# THÈSE

présentée à l'

**INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE**

pour obtenir le grade de
DOCTEUR D'ÉTAT ÈS SCIENCES MATHÉMATIQUES

par

**Radhia COUSOT**

# FONDEMENTS DES MÉTHODES DE PREUVE D'INVARIANCE ET DE FATALITÉ DE PROGRAMMES PARALLÈLES

Thèse soutenue le 15 novembre 1985 devant le jury :

| | | | |
|---|---|---|---|
| Président | : | **C. Pair** | Rapporteur |
| Examinateurs | : | **J.P. Jouannaud** | |
| | | **G. Roucairol** | Rapporteur extérieur |
| | | **M. Sintzoff** | Rapporteur extérieur |
| | | **J.P. Verjus** | |

# FONDEMENTS DES METHODES DE PREUVE D'INVARIANCE ET DE FATALITE DE PROGRAMMES PARALLELES

Radhia COUSOT

# 1. INTRODUCTION

# 1. INTRODUCTION

## 1.1 BREF HISTORIQUE DE NOTRE DEMARCHE

## 1.2 MOTIVATION ESSENTIELLE ET COMPARAISON AVEC D'AUTRES APPROCHES

## 1.3 RESUME SUCCINT ET PLAN DE LA THESE

### 1.3.1 NOTATIONS

### 1.3.2 SEMANTIQUE OPERATIONNELLE

### 1.3.3 PROPRIETES D'INVARIANCE ET DE FATALITE DES PROGRAMMES

### 1.3.4 PREUVES D'INVARIANCE

### 1.3.5 PREUVES DE FATALITE

### 1.3.6 CONCLUSION ET REFERENCES

# 1. INTRODUCTION

## 1.1 BREF HISTORIQUE DE NOTRE DEMARCHE

La motivation de ce travail remonte à nos premières réflexions en 1975 concernant les techniques de mise au point des programmes. Partant du problème de la détermination automatique de jeux d'essais, et après l'avoir généralisé, nous avons étudié les problèmes d'analyse sémantique des programmes (c'est-à-dire la détermination statique (à la compilation) de propriétés dynamiques (à l'exécution) des programmes). Plutôt que de chercher à établir un catalogue de méthodes d'analyse, nous avons préféré étudier comment construire une méthode d'analyse sémantique quelconque à partir d'une méthode de preuve en appliquant des techniques d'approximation (cf. 4.3.2.5). Ceci nous a conduit depuis 1979-80 à nous intéresser presque exclusivement aux méthods de preuve. Comme nous avions du mal à comprendre très profondément les méthodes existentes, nous avons cherché à les formuler de manière aussi concise et rigoureuse que possible de façon à pouvoir les comparer et les généraliser. Il se pose alors très vite le problème de la justification puis de la construction d'une méthode de preuve à partir d'une sémantique et donc celui du choix de la méthode de définition de la sémantique des programmes. Ceci nous amène à l'étude des sémantiques de programmes qui est le point de départ de cette thèse, nous étudions ensuite les méthodes de preuve de propriétés d'invariance et de fatalité de programmes impératifs, séquentiels, nondéterministes ou parallèles.

## 1.2 MOTIVATION ESSENTIELLE ET COMPARAISON AVEC D'AUTRES APPROCHES

Il y a de très nombreuses façons d'aborder le problème des preuves de programmes et nous n'avons pas pu, ni voulu explorer toutes les voies de recherche possibles. Il nous semble que tous les problèmes relatifs aux preuves de programmes ne pourront faire de progrès significatifs que si les méthodes de preuve sont mieux comprises qu'elles ne le sont actuellement. Nous avons donc cherché à établir les fondements des méthodes de preuve de programmes ce qui nous a contraint à faire des choix voire des impasses :

- Avant de démontrer une propriété d'un programme il faut spécifier cette propriété. Nous n'étudierons pas le problème de la spécification de propriétés de programmes et nous ne considérerons que deux classes de propriétés à savoir l'invariance conditionnelle et la fatalité sous invariance.

- Les preuves ne sont pas encore entrées dans la pratique des programmeurs. Elles ne sont généralement appliquées formellement que sur de très petits programmes (quelques lignes) ou informellement sur de petits programmes (quelques pages). Il serait donc intéressant d'essayer de les appliquer sur des programmes moyens ou grands (quelques centaines voire milliers de pages) pour tirer de ces expériences des enseignements pratiques. Nos exemples seront toujours très courts et n'ont évidemment pas cet objectif, ils ne servent qu'à illustrer des notions abstraites. Parmi les difficultés pratiques qui rebutent les programmeurs, il y a le problème de la taille des preuves, le manque de temps, d'expérience et d'entraînement mais également le manque de connaissances (souvent réduites aux méthodes de Floyd et Dijkstra). Notre

contribution dans ce domaine vise plutôt à essayer d'élargir la panoplie des outils disponibles.

- Même pour des programmes simples, les preuves peuvent être difficiles et parfois complexes. Ceci conduit à l'idée que les preuves ne pourront être faites de manière efficace et rigoureuse que si ce sont les ordinateurs qui les font. Cette voie a beaucoup été explorée aux Etats-Unis sans rencontrer les succès espérés, à cause principalement des déficiences des démonstrateurs de théorèmes. Comme la part d'invention nécessaire pour faire des inductions est relativement faible au regard des très nombreuses conditions souvent simples qu'il faut vérifier, l'idée d'outils de preuve semi-automatiques est séduisante : l'ordinateur n'est plus utilisé pour faire mais pour aider à faire la preuve. On pourra penser à un simple aide-mémoire pour guider une preuve faite à la main. De tels systèmes interactifs sont généralement beaucoup plus ambitieux et cherchent à réduire les interventions nécessaires de l'utilisateur. Celui-ci intervient en programmant diverses stratégies possibles qui peuvent être utilisées pour tenter de faire la preuve ou bien de manière conversationnelle pour apporter la part d'invention nécessaire pour faire les inductions. Malheureusement, en cas d'échec de la preuve, les causes de l'échec sont difficilement présentables de manière synthétique à l'utilisateur. De ce fait, l'expérience a montré que les résultats deviennent vite incompréhensibles pour des programmes dépassant quelques lignes. Nous n'avons pas cherché à étudier un quelconque système d'aide à la preuve. Il nous semble que les systèmes existants ont bien souvent le défaut d'être basés sur une seule méthode qui ne marche évidemment pas à coup sûr (de même qu'en mathématiques un raisonnement peut être beaucoup plus simple qu'un autre même s'ils sont équivalents c'est-à-dire

qu'il est possible de passer formellement de l'un à l'autre). Notre apport dans ce domaine est donc essentiellement de présenter de manière uniforme des méthodes apparemment dissemblables. Dans le futur, ceci pourrait encourager à l'emploi d'une combinaison de méthodes plutôt que d'une seule.

Notre formalisme pour étudier les méthodes de preuve est de nature sémantique plutôt que syntaxique. Il fait appel aux modèles ensemblistes et non aux systèmes logiques formels. Il va un peu à l'opposé des nombreux travaux actuels qui cherchent à formaliser les méthodes de preuve à l'aide de logiques de toutes sortes (algorithmiques, dynamiques, modales, temporelles, ...). Ces travaux reposent sur l'idée que l'utilisation de systèmes formels permettra une automatisation plus facile des preuves, (ce qui reste à démontrer). L'utilisation de systèmes formels introduit des problèmes supplémentaires (comme l'incomplétude syntaxique) qu'il nous semble utile d'éliminer dans un premier temps (par exemple pour ne pas cacher, quand il se pose, le problème plus fondamental de l'incomplétude sémantique). De plus ces logiques ne sont pas indépendantes du langage auquel elles s'appliquent et sont donc peut être à la fois beaucoup moins concises, moins abstraites et moins compréhensibles que la formalisation que nous proposons. Enfin, ces logiques d'emploi assez lourd nous semblent mal adaptées pour des preuves informelles qui semblent inévitables en pratique. Il reste que notre travail devrait pouvoir trouver son utilité comme base pour étudier ces logiques (du moins celles ayant trait à l'invariance et la fatalité).

- Comme les programmes sont difficiles à comprendre (et donc à prouver) une fois qu'ils sont terminés, il semble que l'étude des méthodes de construction de programmes (et des preuves concomitantes)

soit plus prometteuse que celle des preuves de correction à postériori. En suivant cette démarche il reste qu'à chaque étape de la construction, il faut démontrer des propriétés relatives à cette étape. Pour ce faire, les méthodes de preuve de propriétés de programmes sont indispensables (et ce d'autant plus que nous proposons une notion abstraite du comportement d'un programme). D'autre part lors du passage d'une étape à la suivante (par exemple par transformation) un certain nombre de propriétés doivent être conservées sans que les preuves soient à refaire. Notre contribution dans ce domaine concerne l'étude de relations entre sémantiques qui conservent les propriétés d'invariance et de fatalité. Il s'agit bien évidemment de quelques résultats techniques et nous n'avons pas l'ambition de proposer une méthodologie de la programmation par transformation de programmes.

- Notre étude adopte un point de vue endogène c'est-à-dire que le comportement du programme est supposé complètement donné sans rien connaître de ce qui lui est extérieur : ce point de vue s'oppose au point de vue exogène où on s'intéresse seulement à une composante du programme (sous-programme, module, processus, etc.) dont le comportement dépend de causes externes et agit sur l'extérieur (dont la connaissance est en général imparfaite). Cette distinction ne nous paraît pas essentielle dans la mesure où l'état du système peut englober l'état du programme et la partie de l'état de l'extérieur ayant un intérêt pour la preuve.
La sémantique du système n'étant pas nécessairement close, on peut étudier des propriétés de systèmes dont l'évolution ne dépend pas uniquement de leur état courant mais dépend de leur histoire (et donc de causes extérieures inconnues et ignorées). Dans le même ordre d'idées, le fait que nous considérions des preuves

relatives à un programme donné, nous conduit à étudier la façon de décomposer cette preuve en fonction de la structure (des états ou des actions) du programme mais nous n'avons pas étudié le problème dual qui consiste à étudier comment la composition d'un programme à partir de parties induit une construction de la preuve du programme par composition des preuves des parties, (exemple de la méthode de Hoare).

Ayant brièvement présenté les voies de recherche qu'il nous aurait été possible d'explorer mais que nous n'avons pas choisies, nous résumons succintement maintenant le contenu de cette thèse.

## 1.3 RESUME SUCCINT ET PLAN DE LA THESE

### 1.3.1 NOTATIONS

Nous utiliserons évidemment des notations mathématiques classiques (concernant la logique, les ensembles, les ordres, les ordinaux, les séquences et les cardinaux) qui sont résumées dans l'annexe I. Il est préférable de commencer par lire cette annexe, mais pour l'éviter nous donnons en annexe II un index des notations mathématiques qui pourra être consulté au fur et à mesure des besoins. Les notations informatiques sont introduites au fil du texte et résumées dans un index donné en annexe III.

### 1.3.2 SEMANTIQUE OPERATIONNELLE

Le chapitre 2 est consacré à l'étude de la sémantique opérationnelle des programmes.

Toutes les méthodes de preuve de programmes utilisent la notion de "pas de programme". Ceci conduit donc naturellement à formaliser la sémantique d'un programme par un système de transition (cf. 2.2) formé par un ensemble d'états (en général couple état mémoire - état contrôle), un ensemble d'actions, une caractérisation des états initiaux et une relation de transition qui pour toute action "a" caractérise les paires d'états "$s$" et "$s'$" telles que par exécution de l'action "a" on peut passer de l'état "$s$" dans l'état "$s'$".

La sémantique engendrée par ce système de transition (cf. 2.4) est formée par les ensembles d'états, d'actions et de traces engendrés par

ce système de transition. Une trace est une suite d'états séparés par des actions (conformément à la relation de transition) qui commence par un état initial et est infinie ou bien se termine par un état de blocage (sans successeur possible). Les traces représentent donc un calcul fini et achevé ou bien un calcul infini mais jamais un calcul en cours.

Malheureusement, toutes les sémantiques de programmes ne sont pas directement engendrées par un système de transition. Par exemple, les sémantiques de programmes parallèles équitables ne le sont pas car l'évolution du calcul ne dépend pas uniquement de l'état courant (qui ne contient pas toutes les informations nécessaires pour déterminer l'évolution future des calculs). Autrement dit, les systèmes de transition permettent de rendre compte de l'évolution des programmes quand elle ne dépend que de l'état courant mais pas quand elle dépend de l'histoire du calcul pour arriver dans cet état. Par soucis de généralité, nous sommes donc conduits à définir une sémantique (cf. 2.1) comme un ensemble d'états, un ensemble d'actions et un ensemble de traces quelconques.

Pour formaliser la notion de "pas d'exécution" d'un programme dont la sémantique est arbitraire nous définissons (cf. 2.3) la notion de système de transition engendré par une sémantique. Les transitions sont simplement celles qu'on peut observer le long d'une trace quelconque.

Pour faire des preuves de programmes il est souvent plus pratique de ne pas raisonner sur la sémantique du programme mais plutôt sur une sémantique qui lui est proche. De telles techniques sont souvent utilisées sans justification. Pour démontrer leur validité (aux chapitres 4 pour l'invariance et 5 pour la fatalité) il est nécessaire de les formaliser. Nous le faisons au moyen de relations entre sémantiques qui sont étudiées aux paragraphes 2.5 et 2.6.

Par exemple pour faire une preuve de correction partielle d'un programme parallèle, on peut ignorer l'hypothèse que son exécution est faiblement équitable (tout processus toujours activable est fatalement activé). Nous formaliserons la relation entre la sémantique non équitable et la sémantique équitable de ce programme au moyen d'une fermeture d'une sémantique par réduction aux traces équitables (cf. 2.6.4). Dans une preuve d'invariance, on peut ne pas tenir compte des branches du programme qui sont mortes. Pour le démontrer nous introduirons la notion de fermeture d'une sémantique par réduction aux états accessibles (cf. 2.6.3). Enfin, il est fréquent d'utiliser des variables auxiliaires pour démontrer la correction partielle de programmes parallèles. La relation entre la sémantique du programme et celle du programme transformé (contenant les variables auxiliaires) peut être définie par réduction des actions inobservables (pour éliminer les affectations aux variables auxiliaires) (cf. 2.5.4.2) et par concordance à une relation entre états près (pour éliminer les variables auxiliaires). Lorsque nous définirons une relation entre sémantiques, ceci induit une relation entre systèmes de transition (en considérant la relation entre les sémantiques qu'ils engendrent) dont nous étudions les propriétés.

Nous constatons au paragraphe 2.5 qu'en général, une sémantique et le système de transition qu'elle engendre ne donnent pas les mêmes informations. En effet, une sémantique est en général différente de sa rétraction par transitions c'est-à-dire de la sémantique engendrée par le système de transition qu'elle engendre. Nous disons que la sémantique est close dans le cas contraire. Ceci nous nous amène (cf. 2.6.8) à la caractérisation des sémantiques closes (à l'aide de divers opérateurs sur les sémantiques définis au paragraphe 2.6). Pour résumer, très intuitivement, une sémantique

est close si elle est fermée par fusion (cf. 2.6.5, le comportement futur de l'exécution dépend seulement de l'état courant qui a été atteint et non de la façon dont il a été atteint), réduite par élimination des traces préfixes stricts (cf. 2.6.6, l'arrêt de l'exécution en un état ne dépend que cet état) et fermée par limites (cf. 2.6.7, les limites des comportements finis sont des comportements infinis acceptables).

Nous passons au paragraphe 2.7 le problème de la spécification de la sémantique d'un programme. Quand la sémantique est close (cf. 2.7.1), on peut le faire aisément à l'aide d'un système de transition (qui peut être lui-même défini par induction sur la syntaxe du programme). Pour une sémantique non close, ce n'est pas possible directement. On peut toujours le faire indirectement en incluant un résumé de l'histoire des calculs dans les états et un contrôleur pour surveiller les transitions (cf. 2.7.2.2) ou bien spécifier la sémantique non close comme un sous-ensemble des préfixes des traces engendrées par un système de transition (cf. 2.7.2.1). Quand la sémantique est non close mais réduite par élimination des traces préfixes strictes, il suffit de considérer un sous-ensemble des traces engendrées par un système de transition (cf. 2.7.3).

Nous terminons ce chapitre par des exemples de définitions de la sémantique d'un langage de programmation (cf. 2.8). Il s'agit d'illustrer ce qui vient d'être dit sur les méthodes de spécification de sémantiques de programmes mais surtout de disposer d'exemples qui nous serviront dans la suite pour illustrer la construction systématique de méthodes de preuves. Nous définissons la syntaxe et la sémantique opérationnelle des programmes séquentiels (cf. 2.8.1, qui sont composés d'affectations (ordinaires ou aléatoires), de conditionnelles

et d'itérations), de programmes parallèles asynchrones (cf. 2.8.2, qui sont composés de processus séquentiels partageant des données communes) auxquels nous ajoutons en 2.8.3 la possibilité de communiquer sur rendez-vous par envoi et réception de messages sur des canaux (avec possibilité de sélection entre plusieurs alternatives comme dans CSP ou ADA) et en 2.8.4 l'hypothèse d'exécution faiblement équitable. Finalement, nous ajoutons en 2.8.5 la possibilité de synchroniser les processus au moyen de sémaphores. Pour nous convaincre que la définition des sémaphors que nous avons utilisée est bien celle proposée à l'origine par Dijkstra, nous démontrons certaines propriétés des sémaphores qui ont été énoncés par Habermann et sont généralement tenues pour vraies sans justification. Enfin nous donnons une sémantique libérale des sémaphores qui peut être utilisée pour démontrer des propriétés d'invariance.

### 1.3.3 PROPRIETES D'INVARIANCE ET DE FATALITE DES PROGRAMMES

Au chapitre 3 nous définissons et illustrons très brièvement les propriétés de programmes que nous considérons dans cette thèse à savoir l'invariance (cf. 3.2) et la fatalité (cf. 3.3).

La propriété d'invariance conditionnelle (cf. 3.2.1) est la propriété d'invariance la plus générale que nous considérons. Nous disons que $\psi$ est invariante sous condition $\phi$ pour une sémantique si pour toute trace de cette sémantique et tout état courant dans cette trace la relation $\psi$ est vraie entre l'état initial et l'état

courant quand la relation $\phi$ a été vraie entre l'état initial et tous les états précédant l'état courant. Nous parlons d'invariance relationnelle (cf. 3.2.2) quand $\phi$ est toujours vraie et d'invariance assertionnelle (cf. 3.2.3) quand de plus $\psi$ ne dépend pas de l'état initial.

Ces définitions recouvrent un grand nombre de propriétés classiques des programmes comme la correction partielle, l'absence d'erreurs à l'exécution, la non-terminaison, l'exclusion mutuelle, l'absence d'interblocages globaux permanents et des propriétés moins classiques comme les propriétés de précédence du genre l'état courant ne peut pas satisfaire $\psi$ sans qu'un état précédent ait satisfait $\phi$.

La propriété de fatalité sous invariance (cf. 3.3.1) est la propriété de fatalité la plus générale que nous considérions. Nous dirons que $\psi$ est fatale sous invariance de $\phi$ pour une sémantique si pour toute trace de cette sémantique il existe un état (dit but) telle que la relation $\psi$ soit vraie entre l'état initial et le but et la relation $\phi$ soit vraie entre l'état initial et tous les états qui précèdent le but. A nouveau, nous parlons de fatalité relationnelle quand $\phi$ est toujours vraie et de fatalité assertionnelle si de plus $\psi$ ne dépend pas de l'état initial.

Ces définitions recouvrent également un grand nombre de propriétés classiques des programmes comme la terminaison, la correction totale, la garantie d'entrée en section critique ou de réponse à un signal, l'absence de famine d'un processus, etc.

D'autres propriétés des programmes peuvent également se ramener à l'invariance et à la fatalité par exemple en considérant les suffixes de la sémantique du programme ou d'autres relations entre sémantiques comme étudiées au chapitre 2.

### 1.3.4 PREUVES D'INVARIANCE

Le chapitre 4 est consacré aux fondements des méthodes de preuve de propriétés d'invariance de programmes séquentiels, non-déterministes ou parallèles.

Nous commençons par étudier au paragraphe 4.1 des relations entre sémantiques qui conservent l'invariance avec l'idée que pour démontrer une propriété d'invariance d'un programme relativement à une sémantique nous pouvons essayer de nous ramener à la preuve d'une propriété similaire relativement à une autre sémantique (généralement plus simple).

Par exemple, les propriétés d'invariance sont conservées pour des sémantiques concordantes à des relations ou fonctions entre états et/ou actions près (cf. 4.1.1). Elles sont également conservées après réduction des états inobservables (cf. 4.1.2). La propriété la plus importante est que pour faire une preuve d'invariance pour une sémantique, il est toujours correct de raisonner sur le système de transition qu'elle engendre. Autrement dit les propriétés d'invariance sont conservées par rétraction de la sémantique par transitions (cf. 4.1.3). Cette méthode de preuve est correcte mais elle n'est en général pas sémantiquement complète c'est-à-dire qu'il se peut qu'une propriété

soit invariante pour une sémantique mais pas pour sa rétraction par transitions. Toutefois la méthode est sémantiquement complète quand la sémantique est fermée par fusion (cf. 4.1.3 ∨ 4) et donc en particulier pour les langages considérés au paragraphe 2.8.

Au paragraphe 4.2 nous étudions les principes d'induction qu'on peut utiliser pour démontrer les propriétés d'invariance des programmes. Un principe d'induction décrit l'essence d'une méthode de preuve de manière très concise et abstraite.

Comme nous avons vu dans le paragraphe 4.1, nous rencontrons très fréquemment (mais pas toujours) des sémantiques closes. Nous commençons donc par étudier ce cas particulier (cf. 4.2.1). La méthode la plus connue pour démontrer des propriétés d'invariance de programmes est la méthode de Floyd, Naur et Hoare. Partant d'un exemple, nous inférons le principe d'induction de base pour cette méthode (cf. 4.2.1.1). Essentiellement celui-ci exprime la propriété assez évidente suivante : pour démontrer que la fermeture transitive réflexive $t^*$ d'une relation de transition $\exists a \in A.\, t_a$ entraîne une relation invariante $\psi$ $(\forall s,s' \in S.\, t^*(s,s') \Rightarrow \psi(s,s'))$ il faut et il suffit qu'il existe un invariant $I$ plus fort que $\psi$ $(\forall s,s' \in S.\, I(s,s') \Rightarrow \psi(s,s'))$ qui soit vrai quand l'état courant est un état initial $(\forall \underline{s} \in S.\, I(\underline{s},\underline{s}))$ et reste vrai pour tous les descendants possibles des états initiaux $(\forall \underline{s}, s, s' \in S, a \in A.\, (I(\underline{s},s) \wedge t_a(s,s')) \Rightarrow I(\underline{s},s'))$. Dans ce chapitre nous traitons de l'invariance relationnelle en remarquant que la généralisation à l'invariance conditionnelle est triviale (cf. 4.2.1.1-2).

Nous étudions ensuite les variantes possibles de ce principe d'induction de base de façon à en dériver toutes les méthodes existantes plus quelques autres. Pour que l'étude soit systématique nous considérons des transformations de principes d'induction (cf. 4.2.1.2). Pour une propriété de la forme $([E(\underline{s}) \wedge \delta(\bar{s})] \Rightarrow \psi(\underline{s},\bar{s}))$ relative à une relation entre états initiaux et finaux, il est possible de restreindre l'invariant aux états initiaux (cf. 4.2.1.2.1). Une autre transformation également triviale (cf. 4.2.1.2.2) consiste à remarquer que la condition de vérification $(\forall s,s' \in S, a \in A.\ [\exists s \in S.\ I(s,s') \wedge t_a(s,s')] \Rightarrow I(\underline{s},s'))$ est équivalente à $(\forall \underline{s},s \in S.\ I(\underline{s},s) \Rightarrow \neg[\exists s' \in S, a \in A.\ t_a(s,s') \wedge \neg I(\underline{s},s')])$. Autrement dit, nous pouvons utiliser une plus forte post-condition (comme Floyd pour l'affectation) ou bien une plus faible pré-condition (comme Hoare pour l'affectation). Une autre transformation (cf. 4.2.1.2.3) consiste à remarquer que nous pouvons raisonner sur les relations inverses ($\tau^* \Rightarrow \psi$ si et seulement si $(\tau^{-1})^* \Rightarrow \psi^{-1}$). Ceci nous permet de formaliser la méthode de Morris-Wegbreit dite "subgoal induction" qui consiste donc à appliquer la méthode de Floyd sur l'inverse du programme. Il est également possible (cf. 4.2.1.2.4) de remplacer l'invariant I par sa négation ce qui conduit à des preuves contrapositives par l'absurde (qui sont ignorées dans la littérature). Enfin (cf. 4.2.1.2.5), quand la propriété à démontrer est une assertion (au lieu d'une relation), nous pouvons utiliser une assertion (comme dans la méthode de Floyd-Naur) au lieu d'une relation invariante (comme dans la méthode de Manna).

