous pouvons généraliser la méthode au cas des programmes parallèles. Enfin nous étudions l'ordre (intuitivement la "longueur") de la relation de terminaison $\prec\!\!\!\!-$ qui est nécessaire pour pouvoir faire la preuve en fonction d'une borne (en général transfinie) sur le non-déterminisme de la relation de transition t.

5.2.2. - INDUCTION "A LA BURSTALL"

Nous avons également étudié la méthode conçue par Burstall et popularisée sous le nom de méthode des assertions intermittentes par Manna & Waldinger (cf [8] pour le détail de l'étude et les références). Ini-

tialement conçue pour démontrer la correction totale des programmes séquentiels, nous avons proposé une présentation par une logique de Hoare (2) qui est correcte et relativement complète (3), puis généralisé la méthode au cas des programmes parallèles. Pour éviter d'entrer dans les détails, donnons simplement le principe d'induction correspondant (commenté ci-après) :

(k) $\exists \Lambda, L \in (\Lambda \to (S^2 \to \{tt,ff\})), \ll \in \omega f(\Lambda^2 \to \{tt,ff\})$.

(l) $\exists \lambda \in \Lambda. \forall s,s' \in S.\ L_\lambda(s,s') = [\varepsilon(s) \Rightarrow \psi(s,s')]$

(m) $\wedge \forall \lambda \in \Lambda. (\exists I_\lambda \in SxS \to \{tt,ff\}), \prec_\lambda \in \omega f((SxS)^2 \to \{tt,ff\})$.
$\forall s,s' \in S.$

(o) $[\ I_\lambda(s,s)$

(p) $\wedge\ I_\lambda(s,s') \Rightarrow L_\lambda(s,s')$

(q) $\vee [\exists s'' \in S.\ t(s',s'')$

(r) $\wedge \forall s'' \in S.(t(s',s'') \Rightarrow (I_\lambda(s,s'')$

(s) $\wedge (s,s'') \prec_\lambda (s,s')))]$

(t) $\vee [\exists \lambda' \ll \lambda. \forall s'' \in S.(L_{\lambda'}(s',s'') \Rightarrow$

(u) $(I_\lambda(s,s'') \wedge (s,s'') \prec_\lambda (s,s')))]])$

Pour comprendre ce principe d'induction il faut d'abord remarquer que ψ est fatale pour (S,t,ε) si et seulement si ψ' telle que $\psi'(s,s') = [\varepsilon(s) \Rightarrow \psi(s,s')]$ l'est pour $(s,t,\lambda s.tt)$. Pour démontrer que ψ' est fatale, il suffit de démontrer que tous les lemmes L_λ, $\lambda \in \Lambda$ sont fatals et que ψ' est l'un d'eux (ligne (l)). La preuve de fatalité de chaque lemme L_λ peut être faite, en utilisant le principe d'induction précédemment étudié, à l'aide d'un invariant I_λ et d'une relation de terminaison $\prec_\lambda$. Les lignes (o), (p), (q), (r), (s), sont donc semblables aux lignes (e), (f), (g), (h), (i), (j). Elles formalisent ce que Burstall appelle l'exécution symbolique ("hand-simulation") en assurant grâce à la relation de terminaison que cette exécution symbolique est finie (ce qui est sous-entendu par Burstall). De

plus, les lemmes $L_\lambda, \lambda \in \Lambda$ sont en relation par $\ll$ qui est bien fondée (ligne (k)). On peut interpréter $\lambda' \ll \lambda$ comme le fait que l'on démontre la fatalité de $L_{\lambda'}$ avant de démontrer la fatalité de L_λ. Autrement dit on peut utiliser $L_{\lambda'}$ dans la démonstration de L_λ. C'est ce qui est fait aux lignes (t) et (u) qui formalisent ce que Burstall appelle l'étape d'induction ("a little induction"). Intuitivement si l'état courant s' est accessible depuis l'état initial s par un chemin où L_λ n'est jamais vrai alors d'après le lemme $L_{\lambda'}$ précédemment démontré nous atteindrons fatalement un état s" descendant de s (ligne (t)) et de plus près du but L_λ que s' (ligne (u)).

Exemple 2 (suite) :

Dans une preuve informelle on démontre d'abord par exécution symbolique que partant d'un état quelconque, le programme atteint fatalement l'état $X \geqslant 0$. Autrement dit L_1 tel que $L_1(x,x') = [x' \geqslant 0]$ est fatal ce qui se démontre en choisissant $I_1(x,x') = [x'=x \vee x' \geqslant 0]$ et l'ordre bien fondé $(x_1,x_1') \prec_1 (x_2,x_2')$ ssi $(x_2' < 0 \wedge x_1' \geqslant 0)$ qui satisfont les conditions (o) quand x=x', (p) quand $x' > 0$ et (q), (r), (s) quand $x' < 0$.

On démontre ensuite par induction sur n que partant de $X=n \geqslant 0$, l'exécution du programme se termine avec X=0. Autrement dit pour tout $n \geqslant 0$ le lemme L_n tel que $L_n(x,x') = [(x=n) \Longrightarrow (x'=0)]$ est fatal. Le cas n=0 se traite trivialement en choisissant $I_o(x,x') = [(x=0) \Longrightarrow (x'=0)]$ qui satisfait les lignes (o) et (p). Le cas $n > 0$ consiste à montrer que partant de $X=n > 0$ on obtient par exécution symbolique X=n-1 et comme $0 \leqslant n-1 < n$ on conclut par induction utilisant le lemme L_n que l'exécution se termine fatalement avec X=0. Ce cas se traite donc avec $I_n(x,x') = [(x=n) \Longrightarrow (x'=n \vee x'=n-1 \vee x'=0)]$ et $(x_1,x_1') \prec_n (x_2,x_2')$ ssi $x_1' < x_2'$ qui satisfont les lignes (o) quand $x \neq n$ ou $x=x'=n$, (q), (r), (s) quand $x=x'=n$, (t), (u) quand $x=n$ et $x'=n-1>0$ et (p) quand $x \neq n$ ou $x=n \wedge x'=0$.

Finalement on démontre aisément la proposition que L_p avec $L_p(x,x') = [x'=0]$ est fatale. Si $X < 0$ alors d'après L_1 on atteint $X=n \geqslant 0$ puis $X=0$ d'après L_n sinon $X=n \geqslant 0$ et d'après L_n on atteint fatalement $X=0$. Plus formellement ce raisonnement consiste à choisir I_p donné par $I_p(x,x') = [(x=x') \vee (x < 0 \wedge x' \geqslant 0) \vee (x'=0)]$ et $(x_1,x_1') \prec_p (x_2,x_2')$ ssi $(0=x_1' < x_2') \vee (x_2' < 0 \leqslant x_1')$ qui satisfont (o) et (p) quand $x'=0$ (ou trivialement quand $x' < 0 \wedge x \neq x'$) (q), (r), (s) quand $x=x' < 0$ et (t), (u) quand $x' > 0$.

On a donc choisi $\Lambda = \{1,0,1,\ldots p\}$ et démontré les lemmes selon l'ordre bien fondé $1 \ll 0 \ll 1 \ll 2 \ll \ldots \ll p$.

□

Dans (8), nous donnons une série de principes d'induction équivalents au principe d'induction ci-dessus qui permettent de mieux comprendre divers aspects et présentations possibles de la méthode de Burstall et de la généraliser plus aisément. De plus, nous démontrons un résultat de complétude forte en ce sens qu'il est toujours possible de trouver une preuve utilisant les lemmes L_λ, $\lambda \in \Lambda$ imposés pour faire la preuve. Le choix des lemmes peut être arbitraire pourvu que tout lemme L_λ soit fatal relativement aux lemmes utilisés pour le démontrer (plus précisément, L_λ est fatal pour le système de transition $(S, t \vee \bigvee_{\lambda' < \lambda} L_{\lambda'}, \lambda s.tt)$.