Dans le paragraphe 4.2.1.3 nous déterminons tous les principes d'induction que nous pouvons dériver par les transformations ci-dessus, ce qui permet de retrouver toutes les méthodes classiques et de découvrir quelques autres.

Nous montrons ensuite que tous ces principes d'induction sont fortement équivalents (cf. 4.2.1.4) en ce sens que si nous avons

découvert l'invariant I qui convient pour un principe d'induction nous pouvons déterminer l'invariant I' qui convient pour faire la preuve avec tout autre principe d'induction. (Ceci n'empêche d'ailleurs pas que la preuve avec un principe d'induction soit plus facile qu'avec un autre). Ils sont également corrects (si l'invariant I satisfait les conditions de vérification alors $\psi$ est invariante) et sémantiquement complets (si $\psi$ est invariante, nous pouvons toujours trouver un invariant I satisfaisant les conditions de vérification (mais nous n'avons pas forcément de complétude syntaxique en ce sens que si le langage d'assertions est mal choisi, nous ne pouvons peut-être pas formuler I dans ce langage)).

Les résultats précédents se généralisent aux sémantiques non closes fermées par fusions (cf. 4.2.2) puisque dans ce cas une propriété est invariante pour cette sémantique si et seulement si elle l'est pour la réduction de cette sémantique par transitions.

Ceci n'est évidemment pas vrai pour une sémantique non fermée par fusions (cf. 4.2.3).

Si cette sémantique a été définie comme un sous-ensemble des préfixes des traces engendrées par un système de transition, il est correct mais pas sémantiquement complet d'utiliser les principes d'induction précédents pour ce système de transition. Pour être complets, nous proposons un principe d'induction utilisant des variables auxiliaires (dans la preuve mais pas dans le programme) permettant de cumuler des histoires. Ceci permet de tenir compte dans le principe d'induction proposé du fait que les successeurs possibles d'un état ne dépendent pas uniquement de cet état mais également de la façon dont il a été atteint (ce que nous savons quand nous cumulons l'histoire).

Si cette sémantique non fermée par fusions a été définie par concordance avec une sémantique close (cf. 4.2.3.2), nous pouvons aisément nous ramener aux principes d'induction que nous étudiés pour les sémantiques closes.

De plus nous montrons que ces nouveaux principes d'induction se ramènent aux précédents quand la sémantique est fermée par fusions.

Enfin nous montrons (cf. 4.2.3.3) que les deux approches (cumul de l'histoire dans des variables auxiliaires ou utilisation d'une sémantique close concordante) sont fortement équivalentes.

Les principes d'induction tels que nous les avons proposés ne sont pas très pratiques à mettre en œuvre directement dans une preuve. Par exemple, dans la condition de vérification $(I(\underline{s},s) \wedge t_a(s,s')) \Rightarrow I(\underline{s},s')$, $t_a$ serait une énorme formule définissant l'exécution d'un pas quelconque du programme. Il est donc préférable de décomposer cette condition de vérification complexe en une conjonction de conditions de vérification plus simples correspondant par exemple chacune à un pas élémentaire du programme. C'est l'objet du paragraphe 4.3 qui porte sur la construction systématique d'une méthode de preuve d'invariance à partir d'une sémantique opérationnelle et d'un principe d'induction par décomposition de l'invariant global en invariants locaux.

Plutôt que d'imaginer empiriquement une méthode de preuve pour un langage de programmation puis de démontrer sa correction et sa complétude sémantique à posteriori, nous proposons de construire la méthode de preuve de manière systématique. La démarche (cf. 4.3.1) consiste tout d'abord à définir la sémantique opérationnelle

à l'aide d'un système de transition (cf. 4.3.1.1), puis à définir la propriété invariante à démontrer (cf. 4.3.1.2) ce qui permet de choisir le principe d'induction adéquat (cf. 4.3.1.3) parmi ceux précédemment proposés. Ce principe d'induction fait intervenir un invariant global (portant sur les états du programme) alors qu'on préfère généralement des invariants locaux (qui portent par exemple sur les états du programmes correspondant à chaque point du programme, ce qui permet par exemple dans la méthode de Floyd de les associer en commentaire au point du programme auquel ils correspondent). Les invariants locaux étant choisis (cf. 4.3.1.4), il faut définir leur sémantique (cf. 4.3.1.5) c'est-à-dire définir l'invariant global qui correspond à des invariants locaux et inversement. La détermination de la méthode de preuve consiste alors à dériver les conditions de vérification correspondantes. Ceci (cf. 4.3.1.7) en remplaçant le système de transition par sa définition et l'invariant global par les invariants locaux dans le principe d'induction qui a été choisi. Il ne reste ensuite qu'à simplifier pour obtenir les conditions de vérification élémentaires. La méthode obtenue est correcte par construction. Il faut ensuite vérifier qu'elle est sémantiquement complète (cf. 4.3.1.8). Cette vérification peut être inutile ou simplifiée selon la nature de la relation entre invariant global et invariants locaux. C'est pourquoi nous remarquons au paragraphe 4.3.1.6.1 qu'en général, l'ensemble des invariants locaux forme un treillis qui correspond au treillis des invariants globaux (qui sont des sous-ensembles de l'ensemble des paires d'états). Nous étudions ensuite (cf. 4.3.1.6.2) diverses propriétés possibles des correspondances entre invariants locaux et globaux (correspondances monotones, (demi- ou quasi-) correspondances de Galois (injectives, surjectives), isomorphismes complets).

Dans le paragraphe suivant 4.3.2, nous donnons des exemples de construction de méthode de preuve en commençant par la

construction d'une méthode de preuve de non-terminaison, d'absence d'erreurs à l'exécution et d'invariance globale, par l'absurde pour les programmes séquentiels (cf. 4.3.2.1). Comme ces méthodes ne sont pas classiques, nous les illustrons par des exemples simples.

Ensuite (cf. 4.3.2.2), nous étendons la méthode de Morris-Wegbreit (dite "subgoal induction" pour démontrer la correction partielle de programmes séquentiels) aux programmes parallèles (comme Owicki-Gries ont étendu la méthode de Floyd-Naur-Hoare aux programmes parallèles asynchrones) et nous la généralisons à d'autres propriétés d'invariance. Nous commençons par considérer la correction partielle des programmes séquentiels (cf. 4.3.2.2.1-4) puis l'invariance globale (alors que Morris-Wegbreit croyaient que ce n'était pas possible). Nous abordons ensuite la correction partielle de programmes parallèles asynchrones (cf. 4.3.2.2.2) où nous retrouvons la décomposition de la preuve en une preuve séquentielle par processus et une preuve d'absence d'interférences. Cette méthode étant nouvelle nous des exemples d'application (comme le calcul parallèle asynchrone de n!).

Ayant remarqué que dans les méthodes "en avant" (à la Floyd) l'invariant décrit ce qui a été fait (relation entre l'état initial et l'état courant) alors que pour les méthodes "en arrière" (à la Morris-Wegbreit) l'invariant décrit ce qui reste à faire (relation entre l'état courant et l'état final) et que ces deux types d'informations peuvent être très utiles pour aider à la compréhension du programme, nous proposons un principe d'induction (cf. 4.3.2.2.5-3) combinant les avantages de ces deux méthodes (mais où la preuve présente évidemment des redondances).

Nous généralisons ensuite cette méthode aux programmes parallèles synchrones pour la preuve d'absence d'interblocages

globaux permanents (cf. 4.3.2.2.3), la preuve d'exclusion mutuelle (cf. 4.3.2.2.4) et la preuve de non-terminaison (cf. 4.3.2.2.5).

Nous tentons ensuite (cf. 4.3.2.2.6) d'expliquer pourquoi la méthode de Morris-Wegbreit n'a pas eu le même succès que la méthode de Floyd. La raison principale nous semble bien mise en évidence dans le cas des programmes parallèles où les mêmes invariants peuvent être utilisés, pour les méthodes "à la Floyd", pour démontrer la correction partielle, l'absence d'erreurs à l'exécution, l'absence d'interblocages, l'exclusion mutuelle etc., alors que ce n'est pas le cas pour les méthodes "à la Morris-Wegbreit". Pour remédier à cet inconvénient, nous proposons d'utiliser un principe d'induction combinant l'induction en avant et en arrière.

Nous terminons cette série d'exemples en construisant une méthode de preuve de propriétés d'invariance pour les programmes parallèles communicants (cf. 4.3.2.3). Pour toutes ces méthodes que nous avons construites, nous avons donné une preuve de correction et de complétude sémantique.

Les méthodes de preuve de propriétés d'invariance pour les programmes parallèles connues dans la littérature (Aschroft, Hoare, Howard, Keller, Lamport, Mazurkiewitz, Newton, Owicki-Gries, ...) sont souvent difficiles à comparer à cause des formalismes souvent très différents qui sont utilisés pour les présenter. L'objectif du paragraphe 4.3.2.4 est de montrer qu'elles dérivent toutes du même principe d'induction (qui est à la base de la méthode de Floyd) et ne diffèrent que par la façon de décomposer l'invariant global utilisé dans ce principe d'induction en invariants locaux associés à des points du programme.

Par exemple, la méthode de Ashcroft et de Keller consiste à utiliser un seul invariant global (cf. 4.3.2.4.1). Ce fut la première généralisation de la méthode de Floyd aux programmes parallèles mais elle a l'inconvénient qu'il y a une seule condition de vérification qui n'est pas décomposée en conditions élémentaires.

A l'inverse, la méthode de Aschroft-Manna consiste à utiliser un invariant local associé à chaque état de contrôle (cf. 4.3.2.4.3). Dans ce cas, la décomposition est par contre trop fine, et par conséquent le nombre de conditions de vérification est trop grand.

Un premier compromis dans la méthode de Owicki-Gries consiste à utiliser un invariant local sur les variables, associé à chaque point (et non plus à chaque état) de contrôle du programme (cf. 4.3.2.4.3). Mais cette méthode est sémantiquement incomplète.

Pour y remedier, nous pouvons suivre Newton et Lamport et utiliser des invariants locaux portant sur l'état de contrôle et les variables, associés à chaque point de contrôle du programme (cf. 4.3.2.4.4).

Nous pouvons également comme l'ont proposé Owicki-Gries utiliser des invariants locaux portant sur les variables du programme et sur des variables auxiliaires, associés à chaque point de contrôle du programme (cf. 4.3.2.4.5). Nous montrons que la méthode est correcte et sémantiquement complète (en définissant la sémantique auxiliaire, qui peut toujours être utilisée pour faire la preuve, à une réduction des états inobservables et à une fonction des états près. Dans cette sémantique auxiliaire, les états de contrôle sont simplement simulés par des variables, ce qui montre que les variables auxiliaires ne servent dans la méthode de Owicki-Gries qu'à simuler l'état de contrôle dont les invariants locaux sont indépendants).

Avec ces décompositions, le nombre de conditions de vérification est proportionnel au produit des longueurs des processus du programme parallèle alors qu'en pratique on souhaite que le nombre de conditions de vérification croisse linéairement avec la taille du programme. C'est le cas avec une méthode proposée par Lamport qui consiste à utiliser des invariants locaux portant sur l'état de contrôle et les variables, associés à chaque processus du programme parallèle

Il est possible enfin, d'utiliser une information redondante (comme dans des méthodes proposées par Hoare, Howard, ...) sous la forme d'un invariant global et d'invariants locaux associés à divers points du programme (cf. 4.3.2.4.7).

Cette profusion de méthodes nous amène à les classer selon le principe d'induction sous-jacent et selon la finesse de la décomposition de l'invariant global en invariants locaux (cf. 4.3.2.4.8).

Pour conclure ce paragraphe, il nous semble que les exemples que nous avons donnés montrent que le choix de la finesse de la décomposition de l'invariant global en invariants locaux ne devrait pas être fixée une fois pour toutes dans une méthode de preuve. Il est bien préférable de choisir cette décomposition en fonction du problème à traiter. La formalisation des méthodes de preuve d'invariance que nous avons proposée permet de le faire sans difficultés.

Nous terminons ce chapitre sur les preuves d'invariance par le paragraphe 4.3.2.5 consacré à l'analyse sémantique de programmes. Nous le faisons parce que ce problème (qui rappelons le, consiste

à déterminer statiquement et automatiquement des invariants pour un programme) a motivé le travail que nous présentons ici. Nous le faisons surtout pour montrer que les résultats obtenus se généralisent sans peine aux programmes parallèles. La présentation est très brève. Nous rappelons simplement comment faire une analyse sémantique "en avant" (cf. 4.3.2.5.1, déterminer un sur-ensemble des descendants des états initiaux), comment faire une analyse sémantique "en arrière" (cf. 4.3.2.5.2, déterminer un sur-ensemble des ascendants des états finaux) et comment faire une analyse combinée "avant-arrière" (cf. 4.3.2.5.3, déterminer un sur-ensemble des états qui sont à la fois descendants des états initiaux et ascendants des états finaux). Comme le formalisme utilisé est très général et qu'il englobe les programmes parallèles nous nous contentons de donner quelques exemples pour montrer l'application des méthodes d'analyse sémantique à ce type de programmes.

### 1.3.5 PREUVES DE FATALITE

Le chapitre 5 est consacré à l'étude des méthodes de preuve de propriétés de fatalité des programmes séquentiels et parallèles.

Un certain nombre de résultats obtenus au chapitre précédent (comme les transformations de principes d'induction, la décomposition de l'invariant global en invariant locaux, ...) s'appliquent également ici (avec les légers adaptations qui pourraient être nécessaires). Pour éviter les répétitions, nous ne reprendrons pas ces mêmes idées dans ce chapitre même si elles s'appliquent.

Ce chapitre comprend deux paragraphes importants, le paragraphe 5.2 consacré aux principes d'induction "à la Floyd" et le paragraphe 5.3 consacré aux principes d'induction "à la Burstall". En fait nous aurions pu rédiger différemment en présentant 5.2 en quelques phrases comme un cas particulier de 5.3. Nous avons choisi d'aller du particulier (5.2) au général (5.3) de façon à refléter l'évolution historique mais surtout pour graduer les difficultés. Pour éviter les redites nous présentons en 5.2 un certain nombre de résultats (comme les principes d'induction pour les sémantiques non closes, ...) qui ne seront pas repris en 5.3 car leur généralisation ne nous a pas semblé présenter des difficultés une fois que l'idée a été donnée en 5.2.

Nous commençons l'étude des méthods de preuve de fatalité par celle des relations entre sémantiques qui conservent la fatalité (cf. 5.1). Malheureusement, les résultats positifs sont beaucoup moins nombreux que pour l'invariance. Sans chercher l'exhaustivité, nous montrons (cf. 5.1.1) que les propriétés de fatalité sont conservés par inclusion de sémantiques (mais dans un sens seulement car par exemple il n'est pas toujours possible de démontrer une propriété de fatalité d'un programme parallèle synchrone en raisonnant sur la sémantique libérale des sémaphores) et que (cf. 5.1.2) les propriétés de fatalité sont conservées pour des sémantiques concordantes à des relations entre états et actions près (dans les deux sens, sous certaines conditions).

Dans le paragraphe 5.2, nous étudions les preuves de fatalité par des principes d'induction généralisant la méthode de Floyd.

Nous commençons par rappeler en 5.2.1 la méthode de Floyd (dite des assertions invariantes et de l'ordre bien fondé) pour démontrer la correction totale des programmes séquentiels.

Ceci nous permet d'en extraire le principe d'induction de base pour démontrer les propriétés de fatalité des sémantiques closes (cf. 5.2.2).

Nous étudions ensuite une série de principes d'induction équivalents au principe d'induction de base (cf. 5.2.3) qui reflètent quelques unes des variantes possibles de la méthode de Floyd. Par exemple, il n'est pas nécessaire d'associer une fonction de terminaison à tous les points de contrôle du programme mais seulement aux points de coupure des boucles. L'utilisation de bons-ordres n'est pas obligatoire puisque les relations bien-fondées suffisent et sont quelquefois plus commodes. La fonction de terminaison peut être remplacée par une variable auxiliaire (dans la preuve mais qui n'apparaît pas nécessairement dans le programme) qui décroît strictement à chaque pas. Cette variable auxiliaire peut toujours être choisie comme un ordinal, etc.

Nous abordons ensuite le problème de la correction et de la complétude sémantique des principes d'induction à la Floyd qui précèdent (cf. 5.2.4).

Nous remarquons en 5.2.5 que si la propriété fatale est une relation entre les états initiaux et finaux il faut en général, que la fonction de terminaison porte sur l'état courant mais également

sur l'état initial (ce que beaucoup d'ouvrages introductifs ignorent. Ils présentent donc une méthode sémantiquement incomplète).

Il est également intéressant de caractériser les relations bien-fondées (ou de manière équivalente les ordinaux) qui sont nécessaires pour faire des preuves de fatalité basées sur les principes d'induction généralisant la méthode de Floyd (cf. 5.8.6). Pour ce faire, nous disons que le non-déterminisme d'une sémantique est $\mathfrak{m}$-borné si le cardinal de l'ensemble des successeurs d'un état quelconque pour la relation de transition qu'elle engendre est strictement inférieur à $\mathfrak{m}$. En particulier le non-déterminisme est fini (Dijkstra dit "borné") si tout état a un nombre fini de successeurs possibles. Nous montrons que pour une sémantique close dont le non-déterminisme est $\mathfrak{m}$-borné, il est toujours possible de faire des preuves de fatalité avec des relations bien-fondées dont l'ordre est inférieur à $\mathfrak{m}^+$ (où $\mathfrak{m}^+ = \omega$ si $\mathfrak{m} < \omega$, $\mathfrak{m}^+ = \mathfrak{m}$ quand $\mathfrak{m}$ est un cardinal régulier sinon $\mathfrak{m}^+$ qui est le plus petit cardinal strictement supérieur à $\mathfrak{m}$). Cette limite est stricte quand $\mathfrak{m}$ est régulier. Comme cas particulier, nous obtenons que la méthode de Knuth-Luckham-Suzuki (qui consiste à utiliser un compteur strictement incrémenté à chaque tour de boucle et dont la valeur est bornée) n'est pas sémantiquement complète quand le non-déterminisme n'est pas fini (et ne peut donc pas être généralisée au cas des programmes parallèles équitables).

Le paragraphe 5.2.7 est consacré à la décomposition des conditions de vérification. Le cas général ayant été étudié en 4.3, nous nous contentons d'illustrer quelques décompositions (de façon à montrer comment les méthodes de Lamport et Owicki-Gries initialement conçues pour les preuves de correction partielle peuvent être étendues aux preuves de correction totale).

N'ayant abordé jusqu'ici que le cas des sémantiques closes, le cas des preuves de fatalité pour les programmes parallèles équitables étaient exclues. C'est pourquoi nous étudions au paragraphe 5.2.8 les principes d'induction "à la Floyd" pour démontrer les propriétés de fatalité de sémantiques non closes.

Comme pour l'invariance, nous pouvons définir la sémantique non close par concordance avec une sémantique close à une fonction d'états près (cf. 5.2.8.1). Dans ces conditions, n'importe lequel des principes d'induction introduits pour les sémantiques closes est utilisable. Ceci revient, par exemple pour un programme parallèle équitable, à raisonner sur un programme transformé qui incorpore un contrôleur d'exécution assurant l'équité.

Une autre approche (cf. 5.2.8.2) peut être utilisée quand nous spécifions la sémantique non close par un sous-ensemble des préfixes des traces engendrées par un système de transition. Elle consiste à cumuler l'histoire des calculs dans une variable auxiliaire. Ceci permet, quand la sémantique n'est pas fermée par limites de ne pas imposer que la fonction de terminaison décroisse à chaque pas mais seulement aux points de coupure qui ne sont pas déterminés statiquement comme dans la méthode de Floyd mais dynamiquement c'est-à-dire en fonction de l'histoire des calculs. Comme cas particulier nous retrouvons la méthode de Pnueli-Lehmann-Shavi pour démontrer la correction totale de programmes parallèles faiblement équitables et qui consiste essentiellement à appliquer la méthode de Floyd mais avec la possibilité que la fonction de terminaison ne décroisse pas tant qu'un processus reste activable sans être activé. De plus, quand la sémantique n'est pas réduite par élimination des traces préfixes strictes, l'histoire est utilisée pour s'assurer que la

but est atteint pour les traces finies avant la fin de la trace (qui peut ne pas être un état sans successeur). Enfin, quand la sémantique n'est pas fermée par fusion, l'invariant doit être vrai pour tous les états qui peuvent être atteints en suivant le préfixe d'une trace où le but n'est jamais atteint mais pas forcément pour tous les préfixes obtenus par transitions successives. Là encore, le cumul de l'histoire dans une variable auxiliaire est utile.

Enfin, nous montrons au paragraphe 5.2.8.3 que les deux approches (utilisation d'une sémantique auxiliaire incluant un contrôleur d'exécution ou bien utilisation de variables auxiliaires pour cumuler l'histoire) sont équivalentes.

Au paragraphe 5.3.1, nous présentons la méthode des assertions intermittentes de Burstall à l'aide d'exemples puis nous en déduisons le principe d'induction de base formalisant de manière très concise cette méthode. Nous démontrons que ce principe de preuve est correct. La question de la complétude sémantique est plus complexe. En utilisant l'induction transfinie (plutôt que finie puisque le principe d'induction généralise la méthode de Burstall au nondéterminisme infini) nous montrons que ce principe d'induction est sémantiquement complet sous une condition suffisante (mais pas nécessaire) sur la sémantique et la propriété de fatalité. Cette condition exprime qu'un état ne peut pas être un but sur une trace et appartenir à un préfixe d'une autre trace le long duquel le but n'a pas été atteint. Cette condition est évidemment vérifiée dans le cas de Burstall qui considère des programmes déterministes mais également pour des généralisations aux programmes nondéterministes (Pnueli, Apt-Delporte,...) où sont considérées des propriétés de fatalité unaires ne dépendant pas des états initiaux.

Lorsqu'on considère des propriétés de fatalité unaires, les relations entre les valeurs initiales et finales des variables des programmes ne peuvent être exprimées qu'en considérant un programme transformé dans lequel les valeurs initiales sont affectées à des variables auxiliaires. Outre la transformation du programme sous raison fondamentale, l'utilisation de variables auxiliaires est en un sens trop souple parce que nous pouvons relier des états intermédiaires quelconques lors d'un calcul et même mémoriser toute l'histoire du calcul. Une telle liberté d'utilisation de variables auxiliaires n'est pas dans l'esprit de la méthode proposée par Burstall et des exemples donnés par Manna-Waldinger où les lemmes démontrés par induction sur les données sont toujours de la forme :

"if sometime $\phi(x_1,\dots,x_m) \wedge x_1=a_1 \wedge \dots \wedge x_m=a_m$ at $\ell$ then
sometime $\psi(a_1,\dots,a_m,x_1,\dots,x_m)$ at $\ell'$ "

(où $x_1,\dots,x_m$ sont les variables du programme et $a_1,\dots,a_m$ leurs valeurs symboliques respectives au point $\ell$ du programme). Ceci s'exprime dans notre principe d'induction de base par l'utilisation de propriétés de fatalité binaires (mieux qu'en imposant des restrictions adéquates sur l'utilisation des variables auxiliaires qui dépendraient de la syntaxe des programmes). Cependant, nous faisons la conjecture que même pour les programmes déterministes, il existe des propriétés de fatalité pour lesquelles l'utilisation d'assertions binaires n'est pas sémantiquement complète.

Cette conjecture nous conduit en 5.3.2 à généraliser la méthode des assertions intermittentes de Burstall d'une part en utilisant l'induction transfinie (pour traiter le non déterminisme non borné) et des assertions intermittentes ternaires (permettant d'exprimer des lemmes d'une forme plus générale "if sometime $\phi(\underline{a}_1,\dots,\underline{a}_m, x_1,\dots,x_m) \wedge x_1=a_1 \wedge \dots \wedge x_m=a_m$ at $\ell$ then sometime $\psi(\underline{a}_1,\dots,\underline{a}_m, a_1,\dots,a_m, x_1,\dots,x_m)$ at $\ell'$ " où $\underline{a}_1,\dots,\underline{a}_m$ (respectivement $a_1,\dots,a_m$) dénotent les valeurs des variables au point d'entrée (respectivement au point $\ell$)

du programme). Nous démontrons que ce principe d'induction généralisé est correct et sémantiquement complet.