Suite à une polémique concernant les méthodes de Floyd et Burstall (Manna & Waldinger, Gries, ...) (où chaque clan a lancé le défi de trouver une preuve élégante avec la méthode abhorrée d'exemples qui se traitent facilement avec sa méthode favorite), nous avons démontré (laissant de côté les questions d'esthétique) que toute preuve donnée avec l'un des principes d'induction peut se réécrire en une preuve utilisant l'autre principe d'induction ((9)).

5.3. - Généralisation au cas où l'ensemble θ des traces possibles est arbitraire

Quand l'ensemble des traces θ définissant les comportements possibles d'un programme sur un ensemble donné S d'états n'est pas clos, (c'est-à-dire qu'il n'existe pas de relation de transition t et de condition d'entrée ε sur les états de S tels que l'ensemble T des traces engendrées par le système de transition (S,t, ε) soit égal à θ) les principes d'induction que nous venons de considérer pour les preuves de fatalité ne sont pas applicables. C'est le cas des programmes parallèles avec hypothèse d'exécution équitable.

Contre-exemple 1 (suite) :

S'il était possible de donner une preuve de terminaison du programme [[11: X:=**false** 12: ‖ 21: **while** X **do skip od** 22:]] avec le principe d'induction (e), on aurait I(<11,21,tt> , <11,21,tt>) d'après (f) et (<11,21, tt > , < 11,21,tt>) $\prec\!\!\!\!-$ (<11,21,tt> , <11,21,tt>) d'après (j) ce qui est contraire à l'hypothèse (e) que $\prec\!\!\!\!-$ est bien fondé. De même il n'existe pas de preuve de terminaison avec (k) car d'après (9) on pourrait alors réécrire cette preuve en une preuve pour (e).
□

Le cas d'un ensemble arbitraire θ de comportements possibles est traité dans sa généralité par (7). A titre d'exemple, nous n'aborderons dans ce rapport que le cas particulier des programmes parallèles avec hypothèse d'exécution faiblement équitable. Soit donc un programme parallèle quelconque et (S,t,ε) le système de transition correspondant où t est partitionné en $\bigvee_{k \in n} t_k$; chaque t_k correspondant par exemple à un processus du programme (ces processus ne sont donc pas créés dynamiquement). L'ensemble θ_e des traces faiblement équitables comprend toutes les traces finies engendrées par (S,t,ε) ainsi que les traces infinies p telles qu'aucun processus

$k \in n$ ne soit toujours prêt et jamais activé c'est-à-dire $\neg[\exists k \in n.\ \exists i \geqslant o.\ \forall j \geqslant i.\ (\exists s'.\ t_k(p_j,s') \wedge \neg t_k(p_j,p_{j+1}))]$.

L'équité faible correspond à un cas particulier où les preuves d'invariance peuvent se faire à l'aide du principe d'induction (a) le problème à résoudre est celui des preuves de fatalité. Nous n'illustrerons qu'une approche possible à savoir celle que nous appelons méthode de "l'ensemble dynamique de points de coupure" dans (7).

Pour démontrer qu'une propriété d'un programme est fatale, il suffit de prouver, par l'absurde, qu'une quantité décroitrait infiniment souvent selon une relation bien fondée dans le cas contraire. Comprendre le "infiniment souvent" comme "à chaque pas" ou même comme "à chaque tour de boucle" est en général trop restrictif. Dans le cas général, l'"instant" où la quantité décroît (c'est-à-dire l'instant où le calcul se trouve à un point de coupure) dépend de l'histoire du calcul d'où l'appellation "dynamique". Dans le cas particulier de l'hypothèse d'équité faible, cet instant est celui où un processus l qui pourrait rester indéfiniment prêt sans hypothèse d'équité est activé. Cette remarque conduit au principe d'induction suivant dont une version a été donnée par Lehman, Pnueli et Stavi :

$$(\forall p \in \theta_e.\ \exists i \in \mathrm{Dom}(p).\ \psi\ (P_o,P_i))$$

$$\Longleftrightarrow$$

$$(v)\ (\exists I \in (SxSxn) \to \{tt,ff\}, \prec \in \omega f((SxS)^2 \to \{tt,ff\}).$$

$$\forall s,s' \in S,\ l \in n.$$

$$\varepsilon(s) \Longrightarrow [\exists l \in n.\ I(s,s,l)]$$

$$I(s,s',l) \Longrightarrow \psi(s,s') \vee$$

$$[\exists k \in n, s'' \in S.\ t_k(s',s'')$$

$$\wedge \forall k \in n, s'' \in S.\ t_k(s's'') \Longrightarrow$$

$$[(\exists l' \in n.\ I(s,s'',l') \wedge (s,s'') \prec (s,s'))$$

$$\vee (K \neq l \wedge I(s,s'',l) \wedge \exists s''' \in S.\ t_l(s',s'''))]])$$