De ce principe, nous dérivons au paragraphe 5.3.3 toute une série de principes d'induction de plus en plus abstraits et concis. Par exemple, il est intéressant de considérer un nombre transfini et non plus fini d'assertions intermittentes (que nous pouvons représenter de manière finie au moyen de variables auxiliaires). Ceci étend la méthode de Burstall de façon à incorporer la méthode de Floyd et permet d'utiliser des assertions intermittentes binaires et non plus tertiaires (en prenant formellement un lemme différent pour chaque état initial). Parmi ces principes d'induction il en est un qui formalise l'idée de Schwarz que la méthode de Burstall consiste à démontrer des théorèmes par induction mathématique à partir d'axiomes spécifiant l'effet des commandes élémentaires du programme. Les principes d'induction les plus abstraits permettent une meilleure compréhension de la méthode de Burstall (par exemple nous montrons que "l'évaluation symbolique" et l'"induction sur les données" peuvent être comprises de manière unifiée et réduite à une induction sur les calculs). Ces généralisations successives introduisent plus de souplesse dans l'écriture des preuves mais pas de puissance supplémentaire puisque nous démontrons que tous les principes de preuve considérés sont corrects et sémantiquement complets donc équivalents. Comme les principes d'induction les plus abstraits peuvent paraître très éloignés de la méthode de Burstall, nous donnons quelques exemples pour montrer le contraire.

Le principe d'induction "à la Floyd" comporte une induction le long des traces d'exécution tandis que le principe d'induction "à la Burstall" comporte la combinaison d'une induction le long de parties de traces d'exécution (en relation avec "l'évaluation symbolique" de Burstall)

et d'une récursivité (en relation avec "l'induction sur les données" de Burstall). Le principe d'induction "à la Floyd" correspond au cas particulier du principe d'induction "à la Burstall" où la récursivité n'est pas utilisée. Comme conséquence immédiate, le principe d'induction "à la Burstall" est sémantiquement complet puisque nous avons démontré précédemment que celui "à la Floyd" l'est. Cette remarque indique également pourquoi la méthode de Burstall est mieux adaptée que la méthode de Floyd pour démontrer la correction totale de programmes itératifs obtenus par élimination de la récursivité (la raison étant qu'il est possible de conserver la récursivité dans la preuve). Beaucoup plus important est le fait que le principe d'induction "à la Burstall" offre des possibilités de décomposer une preuve d'un théorème de fatalité en preuves indépendantes de lemmes plus simples, qui n'existent pas avec le principe d'induction "à la Floyd" qui nécessite une preuve globale (cf. 5.3.5).

L'argument de complétude sémantique pour le principe d'induction "à la Burstall" n'est pas pleinement satisfaisant parceque le style des preuves permises est fixe. Les utilisateurs de la méthode de Burstall ont besoin d'un résultat de complétude plus fort puisqu'ils aimeraient savoir si les lemmes qu'ils ont l'intention d'utiliser dans leurs preuves peuvent toujours être choisis librement. Une réponse affirmative est donnée au paragraphe 5.3.4 (avec la condition nécessaire et suffisante que chaque lemme doit concerner une propriété qui est fatale pour le programme mais aussi relativement aux autres lemmes qui sont utilisés dans sa preuve).

Une certaine polémique a entouré la comparaison des méthodes de Floyd et Burstall (Manna-Waldinger, Gries, ...), les uns affirmant qu'il y a des programmes qui ont une preuve "naturelle" par la méthode

de Burstall et pas avec la méthode de Floyd, les autres travaillant suffisamment la preuve avec la méthode de Floyd jusqu'à donner une impression de simplicité. (La plupart des exemples fournis (comme la version itérative de la fonction d'Ackermann) étaient obtenus par élimination de la récursivité et nous en donnons un (cf. 5.4-1) qui est simple, semble convaincant et n'est pas de cette nature). Comme contribution à ce débat nous démontrons au paragraphe 5.4 que toute preuve obtenue par une méthode peut se réécrire systématiquement en une preuve par l'autre méthode. La preuve est assez technique et longue mais intuitivement la transformation entre les deux preuves est très similaire dans un sens à l'élimination de la récursivité dans les programmes et dans l'autre sens à la présentation récursive de programmes itératifs, (nous ne prétendons donc pas que ces transformations préservent le "naturel" des preuves).

Ayant montré que la méthode de Floyd est un cas particulier de la méthode de Burstall (après les généralisations adéquates que nous avons faites), il reste néanmoins que les présentations classiques des preuves par ces deux méthodes à l'aide d'assertions invariantes d'une part et d'assertions intermittentes d'autre part sont suffisamment dissemblables pour qu'il soit difficile d'uniformiser ces deux méthods. C'est pourquoi nous introduisons, au paragraphe 5.5, la notion de charte de preuve.

L'idée de présenter graphiquement les preuves de programmes par des diagrammes acycliques fut introduite par Lamport et développée ultérieurement par Owicki-Lamport et Manna-Pnueli. Cependant, ces méthodes n'étaient pas sémantiquement complètes à cause d'un certain

nombre de restrictions, principalement l'impossibilité de faire des inductions infinies. Notre formalisation est plus générale du fait qu'elle consiste à introduire des chartes de preuve qui sont bien structurées, peuvent eventuellement présenter des cycles et peuvent être utilisés récursivement pour les preuves par induction sur les données. Après avoir défini la notion de charte de preuve (cf. 5.5.1) nous démontrons la correction et la complétude sémantique de la méthode en montrant qu'elle correspond à l'utilisation d'un principe d'induction "à la Burstall" (cf. 5.5.2). Nous donnons ensuite quelques exemples de présentation de preuves par chartes (cf. 5.5.3), pour montrer que les preuves "à la Floyd" peuvent se présenter très naturellement par des chartes et également pour montrer que les chartes de preuve sont très utiles pour démontrer des propriétés de fatalité de programmes parallèles asynchrones. Enfin, les idées developpées au paragraphe 5.2 concernant les preuves de fatalité pour les sémantiques non closes, s'appliquent directement. Nous le montrons simplement par des exemples en étendant les preuves de fatalité par chartes de preuves au cas des programmes parallèls faiblement équitables puis à celui des programmes parallèles synchrones.

## 1.3.6 CONCLUSION ET REFERENCES

Le chapitre 6 est une brève conclusion. Les références sont données à la fin de chaque chapitre.

# 2. SEMANTIQUE OPERATIONNELLE

# 2. SEMANTIQUE OPERATIONNELLE

## 2.1 DEFINITION DE LA SEMANTIQUE PAR UN ENSEMBLE DE TRACES COMPLETES

### 2.1.1 TRACES

### 2.1.2 SEMANTIQUE

## 2.2 DEFINITION DES SYSTEMES DE TRANSITION

## 2.3 SYSTEME DE TRANSITION ENGENDRE PAR UNE SEMANTIQUE

## 2.4 SEMANTIQUE ENGENDREE PAR UN SYSTEME DE TRANSITION

## 2.5 RELATIONS ENTRE SEMANTIQUES ET ENTRE SYSTEMES DE TRANSITION

### 2.5.1 INCLUSION DE SEMANTIQUES ET DE SYSTEMES DE TRANSITION

### 2.5.2 EQUIVALENCE DE SYSTEMES DE TRANSITION

### 2.5.3 CONCORDANCE ENTRE SEMANTIQUES ET ENTRE SYSTEMES DE TRANSITION A DES RELATIONS ENTRE ETATS ET/OU ACTIONS PRES

#### 2.5.3.1 Concordance à une fonction des états près

# 2. SEMANTIQUE OPERATIONNELLE

Une preuve de correction d'un programme consiste à démontrer une relation entre une sémantique et une spécification de ce programme. Pour faire l'étude des méthodes de preuve de programmes, il est donc nécessaire de disposer d'une méthode de définition de la sémantique des programmes. Dans ce chapitre nous expliquerons la méthode que nous avons choisie pour définir la sémantique des programmes.

Pour définir la sémantique d'un langage de programmation il faut simplement définir la sémantique de tous les programmes de ce langage. Pour ce faire on procède généralement par induction sur la syntaxe abstraite des programmes.

Pour définir la sémantique d'un programme, il faut définir un modèle de l'ensemble des exécutions possibles de ce programme sur ordinateur. Il s'agit d'un modèle parcequ'une simplification de la réalité est nécessaire pour éviter d'avoir à tenir compte d'une multitude de détails, en général liés à une implémentation (nombre et capacité des mémoires, conventions de codage des informations, temps de calcul des opérations, ...). Il se pose alors le problème de savoir à quel niveau d'abstraction doit être définie la sémantique. Par exemple de manière classique la sémantique dénotationnelle (Milne - Strachey [76]) a pour but d'associer à un texte de programme syntaxiquement correct une fonction mathématique qui, à une donnée, associe le résultat du programme pour cette donnée. Ce type de définition rend difficilement compte de certaines notions liées à l'exécution (par exemple que deux processus d'un programme parallèle sont en exclusion mutuelle en cours d'exécution).

Plutôt que de retenir les états d'entrée et de sortie, nous considérons un modèle opérationnel dont le niveau d'abstraction est celui qui permet d'observer la suite d'états intermédiaires par lesquels passe l'ordinateur pendant l'exécution d'un programme (ce qui permet par exemple d'étudier l'ensemble des valeurs prises par une variable au cours du calcul, de s'assurer qu'un processus d'un programme parallèle ne meurt pas de faim ou d'étudier des propriétés temps-réel en incluant une mesure de distance entre deux états successifs).

Un premier modèle de la sémantique opérationnelle des programmes consiste à utiliser les systèmes de transition (Keller [79]). Un système de transition définit l'ensemble des états du programme et une relation de transition entre un état et ses successeurs possibles. Les traces (ou exécutions) du programme sont alors les séquences d'états qui commencent par un état initial et telles que deux états successifs sont liés par la relation de transition. C'est le modèle défini dans Cousot-P [79, 81] que nous avons utilisé dans nos premiers travaux. Cette définition de la sémantique des programmes est bien adaptée à l'étude des méthodes de preuve car la relation de transition formalise bien la notion de pas du programme et les preuves procèdent généralement par induction sur le nombre de pas d'exécution du programme. Malheureusement, cette approche ne permet pas de définir de relation entre plus de deux états successifs du programme. Ceci est gênant par exemple pour définir la sémantique de programmes parallèles équitables dont l'exécution est contrôlée par un agent extérieur dont l'état ne fait pas partie de celui du programme.

Un modèle plus général est celui des traces d'exécution (Pratt [79], Lamport [80]). Avec ce modèle la sémantique d'un programme est définie comme un ensemble de traces, chaque

trace étant la séquence des états intermédiaires observés au cours d'un calcul terminé ou infini. Contrairement à ce qui se fait généralement dans la littérature (cf. par exemple Pratt [79]), nous ne considérons pas de traces incomplètes qui correspondraient à un calcul en cours. Ceci nous permet de n'avoir à faire aucune hypothèse sur l'ensemble des traces que nous considérons (alors que Pratt [79] doit supposer que tout préfixe d'une trace en cours est une trace en cours, etc.). De plus, il n'y a aucune perte de généralité en omettant les traces inachevées. Ce modèle est plus général que les systèmes de transition car il permet de définir des relations entre un nombre quelconque d'états successifs du programme. Cependant il est mal adapté pour rendre compte des méthodes de preuve de programmes qui reprennent toutes l'idée due à Floyd [67] - Naur [66] qu'il est plus simple de raisonner sur des ensembles d'états par induction sur le nombre de pas du programme que sur l'ensemble des traces d'exécution.

Dans la suite nous associerons donc à un programme une sémantique (définie comme un ensemble de traces, chaque trace étant une suite finie ou infinie d'états) et un système de transition (défini comme une relation de transition sur un ensemble d'états). Dans ce chapitre nous nous poserons la question suivante : étant donnée une propriété P relative à une sémantique S caractériser les propriétés P' et les sémantiques S' en fonction de P et S telles que P est vrai de S si, seulement si ou si et seulement si P' est vrai de S'. Ceci nous amènera en particulier à la question suivante : étant donnée une sémantique est-elle (ou jusqu'à quel point est-elle) engendrée par un système de transition ? Les résultats obtenus nous permettront plus tard de répondre à la question plus générale : étant donnée une preuve d'une propriété relative à une sémantique est-il possible (ou jusqu'à quel point est-il possible) de la remplacer par une preuve d'une autre propriété relative à une autre sémantique ou à système de transition.

## 2.1 DEFINITION DE LA SEMANTIQUE PAR UN ENSEMBLE DE TRACES COMPLETES

Dans ce paragraphe nous définissons formellement la notion de trace. Une trace représente la séquence finie (ou infinie) des états intermédiaires au cours d'une exécution achevée (respectivement, ayant bouclé) du programme. Chaque état se décompose généralement en un état contrôle (spécifiant le ou les points du programme où se trouve l'exécution) et un état mémoire (indiquant la valeur actuelle des identificateurs). Deux états successifs dans une trace sont séparés par une action qui est généralement le nom de l'opération atomique qui a permis de passer d'un état à son successeur.

Nous définissons ensuite une sémantique comme étant un ensemble de traces d'exécution.

### 2.1.1 TRACES

Une trace d'exécution p est une séquence non vide finie ou infinie d'états séparés par des actions.

Intuitivement, cette trace est un modèle d'une exécution du programme qui commence dans l'état $s_0$, exécute une action nommée $a_0$ pour atteindre l'état $s_1$, puis l'action $a_1$ pour atteindre l'état $s_2$, etc. Si l'exécution se termine alors la trace est finie sinon elle est infinie. Les traces sont complètes en ce sens qu'elles représentent un calcul achevé ou qui ne se termine pas (et jamais un calcul "en cours"). Nous n'utiliserons pas la notion de trace vide qui ne correspond à aucune réalité physique (puisqu'une exécution doit toujours commencer par un état initial).

Exemple

L'exécution d'un programme qui exécute deux fois l'action a puis l'action b et se termine, peut se décrire par la trace finie :

$$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{b} 1$$

L'exécution d'un programme qui boucle en exécutant l'action a sans arrêt peut se décrire par la trace infinie :

$$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \ldots 0 \xrightarrow{a} 0 \ldots$$

□

Une trace $p$ sur un ensemble $S$ d'états et un ensemble $A$ d'actions est un triplet $\langle m, s, a \rangle$. La longueur $m$ de la trace est un entier non nul ($m = \{0, \ldots, m-1\} \in (\omega \setminus 0)$) si la trace est finie. C'est $\omega = \{0, 1, \ldots\}$ si la trace est infinie. Par conséquent $m \in ((\omega+1) \setminus 0)$. $s \in (m \to S)$ est la séquence des états (le $i^{ème}$ état $s(i)$ en comptant à partir de zéro étant noté $s_i$).
La séquence des actions est $a \in ((m-1) \to A)$, (la $i^{ème}$ action $a(i)$ en comptant à partir de zéro étant notée $a_i$).

Définition 2.1.1:1

L'ensemble des traces sur un ensemble $S$ d'états et $A$ d'actions est défini par :

- $\Sigma^m\langle S, A \rangle = \{\langle m, s, a \rangle : s \in (m \to S) \wedge a \in ((m-1) \to A)\}$
  (traces de longueur $m \in ((\omega+1) \setminus 0)$)

- $\Sigma^{<\ell}\langle S, A \rangle = \bigcup_{m \in (\ell \setminus 0)} \Sigma^m \langle S, A \rangle$
  (traces de longueur strictement inférieure à $\ell \in (\omega+2)$)

- $\Sigma^{\leq \ell}\langle S, A \rangle = \Sigma^{<\ell}\langle S, A \rangle \cup \Sigma^{\ell}\langle S, A \rangle$
  (traces de longueur inférieure ou égale à $\ell \in (\omega+1)$)

Nous appelerons $\Sigma^{<\omega}\langle S,A\rangle$ l'ensemble des traces finies, $\Sigma^{\omega}\langle S,A\rangle$ l'ensemble des traces infinies et $\Sigma^{\leq\omega}\langle S,A\rangle$ l'ensemble des traces sur S et A. L'ensemble des traces étant d'usage fréquent, nous poserons $\Sigma\langle S,A\rangle = \Sigma^{\leq\omega}\langle S,A\rangle$ et nous omettons le couple $\langle S,A\rangle$ si nous pouvons le déterminer sans ambiguité d'après le contexte.

La longueur d'une trace $p=\langle m,s,a\rangle$ compteé en nombre d'états est $|p|=m$. La longueur de $p$ compteé en nombre d'actions est $|\!\!\downarrow\!\!p|=m-1$. Le $i^{\text{ème}}$ état de $p$ (compté à partir de zéro) est $p_i=s_i$ pour $i\in|p|$. La $i^{\text{ème}}$ action de $p$ (compteé à partir de zéro) est $p_{\downarrow i}=a_i$ pour $i\in|\!\!\downarrow\!\!p|$.

Le préfixe $p^{<m}$, $m\in((\omega+2)\setminus 0)$ (respectivement $p^{\leq m}$, $m\in(\omega+1)$) d'une trace $p=\langle m,s,a\rangle$ est $p$ si $m\geq m$ (respectivement $m+1\geq m$) sinon $\langle m, s^{<m}, a^{<m-1}\rangle$ (respectivement $\langle m+1, s^{\leq m}, a^{<m}\rangle$), où $f^{<m}$ (respectivement $f^{\leq m}$) est le préfixe (non vide) $\langle f_0,\ldots,f_{m-1}\rangle$ (respectivement $\langle f_0,\ldots,f_m\rangle$) d'une séquence $\langle f_0,\ldots,f_{m-1},f_m,\ldots\rangle$.

Le suffixe $p^{>m}$, $m+1<m$ (respectivement $p^{\geq m}$, $m<m$) d'une trace $p=\langle m,s,a\rangle$ est $\langle m-(m+1), s^{>m}, a^{>m}\rangle$ (respectivement $\langle m-m, s^{\geq m}, a^{\geq m}\rangle$) où $f^{>m}$ (respectivement $f^{\geq m}$) est le suffixe (non vide) $\langle f_{m+1},\ldots\rangle$ (respectivement $\langle f_m,\ldots\rangle$) d'une séquence $\langle f_0,\ldots,f_m,f_{m+1},\ldots\rangle$.

La tranche $p^{<m,m'>}$ (respectivement $p^{\leq m,m'>}$, $p^{<m,m'\geq}$, $p^{\leq m,m'\geq}$) est $(p^{<m'})^{>m}$ (respectivement $(p^{\leq m'})^{\geq m}$, $(p^{<m'})^{\geq m}$, $(p^{\leq m'})^{>m}$).

La concaténation de $p\in\Sigma\langle S,A\rangle$ et $q\in\Sigma\langle S,A\rangle$ est $p\hat{}q=p$ si $|p|=\omega$ sinon $|p|<\omega$ et si $p_{|\!\!\downarrow\!\!p|}=q_0$ alors $p\hat{}q=r$ tel que $r^{\leq|\!\!\downarrow\!\!p|}=p$ et $r^{\geq|\!\!\downarrow\!\!p|}=q$ sinon $p_{|\!\!\downarrow\!\!p|}\neq q_0$ et $p\hat{}q$ est indéfinie.

La concaténation de $p\in\Sigma\langle S,A\rangle$ et $q\in\Sigma\langle S,A\rangle$ via une action $a\in A$ est $p\xrightarrow{a}q=p$ si $|p|=\omega$ sinon $p\xrightarrow{a}q=r\in\Sigma\langle S,A\rangle$ telle que $r^{\leq|p|}=p$, $r^{\geq|p|}=q$ et $r_{\downarrow|\!\!\downarrow\!\!p|}=a$.

Par abus de notation, nous notons $s$ la trace de longueur 1 constituée du seul état $s$, c'est-à-dire $\langle 1, \{\langle 0, s\rangle\}, 0\rangle$. De ce fait une trace $p$ finie peut se noter $p_0 \xrightarrow{\sharp_0} p_1 \cdots p_{|\sharp|-1} \xrightarrow{\sharp_{|\sharp|-1}} p_{|\sharp|}$ et une trace infinie par $p_0 \xrightarrow{\sharp_0} p_1 \cdots p_i \xrightarrow{\sharp_i} p_{i+1} \cdots$ et donc par abus de notation, nous écrirons $\langle p_i \xrightarrow{\sharp_i} p_{i+1} : i \in |\sharp| \rangle$ (par analogie avec l'écriture $\langle f_i : i \in I \rangle$ de la fonction $F$ telle que $\forall i \in I.\ F(i) = f_i$).

Exemple

$$p = p_0 \xrightarrow{\sharp_0} p_1 \xrightarrow{\sharp_1} p_2$$

$$q = q_0 \xrightarrow{\sharp_0} q_1 \xrightarrow{\sharp_1} q_2 \xrightarrow{\sharp_2} q_3 \xrightarrow{\sharp_3} q_4 \cdots q_i \xrightarrow{\sharp_i} q_{i+1} \cdots$$

$|p| = 3$, $|\sharp| = 2$, $|q| = |\sharp| = \omega$

$p^{<0}$ est indéfini, $p^{<1} = p_0$, $p^{<2} = p_0 \xrightarrow{\sharp_0} p_1$, $p^{<m} = p$ pour $m \geq 3$

$q^{<0}$ est indéfini, $q^{<1} = q_0$, $q^{<m} = q_0 \xrightarrow{\sharp_0} q_1 \cdots \xrightarrow{\sharp_{m-2}} q_{m-1}$ pour $m \geq 2$, $q^{<\omega} = q$

$p^{\geq 0} = p$, $p^{\geq 1} = p_1 \xrightarrow{\sharp_1} p_2$, $p^{\geq 2} = p_2$, $p^{\geq m}$ est indéfini pour $m \geq 3$

$q^{\geq m} = q_m \xrightarrow{\sharp_m} q_{m+1} \cdots q_i \xrightarrow{\sharp_i} q_{i+1} \cdots$ pour $m \geq 0$, $q^{\geq \omega}$ est indéfini

$q^{\langle 2,4\rangle} = q_2 \xrightarrow{\sharp_2} q_3$

$q \xrightarrow{a} p = q$

$p \xrightarrow{a} q = p_0 \xrightarrow{\sharp_0} p_1 \xrightarrow{\sharp_1} p_2 \xrightarrow{a} q_0 \xrightarrow{\sharp_0} q_1 \cdots q_i \xrightarrow{\sharp_i} q_{i+1} \cdots$

□

## 2.1.2 SEMANTIQUE

La sémantique d'un langage associe à tout programme un ensemble de traces car l'exécution d'un programme (par exemple parallèle) est en général non déterministe. Un ensemble de traces peut également être nécessaire pour décrire les exécutions d'un programme déterministe si par exemple les états initiaux correspondant à des données différentes sont différents.

La sémantique d'un langage $\mathcal{LP}$ associe à tout programme $Pr$ de $\mathcal{LP}$ un ensemble non vide $S[\![Pr]\!]$ d'états, un ensemble non vide $A[\![Pr]\!]$ d'actions et un ensemble $\Sigma[\![Pr]\!] \subseteq \Sigma\langle S[\![Pr]\!], A[\![Pr]\!]\rangle$ de traces.

Exemple 2.1.2-1

La sémantique d'un programme qui exécute de zéro à deux fois l'action a puis l'action b peut se décrire par la sémantique $\langle S, A, \Sigma\rangle$ où $S = \{0, 1\}$, $A = \{a, b\}$ et $\Sigma = \{ 0 \xrightarrow{b} 1,\ 0 \xrightarrow{a} 0 \xrightarrow{b} 1,\ 0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{b} 1 \}$.

□

Exemple 2.1.2-2

La sémantique d'un programme qui exécute un nombre quelconque mais fini de fois l'action a puis l'action b peut se décrire par la sémantique $\langle S, A, \Sigma\rangle$ où $S = \{0, 1\}$, $A = \{a, b\}$ et

$$\Sigma = \{\ 0 \xrightarrow{b} 1$$
$$0 \xrightarrow{a} 0 \xrightarrow{b} 1$$
$$\dots$$
$$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \dots 0 \xrightarrow{a} 0 \xrightarrow{b} 1\ \}$$

□

Exemple 2.1.2-3

La sémantique d'un programme qui exécute un nombre fini de fois l'action a puis l'action b ou exécute un nombre infini de fois l'action a peut se décrire par la sémantique $\langle S, A, \Sigma\rangle$ où $S = \{0, 1\}$, $A = \{a, b\}$ et

$$\Sigma = \{\ \dots$$
$$0 \xrightarrow{a} 0 \dots 0 \xrightarrow{a} 0 \xrightarrow{b} 1$$
$$\dots$$
$$0 \xrightarrow{a} 0 \dots 0 \xrightarrow{a} 0 \dots\ \}$$

□

Définition 2.1.2:1 Sémantiques

L'ensemble des sémantiques sur des ensembles $\mathcal{S}$ (d'états) et $\mathcal{A}$ (d'actions) est $\mathrm{Sem}\langle \mathcal{S}, \mathcal{A} \rangle = \{ \langle S, A, \Sigma \rangle : S \subseteq \mathcal{S} \wedge A \subseteq \mathcal{A} \wedge \Sigma \subseteq \Sigma\langle S, A \rangle \}$

## 2.2 DEFINITION DES SYSTEMES DE TRANSITION

Etant donnés des ensembles S d'états et A d'actions, un système de transition formalise la notion d'états initiaux d'un programme (comme un sous-ensemble de S caractérisé par une fonction $\varepsilon$ à valeurs de vérité (tt vrai, ff faux)) et de pas du programme correspondant à l'exécution d'une action $a \in A$ (comme une relation de transition $t_a$ entre un état et ses successeurs possibles).

Définition 2.2:1 Systèmes de transition

L'ensemble des systèmes de transition sur des ensembles $\mathcal{S}$ (d'états) et $\mathcal{A}$ (d'actions) est $Tran\langle \mathcal{S}, \mathcal{A} \rangle = \{ \langle S, A, t, \varepsilon \rangle : S \in \mathcal{S} \wedge A \in \mathcal{A} \wedge t \in (A \to (S \times S \to \{tt, ff\})) \wedge \varepsilon \in (S \to \{tt, ff\}) \}$.

Exemple 2.2-1

Un programme qui exécute l'action b ou bien exécute un nombre fini de fois l'action a puis l'action b ou bien un nombre infini de fois l'action a, peut-être informellement représenté par l'automate fini suivant:

que nous formalisons par le système de transition $\langle S, A, t, \varepsilon \rangle$ où $S = \{0, 1\}$, $A = \{a, b\}$, $t_a(s, s') = [s = s' = 0]$, $t_b(s, s') = [s = 0 \wedge s' = 1]$ et $\varepsilon(s) = [s = 0]$.

□

## 2.3 SYSTEME DE TRANSITION ENGENDRE PAR UNE SEMANTIQUE

Pour formaliser les notions d'"état initial" et de "pas de calcul" pour une sémantique $\langle S, A, \Sigma \rangle$, nous définissons le système de transition $\langle S, A, t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ qu'elle engendre. (Pour alléger les notations, nous laisserons implicites certains paramètres quand ils peuvent être aisément déterminés d'après le contexte. Par exemple nous écrirons $\langle S,A,t,\varepsilon\rangle$ au lieu de $\langle S,A,t\langle S,A,\Sigma\rangle,\varepsilon\langle S,A,\Sigma\rangle\rangle$).

Définition 2.3:1 Système de transition engendré par une sémantique

Les états initiaux engendrés par une sémantique $\langle S,A,\Sigma\rangle$ sont caractérisés par

$$\varepsilon \in (S \to \{tt, ff\})$$

$$\varepsilon(s) = [\exists p \in \Sigma .\ p_0 = s]$$

La relation de transition engendrée par une sémantique $\langle S,A,\Sigma\rangle$ est définie par :

$$t \in (A \to (S \times S \to \{tt, ff\}))$$

$$t_a(s,s') = [\exists p \in \Sigma, i \in |p| .\ (p_i = s \wedge p_i = a \wedge p_{i+1} = s')]$$

Le système de transition engendré par une sémantique $\langle S,A,\Sigma\rangle$ est $\langle S, A, t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ où $t\langle S,A,\Sigma\rangle$ et $\varepsilon\langle S,A,\Sigma\rangle$ sont respectivement la relation de transition et les états initiaux engendrés par $\langle S,A,\Sigma\rangle$.

Exemple

Les sémantiques données en exemples 2.1.2-1, 2.1.2-2 et 2.1.2-3 engendrent exactement le même système de transition 2.2-1.

□

## 2.4 SEMANTIQUE ENGENDREE PAR UN SYSTEME DE TRANSITION

La description des comportements possibles d'un programme par un système de transition est souvent plus simple à utiliser voire à comprendre que la description par un ensemble de traces. Pour répondre à la question de savoir dans quelles conditions la donnée d'une sémantique $\langle S, A, \Sigma \rangle$ d'un programme est équivalente à la donnée du système de transition $\langle S, A, t, \varepsilon \rangle$ engendré par cette sémantique, nous définissons l'ensemble des traces complètes engendrées par un système de transition quelconque.

Définition 2.4:1

La sémantique engendrée par un système de transition $\langle S, A, t, \varepsilon \rangle$ $\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle$ avec :

- $\Sigma^m\langle S, A, t, \varepsilon\rangle = \{p \in \Sigma^m\langle S, A\rangle : \varepsilon(p_0) \wedge \forall i \in |p|-1 . t_{p_i}(p_i, p_{i+1}) \wedge \forall a \in A, s \in S . \neg t_a(p_{m-1}, s)\}$
  (traces complètes finies de longueur $m \in (\omega \cup 0)$)

- $\Sigma^\omega\langle S, A, t, \varepsilon\rangle \quad \{p \in \Sigma^\omega\langle S, A\rangle : \varepsilon(p_0) \wedge \forall i \in \omega . t_{p_i}(p_i, p_{i+1})\}$
  (traces infinies)

$\Sigma\langle S, A, t, \varepsilon\rangle \quad \bigcup_{m \in ((\omega+1) \cup 0)} \Sigma^m\langle S, A, t, \varepsilon\rangle$
(traces complètes)

Exemple 2.4-1

La sémantique engendrée par le système de transition donné en exemple 2.2-1 est définie en 2.1.2-3.

□

Dans la suite nous utiliserons les notations suivantes :
$\Sigma^{<\ell}\langle S, A, t, \varepsilon\rangle = \bigcup_{m \in (\ell \cup 0)} \Sigma^m\langle S, A, t, \varepsilon\rangle$ (traces de longueur strictement inférieure à $\ell \in (\omega+2)$),

(en particulier $\Sigma^{<\omega}\langle S, A, t, \varepsilon\rangle$ est l'ensemble des traces finies engendrées par le système de transition $\langle S, A, t, \varepsilon\rangle$).

$\Sigma^{\leq \ell}\langle S, A, t, \varepsilon\rangle = (\Sigma^{<\ell}\langle S, A, t, \varepsilon\rangle \cup \Sigma^{\ell}\langle S, A, t, \varepsilon\rangle)$, (traces de longueur inférieure ou égale à $\ell \in (\omega+1)$).

## 2.5 RELATIONS ENTRE SEMANTIQUES ET ENTRE SYSTEMES DE TRANSITION

Notons que le système de transition $\langle S,A,t\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle,\varepsilon\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle\rangle$ engendré par la sémantique $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$ engendrée par le système de transition $\langle S,A,t,\varepsilon\rangle$ est $\langle S,A,t,\varepsilon\rangle$ lui-même. Par contre les exemples 2.3-1 et 2.4-1 montrent que la sémantique $\langle S,A,\Sigma\langle S,A,t\langle S,A\Sigma\rangle,\varepsilon\langle S,A,\Sigma\rangle\rangle\rangle$ engendrée par le système de transition $\langle S,A,t\langle S,A,\Sigma\rangle,\varepsilon\langle S,A,\Sigma\rangle\rangle$ engendré par la sémantique $\langle S,A,\Sigma\rangle$ n'est en général pas cette sémantique $\langle S,A,\Sigma\rangle$ elle-même. Ceci nous amène donc à étudier dans ce paragraphe les relations qui peuvent exister entre sémantiques. Ensuite, au paragraphe 2.6, il nous sera possible d'énoncer une condition nécessaire et suffisante pour qu'une sémantique soit close (c'est-à-dire égale à la sémantique engendrée par le système de transition qu'elle engendre).

Nous incluons également dans ce paragraphe 2.5, la définition de relations entre sémantiques qu'il est nécessaire d'étudier pour justifier certaines méthodes de preuve de propriétés de programmes. En effet, pour certaines classes de propriétés, les démonstrations de correction se font plus facilement en faisant abstraction de certains aspects de la sémantique des programmes. On raisonne donc non pas sur la sémantique exacte du programme mais sur une sémantique approchée qui lui correspond selon une relation qui conserve la propriété à démontrer. Par exemple les preuves de propriétés d'invariance (correction partielle, exclusion mutuelle, ...) de programmes parallèles avec exécution équitable des processus concurrents se font beaucoup plus aisément en raisonnant sur le programme parallèle non équitable correspondant.

Pour formaliser cette méthode de preuve considérons (pour l'instant en simplifiant) qu'une propriété d'un programme est une relation entre une spécification $Sp \in Spec$ et la sémantique $\langle S, A, \Sigma \rangle \in Sem\langle \mathcal{S}, \mathcal{A} \rangle$ du programme. Nous avons $P \in ((Spec \times Sem\langle \mathcal{S}, \mathcal{A} \rangle) \to \{tt, ff\})$.

La preuve se fait en raisonnant sur une propriété' $P' \in ((Spec \times Sem\langle \mathcal{S}, \mathcal{A} \rangle) \to \{tt, ff\})$ reliant une spécification sp' et une sémantique approchée $\langle S', A', \Sigma' \rangle$ du programme.

Pour justifier cette méthode de preuve, il faut montrer que la sémantique approchée $\langle S', A', \Sigma' \rangle$ du programme est liée à la sémantique exacte du programme $\langle S, A, \Sigma \rangle$ par une relation entre sémantiques $R \in ((Sem\langle \mathcal{S}, \mathcal{A} \rangle \times Sem\langle \mathcal{S}, \mathcal{A} \rangle) \to \{tt, ff\})$ qui conserve la propriété originale P à démontrer:

$$[P'(Sp', \langle S', A', \Sigma' \rangle) \wedge R(\langle S', A', \Sigma' \rangle, \langle S, A, \Sigma \rangle)] \Rightarrow P(Sp, \langle S, A, \Sigma \rangle)$$

(En général, la preuve est faite pour toutes spécifications sp' et sp liées par une certaine relation entre spécifications).

Enfin, une relation R entre sémantiques induit une relation entre systèmes de transition, également notée R et définie par:

$$R \in ((Tran\langle \mathcal{S}, \mathcal{A} \rangle \times Tran\langle \mathcal{S}, \mathcal{A} \rangle) \to \{tt, ff\})$$

$$R(\langle S', A', t', \varepsilon' \rangle, \langle S, A, t, \varepsilon \rangle) = R(\langle S', A', \Sigma\langle S', A', t', \varepsilon' \rangle \rangle, \langle S, A, \Sigma\langle S, A, t, \varepsilon \rangle \rangle)$$

De la même façon une relation R entre systèmes de transition induit une relation entre sémantiques, également notée R et définie par:

$$R \in ((Sem\langle \mathcal{S}, \mathcal{A} \rangle \times Sem\langle \mathcal{S}, \mathcal{A} \rangle) \to \{tt, ff\})$$

$$R(\langle S', A', \Sigma' \rangle, \langle S, A, \Sigma \rangle) = R(\langle S', A', t\langle S', A', \Sigma' \rangle, \varepsilon\langle S', A', \Sigma' \rangle \rangle, \langle S, A, t\langle S, A, \Sigma \rangle, \varepsilon\langle S, A, \Sigma \rangle \rangle)$$

Il est donc intéressant d'étudier pour toute relation R entre sémantiques ou systèmes de transition la relation induite respectivement entre systèmes de transition et sémantiques. Donnons maintenant des exemples de relations, entre sémantiques et entre systèmes de transition, qui seront utilisées ultérieurement.

### 2.5.1 INCLUSION DE SEMANTIQUES ET DE SYSTEMES DE TRANSITION

On utilise la relation d'inclusion entre sémantiques quand on fait des preuves de programmes relatives à un sur- ou sous-ensemble des traces d'exécution du programme. Par exemple, on peut quelquefois démontrer une propriété d'invariance (correction partielle, ...) ou de fatalité (terminaison, ...) d'un programme parallèle en ignorant les hypothèses d'exécution équitable des processus concurrents (tout processus indéfiniment activable est fatalement activé, ...). Ceci vient du fait que ces propriétés étant vraies pour une sémantique $\langle S', A', \Sigma' \rangle$ le sont également pour toute sémantique $\langle S', A', \Sigma \rangle$ telle que $\Sigma \in \Sigma'$.

Plus généralement, la relation d'inclusion entre sémantiques est définie par $\langle S, A, \Sigma \rangle \subseteq \langle S', A', \Sigma' \rangle$ si et seulement si $[S \in S' \wedge A \in A' \wedge \Sigma \in \Sigma']$.

Muni de cet ordre partiel réflexif, $\underline{sem}\langle \mathcal{S}, \mathcal{A} \rangle$ est un treillis complet dont l'infimum est $\langle 0,0,0 \rangle$ et le supremum $\langle \mathcal{S}, \mathcal{A}, \Sigma\langle \mathcal{S}, \mathcal{A} \rangle \rangle$, (où 0 désigne l'ensemble vide, cf. annexe I-2).

Le lemme qui suit caractérise la relation d'inclusion $\subseteq$ entre systèmes de transition induite par l'inclusion de sémantiques et qui est donc définie par :

$$[\langle S, A, t, \varepsilon \rangle \subseteq \langle S', A', t', \varepsilon' \rangle] \Leftrightarrow [S \in S' \wedge A \in A' \wedge \Sigma\langle S, A, t, \varepsilon \rangle \in \Sigma\langle S', A', t', \varepsilon' \rangle]$$

Pour exprimer ce lemme nous définissons :

- $\underline{Acc}\langle S, A, t, \varepsilon \rangle(s) = [\exists p \in \Sigma\langle S, A, t, \varepsilon \rangle, i \in |p| . s = p_i]$

  (qui caractérise les états accessibles d'un système de transition)

- $\underline{Blo}\langle S, A, t, \varepsilon \rangle(s) = [\forall s' \in S, a \in A . \neg t_a(s, s')]$

  (qui caractérise les états de blocage d'un système de transition)

et rappelons que $t_a 1 Acc$ est la restriction gauche de $t_a$ à Acc (cf. annexe I-2).

Lemme 2.5.1 N 1

$$[\langle S,A,t,\varepsilon\rangle \subseteq \langle S',A',t',\varepsilon'\rangle]$$
$$\Leftrightarrow [S\subseteq S' \wedge A\subseteq A' \wedge (\forall a\in A.\ t_a 1 Acc \Rightarrow t'_a) \wedge (Blo\wedge Acc \Rightarrow Blo') \wedge (\varepsilon\Rightarrow\varepsilon')]$$

avec $Acc = \underset{\sim}{Acc}\langle S,A,t,\varepsilon\rangle$, $Blo = \underset{\sim}{Blo}\langle S,A,t,\varepsilon\rangle$ et $Blo' = \underset{\sim}{Blo}\langle S',A',t',\varepsilon'\rangle$.

Démonstration

($\Rightarrow$) Si $\langle S,A,t,\varepsilon\rangle \subseteq \langle S',A',t',\varepsilon'\rangle$ alors par définition on a $\Sigma\langle S,A,t,\varepsilon\rangle \subseteq \Sigma\langle S',A',t',\varepsilon'\rangle$. Si $t_a 1 Acc(A,A')$ est vrai, il existe une trace $p$ de $\Sigma\langle S,A,t,\varepsilon\rangle$ et $i\in|p|$ tels que $A=p_i$ et $t_a(A,A')$ et donc une trace $p'$ de $\Sigma\langle S,A,t,\varepsilon\rangle$ telle que $(i+1)\in|p'| \wedge A=p'_i \wedge A'=p'_{i+1}$. Comme $p'\in\Sigma\langle S',A',t',\varepsilon'\rangle$, la définition 2.4:1 entraîne que $t'_a(A,A')$ est vrai. De même si $(Blo\wedge Acc)(A)$ est vrai, il existe une trace finie $p$ de $\Sigma\langle S,A,t,\varepsilon\rangle$ de longueur $n$ telle que $p_{n-1}=A \wedge \forall a\in A, A'\in S.\ \neg t_a(A,A')$. Comme $p\in\Sigma\langle S',A',t',\varepsilon'\rangle$, la définition 2.4:1 entraîne que $\forall a\in A, A'\in S.\ \neg t'_a(p_{n-1},A')$ et donc $Blo'(A)$. Enfin si $\varepsilon(A)$ alors d'après 2.4:1 il existe $p\in\Sigma\langle S,A,t,\varepsilon\rangle$ telle que $p_0=A$. Comme $p\in\Sigma\langle S',A',t',\varepsilon'\rangle$ nous avons $\varepsilon'(p_0)$ et donc $\varepsilon'(A)$.

($\Leftarrow$) Il faut démontrer que $\Sigma\langle S,A,t,\varepsilon\rangle \subseteq \Sigma\langle S',A',t',\varepsilon'\rangle$. Si $p\in\Sigma\langle S,A,t,\varepsilon\rangle$ nous avons $\varepsilon(p_0)$ et donc $\varepsilon'(p_0)$. Si $i\in|p|$ alors nous avons $Acc(p_i)\wedge t_{p_i}(p_i,p_{i+1})$ et par conséquent $t'_{p_i}(p_i,p_{i+1})$. Si $p$ est une trace infinie, nous avons démontré que $p\in\Sigma\langle S',A',t',\varepsilon'\rangle$. Si $p$ est une trace finie de longueur $n$, nous avons $(Acc\wedge Blo)(p_{n-1})$, donc $Blo'(p_{n-1})$, qui d'après 2.4:1, implique $p\in\Sigma\langle S',A',t',\varepsilon'\rangle$.

□

## 2.5.2 EQUIVALENCE DE SYSTEMES DE TRANSITION

La relation d'égalité entre sémantiques induit une relation d'équivalence entre systèmes de transition définie par :

$$[\langle S,A,t,\varepsilon\rangle \equiv \langle S',A',t',\varepsilon'\rangle] \Leftrightarrow [S=S' \wedge A=A' \wedge \Sigma\langle S,A,t,\varepsilon\rangle = \Sigma\langle S',A',t',\varepsilon'\rangle]$$

(Par dérogation à la convention de 2.5, nous notons $\equiv$ la relation induite, réservant $=$ à l'égalité de systèmes de transition).

Lemme 2.5.2.1

$$\begin{aligned} & [\langle S,A,t,\varepsilon\rangle \equiv \langle S',A',t',\varepsilon'\rangle] \\ \Leftrightarrow\ & [S=S' \wedge A=A' \wedge (\forall a\in A.\ t_a 1 Acc = t'_a 1 Acc') \wedge (Blo \wedge Acc = Blo' \wedge Acc') \wedge \varepsilon=\varepsilon'] \\ \Rightarrow\ & [Acc = Acc'] \end{aligned}$$

avec $Acc = \underset{\sim}{Acc}\langle S,A,t,\varepsilon\rangle$, $Acc' = \underset{\sim}{Acc}\langle S',A',t',\varepsilon'\rangle$, $Blo = \underset{\sim}{Blo}\langle S,A,t,\varepsilon\rangle$ et $Blo' = \underset{\sim}{Blo}\langle S',A',t',\varepsilon'\rangle$

Démonstration

Si $\langle S,A,t,\varepsilon\rangle \equiv \langle S',A',t',\varepsilon'\rangle$ alors $\Sigma\langle S,A,t,\varepsilon\rangle = \Sigma\langle S',A',t',\varepsilon'\rangle$ et donc par définition de $\underset{\sim}{Acc}$, nous avons $\underset{\sim}{Acc}\langle S,A,t,\varepsilon\rangle = \underset{\sim}{Acc}\langle S',A',t',\varepsilon'\rangle$.

$\langle S,A,t,\varepsilon\rangle \equiv \langle S',A',t',\varepsilon'\rangle$ si et seulement si $\langle S,A,t,\varepsilon\rangle \subseteq \langle S',A',t',\varepsilon'\rangle$ et $\langle S',A',t',\varepsilon'\rangle \subseteq \langle S,A,t,\varepsilon\rangle$ et donc d'après le lemme 2.5.1.1, si et seulement si $S=S' \wedge A=A' \wedge \forall a\in A.(t_a 1 Acc \Rightarrow t'_a \wedge t'_a 1 Acc' \Rightarrow t_a) \wedge Blo \wedge Acc \Rightarrow Blo' \wedge Blo' \wedge Acc' \Rightarrow Blo \wedge \varepsilon=\varepsilon'$. Pour tout $a \in A=A'$, nous avons $[t_a 1 Acc \Rightarrow t'_a \wedge t'_a 1 Acc' \Rightarrow t_a] \Rightarrow [t_a 1 Acc \Rightarrow t'_a 1 Acc = t'_a 1 Acc' \Rightarrow t_a 1 Acc' = t_a 1 Acc] = [t_a 1 Acc = t'_a 1 Acc'] \Rightarrow [t_a 1 Acc \Rightarrow t'_a \wedge t'_a 1 Acc' \Rightarrow t_a]$. De plus $[Blo \wedge Acc \Rightarrow Blo' \wedge Blo' \wedge Acc' \Rightarrow Blo] = [Blo \wedge Acc = Blo' \wedge Acc']$ car $Acc = Acc'$.

□

Par abus de langage nous parlerons du système de transition qui engendre une sémantique en sous-entendant un représentant quelconque de sa classe d'équivalence.

### 2.5.3 CONCORDANCE ENTRE SEMANTIQUES ET ENTRE SYSTEMES DE TRANSITION A DES RELATIONS ENTRE ETATS ET/OU ACTIONS PRES

La relation de concordance entre sémantiques sera utilisée dans les justifications de méthodes de preuves pour éliminer des informations contenues dans les états ou actions. Il s'agit par exemple d'éliminer les variables auxiliaires introduites pour faciliter une démonstration ou d'identifier toutes les actions d'un même processus d'un programme parallèle.

Soient $rs \in (\mathcal{S} \times \mathcal{S} \to \{tt, ff\})$ une relation entre états et $ra \in (\mathcal{A} \times \mathcal{A} \to \{tt, ff\})$ une relation entre actions.

La relation de concordance entre traces aux relations rs entre états et ra entre actions près est définie par :

$$\simeq_{<rs,ra>} \in (\Sigma<\mathcal{S},\mathcal{A}> \times \Sigma<\mathcal{S},\mathcal{A}> \to \{tt, ff\})$$

$$\simeq_{<rs,ra>}(p,q) = [|p| = |q| \wedge \forall i \in |p| . rs(p_i, q_i) \wedge \forall i \in |p| . ra(\beta_{p_i}, \beta_{q_i})]$$

La relation de concordance entre sémantiques aux relations rs entre états et ra entre actions près est définie par :

$$\simeq_{<rs,ra>} \in (Sem<\mathcal{S},\mathcal{A}> \times Sem<\mathcal{S},\mathcal{A}> \to \{tt, ff\})$$

$$\simeq_{<rs,ra>}(<S,A,\Sigma>, <S',A',\Sigma'>) = [S' = rs[S] \wedge A' = ra[A] \wedge \Sigma' = \simeq_{<rs,ra>}[\Sigma]]$$

La relation induite entre systèmes de transition est la relation de concordance entre systèmes de transition aux relations rs entre états et ra entre actions près qui est définie par :

$$\simeq_{<rs,ra>} \in (Tran<\mathcal{S},\mathcal{A}> \times Tran<\mathcal{S},\mathcal{A}> \to \{tt, ff\})$$

$$\simeq_{<rs,ra>}(<S,A,t,\varepsilon>, <S',A',t',\varepsilon'>) = \simeq_{<rs,ra>}(<S,A,\Sigma<S,A,t,\varepsilon>>, <S',A',\Sigma<S',A',t',\varepsilon'>>)$$

Lemme 2.5.3~1

Si

$$(\exists s \in S.\ \varepsilon(s) \wedge rs(s,s')) = \varepsilon^{\#}(s') \qquad (a)$$

$$\wedge\ (ra(a,a') \wedge rs^{-1}(s'_0,s_0) \wedge t_a(s_0,s_1) \wedge rs(s_1,s'_1)) \Rightarrow t^{\#}_{a'}(s'_0,s'_1) \qquad (b)$$

$$\wedge\ (rs(s_0,s'_0) \wedge t^{\#}_{a'}(s'_0,s'_1)) \Rightarrow (\exists s_1 \in S, a \in A.\ rs(s_1,s'_1) \wedge ra(a,a') \wedge t_a(s_0,s_1)) \qquad (c)$$

$$\wedge\ (rs(s_0,s'_0) \wedge t_a(s_0,s_1)) \Rightarrow (\exists s'_1 \in S', a' \in A'.\ t^{\#}_{a'}(s'_0,s'_1)) \qquad (d)$$

alors

$$\Sigma\langle rs[S], ra[S], t^{\#}, \varepsilon^{\#}\rangle = \{q \in \Sigma\langle rs[S], ra[A]\rangle.\ \exists p \in \Sigma\langle S,A,t,\varepsilon\rangle.\ \simeq\langle rs,ra\rangle(p,q)\}$$

et donc

$$\simeq\langle rs,ra\rangle(\langle S,A,t,\varepsilon\rangle, \langle S',A',t',\varepsilon'\rangle) \iff (\langle S',A',t',\varepsilon'\rangle \equiv \langle rs[S], ra[A], t^{\#}, \varepsilon^{\#}\rangle)$$

Démonstration

Commençons par montrer que $\Sigma\langle rs[S], ra[A], t^{\#}, \varepsilon^{\#}\rangle = \{q \in \Sigma\langle rs[S], ra[A]\rangle.\ \exists p \in \Sigma\langle S,A,t,\varepsilon\rangle.\ \simeq\langle rs,ra\rangle(p,q)\}$.