Exemple 1 (suite et fin)

Pour démontrer la terminaison de [[11: X:=false 12: ‖ 21: while X do skip od 22]] soit ψ $(\langle c_1,c_2,x\rangle,\langle c'_1,c'_2,x'\rangle) = [c'_1=12 \wedge c'_2=22]$ on choisira par exemple $I(\langle c_1,c_2,x\rangle, \langle c'_1,c'_2,x'\rangle) = [(c'_1=11) \Longrightarrow (1=1)]$ et $(s_1,s'_1) \longrightarrow\!\!\!\!\!\prec (s_2,s'_2)$ ssi $s'_1 \longrightarrow\!\!\!\!\!\ll s'_2$ avec $\langle c'_1,c'_2,x'\rangle \longrightarrow\!\!\!\!\!\ll \langle c_1,c_2,x\rangle$ ssi $[(c'_1=12 \wedge c_1=11) \vee (c'_2=22 \wedge c_2=21)]$.

□

Une autre approche, que nous appelons "transformationnelle", est applicable dans le cas où ψ est fatale pour les traces θ de (S,t,ε) si (et seulement si) ψ' l'est pour les traces θ' de (S',t',ε'). Une méthode correcte (et complète) de démonstration (indirecte) de la fatalité de ψ consiste à utiliser un principe d'induction $P'(S',t',\varepsilon',\psi')$ pour démontrer la fatalité de ψ'. Si de plus on dispose d'une définition de S',t',ε',ψ' en termes de S,t,ε,ψ et que l'on remplace S',t',ε',ψ' dans P' par cette définition, on obtient une méthode de preuve P directement en termes de S,t,ε,ψ.

Cette approche a été utilisée pour obtenir des méthodes de preuve de propriétés de fatalité de programmes parallèles avec hypothèse d'exécution équitable en appliquant (e) à un programme transformé incorporant un "scheduler" (cf exemples et références dans (7)).

6. - CONSTRUCTION D'UNE METHODE DE PREUVE A PARTIR D'UNE SEMANTIQUE OPERATIONNELLE ET D'UN PRINCIPE D'INDUCTIONS PAR DECOMPOSITION D'ASSERTIONS GLOBALES EN ASSERTIONS LOCALES

Nous avons étudié comment construire formellement une méthode de preuve, étant donnés une sémantique opérationnelle, un principe d'induction, un langage d'assertions et sa sémantique. Les résultats obtenus pour les propriétés d'invariance (5) (6), (10), sont également applicables aux propriétés de fatalité.

Un principe d'induction correct et sémantiquement complet pour démontrer une propriété $P(S,t,\varepsilon,\psi)$ a la forme générale :

$$P(S,t,\varepsilon,\psi) \Longleftrightarrow [\exists H \in A(S).\ C[\![S,t,\varepsilon,\psi]\!](H)]$$

(par exemple $A(S).[(SxS \to \{tt,ff\}) \times \omega f((SxS)^2 \to \{tt,ff\})]$ dans (e)).