Montrons que si $\exists p \in \Sigma\langle S,A,t,\varepsilon\rangle.\ \simeq\langle rs,ra\rangle(p,q)$ alors $q \in \Sigma\langle rs[S], ra[A], t^{\#}, \varepsilon^{\#}\rangle$. D'après la définition de $\simeq\langle rs,ra\rangle(p,q)$, 2.4:1, (a) et (b), nous avons $|p| = |q|$, $\varepsilon(p_0) \wedge rs(p_0,q_0) \Rightarrow \varepsilon^{\#}(q_0)$, $\forall i \in |p|.\ (ra(p_i,q_i) \wedge rs(p_i,q_i) \wedge t_{p_i}(p_i,p_{i+1}) \wedge rs(p_{i+1},q_{i+1})) \Rightarrow t^{\#}_{q_i}(q_i,q_{i+1})$ et donc $q \in \Sigma\langle rs[S], ra[A], t^{\#}, \varepsilon^{\#}\rangle$ si $p$ et donc $q$ est une trace infinie. Si $p$ est une trace finie de longueur $n$, nous avons $\forall a' \in A', s \in S.\ \neg t^{\#}_{a'}(q_{n-1}, s')$ car sinon $rs(p_{n-1}, q_{n-1})$ et (c) impliqueraient $\forall s \in S, a \in A.\ t_a(p_{n-1}, s)$ en contradiction avec l'hypothèse que $p$ est de longueur $n$.

Si $q \in \Sigma\langle rs[S], ra[A], t^{\#}, \varepsilon^{\#}\rangle$ alors $\exists p \in \Sigma\langle S,A,t,\varepsilon\rangle.\ \simeq\langle rs,ra\rangle(p,q)$. Nous construisons $p$ tel que $|p| = |q|$ comme suit : d'après (a), $\varepsilon^{\#}(q_0) \Rightarrow (\exists p_0 \in S.\ \varepsilon(p_0) \wedge rs(p_0,q_0))$. Disposant de $p_i$ tel que $rs(p_i,q_i)$ et $i \in |p|$, nous avons $t^{\#}_{q_i}(q_i,q_{i+1})$ et d'après (c), il existe $p_{i+1}$ et $p_i$ tels que $rs(p_{i+1},q_{i+1})$, $ra(p_i,q_i)$ et $t_{p_i}(p_i,p_{i+1})$. Si $q$ et donc $p$ est infinie, ceci montre que $p \in \Sigma\langle S,A,t,\varepsilon\rangle$. Si $q$ est finie de longueur $n$, il reste à montrer que $\forall a \in A, s \in S.\ \neg t_a(p_{n-1}, s)$. Dans le cas contraire $rs(p_{n-1}, q_{n-1})$ et (d) impliqueraient l'existence de $s' \in S', a' \in A'$ tels que $t^{\#}_{a'}(q_{n-1}, s')$, en contradiction avec la définition 2.4:1 d'une trace finie de longueur $n$.

Comme conséquence immédiate, nous obtenons $\simeq\langle rs, ra\rangle(\langle s,A,t,\varepsilon\rangle,\langle s',A',t',\varepsilon'\rangle) \Leftrightarrow (\langle s',A',t',\varepsilon'\rangle \equiv \langle rs[s], ra[A], t^{\#}, \varepsilon^{\#}\rangle)$ en observant que $s' = rs[s]$ et $A' = ra[A]$.

□

Nous utiliserons principalement trois cas particuliers :

### 2.5.3.1 Concordance à une fonction des états près

Il s'agit du cas où $rs$ est une fonction $fs \in (S \to S')$ et $ra$ la relation d'identité $1_{\mathcal{M}}$.

La sémantique concordante à $\langle S,A,\Sigma\rangle$ à la fonction $fs \in (S \to S')$ des états près est donc $\simeq\langle fs\rangle(\langle S,A,\Sigma\rangle) = \langle fs[S], A, \{\langle m, fs(A), a\rangle : \langle m,A,a\rangle \in \Sigma\}\rangle$ et $fs$ est étendue à $(\Sigma\langle S,A\rangle \to \Sigma\langle S',A'\rangle)$ par $\forall i \in |A|.\ fs(A)_i = fs(A_i)$.

---

### 2.5.3.2 Concordance à l'annulation des états près

– Quand les preuves de correction d'un programme portent uniquement sur les actions des traces, il est parfois possible d'éliminer les états en les identifiant à un état unique noté par convention $\Lambda_{\mathcal{M}}$.

Nous définissons la sémantique concordante à $\langle S,A,\Sigma\rangle$ à l'annulation des états près comme la sémantique concordante à la fonction $fs \in (S \to \{\Lambda_{\mathcal{M}}\})$ près définie par $\forall A \in S.\ fs(A) = \Lambda_{\mathcal{M}}$.

– Si une sémantique $\langle S,A,\Sigma\rangle$ ne comporte qu'un seul état, nous pouvons identifier les traces $\langle m,A,a\rangle$ à des séquences d'actions $a$ et l'ensemble $\Sigma$ des traces à l'ensemble des séquences d'actions $\{a : \exists m,A.\ \langle m,A,a\rangle \in \Sigma\}$ quand elles ne sont pas vides sinon $\{a_{\mathcal{M}}\}$ où $a_{\mathcal{M}}$ désigne par convention une action unique. Dans ce cas, cette sémantique peut donc être directement définie par un couple $\langle A,\Sigma\rangle$ où $A$ est un ensemble d'actions et $\Sigma \subseteq \{A^{*\omega} \cup \{a_{\mathcal{M}}\}\}$.

- Remarquons que nous pouvons toujours, sans perte d'informations, ramener une sémantique $\langle S, A, \Sigma \rangle$ à une sémantique $\langle S', A', \Sigma' \rangle$ ne comportant qu'un seul état en choisissant $S' = \{\underline{\Delta}\}$, $A' = S \times (A \cup \{\underline{a}\})$ où $\underline{a} \notin A$ (c'est-à-dire que les actions incluent l'état dans lequel elles sont effectuées et que nous rajoutons une action unique après le dernier état d'une trace finie) et $\Sigma' = \{\underline{\Delta} \xrightarrow{\langle p_0, \sharp_0 \rangle} \underline{\Delta} \cdots \underline{\Delta} \xrightarrow{\langle p_{|\sharp|-1}, \sharp_{|\sharp|-1} \rangle} \underline{\Delta} \xrightarrow{\langle p_{|\sharp|}, \underline{a} \rangle} \underline{\Delta} : p \in \Sigma \wedge |p| < \omega\}$ $\cup \{\langle \underline{\Delta} \xrightarrow{\langle p_i, \sharp_i \rangle} \underline{\Delta} : i \in \omega \rangle : p \in \Sigma \wedge |p| = \omega\}$. Cet argument montre que nous pouvons toujours raisonner sur les actions, ce que nous faisons rarement.

### 2.5.3.3 Concordance à l'annulation des actions près

- Quand les preuves de correction d'un programme (comme la correction partielle) portent uniquement sur les états, il est parfois possible d'éliminer les actions. Il se peut également que les actions n'aient pas besoin d'être notées dans les traces parceque les états comportent des informations de contrôle suffisamment riches pourque la transition entre deux états ne puisse correspondre qu'a une seule action qu'il est possible de déterminer sans ambiguité à partir de ces informations de contrôle. Eliminer les actions revient à les identifier toutes à une action unique $\underline{a}$ en choisissant $S' = S$, $A' = \{\underline{a}\}$, $rs = \underline{1}$ et $ra(a, a') = [a' := \underline{a}]$.

- Si une sémantique $\langle S, A, \Sigma \rangle$ ne comporte qu'une seule action, nous pouvons identifier les traces $\langle m, s, a \rangle$ à des séquences non vides d'états $s$ et l'ensemble des traces $\Sigma$ à $\{s : \exists m, a . \langle m, s, a \rangle \in \Sigma\}$. Cette sémantique peut donc être définie par un couple $\langle S, \Sigma \rangle$ où $S$ est un ensemble d'états et $\Sigma \subseteq S^{\leq \omega}$.

- Remarquons que nous pouvons toujours ramener une sémantique $\langle S, A, \Sigma \rangle$ à une sémantique $\langle S', A', \Sigma' \rangle$ ne comportant qu'une seule action, en choisissant $S' = S \times (A \cup \{\underline{a}\})$ où $\underline{a} \notin A$ (c'est-à-dire que les états incluent une partie contrôle indiquant la prochaine action à effectuer ou $\underline{a}$ si

c'est la dernière), $A' = \{\underset{\sim}{a}\}$ et $\Sigma' = \{\langle p_0, \sharp_0\rangle \xrightarrow{\underset{\sim}{a}} \cdots \xrightarrow{\underset{\sim}{a}} \langle p_{|\sharp|}, \underset{\sim}{a}\rangle : p \in \Sigma \wedge |p| < \omega\} \cup \{\langle\langle p_i, \sharp_i\rangle \xrightarrow{\underset{\sim}{a}} \langle p_{i+1}, \sharp_{i+1}\rangle : i \in \omega\rangle : p \in \Sigma \wedge |p| = \omega\}$. Cet argument montre que nous pouvons toujours raisonner uniquement sur des états, ce que nous faisons souvent.

## 2.5.4 REDUCTION DE SEMANTIQUES

Souvent, une preuve de correction d'un programme est simplifiée en ne tenant pas compte de certains états intermédiaires du programme. Par exemple, la compilation de langages de haut niveau traduit l'évaluation d'une expression arithmétique en une suite d'instructions machine comportant des points de contrôle et des registres intermédiaires. Sous certaines conditions (absence d'effets de bord, ...) il peut être plus simple de ne pas tenir compte de cette décomposition des calculs dans les preuves. Ceci revient à considérer que l'évaluation de l'expression est indivisible, autrement dit que les états intermédiaires du calcul sont inobservables.

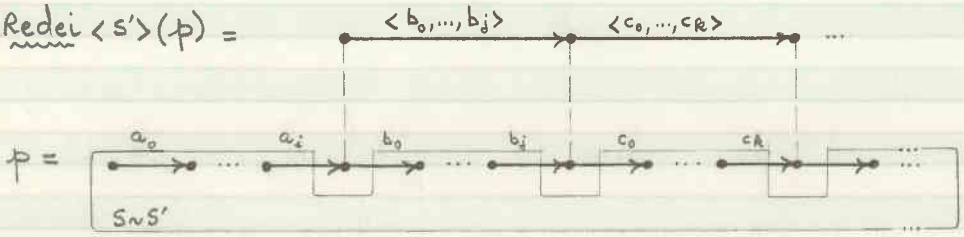
Exemple

Soit la sémantique $S = \{0,1,2,3\}$, $A = \{R := X\ ;\ ,\ R := R+1\ ;\ ,\ X := R\ ;\}$, $\Sigma = \{0 \xrightarrow{R:=X;} 1 \xrightarrow{R:=R+1;} 2 \xrightarrow{X:=R;} 3\}$. Par élimination de l'ensemble $\{1,2\}$ des états inobservables, nous obtenons la sémantique $S' = \{0,3\}$, $A' = \{R := X; R := R+1; X := R;\}$, $\Sigma = \{0 \xrightarrow{R:=X;R:=R+1;X:=R;} 3\}$.

□

### 2.5.4.1 Réduction des états inobservables

Etant donnés $S, A, S'$ $(S' \neq 0 \wedge S' \subseteq S)$ et $A' = A^{*\omega}$, nous notons $\underline{\text{Redei}}\langle S' \rangle(p)$ lorsque $(S' \cap \{p_i : i \in |p|\}) \neq 0$, la trace de $\Sigma\langle S', A' \rangle$ dérivée de $p \in \Sigma\langle S, A \rangle$ par réduction des états inobservables de $S \vee S'$. Le schéma suivant donne l'intuition de la définition.




Si $p = \langle m, A, a \rangle \in \Sigma\langle S, A \rangle$ alors $\underline{\text{Redei}}\langle S' \rangle(p)$ lorsque $(S' \cap A) \neq 0$ est la trace $p' = \langle m', A', a' \rangle \in \Sigma\langle S', A^{*\omega} \rangle$ définie comme suit :
Si $i \leq m$ alors $\underline{\text{card}}(\{A_j \in S : j \in i \wedge A_j \in S'\})$ est le nombre d'états observables dans $p$ de rang strictement inférieur à $i$ (qui peut être $\omega$ quand $i = m = \omega$). En particulier $m' = \underline{\text{card}}(\{A_j \in S : j \in m \wedge A_j \in S'\})$ est le nombre d'états observables dans $p$. Nous avons $m' \neq 0$. Si $k \in m'$ alors $r(k) = \underline{\sup}\{i : i \in m \wedge (\underline{\text{card}}(\{A_j \in S : j \in i \wedge A_j \in S'\}) = k)\}$ est le rang (compté à partir de zéro) du $k^{\text{ème}}$ état observable de $p$. Nous avons $A' \in (m' \to S')$ telle que $\forall k \in m'.\ A'(k) = A(r(k))$. Posons $a' \in ((m'-1) \to A')$ telle que $\forall k \in (m'-1).\ a'(k) = a^{\langle r(k), r(k+1) \rangle}$.

Etant donnés une sémantique $\langle S, A, \Sigma \rangle \in \underline{\text{Sem}}\langle \mathcal{S}, \mathcal{A} \rangle$ et un ensemble non vide d'états (dits observables) $S' \subseteq S$, la sémantique dérivée de $\langle S, A, \Sigma \rangle$ par réduction des états inobservables $S \vee S'$ est $\underline{\text{Redei}}\langle S' \rangle(\langle S, A, \Sigma \rangle) = \langle S', A^{*\omega}, \underline{\text{Redei}}\langle S' \rangle[\Sigma] \rangle$.

La relation de réduction dérivée entre systèmes de transition est $\underline{\text{Redei}}\langle S' \rangle(\langle S, A, t, \varepsilon \rangle, \langle S', A', t', \varepsilon' \rangle) = [\underline{\text{Redei}}\langle S' \rangle(\langle S, A, \Sigma\langle S, A, t, \varepsilon \rangle \rangle) = \langle S', A', \Sigma\langle S', A', t', \varepsilon' \rangle \rangle]$

Pour étudier les propriétés de cette relation, nous utiliserons la notation suivante :

Si $t \in (A \to (S \times S \to \{tt, ff\}))$ alors nous étendons $t$ à $((2^S)^3 \to (A^{<\omega} \to (S \times S \to \{tt, ff\})))$ par :

$$t\,|\,S_d, S_i, S_f\,|_{<>}(s, s') = [s' = s \wedge s' \in S_f]$$

$$t\,|\,S_d, S_i, S_f\,|_{<a_0, \ldots, a_m>}(s, s') = [\exists s \in (m+2 \to S).\ (s_0 = s \in S_d \wedge \forall j \in (m+1 \vee 0).\ s_j \in S_i \wedge s_{m+1} = s' \in S_f \wedge \forall j \in (m+1).\ t_{a_j}(s_j, s_{j+1}))]$$

c'est-à-dire que nous passons de $s \in S_d$ à $s' \in S_f$ par les actions $a_0, \ldots, a_m$ sur des états intermédiaires dans $S_i$.

Lemme 2.5.4 v 1

(1) Si $\langle S, A, t, \varepsilon \rangle \in \underline{\text{Tran}}\langle \mathcal{S}, \mathcal{A} \rangle$ est un système de transition, $S' \subseteq S$ un ensemble non vide d'états et $\varepsilon^\# \in (S' \to \{tt, ff\})$, $t^\# \in (A^{\#\omega} \to (S' \times S' \to \{tt, ff\}))$ sont définis par :

$$\varepsilon^\#(s') = [\exists s \in S, \alpha \in A^{<\omega}.\ \varepsilon(s) \wedge t\,|\,S \vee S', S \vee S', S'\,|_\alpha (s, s')] \qquad (a)$$

$$t^\#_\alpha (s, s') = t\,|\,S', S \vee S', S'\,|_\alpha (s, s') \qquad (b)$$

alors

$$\Sigma\langle S', A^{\#\omega}, t^\#, \varepsilon^\# \rangle \subseteq \underline{\text{Redei}}\langle S' \rangle [\Sigma \langle S, A, t, \varepsilon \rangle]$$

et donc

$$\underline{\text{Redei}}\,(\langle S, A, t, \varepsilon \rangle, \langle S', A', t', \varepsilon' \rangle) \Rightarrow [\langle S', A^{\#\omega}, t^\#, \varepsilon^\# \rangle \subseteq \langle S', A', t', \varepsilon' \rangle]$$

(2) Si de plus

$$\forall p \in \Sigma \langle S, A, t, \varepsilon \rangle, i \in |p|, \alpha \in A^{\#\omega}.\ t^\#_\alpha (p_i, s') \Rightarrow [\exists j > i.\ j \in |p| \wedge p_j \in S'] \qquad (c)$$

alors

$$\Sigma\langle S', A^{\#\omega}, t^\#, \varepsilon^\# \rangle = \underline{\text{Redei}}\langle S' \rangle [\Sigma \langle S, A, t, \varepsilon \rangle]$$

et donc

$$\underline{\text{Redei}}\langle S' \rangle (\langle S, A, t, \varepsilon \rangle, \langle S', A', t', \varepsilon' \rangle) \Leftrightarrow [\langle S', A', t', \varepsilon' \rangle \equiv \langle S', A^{\#\omega}, t^\#, \varepsilon^\# \rangle]$$

Démonstration

(1) Si $q \in \Sigma\langle S', A^{\#\omega}, t^{\#}, \varepsilon^{\#}\rangle$ alors nous pouvons construire $p \in \Sigma\langle S, A, t, \varepsilon\rangle$ tel que $q = \underset{\sim}{\text{Redei}}\langle S'\rangle(p)$. D'après 2.4:1, nous avons $\varepsilon^{\#}(q_0)$ et donc d'après (a), il existe $m \in \omega$, $p_0, \dots, p_m \in S$, $\mathfrak{p}_0, \dots, \mathfrak{p}_{m-1} \in A$ tels que $\forall i \in m.\ p_i \in (S' \cup S)$, $p_m = q_0 \in S'$, $\varepsilon(p_0)$ et $\forall i \in m.\ t_{\mathfrak{p}_i}(p_i, p_{i+1})$. Posons $r(0) = m$ de sorte que $q_0 = p_{r(0)}$ et $\forall i \in r(0).\ p_i \notin S'$. Si $p$ a été construit jusqu'au rang $r(k)$ tel que $q_k = p_{r(k)}$ et $k \in |q_{\mathfrak{p}}|$ alors d'après 2.4:1 nous avons $t^{\#}_{q_{\mathfrak{p}} k}(q_k, q_{k+1})$ et donc d'après (b), il existe $m$ avec $m+1 = |q_{\mathfrak{p}} k|$, $p_{r(k)+1}, \dots, p_{r(k)+m+1} \in S$, $\mathfrak{p}_{r(k)} = q_{\mathfrak{p}} k_0, \dots, \mathfrak{p}_{r(k)+m} = q_{\mathfrak{p}} k_m$ tels que $\forall i \in ((m+1) \setminus 0).\ p_{r(k)+i} \in (S \cup S')$, $p_{r(k)+m+1} = q_{k+1} \in S'$ et pour $i = r(k), \dots, r(k)+m$, nous avons $t_{\mathfrak{p}_i}(p_i, p_{i+1})$. Posons $r(k+1) = r(k)+m+1$ de sorte que $q_{k+1} = p_{r(k+1)}$ et $q_{\mathfrak{p}} k = \mathfrak{p}^{\leq r(k), r(k+1)\rangle}$. Si $q$ est infinie alors par cette construction $p$ l'est également et donc $p \in \Sigma\langle S, A, t, \varepsilon\rangle$. Si $q$ est finie de longueur $\ell$, poursuivons la construction comme suit : soit $r \in \langle S, A, t, \varepsilon'\rangle$ une trace finie ou infinie, avec $\varepsilon'(s) = [s = p_{r(\ell-1)}]$. Posons $p_{r(\ell-1)+i} = r_i$ pour $i \in |r|$ et $\mathfrak{p}_{r(\ell-1)+i} = r_{\mathfrak{p}} i$ pour $i \in |r_{\mathfrak{p}}|$. Observons que $\forall i \in (|r| \setminus 0).\ r_i \notin S'$ car sinon pour le plus petit $i \in (|r| \setminus 0)$ tel que $r_i \in S'$ nous aurions $t^{\#}_{\langle r_{\mathfrak{p}} 0, \dots, r_{\mathfrak{p}} i-1\rangle}(r_0, r_i)$ en contradiction avec le fait que $q_{\ell-1} = p_{r(\ell-1)} = r_0$ n'a pas de successeur puisque $q$ est de longueur $\ell$. Il reste à démontrer par récurrence sur $k$ que $r(k) = \sup\{i : i \in |p| \wedge (\underset{\sim}{\text{card}}(\{p_j \in S : j \in i \wedge p_j \in S'\}) = k)\}$. Comme $\forall j \in r(0).\ p_j \notin S'$ et $p_{r(0)} \in S'$, c'est vrai pour $k = 0$. Si c'est vrai pour $k$ alors $\underset{\sim}{\text{card}}(\{p_j \in S : j \in r(k) \wedge p_j \in S'\}) = k$, $p_{r(k)} \in S'$, $p_{r(k)+1}, \dots, p_{r(k+1)-1} \notin S'$ et $p_{r(k+1)} \in S'$ impliquent $\underset{\sim}{\text{card}}(\{p_j \in S : j \in i \wedge p_j \in S'\}) = k+1$ pour $i = r(k)+1, \dots, r(k+1)$.

Observons que $\underset{\sim}{\text{Redei}}\langle S'\rangle(\langle S, A, t, \varepsilon\rangle, \langle S', A', t', \varepsilon'\rangle)$ est équivalent à $A' = A^{\#\omega} \wedge \underset{\sim}{\text{Redei}}\langle S'\rangle[\Sigma\langle S, A, t, \varepsilon\rangle] = \Sigma\langle S', A', t', \varepsilon'\rangle$, ce qui implique $\Sigma\langle S', A^{\#\omega}, t^{\#}, \varepsilon^{\#}\rangle \subseteq \Sigma\langle S', A', t', \varepsilon'\rangle$.

(2) A la suite de (1), montrons que si $\exists p \in \Sigma\langle S, A, t, \varepsilon\rangle$ tel que $q = \underset{\sim}{\text{Redei}}\langle S'\rangle(p)$ alors $q \in \Sigma\langle S', A^{\#\omega}, t^{\#}, \varepsilon^{\#}\rangle$. Comme $(S' \cap \{p_i : i \in |p|\}) \neq 0$, il existe $j \in |p|$ tel que $p_j \in S'$. $r(0)$ est le plus petit $j \in |p|$ tel que $p_j \in S'$ et nous avons $p_{r(0)} = q_0$ et donc $\varepsilon^{\#}(q_0)$. Si $k \in |q_{\mathfrak{p}}|$, nous avons $t\,1\,S', S \cup S', S'\,1_{\mathfrak{p}^{\leq r(k), r(k+1)\rangle}}(p_{r(k)}, p_{r(k+1)})$, $q_k = p_{r(k)}$, $q_{k+1} = p_{r(k+1)}$ et $q_{\mathfrak{p}} k = \mathfrak{p}^{\leq r(k), r(k+1)\rangle}$ ce qui implique $t^{\#}_{q_{\mathfrak{p}} k}(q_k, q_{k+1})$. D'après 2.4:1, nous avons donc $q \in \Sigma\langle S', A^{\#\omega}, t^{\#}, \varepsilon^{\#}\rangle$ si $q$ est infinie. Si $q$ est finie de longueur $\ell$, il faut montrer que $q_{\ell-1}$ n'a pas de successeur pour $t^{\#}$. Par l'absurde, supposons qu'il y en ait un, soit $s$. Il existe donc $\alpha \in A^{\#\omega}$ tel que $t^{\#}_{\alpha}(q_{\ell-1}, s)$. Comme $q_{\ell-1} = p_{r(\ell-1)}$ il existe d'après (c) un $k$ tel que $r(\ell-1) < k < |p| \wedge p_k \in S'$. D'autre part,

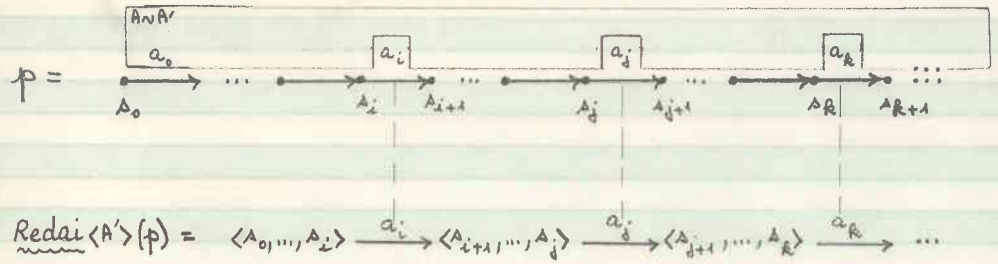
$r(\ell-1) = \sup(\{i : i \in |p| \wedge (\mathrm{card}(\{p_j \in S : j \in i \wedge p_j \in S'\}) = \ell-1)\})$ et $p_{r(\ell-1)} \in S'$ d'où $\mathrm{card}(\{p_j \in S : j \in (r(\ell-1)+1) \wedge p_j \in S'\}) = \ell$. Par définition de $q = \mathrm{Redei}\langle S' \rangle(p)$ et $|q| = \ell$ nous avons $\ell = \mathrm{card}(\{p_j \in S : j \in |p| \wedge p_j \in S'\})$. Nous en déduisons la contradiction $\forall k . (r(\ell-1) < k < |p| \Rightarrow p_k \notin S')$.

Comme $\mathrm{Redei}\langle S' \rangle(\langle S, A, t, \varepsilon\rangle, \langle S', A', t', \varepsilon'\rangle)$ est équivalent à $A' = A^{\#\omega} \wedge \mathrm{Redei}\langle S' \rangle[\Sigma\langle S, A, t, \varepsilon\rangle] = \Sigma\langle S', A', t', \varepsilon'\rangle$ soit $A' = A^{\#\omega} \wedge \Sigma\langle S', A^{\#\omega}, t^{\#}, \varepsilon^{\#}\rangle = \Sigma\langle S', A', t', \varepsilon'\rangle$ c'est-à-dire $\langle S', A', t', \varepsilon'\rangle \equiv \langle S', A^{\#\omega}, t^{\#}, \varepsilon^{\#}\rangle$.

□

### 2.5.4.2 Réduction des actions inobservables

De manière analogue, nous pouvons définir la réduction des actions inobservables d'une trace. Etant donnés $S$, $A$, $S' = S^{\#\omega}$ et $A'$ $(A' \in A \wedge A' \neq 0)$, nous notons $\mathrm{Redai}\langle A' \rangle(p)$ lorsque $(A' \cap \{\#_i : i \in |\#|\}) \neq 0$ la trace de $\Sigma\langle S', A' \rangle$ dérivée de $p \in \Sigma\langle S, A \rangle$ par réduction des actions inobservables de $A \setminus A'$. L'idée se représente informellement comme suit :




et la formalisation est tout à fait similaire à ce qui précède.

## 2.6 FERMETURES DE SEMANTIQUES

Nous définissons maintenant des relations entre sémantiques au moyen d'opérateurs de fermeture sur le treillis complet $\underline{Sem}\langle \mathcal{S}, \mathcal{A} \rangle$ muni de l'inclusion $\subseteq$. Nous étudions, quand elle est intéressante, la relation induite entre systèmes de transition.

Dans une première partie (2.6.1 à 2.6.4) nous nous intéressons à des relations entre sémantiques qui seront ultérieurement utilisées pour justifier certaines méthodes de preuve. Il s'agit de raisonner sur les préfixes des traces (2.6.1), les suffixes des traces (2.6.2), les états et actions accessibles (2.6.3) ou les traces équitables.

Dans une deuxième partie (2.6.5 à 2.6.8) nous introduisons successivement (en 2.6.5) la notion de sémantique fermée par fusion (Pratt [79]), (en 2.6.6) la notion de sémantique réduite par élimination des traces préfixes strictes, (en 2.6.7) la notion de fermeture d'une sémantique par limites (Abrahamson [80]) ce qui permet (en 2.6.8) de donner des conditions nécessaires et suffisantes pour qu'une sémantique $\langle S, A, \Sigma \rangle$ d'une part et la sémantique $\underline{R}tran(\langle S, A, \Sigma \rangle)$ engendrée par le système de transition $\langle S, A, t\langle S, A, \Sigma \rangle, \varepsilon\langle S, A, \Sigma \rangle \rangle$ engendré par $\langle S, A, \Sigma \rangle$ d'autre part soient $\subseteq$-comparables (cf. théorèmes 2.6.8∿2, 2.6.8∿3) ou égales (cf. théorème 2.6.8∿4). Ces résultats seront utilisés dans le paragraphe suivant pour définir des sémantiques à l'aide de systèmes de transition.

### 2.6.1 FERMETURE D'UNE SEMANTIQUE PAR PREFIXES

Certaines propriétés des sémantiques comme l'invariance se conservent par fermeture par préfixes.

La relation de préfixe ou facteur gauche sur $\Sigma^{\leq\omega}\langle S,A\rangle$, définie par $(p \mapsto q \Leftrightarrow \exists i \in |q|.\ p = q^{\leq i})$ est une relation d'ordre réflexive.

La fermeture par préfixes d'une sémantique $\langle S,A,\Sigma\rangle$ est la sémantique $\mathrm{Pref}(\langle S,A,\Sigma\rangle) = \langle S,A,\{p \in \Sigma^{\leq\omega}\langle S,A\rangle : \exists q \in \Sigma.\ p \mapsto q\}\rangle$

Pref est un opérateur de fermeture supérieure sur $\mathrm{sem}\langle \mathcal{S}, \mathcal{A}\rangle$ muni de l'inclusion $\subseteq$.

La relation induite sur les systèmes de transition n'est pas intéressante car une sémantique fermée par préfixes ne peut pas être engendrée par un système de transition, sauf si toutes les traces sont réduites à un seul état.

Il est quelquefois beaucoup plus facile de faire des preuves en raisonnant sur les préfixes finis ou traces incomplètes (c'est-à-dire sur des calculs "en cours") plutôt que sur des traces complètes (c'est-à-dire sur des calculs terminés ou infinis). Dans ce cas, nous pouvons raisonner sur la fermeture par préfixes finis de la sémantique:

La fermeture par préfixes finis d'une sémantique $\langle S,A,\Sigma\rangle$ est $\mathrm{Pref}^{<\omega}(\langle S,A,\Sigma\rangle) = \langle S,A,\{p \in \Sigma^{<\omega}\langle S,A\rangle : \exists q \in \Sigma.\ p \mapsto q\}\rangle$

$\mathrm{Pref}^{<\omega}$ n'est pas extensif (nous n'avons pas $\langle S,A,\Sigma\rangle \subseteq \mathrm{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)$ quand $\Sigma$ contient une trace infinie), toutefois nous avons :

Lemme 2.6.1~1

$\underset{\sim}{Pref}^{<\omega}$ est un opérateur de préfermeture supérieure sur $\underset{\sim}{Sem}\langle \mathcal{S}, \mathcal{A} \rangle$.

De nouveau la relation induite sur les systèmes de transition est sans intérêt.

Nous utiliserons la remarque triviale que l'ensemble des préfixes d'un ensemble de traces est constitué des préfixes finis et des traces infinies :

Lemme 2.6.1~2

$$\underset{\sim}{Pref}(\langle S, A, \Sigma \rangle) = \underset{\sim}{Pref}^{<\omega}(\langle S, A, \Sigma \rangle) \cup \langle S, A, \Sigma \cap \Sigma^{\omega}\langle S, A \rangle \rangle$$

## 2.6.2 FERMETURE D'UNE SEMANTIQUE OU D'UN SYSTEME DE TRANSITION PAR SUFFIXES

Certaines preuves de programmes sont relatives à un état initial qui ne correspond pas forcément au point de départ de l'exécution du programme. Nous raisonnons alors sur une fermeture de la sémantique du programme par suffixes.

La relation de suffixe ou facteur droit sur $\Sigma^{\leq\omega}\langle S,A\rangle$ est définie par $p \rightarrow\!\!\!\!\!+\; q \Leftrightarrow \exists i \in |q| \cdot p = q^{\geq i}$.

La fermeture par suffixes d'une sémantique $\langle S,A,\Sigma\rangle$ est la sémantique $\mathrm{Suff}(\langle S,A,\Sigma\rangle) = \langle S,A,\{p \in \Sigma^{\leq\omega}\langle S,A\rangle : \exists q \in \Sigma . p \rightarrow\!\!\!\!\!+\; q\}\rangle$.

Suff est un opérateur de fermeture supérieure sur $\mathrm{Sem}\langle \mathcal{S},\mathcal{A}\rangle$ muni de l'inclusion $\in$.

La relation induite sur les systèmes de transition est définie par
$\mathrm{Suff}(\langle S,A,t,\varepsilon\rangle,\langle S',A',t',\varepsilon'\rangle) = [\mathrm{Suff}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \langle S',A',\Sigma\langle S',A',t',\varepsilon'\rangle\rangle]$

Lemme 2.6.2~1

$$\mathrm{Suff}(\langle S,A,t,\varepsilon\rangle,\langle S',A',t',\varepsilon'\rangle) \Leftrightarrow [\langle S',A',t',\varepsilon'\rangle \equiv \langle S,A,t,\varepsilon^*\rangle]$$

avec $\varepsilon^*(s) = [\exists s' \in S.\ \varepsilon(s') \wedge t^*(s',s)]$

en notant $t^*$ la fermeture transitive réflexive de $t$ définie comme suit :

$$t^*(s,s') = \bigcup_{n\geq 0} t^n(s,s') \qquad t^0(s,s') = [s'=s]$$

$$t^{n+1}(s,s') = [\exists a \in A, s'' \in S.\ (t_a(s,s'') \wedge t^n(s'',s'))]$$

Démonstration

Nous avons $\{p : \exists q \in \Sigma\langle S,A,t,\varepsilon\rangle . \; p \twoheadrightarrow q\} = \Sigma\langle S,A,t,\varepsilon^*\rangle$ et donc $\mathrm{Suff}(\langle S,A,t,\varepsilon\rangle, \langle S',A',t',\varepsilon'\rangle) \Leftrightarrow [\langle S',A',\Sigma\langle S',A',t',\varepsilon'\rangle\rangle = \langle S,A,\Sigma\langle S,A,t,\varepsilon^*\rangle\rangle]$.

□

Suff est un opérateur de fermeture supérieure sur $\mathrm{Tran}\langle \mathcal{S},\mathcal{A}\rangle/_{\equiv}$ muni de l'inclusion.

## 2.6.3 FERMETURE D'UNE SEMANTIQUE OU D'UN SYSTEME DE TRANSITION PAR REDUCTION AUX ETATS ET/OU ACTIONS ACCESSIBLES

Une preuve de programme souvent se simplifie en raisonnant non pas sur des états quelconques du programme mais sur l'ensemble de ceux qui sont accessibles au cours calcul (ou sur un sur-ensemble de ceux-ci caractérisé invariant). Ceci revient à raisonner sur la réduction de la sémantique du programme aux états (et actions) accessibles.

Etant donnée une sémantique $\langle S, A, \Sigma \rangle$ la réduction aux états et actions accessibles de cette sémantique est $\underline{Redeaa}(\langle S, A, \Sigma \rangle) =$
$\langle \{s \in S : \exists p \in \Sigma, i \in |p| . (p_i = s)\}, \{a \in A : \exists p \in \Sigma, j \in |p| . (p_j = a)\}, \Sigma \rangle$.

Observons que Redeaa est un opérateur de fermeture inférieure sur $\underline{Sem}\langle \mathscr{S}, \mathscr{A} \rangle$ muni de l'inclusion $\subseteq$.

Nous définissons de même la réduction aux états accessibles Redea et la réduction aux actions accessibles Redaa.

La relation induite sur les systèmes de transition est
$\underline{Redeaa}(\langle S, A, t, \varepsilon \rangle, \langle S', A', t', \varepsilon' \rangle) = [\underline{Redeaa}(\langle S, A, \Sigma\langle S, A, t, \varepsilon \rangle \rangle) = \langle S', A', \Sigma\langle S', A', t', \varepsilon' \rangle \rangle]$

Lemme 2.6.3v1

$\underline{Redeaa}(\langle S, A, t, \varepsilon \rangle, \langle S', A', t', \varepsilon' \rangle) \Leftrightarrow$
$\langle S', A', t', \varepsilon' \rangle \equiv \langle \{s \in S : \exists s' \in S . (\varepsilon(s') \wedge t^*(s', s))\}, \{a \in A : \exists s, s', s'' \in S . (\varepsilon(s) \wedge t^*(s, s') \wedge t_a(s', s''))\}, t, \varepsilon \rangle]$

Remarquons que Redeaa est un cas particulier de Redei. En effet,
$\underline{Redeaa}(\langle S, A, t, \varepsilon \rangle, \langle S', A', t', \varepsilon' \rangle) = \underline{Redei}\langle Acc \rangle (\langle S, A, t, \varepsilon \rangle, \langle S', A', t', \varepsilon' \rangle)$ (où $Acc = \underline{Acc}\langle S, A, t, \varepsilon \rangle$).

## 2.6.4 FERMETURE D'UNE SEMANTIQUE PAR REDUCTION AUX TRACES EQUITABLES

Pour définir la sémantique de programmes parallèles avec hypothèse d'exécution équitable des processus, Lehman-Pnueli-Stavi [81], nous pouvons spécifier une sémantique non équitable puis éliminer les traces non équitables. Cette réduction aux traces équitables évite d'avoir à spécifier un contrôleur d'exécution (scheduler) particulier de manière explicite dans la sémantique, et laisse ouvertes d'autres implémentations possibles.

Nous écrivons $\underline{\text{Enabled}}(a,i,p,\Sigma)$ quand l'action $a$ est activable au point $i \in |p|$ d'une trace $p$, c'est-à-dire qu'il existe une trace $q \in \Sigma$ ayant même préfixe que $p$ jusqu'en $i$ et $a$ est activée en ce point :

$$\underline{\text{Enabled}}(a,i,p,\Sigma) = [i \in |p| \wedge \exists q \in \Sigma.(i \in |q| \wedge q^{<i} = p^{<i} \wedge q_i = a)]$$

La réduction d'une sémantique aux traces faiblement équitables pour un ensemble $\alpha$ d'actions conserve les traces finies et les traces infinies pour lesquelles aucune action de $\alpha$ n'est, au delà d'un certain point continuellement activable et jamais activée :

$$\underline{\text{Wfair}}\langle\alpha\rangle(\langle S,A,\Sigma\rangle) =$$
$$\langle S,A,\{p \in \Sigma : (|p| = \omega) \Rightarrow \neg(\exists a \in \alpha, i \in \omega.\ \forall j \geq i.(\underline{\text{Enabled}}(a,j,p,\Sigma) \wedge p_j \neq a))\}\rangle$$

$$\underline{\text{Wfair}}(\langle S,A,\Sigma\rangle) = \underline{\text{Wfair}}\langle A\rangle(\langle S,A,\Sigma\rangle)$$

Exemple

La réduction de la sémantique 2.1.2-3 aux traces faiblement équitables est la sémantique 2.1.2-2. La trace infinie $0 \xrightarrow{a} 0 \ldots 0 \xrightarrow{a} 0 \ldots$ n'est pas faiblement équitable pour $\{a,b\}$ car l'action $b$ n'est jamais activée et toujours activable pour donner une trace $0 \xrightarrow{a} 0 \ldots 0 \xrightarrow{b} 1$.

□

La réduction d'une sémantique aux traces fortement équitables pour un ensemble $\alpha$ d'actions conserve les traces finies et les traces infinies pour lesquelles aucune action de $\alpha$ n'est, au delà d'un certain point activable infiniment souvent et jamais activée :

$$\underset{\sim}{Sfair}\ \langle\alpha\rangle(\langle S,A,\Sigma\rangle) =$$

$$\langle S,A,\{p\in\Sigma : (|p|=\omega)\Rightarrow\neg(\exists a\in\alpha, i\in\omega.(\forall j\geq i.\exists k\geq j.\underset{\sim}{Enabled}(a,k,p,\Sigma))\wedge(\forall j\geq i.\ \sharp_j\neq a))\}\rangle$$

$$\underset{\sim}{Sfair}\ (\langle S,A,\Sigma\rangle) = \underset{\sim}{Sfair}\langle A\rangle(\langle S,A,\Sigma\rangle)$$

Remarquons que l'équité forte entraîne l'équité faible.

Lemme 2.6.4~1

(1) $\underset{\sim}{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\rangle) \subseteq \underset{\sim}{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\rangle) \subseteq \langle S,A,\Sigma\rangle$

(2) $\underset{\sim}{Wfair}\langle\alpha\rangle$ et $\underset{\sim}{Sfair}\langle\alpha\rangle$ sont des opérateurs de fermeture inférieure sur $\underset{\sim}{Sem}\langle\mathcal{S},\mathcal{A}\rangle$ muni de l'inclusion $\subseteq$.

La relation induite sur les systèmes de transition n'a aucun intérêt car en général $\underset{\sim}{Wfair}\langle\alpha\rangle(\Sigma\langle S,A,t,\varepsilon\rangle)$ ne peut pas être engendré par un système de transition sur $S$ et $A$.

Pour justifier les méthodes de preuve de propriétés d'invariance de programmes parallèles équitables, nous utiliserons le fait que tout préfixe fini d'une trace engendrée par un système de transition est préfixe d'une trace équitable relativement à tout ensemble fini d'actions et réciproquement :

Lemme 2.6.4~2

Si $\underset{\sim}{card}(\alpha)<\omega$ alors

- $\underset{\sim}{Pref}^{<\omega}\circ\underset{\sim}{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$
- $\underset{\sim}{Pref}^{<\omega}\circ\underset{\sim}{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$

Démonstration

Nous avons $\underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) \subseteq \underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$ $\subseteq \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$ d'après le lemme 2.6.4v1.1 et le fait que $\underset{\sim}{Pref}^{<\omega}$ est monotone pour $\subseteq$.

Pour montrer que $\underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) \subseteq \underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$ nous considérons un préfixe fini $p$ de longueur $m$ d'une trace de $\Sigma\langle S,A,t,\varepsilon\rangle$ que nous prolongeons en une trace $r$ de $\Sigma\langle S,A,t,\varepsilon\rangle$ dont nous montrons qu'elle est fortement équitable. Pour la base de la construction, choisissons $r^{<m} = p$ de sorte que $r$ soit construit jusqu'au point $m-1$. Dans la construction de $r$, nous utilisons une file d'attente $f$ qui contient toujours une fois et une seule tout élément de $\alpha$ et qui par hypothèse est donc finie. Notons $f_\ell$ la valeur de cette file au point $\ell$ de la construction. Initialement $f_{m-1}$ contient toutes les actions de $\alpha$ dans un ordre quelconque. Supposons que nous ayons construit $r$ jusqu'au point $\ell$ et défini $f_\ell$. Si $r_\ell$ n'a pas de successeur pour $t$ alors la trace $r$ est finie et donc fortement équitable, ce qui termine la démonstration. Sinon, il existe une action $b$ activable en $r_\ell$. Si aucune des actions activables en $r_\ell$ n'appartient à $f_\ell$, nous choisissons $f_{\ell+1} = f_\ell$, $\underset{\sim}{r}_\ell = b$ et $r_{\ell+1}$ un élément quelconque de $S$ tel que $t_{\underset{\sim}{r}_\ell}(r_\ell, r_{\ell+1})$ soit vrai. Sinon il existe une action de $f_\ell$ activable en $r_\ell$. Dans ce cas, nous choisissons $\underset{\sim}{r}_\ell$ comme étant l'action activable de $f_\ell$ la plus près de la tête de la file (toutes les actions devant $\underset{\sim}{r}_\ell$ dans $f_\ell$ (s'il y en a) n'étant donc pas activables en $r_\ell$). Nous choisissons $r_{\ell+1}$ quelconque tel que $t_{\underset{\sim}{r}_\ell}(r_\ell, r_{\ell+1})$ soit vrai et $f_{\ell+1}$ comme étant $f_\ell$ à la différence que $\underset{\sim}{r}_\ell$ a été déplacée en queue de la file d'attente. Pour achever cette démonstration, il suffit de montrer que si par cette construction, nous obtenions une trace $r$ infinie alors elle est fortement équitable. En effet, dans le cas contraire il existerait une action $a$ de $\alpha$ qui est activable infiniment souvent et jamais activée au delà d'un point $i$ et donc du point $j = \underset{\sim}{sup}^+\{m, i\}$ de $r$. Pour chaque entier $\ell$ pour lequel $a$ est activable, $\underset{\sim}{r}_\ell$ précède $a$ dans $f_\ell$. Or ceci n'est possible qu'un nombre fini de fois.

□

## 2.6.5 FERMETURE D'UNE SEMANTIQUE PAR FUSIONS

En général, les évolutions possibles de l'exécution à partir d'un état d'un programme ne dépendent pas de la manière dont cet état a été atteint. Cette condition s'exprime par le fait que la sémantique du programme est fermée par fusions c'est-à-dire que tout préfixe fini d'une trace (terminé par un état $s$) peut se prolonger par tout suffixe (commençant par $s$) d'une trace :

L'extension d'une sémantique $\langle S, A, \Sigma\rangle$ par fusions est $Efus(\langle S, A, \Sigma\rangle) = \langle S, A, \{p\hat{}q : p \in \Sigma^{<\omega}\langle S, A\rangle \wedge q \in \Sigma^{\leq\omega}\langle S, A\rangle \wedge \exists p', q' \in \Sigma . (p \mapsto p' \wedge q \twoheadrightarrow q')\}\rangle$

Exemple

L'extension de la sémantique $\langle \{0,1\}, \{a,b\}, \{0 \xrightarrow{b} 1, 0 \xrightarrow{a} 0 \xrightarrow{b} 1\}\rangle$ par fusions est la sémantique 2.1.2-1 c'est-à-dire $\langle \{0,1\}, \{a,b\}, \{0 \xrightarrow{b} 1, 0 \xrightarrow{a} 0 \xrightarrow{b} 1, 0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{b} 1\}\rangle$.

□

Observons que Efus est un opérateur monotone et extensif sur $\langle Sem\langle \mathcal{S}, \mathcal{A}\rangle, \subseteq\rangle$, mais n'est pas idempotent. D'où la définition suivante :

La fermeture d'une sémantique $\langle S, A, \Sigma\rangle$ par fusions est $Efus(\langle S, A, \Sigma\rangle)$ où Ffus est le plus petit opérateur de fermeture sur $Sem\langle \mathcal{S}, \mathcal{A}\rangle$ plus grand ou égal à Efus. (plus petit et plus grand étant compris par rapport à l'extension point par point de $\subseteq$).

Exemple

La fermeture par fusions de la sémantique 2.1.2-1 est la sémantique 2.1.2-2.

□

Lemme 2.6.5∿1

(1) $\underset{\sim}{Ffus} = \bigcup_{n \geq 0} \underset{\sim}{Efus}^n$

(2) $\underset{\sim}{Efus} \circ \underset{\sim}{Ffus} = \underset{\sim}{Ffus}$

Démonstration

Comme $\underset{\sim}{Sem}\langle \mathcal{D}, \mathcal{A} \rangle$ est un treillis complet pour $\subseteq$ et $\underset{\sim}{Efus}$ est monotone et extensif, le théorème 4.3, son corollaire 4.4-1 et la définition 1.13 dans Cousot - Cousot [79] impliquent que $\underset{\sim}{Ffus}(\langle S, A, \Sigma \rangle)$ est la limite de la séquence $X_0 = \langle S, A, \Sigma \rangle$, $X_\delta = \underset{\sim}{Efus}(X_{\delta-1})$ si $\delta$ est un ordinal successeur et $X_\delta = \bigcup_{\alpha < \delta} X_\alpha$ si $\alpha$ est un ordinal limite. Par définition de $\underset{\sim}{Efus}$, $X_\delta$ est de forme $\langle S, A, \Sigma_\delta \rangle$ avec $(\alpha \leq \beta) \Rightarrow (\Sigma_\alpha \subseteq \Sigma_\beta)$. Pour montrer que la limite est atteinte pour $\delta = \omega$, il suffit de montrer que $\Sigma_{\omega+1} = \{p \hat{} q : p \in \Sigma^{<\omega}\langle S, A \rangle \wedge q \in \Sigma^{\leq\omega}\langle S, A \rangle \wedge (\exists p', q' \in \bigcup_{i<\omega} \Sigma_i . p \mapsto p' \wedge q \twoheadrightarrow q')\} \subseteq \Sigma_\omega = \bigcup_{k<\omega} \Sigma_k = \bigcup_{k<\omega} \{p \hat{} q : p \in \Sigma^{<\omega}\langle S, A \rangle \wedge q \in \Sigma^{\leq\omega}\langle S, A \rangle \wedge (\exists p', q' \in \bigcup_{i<k} \Sigma_i . p \mapsto p' \wedge q \twoheadrightarrow q')\}$. Nous avons bien $(\exists p', q' \in \bigcup_{i<\omega} \Sigma_i) \Rightarrow (\exists i, j \in \omega . p' \in \Sigma_i \wedge q' \in \Sigma_j) \Rightarrow (\exists i, j \in \omega . p', q' \in \Sigma_{\sup(\{i,j\})}) \Rightarrow (\exists k \in \omega . p', q' \in \Sigma_k)$ car $\Sigma_i \subseteq \Sigma_{\sup(\{i,j\})}$ et $\Sigma_j \subseteq \Sigma_{\sup(\{i,j\})}$.