Une preuve de programme a la même forme puisqu'elle consiste à orienter une hypothèse d'induction $\tilde{H}$ satisfaisant certaines conditions de vérification $\tilde{C}[\![S,t,\varepsilon,\psi]\!](\tilde{H})$. La différence est que l'hypothèse d'induction H s'exprime à l'aide d'assertions locales associées à certains points du programme alors que H utilise des assertions globales. la méthode de preuve doit être correcte, c'est-à-dire :

$$[\exists \tilde{H} \in \tilde{A}(S).\tilde{C}[\![S,t,\varepsilon,\psi]\!](\tilde{H}) \Longrightarrow P(S,t,\varepsilon,\psi)$$

d'où l'on déduit qu'il existe une fonction $\gamma \in (\tilde{A}(S) \to A(S))$ (qui définit la sémantique des assertions locales) et telle que

$$\tilde{C}[\![S,t,\varepsilon,\psi]\!] \Rightarrow C[\![S,t,\varepsilon,\psi]\!] \circ \gamma$$

De plus la méthode de preuve est complète si et seulement si :

$$P(S,t,\varepsilon,\psi) \Longrightarrow [\exists \tilde{H} \in \tilde{A}(S).\ \tilde{C}[\![S,t,\varepsilon,\psi]\!](\tilde{H})]$$

ce qui implique l'existence d'une fonction $\alpha \in (A(S) \to \tilde{A}(S))$ telle que :

$$(w)\ C[\![S,t,\varepsilon,\psi]\!] \Longrightarrow \tilde{C}[\![S,t,\varepsilon,\psi]\!] \circ \alpha$$

Les fonctions γ, α peuvent être définies en partitionnant la relation de transition et l'ensemble d'états ou à l'aide de correspondances de Galois (cf [5]).

Exemple 3 :

Pour dériver de (a) la méthode de preuve de correction partielle de Owicki-Gries & Lamport (déjà évoquée pcur l'exemple 1), nous considérons des programmes for-

més de n processus $[\![\mathbb{P}_0 \| \ldots \| \mathbb{P}_{n-1}]\!]$, chaque processus $\mathbb{P}_k$ comportant r_k points de contrôle. La relation de transition correspondante a la forme $t = \bigvee_{k \in n} t_k$. On écrit $\Pi_k^i(s)$ pour signifier que dans l'état s le point de contrôle du processus $\mathbb{P}_k$ est en i. On a $\forall s \in S,\ k \in n.\ \exists i \in r_k.\ \Pi_k^i(s)$.

Dans (a) on a $A(S) = (SxS \to \{tt,ff\})$. L'idée principale de Owicki-Gries & Lamport est de décomposer l'assertion globale en assertions locales associées à chaque point de contrôle $i \in r_k$ de chaque processus $\mathbb{P}_k, k \in n$. On choisit donc:

$$\tilde{A}(S) = \prod_{k \in n} \prod_{l \in r_k} (SxS \to \{tt,ff\})$$

$$\tilde{\gamma}(I) = \lambda(s,s').\left[\bigwedge_{k \in n} \bigvee_{i \in r_k} (\Pi_k^i(s') \wedge I_k^i(s,s'))\right]$$

Si l'on calcule $C[\![S,t,\varepsilon,\psi]\!] \circ \gamma$ où C est donné par (a) on obtient après simplifications :

$$(\exists I \in \prod_{k \in n} \prod_{i \in r_k} (SxS \to \{tt,ff\}).\ \forall s,s',s'' \in S,\ k \in n,\ i,i' \in r_k,\ l \neq k,\ j \in r_l$$

$$[[\varepsilon(s) \wedge \Pi_k^i(s)] \Longrightarrow I_k^i(s,s)$$

$$(x)\ \wedge[I_k^i(s,s') \wedge \Pi_k^i(s') \wedge t_k(s',s'') \wedge \Pi_k^{i'}(s'')] \Longrightarrow I_k^{i'}(s,s'')$$

$$(y)\ \wedge[I_k^i(s,s') \wedge \Pi_k^i(s') \wedge I_l^j(s,s') \wedge \Pi_l^j(s') \wedge t_l(s',s'') \wedge \Pi_k^{i'}(s'')] \Longrightarrow I_k^{i'}(s,s'')$$

$$\wedge[I_k^i(s,s') \wedge \Pi_k^i(s')] \Longrightarrow \phi(s,s')])$$

Le terme (x) correspond à la preuve séquentielle du processus $\mathbb{P}_k$ tandis que (y) correspond au test d'absence d'interférence entre la preuve du processus $\mathbb{P}_k$ et l'exécution du processus $\mathbb{P}_l, l \neq k$. De plus l'exécution d'un pas du processus l ne peut modifier le contrôle dans le processus k et donc i'=i à la ligne (y). En choisissant :

$$\alpha(I) = \lambda k.\lambda i.\lambda(s,s').[I(s,s') \wedge \Pi_k^i(s')]$$

on vérifie aisément la condition (w) ce qui prouve la complétude sémantique de la méthode. En appliquant la

même approche à (e) (resp. (v)), on obtient une généralisation à la correction totale (resp. avec hypothèse d'équité faible), cf (7).
□