□

La relation induite sur les systèmes de transition est l'identité car :

Lemme 2.6.5∿2

$$\underset{\sim}{Ffus}(\langle S, A, \Sigma\langle S, A, t, \varepsilon \rangle\rangle) = \langle S, A, \Sigma\langle S, A, t, \varepsilon \rangle\rangle$$

Démonstration

Il suffit de remarquer que $\underset{\sim}{Efus}(\langle S, A, \Sigma\langle S, A, t, \varepsilon \rangle\rangle) = \langle S, A, \Sigma\langle S, A, t, \varepsilon \rangle\rangle$ et d'appliquer le lemme 2.6.5∿1.1.

□

Nous utiliserons la propriété suivante :

Lemme 2.6.5∿3

(1) $\underset{\sim}{E}fus \circ \underset{\sim}{W}fair\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) \quad \underset{\sim}{W}fair\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$

(2) $\underset{\sim}{F}fus \circ \underset{\sim}{W}fair\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{W}fair\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$

(3) $\underset{\sim}{E}fus \circ \underset{\sim}{S}fair\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{S}fair\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$

(4) $\underset{\sim}{F}fus \circ \underset{\sim}{S}fair\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{S}fair\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$

Démonstration

(1), (3) : Si $P = p\hat{}p'$ et $Q = q'\hat{}q$ sont des traces équitables de $\Sigma\langle S,A,t,\varepsilon\rangle$ alors $p\hat{}q$ est une trace de $\Sigma\langle S,A,t,\varepsilon\rangle$ qui est équitable car sinon une action est au delà de $s$ toujours ou infiniment souvent activable et jamais activée dans $q$ donc dans $Q$.

(2) : Par récurrence, nous avons $\underset{\sim}{E}fus^n \circ \underset{\sim}{W}fair\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{W}fair\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$ et le résultat dérive de 2.6.5∿1.1. Idem pour (4). □

### 2.6.6 REDUCTION D'UNE SEMANTIQUE PAR ELIMINATION DES TRACES PREFIXES STRICTS

En général, si l'exécution d'un programme peut se poursuivre à partir d'un certain état, alors elle doit se poursuivre. Cette condition correspond à l'hypothèse habituelle que les calculs progressent à une vitesse non nulle ou encore qu'il n'y a pas de blocages (pannes) dus à un agent extérieur. Cette condition s'exprime par le fait que la sémantique du programme est réduite par élimination des traces préfixes strictes c'est-à-dire qu'il n'existe pas de trace qui soit préfixe strict d'une autre trace.

La réduction d'une sémantique $\langle S, A, \Sigma \rangle$ par élimination des traces préfixes strictes est $\mathrm{Retps}(\langle S, A, \Sigma \rangle) = \langle S, A, \{p \in \Sigma : \forall q \in \Sigma.\ (p \mapsto q) \Rightarrow (p = q)\} \rangle$

Exemple

$\mathrm{Retps}\ (\langle \{0\}, \{a\}, \{0, 0 \xrightarrow{a} 0\} \rangle) = \langle \{0\}, \{a\}, \{0 \xrightarrow{a} 0\} \rangle$

$\mathrm{Retps}\ (\langle \{0\}, \{a\}, \{0, 0 \xrightarrow{a} 0, 0 \xrightarrow{a} 0 \xrightarrow{a} 0\} \rangle) = \langle \{0\}, \{a\}, \{0 \xrightarrow{a} 0 \xrightarrow{a} 0\} \rangle$

□

L'opérateur Retps est réductif, idempotent mais n'est pas monotone pour $\subseteq$ comme le montre le contre-exemple ci-dessus.

La relation induite sur les systèmes de transition est l'identité car :

Lemme 2.6.6 v1

$\mathrm{Retps}\ (\langle S, A, \Sigma \langle S, A, t, \varepsilon \rangle \rangle) = \langle S, A, \Sigma \langle S, A, t, \varepsilon \rangle \rangle$

Observons que si $\langle S, A, \Sigma' \rangle = \mathrm{Retps}(\langle S, A, \Sigma \rangle)$ alors $\Sigma$ et $\Sigma'$ ont mêmes traces infinies et par conséquent, de manière évidente, nous avons :

Lemme 2.6.6 ~ 2

$$\underset{\sim}{\text{Retps}} \circ \underset{\sim}{\text{Wfair}}\langle\alpha\rangle(\langle s,A,\Sigma\langle s,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{\text{Wfair}}\langle\alpha\rangle(\langle s,A,\Sigma\langle s,A,t,\varepsilon\rangle\rangle)$$

$$\underset{\sim}{\text{Retps}} \circ \underset{\sim}{\text{Sfair}}\langle\alpha\rangle(\langle s,A,\Sigma\langle s,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{\text{Sfair}}\langle\alpha\rangle(\langle s,A,\Sigma\langle s,A,t,\varepsilon\rangle\rangle)$$

## 2.6.7 FERMETURE D'UNE SEMANTIQUE PAR LIMITES

Certaines sémantiques ont la propriété que certaines traces d'exécution peuvent être suivies pendant un temps fini arbitrairement long mais pas pendant un temps infini.

Exemple

Pour tout $n \geq 0$, la sémantique 2.1.2-2 offre la possibilité d'exécuter $n$ fois l'action a (puis l'action b) mais il est impossible d'exécuter l'action a un nombre infini de fois.

□

Dans le cas contraire, la sémantique est fermée par limites c'est-à-dire que si tous les préfixes finis d'une trace infinie peuvent être suivis par une exécution alors la trace infinie est un calcul légitime :

$$\mathrm{Flim}(\langle S, A, \Sigma\rangle) = \langle S, A, \Sigma \cup \{p \in \Sigma^{\omega}\langle S,A\rangle : \forall n \in \omega.\ \exists q \in \Sigma.\ p^{\leq n} \mapsto q\}\rangle$$

Exemple

La fermeture par limites de la sémantique 2.1.2-2 est la sémantique 2.1.2-3.

□

Lemme 2.6.7-1

Flim est un opérateur de fermeture supérieure sur $\langle \mathrm{Sem}\langle \mathcal{S}, \mathcal{A}\rangle, \subseteq\rangle$

Démonstration

Flim est évidemment monotone et extensif. Pour montrer l'idempotence posons $L(\Sigma) = \{p \in \Sigma^{\omega}\langle S,A\rangle : \forall n \in \omega.\ \exists q \in \Sigma.\ p^{\leq n} \mapsto q\}$. Pour montrer que $\mathrm{Flim}(\mathrm{Flim}(\langle S,A,\Sigma\rangle)) = \langle S,A,\Sigma \cup L(\Sigma) \cup L(\Sigma \cup L(\Sigma))\rangle = \mathrm{Flim}(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma \cup L(\Sigma)\rangle$ il faut montrer $L(\Sigma) = L(\Sigma \cup L(\Sigma))$ soit $L(L(\Sigma)) \subseteq L(\Sigma)$ car $L(\Sigma \cup L(\Sigma)) = L(\Sigma) \cup L(L(\Sigma))$.

Si $p \in L(L(\Sigma))$ alors $\forall n \in \omega.\ \exists q \in L(\Sigma).\ p^{\leq n} \mapsto q$ et donc $\exists q' \in \Sigma.\ p^{\leq n} \mapsto q'$ et donc $p \in L(\Sigma)$.

□

La relation induite sur les systèmes de transition est l'identité car :

Lemme 2.6.7~2

$$\underset{\sim}{Flim}(\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle) = \langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle$$

Observons que la fermeture par limites d'une sémantique n'introduit pas de nouveaux préfixes finis de traces et par conséquent deux sémantiques ayant mêmes fermetures par limites ont mêmes préfermetures par préfixes finis (la réciproque étant fausse):

Lemme 2.6.7~3

(1) $\underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Flim} = \underset{\sim}{Pref}^{<\omega}$

(2) $[\underset{\sim}{Flim}(\langle S_1, A_1, \Sigma_1\rangle) = \underset{\sim}{Flim}(\langle S_2, A_2, \Sigma_2\rangle)] \Rightarrow\!\!\!\!\!\!\not\quad\ [\underset{\sim}{Pref}^{<\omega}(\langle S_1, A_1, \Sigma_1\rangle) = \underset{\sim}{Pref}^{<\omega}(\langle S_2, A_2, \Sigma_2\rangle)]$

Démonstration

(1) Comme $\underset{\sim}{Flim}$ est extensif et $\underset{\sim}{Pref}^{<\omega}$ monotone, nous avons $\underset{\sim}{Pref}^{<\omega}(\langle S, A, \Sigma\rangle) \subseteq \underset{\sim}{Pref}^{<\omega}(\underset{\sim}{Flim}(\langle S, A, \Sigma\rangle))$. Si $\langle S', A', \Sigma'\rangle = \underset{\sim}{Pref}^{<\omega}(\underset{\sim}{Flim}(\langle S, A, \Sigma\rangle))$ et $r \in \Sigma'$ alors $r$ est le préfixe fini d'une trace de $\Sigma$ ou bien le préfixe fini d'une trace $p \in \Sigma^{\leq\omega}\langle S, A\rangle$ telle que si $n < \omega$, $\exists q \in \Sigma.\ p^{\leq n} \mapsto q$. Comme $r = p^{\leq n}$, nous en déduisons que $r$ est encore un préfixe fini d'une trace de $\Sigma$.

(2,⇒) Si $\underset{\sim}{Flim}(\langle S_1, A_1, \Sigma_1\rangle) = \underset{\sim}{Flim}(\langle S_2, A_2, \Sigma_2\rangle)$ alors $\underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Flim}(\langle S_1, A_1, \Sigma_1\rangle) = \underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Flim}(\langle S_2, A_2, \Sigma_2\rangle)$ et donc d'après le lemme 2.6.7~3.1, nous avons $\underset{\sim}{Pref}^{<\omega}(\langle S_1, A_1, \Sigma_1\rangle) = \underset{\sim}{Pref}^{<\omega}(\langle S_2, A_2, \Sigma_2\rangle)$.

(2,⇏) Le contre-exemple est $S_1 = S_2 = \{A\}$, $A_1 = A_2 = \{a\}$, $\Sigma_1 = \{A \xrightarrow{a} A,\ A \xrightarrow{a} A \xrightarrow{a} A \xrightarrow{a} A\}$ et $\Sigma_2 = \{A \xrightarrow{a} A \xrightarrow{a} A,\ A \xrightarrow{a} A \xrightarrow{a} A \xrightarrow{a} A\}$

□

Toutefois deux sémantiques ayant mêmes ensembles de traces finies et mêmes préfermetures par préfixes finis ont même fermeture par limites :

Lemme 2.6.7∾4

$$[(\Sigma_1 \cap \Sigma^{<\omega}\langle S_1, A_1\rangle) = (\Sigma_2 \cap \Sigma^{<\omega}\langle S_2, A_2\rangle) \wedge \underset{\sim}{Pref}^{<\omega}(\langle S_1, A_1, \Sigma_1\rangle) = \underset{\sim}{Pref}^{<\omega}(\langle S_2, A_2, \Sigma_2\rangle)]$$

$$\Rightarrow$$

$$[\underset{\sim}{Flim}(\langle S_1, A_1, \Sigma_1\rangle) = \underset{\sim}{Flim}(\langle S_2, A_2, \Sigma_2\rangle)]$$

Démonstration

Posons $L(\Sigma) = \{p \in \Sigma^{\omega}\langle S, A\rangle : \forall m \in \omega.\ \exists q \in \Sigma.\ p^{\leq m} \mapsto q\}$. Nous observons que $\underset{\sim}{Pref}^{<\omega}(\langle S_1, A_1, \Sigma_1\rangle) = \underset{\sim}{Pref}^{<\omega}(\langle S_2, A_2, \Sigma_2\rangle)$ entraîne que $S_1 = S_2$, $A_1 = A_2$ et $L(\Sigma_1) = L(\Sigma_2)$. Nous avons aussi $\Sigma \cap \Sigma^{\omega}\langle S, A\rangle \subseteq L(\Sigma)$. Le lemme dérive alors du fait que $\underset{\sim}{Flim}(\langle S, A, \Sigma\rangle) = \Sigma \cup L(\Sigma) = (\Sigma \cap \Sigma^{<\omega}\langle S, A\rangle) \cup L(\Sigma)$.

□

Comme conséquence, nous obtenons que la fermeture par limites de la réduction aux traces équitables (pour un nombre fini d'actions) d'une sémantique engendrée par un système de transition est cette sémantique :

Lemme 2.6.7∾5

Si $\underset{\sim}{card}(\alpha) < \omega$ alors

(1) $\underset{\sim}{Flim} \circ \underset{\sim}{Wfair}\langle\alpha\rangle(\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle) = \langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle$

(2) $\underset{\sim}{Flim} \circ \underset{\sim}{Sfair}\langle\alpha\rangle(\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle) = \langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle$

Démonstration

Par définition, $\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle$, $\underset{\sim}{Wfair}\langle\alpha\rangle(\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle)$ et $\underset{\sim}{Sfair}\langle\alpha\rangle(\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle)$ ont même ensembles de traces finies. Il suffit alors d'appliquer les lemmes 2.6.4∾2, 2.6.7∾4 et 2.6.7∾2.

□

Le résultat suivant portant sur la composition de fermetures d'une sémantique par fusions puis par limites sera utilisé ultérieurement pour caractériser les sémantiques engendrées par un système de transition :

Lemme 2.6.7 ∿ 6

(1) $\underset{\sim}{E}fus \circ \underset{\sim}{F}lim \circ \underset{\sim}{F}fus = \underset{\sim}{F}lim \circ \underset{\sim}{F}fus$

(2) $\underset{\sim}{F}fus \circ \underset{\sim}{F}lim \circ \underset{\sim}{F}fus = \underset{\sim}{F}lim \circ \underset{\sim}{F}fus$

(3) $\underset{\sim}{F}lim \circ \underset{\sim}{F}fus$ est un opérateur de fermeture supérieure sur $\langle \underset{\sim}{Sem}\langle \mathcal{S}, \mathcal{A} \rangle, \subseteq \rangle$

Démonstration

(1) Posons $\langle S,A,\Sigma_1\rangle = \underset{\sim}{F}fus(\langle S,A,\Sigma\rangle)$, $\langle S,A,\Sigma_2\rangle = \underset{\sim}{F}lim(\langle S,A,\Sigma_1\rangle) = \langle S,A,\Sigma_1 \cup L(\Sigma_1)\rangle$ où $L(\Sigma) = \{ p \in \Sigma^\omega\langle S,A\rangle : \forall m \in \omega.\ \exists q \in \Sigma.\ p^{\leq m} \mapsto q \}$, $\langle S,A,\Sigma_3\rangle = \underset{\sim}{E}fus(\langle S,A,\Sigma_2\rangle)$. Pour démontrer que $\Sigma_3 = \Sigma_2$, il suffit de démontrer que $\Sigma_3 \subseteq \Sigma_2$ car $\underset{\sim}{E}fus$ est extensive pour $\in$. Si $p \in \Sigma_3$ alors $p$ est de la forme $r \hat{} q$ avec $R = (r \hat{} r') \in \Sigma_2$ et $Q = (q' \hat{} q) \in \Sigma_2$. Comme $\Sigma_2 = \Sigma_1 \cup L(\Sigma_1)$, quatre cas sont à considérer :

- $R \in \Sigma_1$ et $Q \in \Sigma_1$ : d'après le lemme 2.6.5 ∿ 2 et la définition de $\Sigma_1$, nous avons $\underset{\sim}{E}fus(\langle S,A,\Sigma_1\rangle) = \langle S,A,\Sigma_1\rangle$ et donc $(r \hat{} r') \in \Sigma_1$ et $(q' \hat{} q) \in \Sigma_1$ ; nous en déduisons $p = r \hat{} q \in \Sigma_1 \subseteq \Sigma_2$.

- $R \in L(\Sigma_1)$ et $Q \in \Sigma_1$ : il existe $r'' \in \Sigma^{\leq\omega}\langle S,A\rangle$ telle que $(r \hat{} r'') \in \Sigma_1$, ce qui ramène au cas précédent.

- $R \in \Sigma_1$ et $Q \in L(\Sigma_1)$ : soit $i$ $(= |r_{>}|)$ le rang de $p_{|\Sigma_>|}$ dans $p$ et donc dans $R$ de sorte que $p_i = R_i$. Comme $R \in \Sigma_1$, il existe $L \in \Sigma_1$ telle que $R^{\leq m} \mapsto L$ pour $m \in |R|$. En particulier $\forall m \leq i.\ p^{\leq m} \mapsto L \in \Sigma_1$ car $p^{\leq m} = R^{\leq m}$. Comme $Q \in L(\Sigma_1)$, c'est une trace infinie et donc $p$ l'est également. Par définition de $L(\Sigma_1)$, nous avons $\forall m \in \omega.\ \exists L \in \Sigma_1.\ Q^{\leq m} \mapsto L$. Quand $m$ est strictement plus grand que le rang $j$ de $q_0$ dans $Q$, nous avons $L = ((q' \hat{} q)^{\leq m} \hat{} \ell) \in \Sigma_1$ soit $(q' \hat{} q^{\leq m-j-1} \hat{} \ell) \in \Sigma_1$, avec $\ell \in \Sigma^{\leq\omega}\langle S,A\rangle$. Comme $\underset{\sim}{E}fus(\langle S,A,\Sigma_1\rangle) = \langle S,A,\Sigma_1\rangle$, nous en déduisons $(r \hat{} q^{\leq m-j-1} \hat{} \ell) \in \Sigma_1$ soit $(p^{\leq m+i-j} \hat{} \ell) \in \Sigma_1$. En particulier quand $n = (m+i-j) > i$ soit $m > j$, $\exists L \in \Sigma_1.\ p^{\leq n} \mapsto L$ et donc $p \in L(\Sigma_1) \subseteq \Sigma_2$.

- $R \in L(\Sigma_1)$ et $Q \in L(\Sigma_1)$ : même raisonnement que précédemment car $R \in L(\Sigma_1)$ implique $\exists L \in \Sigma_1.\ R^{\leq m} \mapsto L$.

(2) D'après (1) et par récurrence $\underset{\sim}{E}fus^m \circ \underset{\sim}{F}lim \circ \underset{\sim}{F}fus = \underset{\sim}{F}lim \circ \underset{\sim}{F}fus$ et d'après le lemme 2.6.5 n°1, nous en déduisons $\underset{\sim}{F}fus \circ \underset{\sim}{F}lim \circ \underset{\sim}{F}fus = \underset{\sim}{F}lim \circ \underset{\sim}{F}fus$.

(3) $\underset{\sim}{F}lim$ et $\underset{\sim}{F}fus$ étant des opérateurs de fermeture, il reste à montrer l'idempotence qui résulte de (2).

□

## 2.6.8 RETRACTION D'UNE SEMANTIQUE PAR TRANSITIONS, SEMANTIQUE CLOSE

Dans une preuve de correction d'un programme, nous cherchons souvent à éviter les raisonnements sur les traces d'exécution d'une sémantique $\langle S,A,\Sigma\rangle$ et à les remplacer par des raisonnements sur le système de transition $\langle S,A,t,\varepsilon\rangle$ que cette sémantique engendre. Cette simplification ne se justifie que si nous pouvons montrer qu'une preuve relative à la rétraction par transitions $Rtran(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$ de la sémantique $\langle S,A,\Sigma\rangle$ peut remplacer la preuve relative à $\langle S,A,\Sigma\rangle$. Nous étudions les propriétés de l'opérateur Rtran qui nous serviront pour de telles justifications.

Définition 2.6.8 : 1

La rétraction par transitions d'une sémantique $\langle S,A,\Sigma\rangle$ est la sémantique engendrée par le système de transition engendré par cette sémantique :

$Rtran \in (Sem\langle\mathcal{S},\mathcal{A}\rangle \to Sem\langle\mathcal{S},\mathcal{A}\rangle)$

$Rtran(\langle S,A,\Sigma\rangle) = \langle S, A, \Sigma\langle S, A, t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle\rangle$

Exemples

La rétraction par transitions de la sémantique $\langle\{o\},\{a\},\{o \xrightarrow{a} o\}\rangle$ est $\langle\{o\},\{a\},\{o \xrightarrow{a} o \ldots o \xrightarrow{a} o \ldots\}\rangle$. Les sémantiques 2.1.2-1, 2.1.2-2 et 2.1.2-3 ont toutes pour rétraction par transitions la sémantique 2.1.2-3.

□

Rtran est une rétraction (monotone et idempotent) sur $\langle Sem\langle\mathcal{S},\mathcal{A}\rangle, \leq\rangle$ mais ce n'est pas un opérateur extensif (comme le montre l'exemple ci-dessus). Toutefois nous utiliserons le résultat suivant :

Théorème 2.6.8~1

Pref ∘ Rtran est une fermeture supérieure sur $\langle Sem\langle \mathcal{S}, \mathcal{A}\rangle, \subseteq\rangle$

Démonstration

Pref et Rtran étant monotones et idempotents, la composition l'est également. Il reste à montrer que si $\langle S,A,\Sigma_1\rangle = Pref \circ Rtran(\langle S,A,\Sigma\rangle)$ alors $\Sigma \subseteq \Sigma_1$. C'est évident car si $p \in \Sigma$, nous avons $\epsilon(p_0)$ et $\forall i \in |p|. t_{p_i}(p_i, p_{i+1})$ où $\langle S,A,t,\epsilon\rangle$ est le système de transition engendré par $\langle S,A,\Sigma\rangle$. Par conséquent, p est préfixe d'une trace de $\Sigma\langle S,A,t,\epsilon\rangle$ et donc $p \in \Sigma_1$ par définition de Pref. □

Une sémantique et sa rétraction par transitions n'étant en général pas comparables (et en particulier, pas $\subseteq$-comparables), le raisonnement sur une sémantique n'est pas équivalent au raisonnement sur le système de transition qu'elle engendre.

Exemple

Nous ne pouvons pas démontrer que toutes les traces des sémantiques 2.1.2-1 ou 2.1.2-2 sont finies en raisonnant sur le système de transition 2.3-1 qu'elles engendrent pour la raison que celui-ci engendre la trace infinie $0 \xrightarrow{a} 0 \ldots 0 \xrightarrow{a} 0 \ldots$.

□

Nous pouvons caractériser les cas où une sémantique et sa rétraction par transitions sont $\subseteq$-comparables comme suit :

Théorème 2.6.8 n 8

(1) $[\langle S,A,\Sigma\rangle \subseteq \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)] \Longrightarrow [\langle S,A,\Sigma\rangle = \underset{\sim}{Retps}(\langle S,A,\Sigma\rangle)]$

(2) $[\langle S,A,\Sigma\rangle \subseteq \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)] \Longleftarrow [\langle S,A,\Sigma\rangle = \underset{\sim}{Retps}(\langle S,A,\Sigma\rangle) \wedge \langle S,A,\Sigma\rangle = \underset{\sim}{Efus}(\langle S,A,\Sigma\rangle)]$

Démonstration

(1) Par l'absurde supposons $\langle S,A,\Sigma\rangle \subseteq \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)$ et $\underset{\sim}{Retps}(\langle S,A,\Sigma\rangle) \neq \langle S,A,\Sigma\rangle$. Comme $\underset{\sim}{Retps}$ est réductif, $\exists p,q \in \Sigma.\ p \mapsto q \wedge p \neq q$. Si $p$ était une trace infinie, $p \mapsto q$ entraînerait $p = q$, donc $p$ est une trace finie. Comme $\langle S,A,\Sigma\rangle \subseteq \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)$, $p$ est une trace engendrée par le système de transition $\langle S,A,t,\varepsilon\rangle$ engendré par $\langle S,A,\Sigma\rangle$ et donc $p_{|\sharp p|}$ est un état de blocage pour $t$. Par ailleurs, comme $p \mapsto q$ et $p \neq q$, nous avons $(|\sharp p|+1) \in |q|$ et donc par définition de $t$ il vient $t_{q_{|\sharp p|}}(q_{|\sharp p|}, q_{|\sharp p|+1})$, en contradiction avec $q_{|\sharp p|} = p_{|\sharp p|}$ d'après $p \mapsto q$.