Ces éléments succints permettent d'exprimer schématiquement notre démarche pour construire une méthode de preuve. Ayant défini une sémantique opérationnelle (qui associe à tout programme un système de transition (S,t,ε)), on choisit un principe d'induction $[\exists H \in A(S). C[\![S,t,\varepsilon,\psi]\!](H)$ et un langage d'assertions $\tilde{A}(S)$ dont on définit la sémantique $\gamma \in (\tilde{A}(S) \to A(S))$. Ces choix étant faits, on détermine formellement les conditions de vérification $\tilde{C}[\![S,t,\varepsilon,\psi]\!]$ de la méthode de preuve que l'on obtient par simplification de la formule $C[\![S,t,\varepsilon,\psi]\!]_o\gamma$. La méthode de preuve obtenue est correcte par construction. Pour démontrer qu'elle est complète, nous définissons α (qui en général se déduit de γ, cf (5)) et testons (w)). Nous avons également donné des conditions suffisantes sur $\tilde{A}(S)$ et γ garantissant la complétude sémantique par construction. Le raisonnement qui est expliqué ci-dessus pour un programme doit se faire en pratique pour un langage en procédant par induction sur la syntaxe des programmes. L'obstacle évident de taille des définitions formelles de la sémantique des langages de programmation peut en partie être surmonté en considérant le langage trait pour trait.

BIBLIOGRAPHIE

1. COUSOT P., **Semantic foundations of program analysis,** in "Program flow analysis, Theory and applications", S.S. Muchnick & N.J. Jones (Eds.), Prentice-Hall, Inc., p. 303-342, 1981.

2. COUSOT P., **A Hoare-style axiomatication of Burstall's intermittent assertions method for proving inevitability properties of programs,** Rapport de recherche LRIM-83-04, Université de Metz, septembre 1983.

3. COUSOT P., On the soundness and completeness of a Hpare-style axiomatization of Burstall's intermittent assertions method, Rapport de Recherche LRIM-83-05, Université de Metz, France, septembre 1983.

4. COUSOT R., Proving invariance properties of parallel programs by backward induction, Rapport de recherche LRIM-83-02, Université de Metz, mars 1981.

5. COUSOT P., COUSOT R., Reasoning about invariance proof methods, Proc. Int. Workshop on program construction, Château de Bonas, France, Vol. 1, INRIA ed., septembre 1980.

6. COUSOT P., COUSOT R., Induction principles for proving invariance properties of programs, In "Tools and notions for program construction", Nice, France, INRIA ed., 7-18 décembre 1981, p. 75-119, Cambridge University Press, 1982.

7. COUSOT P., COUSOT R., "A la Floyd", induction principles for proving inevitability properties of programs, LRIM-82-04, juin 1983, revisé avril 1982, In "Algebraic methods in programming", M. Nivat & J. Reynolds (eds.), Cambridge University Press.

8. COUSOT P., COUSOT R., "A la Burstall" induction principles for proving inevitability properties of programs, Rapport de recherche LRIM-83-08, Université de Metz, novembre 1983.

9. COUSOT P., COUSOT R., Sometime = Aways + Recursion ≡ Always, on the equivalence of the intermittent and invariant assertions methods for proving veritability properties of programs, Rapport de recherche LRIM-83-03, Université de Metz, juillet 1983.

10. COUSOT P., COUSOT R., Invariance proof methods and analysis techniques for parallel programs, In "Automatic program constructie techniques", Biermann A., Guiho G., Kodratoff Y. (ed.), Mac Millan, 1984.