(2) Soit $\langle S,A,\Sigma_t\rangle = \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)$. Par l'absurde supposons $\langle S,A,\Sigma\rangle = \underset{\sim}{Efus}(\langle S,A,\Sigma\rangle)$, $\langle S,A,\Sigma\rangle = \underset{\sim}{Retps}(\langle S,A,\Sigma\rangle)$ et $\exists P \in \Sigma.\ P \notin \Sigma_t$.

D'après le lemme 2.6.8 n 1 et par définition de $\Sigma_t$, $P$ est préfixe d'une trace $q$ de $\Sigma_t$. $P$ est un préfixe fini sinon nous aurions $P = q$ et $P \in \Sigma_t$. On a donc $P \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma_t\rangle)$.

$\langle S,A,\Sigma\rangle = \underset{\sim}{Efus}(\langle S,A,\Sigma\rangle)$ et le lemme 2.6.8 n 7 entraînent $P \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)$. On a donc $\exists p \in \Sigma, P \in \Sigma.\ P \mapsto p$, en contradiction avec $\langle S,A,\Sigma\rangle = \underset{\sim}{Retps}(\langle S,A,\Sigma\rangle)$.

□

Lemme 2.6.8 n3

$$[\,\underset{\sim}{Rtran}\,(\langle S,A,\Sigma\rangle) \subseteq \langle S,A,\Sigma\rangle\,]$$

$$\Leftrightarrow$$

$$[\,\underset{\sim}{Retps} \circ \underset{\sim}{Efus}\,(\langle S,A,\Sigma\rangle) \subseteq \langle S,A,\Sigma\rangle \;\wedge\; \underset{\sim}{Flim} \circ \underset{\sim}{Efus}\,(\langle S,A,\Sigma\rangle) = \underset{\sim}{Efus}\,(\langle S,A,\Sigma\rangle)\,]$$

Démonstration

($\Rightarrow$) Posons $\langle S,A,\Sigma_t\rangle = \underset{\sim}{Rtran}$, $\langle S,A,\Sigma_e\rangle = \underset{\sim}{Efus}\,(\langle S,A,\Sigma\rangle)$ et supposons $\Sigma_t \subseteq \Sigma$. Nous avons $\Sigma \subseteq \Sigma_e$ car $\underset{\sim}{Efus}$ est extensif.

Une trace de la sémantique $\underset{\sim}{Retps} \circ \underset{\sim}{Efus}\,(\langle S,A,\Sigma\rangle)$ est une trace de la forme $(p\hat{}q) \in \Sigma_e$ avec $p \mapsto p' \in \Sigma$ et $q' \twoheadrightarrow q \in \Sigma$ et qui n'est pas préfixe strict dans $\Sigma_e$. Par définition de $\underset{\sim}{Rtran}$, nous observons que $p\hat{}q$ est préfixe d'une trace $r \in \Sigma_t \subseteq \Sigma \subseteq \Sigma_e$. Comme $p\hat{}q$ n'est pas préfixe strict dans $\Sigma_e$, nous avons $(p\hat{}q) = r$ et par conséquent $(p\hat{}q) \in \Sigma$.

Pour tout $p \in \Sigma^{\omega}\langle S,A\rangle$, nous avons $\forall m \in \omega.\ \exists q \in \Sigma_e.\ p^{\leq m} \mapsto q$ qui entraîne $p \in \Sigma_t \subseteq \Sigma_e$ et donc $\underset{\sim}{Flim}\,(\langle S,A,\Sigma_e\rangle) = \langle S,A,\Sigma_e\rangle$.

($\Leftarrow$) Pour la réciproque, soient $\langle S,A,\Sigma_e\rangle = \underset{\sim}{Efus}\,(\langle S,A,\Sigma\rangle)$, $\langle S,A,\Sigma_r\rangle = \underset{\sim}{Retps}\,(\langle S,A,\Sigma_e\rangle)$ et $\langle S,A,t,\varepsilon\rangle$ le système de transition engendré par la sémantique $\langle S,A,\Sigma\rangle$.

Montrons d'abord que pour tous $p \in \Sigma\langle S,A,t,\varepsilon\rangle$, $i \in |p|$, $q \in \Sigma_e$. $p^{\leq i} \mapsto q$, nous avons $\exists q' \in \Sigma_r.\ p^{\leq i} \mapsto q'$. Posons $q^0 = q$. Si la trace $q^0$ de $\Sigma_e$ est préfixe strict dans $\Sigma_e$, elle se prolonge en une trace $q^1$ de $\Sigma_e$ telle que $p^{\leq i} \mapsto q^1$, qui à nouveau si elle est préfixe strict, se prolonge en une trace $q^2$ de $\Sigma_e$ telle que $p^{\leq i} \mapsto q^2$. Si cette construction s'arrête après $m+1$ pas, nous obtenons une trace $q^m$ de $\Sigma_e$ telle que $p^{\leq i} \mapsto q^m$, qui n'est pas préfixe strict et qui est donc élément de $\Sigma_r$. Dans ce cas nous choisissons $q' = q^m$. Si nous obtenons une suite infinie $q^j$, $j \geq 0$ de traces de $\Sigma_e$ telles que $p^{\leq i} \mapsto q^j$ et $(j<j') \Rightarrow ((q^j)^{\leq i} = (q^{j'})^{\leq i} \wedge q^j \neq q^{j'})$. Comme $\underset{\sim}{Flim}(\langle S,A,\Sigma_e\rangle) = \langle S,A,\Sigma_e\rangle$ la trace limite $q'$ définie par $q'^{\leq i} = p^{\leq i}$, $q'_j = q^j_j$ pour $j \geq i$ et $q'_j = q^j_j$ pour $j > i$ appartient à $\Sigma_e$. Etant infinie, elle n'est pas préfixe strict et appartient donc à $\Sigma_r$.

Montrons maintenant que pour tous $p \in \Sigma\langle S,A,t,\varepsilon\rangle$, $i \in |p|$, nous avons $\exists q^i \in \Sigma_r.\ p^{\leq i} \mapsto q^i$.

Si $i=0$ alors $p \in \Sigma\langle S,A,t,\varepsilon\rangle$ implique $\varepsilon(p_0)$ et donc par définition de $\varepsilon$, $\exists r \in \Sigma.\ r_0 = p_0$ de sorte que $\exists r \in \Sigma.\ p^{\leq 0} \mapsto r$. Comme $\Sigma \subseteq \Sigma_e$, nous déduisons du lemme ci-dessus l'existence de $q^0 \in \Sigma_r$ tel que $p^{\leq 0} \mapsto q^0$.

Supposons par hypothèse d'induction pour $i \in |p|$ et $i \neq 0$ l'existence de $q^{i-1} \in \Sigma_r$ tel que $p^{\leq i-1} \mapsto q^{i-1}$. Par définition de $t$, $\exists q' \in \Sigma, j \in |q'|.\ p_{i-1} = q'_{j-1} \wedge p_{i-1} = q'_{j-1} \wedge p_i = q'_j$. Nous en déduisons que $(p^{\leq i} \wedge q'^{\geq j}) \in \Sigma_e$ et donc l'existence de $q^i \in \Sigma_r$ tel que $p^{\leq i} \mapsto q^i$.

Si $p \in \Sigma\langle S,A,t,\varepsilon\rangle$ est finie alors nous avons montré qu'il existe $q^i \in \Sigma_r$ tel que $i = (|p|-1)$ et $p^{\leq i} \mapsto q^i$. Comme $p \in \Sigma\langle S,A,t,\varepsilon\rangle$, $p_i$ est un état de blocage et donc $q^i = p^{\leq i}$ car sinon nous aurions $(p^{\leq i} \wedge q^{\geq i}) \in \Sigma$ et donc $t_{q^{\geq i}}(p_i, q^i_{i+1})$ puisque $\langle S,A,t,\varepsilon\rangle$ est engendré par $\langle S,A,\Sigma\rangle$. Nous en déduisons $p = p^{\leq i} = q^i \in \Sigma$. Sinon $p$ est infinie et $\underline{\text{Flim}}(\langle S,A,\Sigma_e\rangle) = \langle S,A,\Sigma_e\rangle$ de sorte que $p \in \Sigma_e$ car $\forall i \in \omega.\ \exists q^i \in \Sigma_r \subseteq \Sigma_e$. Comme $p$ est infinie et $p \in \Sigma_e$, nous en déduisons $p \in \Sigma_r \subseteq \Sigma$.

□

Le cas particulier où une sémantique est rétractée par transitions ou <u>close</u> est important car alors $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ engendre exactement $\Sigma$ et tout raisonnement sur les traces $\Sigma$ du programme peut se ramener à un raisonnement sur le système de transition $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ (au pire de manière triviale en engendrant $\Sigma$ explicitement à partir de ce système de transition). Ceci conduit à

Définition 2.6.8 : 2

$$[\langle S,A,\Sigma\rangle \text{ est } \underline{\text{close}}] \Longleftrightarrow [\underline{\text{Rtran}}(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma\rangle]$$

Nous pouvons caractériser les sémantiques closes par les faits que

(α) le comportement futur de l'exécution dépend seulement de l'état courant qui a été atteint et non de la façon dont il a été atteint,

(β) le blocage de l'exécution en un état ne dépend que de cet état et

(γ) les limites des comportements finis sont des comportements infinis acceptables :

Théorème 2.6.8v4

[$\langle S,A,\Sigma\rangle$ est close]

[ $\underset{\sim}{Efus}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle \wedge \underset{\sim}{Retps}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle \wedge \underset{\sim}{Flim}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle$ ]

Démonstration

($\Rightarrow$) Lemmes 2.6.5v2, 2.6.6v1 et 2.6.7v2. ($\Leftarrow$) Lemmes 2.6.8v2 et 2.6.8v3.

□

(Emerson[80] a obtenu un résultat similaire avec une notion différente de sémantique utilisant des traces incomplètes). Essayons maintenant de caractériser l'opérateur Rtran :

Lemme 2.6.8v5

[ $\underset{\sim}{Retps}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle \wedge \underset{\sim}{Efus}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle$ ] $\Rightarrow$ [ $\underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)=\underset{\sim}{Flim}(\langle S,A,\Sigma\rangle)$ ]

Démonstration

Si $\underset{\sim}{Retps}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle$ et $\underset{\sim}{Efus}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle$ alors d'après 2.6.8v2, nous avons $\langle S,A,\Sigma\rangle \subseteq \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)$ et donc par monotonie $\underset{\sim}{Flim}(\langle S,A,\Sigma\rangle) \subseteq \underset{\sim}{Flim} \circ \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)$ or $\underset{\sim}{Flim} \circ \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle) = \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)$ d'après le théorème 2.6.8v4 donc $\underset{\sim}{Flim}(\langle S,A,\Sigma\rangle) \subseteq \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)$.

Flim ∘ Efus ∘ Flim ∘ Ffus (<S,A,Σ>) est égal à Flim ∘ Flim ∘ Efus (<S,A,Σ>) d'après le lemme 2.6.7∿6.1 soit Flim ∘ Ffus (<S,A,Σ>) car d'après 2.6.7∿1, Flim est idempotent, soit enfin Efus ∘ Flim ∘ Efus (<S,A,Σ>) en appliquant à nouveau 2.6.7∿6.1. D'autre part, Retps ∘ Efus ∘ Flim ∘ Ffus (<S,A,Σ>) = Retps ∘ Flim ∘ Ffus (<S,A,Σ>) ⊆ Flim ∘ Ffus (<S,A,Σ>) d'après 2.6.7∿6.1 et le fait que Retps soit réductif. D'après le lemme 2.6.8∿3, nous en déduisons que Rtran (Flim ∘ Ffus (<S,A,Σ>)) ⊆ Flim ∘ Ffus (<S,A,Σ>). Comme Flim et Ffus sont ⊆-extensives et Rtran ⊆-monotone, nous avons Rtran (<S,A,Σ>) ⊆ Rtran (Flim ∘ Ffus (<S,A,Σ>)) et donc par transitivité Rtran (<S,A,Σ>) ⊆ Flim ∘ Ffus (<S,A,Σ>). Mais <S,A,Σ> = Efus (<S,A,Σ>) et le lemme 2.6.5∿1 impliquent <S,A,Σ> = Ffus (<S,A,Σ>). On en déduit Rtran (<S,A,Σ>) ⊆ Flim (<S,A,Σ>).

□

La rétraction par transitions de la sémantique équitable (pour un nombre fini d'actions) engendrée par un système de transition est exactement la sémantique engendrée par ce système de transition :

Théorème 2.6.8∿6

Si card (α) < ω alors

(1) Rtran ∘ Wfair <α> ∘ Rtran = Rtran

(2) Rtran ∘ Sfair <α> ∘ Rtran = Rtran

Démonstration

Nous avons Retps ∘ Wfair <α> ∘ Rtran (<S,A,Σ>) = Wfair <α> ∘ Rtran (<S,A,Σ>) d'après 2.6.6∿2 et Efus ∘ Wfair <α> ∘ Rtran (<S,A,Σ>) = Wfair <α> ∘ Rtran (<S,A,Σ>) d'après 2.6.5∿3.1 donc d'après 2.6.8∿5, Rtran ∘ Wfair <α> ∘ Rtran (<S,A,Σ>) est égal à Flim ∘ Ffus ∘ Wfair <α> ∘ Rtran (<S,A,Σ>) et donc à Flim ∘ Wfair <α> ∘ Rtran (<S,A,Σ>) d'après le lemme 2.6.5∿3.2 soit encore Rtran (<S,A,Σ>) d'après le lemme 2.6.7∿5.1. La preuve est similaire pour (2).

□

Si une sémantique est fermée par fusions, elle a même préfixes finis que sa rétraction par transitions :

Lemme 2.6.8v7

$$[\underset{\sim}{Efus}(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma\rangle] \Rightarrow [\underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle) = \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)]$$

Démonstration

($\leq$) Posons $\langle S,A,\Sigma_1\rangle = \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)$. Soient $p \in \Sigma_1$, $i \in |p|$ de sorte que $p^{\leq i}$ est un préfixe fini de $p$. Montrons qu'il est aussi préfixe fini d'une trace de $\Sigma$. $p^{\leq 0} = p_0$ est préfixe fini d'une trace de $\Sigma$ car par définition de $\Sigma_1$, nous avons $\varepsilon(p_0)$ qui implique $\exists p' \in \Sigma.\ p'_0 = p_0$. Supposons maintenant que $p^{\leq i'}$, $i' \in i$ est un préfixe fini d'une trace de $\Sigma$. Donc il existe une trace $q \in \Sigma.\ q^{\leq i'} = p^{\leq i'}$. D'autre part, $p$ étant une trace de $\Sigma_1$, elle est engendrée par le système de transition $\langle S,A,t,\varepsilon\rangle$ engendré par $\langle S,A,\Sigma\rangle$, et nous avons $t_{\underset{\sim}{a}_{i'}}(p_{i'}, p_{i'+1})$ qui implique d'après 2.3:1 que $\exists r \in \Sigma, j \in |\underset{\sim}{r}|.\ (r_j = p_{i'} \wedge \underset{\sim}{r}_j = \underset{\sim}{a}_{i'} \wedge r_{j+1} = p_{i'+1})$. $\langle S,A,\Sigma\rangle$ étant fermée par fusions, la trace $p^{\leq i'} \xrightarrow{\underset{\sim}{a}_{i'}} p_{i'+1} = r_{j+1} \xrightarrow{\underset{\sim}{r}_{j+1}} r^{>j+1}$ est aussi une trace de $\Sigma$ et $p^{\leq i'+1}$ en est un préfixe fini. Finalement $p^{\leq i}$ est préfixe fini d'une trace de $\Sigma$.

($\geq$) $\langle S,A,\Sigma\rangle \leq \underset{\sim}{Pref} \circ \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)$ d'après 2.6.8v1. $\underset{\sim}{Pref}^{<\omega}$ étant monotone, nous avons $\underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle) \leq \underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Pref} \circ \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle) = \underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)$.

□

## 2.7 SPECIFICATION D'UNE SEMANTIQUE A L'AIDE D'UN SYSTEME DE TRANSITION

Comme nous chercherons à ramener les preuves de programmes relatives à une sémantique $\langle S, A, \Sigma \rangle$ à des preuves relatives au système de transition $\langle S, A, t\langle S, A, \Sigma\rangle, \varepsilon\langle S, A, \Sigma\rangle\rangle$ qu'elle engendre, il est naturel de chercher à spécifier la sémantique $\langle S, A, \Sigma\rangle$ à l'aide d'un système de transition $\langle S, A, t, \varepsilon\rangle$ aussi proche que possible de $\langle S, A, t\langle S, A, \Sigma\rangle, \varepsilon\langle S, A, \Sigma\rangle\rangle$.

Dans le cas d'une sémantique close (cf. 2.7.1) ces deux systèmes de transition sont équivalents.

Dans le cas d'une sémantique $\langle S, A, \Sigma\rangle$ non close (cf. 2.7.2) les traces de $\Sigma$ sont préfixes des traces engendrées par $\langle S, A, t\langle S, A, \Sigma\rangle, \varepsilon\langle S, A, \Sigma\rangle\rangle$ de sorte que cette sémantique $\langle S, A, \Sigma\rangle$ peut être spécifiée par un système de transition et une condition sur les préfixes des traces qu'il engendre (cf. 2.7.1.1). Nous montrons également que cette sémantique non close $\langle S, A, \Sigma\rangle$ est engendrée à une concordance près par un système de transition $\langle S^\#, A^\#, t^\#, \varepsilon^\#\rangle$ qui inclut un contrôleur surveillant l'exécution des programmes (cf. 2.7.1.2). Nous étudions ensuite (cf. 2.7.3) le cas particulier important d'une sémantique non close réduite par élimination des traces préfixes strictes et fermée par fusions. Dans ce cas $\Sigma$ est inclus dans l'ensemble des traces engendrées par $\langle S, A, t\langle S, A, \Sigma\rangle, \varepsilon\langle S, A, \Sigma\rangle\rangle$ de sorte que la sémantique $\langle S, A, \Sigma\rangle$ peut alors être spécifiée par un système de transition et une condition sur les traces qu'il engendre (cf. 2.7.3.1) ou bien comme précédemment par concordance avec une sémantique close (cf. 2.7.3.2).

## 2.7.1 SPECIFICATION D'UNE SEMANTIQUE CLOSE A L'AIDE DU SYSTEME DE TRANSITION QUI L'ENGENDRE

Les sémantiques closes sont caractérisées par le théorème 2.6.8~4 et peuvent d'après la définition 2.6.8:2 être définies par le système de transition qui les engendre exactement.

Exemple 2.7.1-1

(1) La sémantique 2.1.2-3 est engendrée par l'automate 2.3-1.

(2) La sémantique des programmes séquentiels (du style programmes PASCAL) est réduite par élimination des traces préfixes stricts et fermée par fusions et limites et peut donc d'après le théorème 2.6.8~4 être définie par un système de transition.

(3) Il en va de même pour les programmes asynchrones pour lesquels aucune hypothèse d'équité à l'exécution n'est faite. Ceci correspond aux hypothèses que chaque processus est exécuté par un processeur qui calcule à une vitesse indéterminée qui peut être nulle (quand le processeur tombe en panne) et que ces processeurs ne peuvent pas tous rester en panne indéfiniment. De ce fait, si l'exécution peut globalement progresser alors elle progressera fatalement (grâce à un processus exécuté sur un processeur qui n'est pas indéfiniment en panne). Formellement ceci s'exprime par le fait que la sémantique est réduite par élimination des traces préfixes stricts.

De plus, comme les vitesses relatives des processeurs sont inconnues (parce que les pannes sont imprévisibles ou bien parce que les processeurs calculent à des vitesses différentes sans être synchronisés), l'état suivant un état donné ne dépend pas de la façon dont celui-ci a été atteint. Formellement ceci s'exprime par le fait que la sémantique est fermée par fusions.

Enfin, l'exécution de toute opération qui peut être reportée pour un temps arbitrairement long peut également être reportée indéfiniment, ce qui fait que la sémantique est fermée par limites.

□

## 2.7.2 SPECIFICATION D'UNE SEMANTIQUE NON CLOSE

Dans le cas général, deux méthodes peuvent être utilisées pour spécifier une sémantique à l'aide d'un système de transition :

### 2.7.2.1 Spécification par un système de transition et une condition sur les préfixes des traces qu'il engendre

D'après le théorème 2.6.8v1, nous avons toujours $\langle S,A,\Sigma\rangle \subseteq Pref \circ Rtran(\langle S,A,\Sigma\rangle)$. Il est donc toujours possible de spécifier la sémantique $\langle S,A,\Sigma\rangle$ par le système de transition $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ qu'elle engendre et une condition sur les préfixes des traces engendrées par ce système de transition pour éliminer les préfixes parasites.

Exemple

La sémantique $S=\{o\}$, $A=\{a\}$, $\Sigma=\{o \xrightarrow{a} o,\ o \xrightarrow{a} o \xrightarrow{a} o\}$ peut être spécifiée par l'automate $\langle S,A,t,\varepsilon\rangle$ schematisé par :


et la restriction aux préfixes de longueur deux ou trois des traces qu'il engendre :

$$\Sigma = \{p : \exists q \in \Sigma\langle S,A,t,\varepsilon\rangle . (p \longrightarrow q \wedge 2 \leq |p| \leq 3)\}$$

□

### 2.7.2.2 Spécification par concordance avec une sémantique close

Pour faire des preuves sur une sémantique non close $\langle S,A,\Sigma\rangle$, il n'est pas équivalent de raisonner sur le système de transition qu'elle engendre. Toutefois, il est toujours possible de trouver une sémantique close $\langle S^\#,A^\#,\varepsilon^\#\rangle$ qui concorde avec la sémantique non close $\langle S,A,\Sigma\rangle$ à une relation entre états

et actions près. Par conséquent les preuves relatives à la sémantique non close $\langle S, A, \Sigma \rangle$ peuvent se faire, à une relation entre états et actions près, sur le système de transition $\langle S^\#, A^\#, t^\#, \varepsilon^\# \rangle$ engendré par cette sémantique close $\langle S^\#, A^\#, \Sigma^\# \rangle$. Il est donc toujours possible de raisonner sur des systèmes de transition.

Exemple

La sémantique $\langle \{0\}, \{a\}, \{0 \xrightarrow{a} 0, 0 \xrightarrow{a} 0 \xrightarrow{a} 0\} \rangle$ peut être définie par la sémantique engendrée par le système de transition représenté par l'automate :


à la concordance $r_S$ entre états et $r_A$ entre actions près définies par :

$r_S(\langle A, m \rangle, s') = [s = s']$

$r_A(a, a') = [a = a']$

□

Montrons qu'il est toujours possible de se restreindre au cas simple où la concordance entre sémantiques est définie à une fonction $\rho^\# \in (S^\# \to S)$ entre états près, (cf. 2.5.3.1) :

Théorème 2.7.2.2 v1

Pour toute sémantique $\langle S, A, \Sigma \rangle$, il existe une sémantique close $\langle S^\#, A^\#, \Sigma^\# \rangle$ et une fonction $\rho^\# \in (S^\# \to S)$ entre états telle que $\langle S, A, \Sigma \rangle$ dérive de $\langle S^\#, A^\#, \Sigma^\# \rangle$ à $\rho^\#$ près, c'est-à-dire :

$\simeq \langle \rho^\# \rangle (\langle S^\#, A^\#, \Sigma^\# \rangle) = \langle S, A, \Sigma \rangle$

Démonstration

$\langle S^\#, A^\#, \Sigma^\# \rangle$ peut toujours être choisie comme la sémantique engendrée par le système de transition $\langle S^\#, A, t^\#, \varepsilon^\# \rangle$ défini par :

$$S^\# = \Sigma \times \omega$$

$$t^\#_a(\langle p, i \rangle, \langle p', i' \rangle) = [(i+1) \in |p| \wedge a = p_i \wedge p' = p \wedge i' = i+1]$$

$$\varepsilon^\#(\langle p, i \rangle) = [i = 0]$$

avec

$$\rho^\#(\langle p, i \rangle) = p_i$$

□

En général, nous nous intéressons à des programmes exécutables par des machines et par conséquent la sémantique $\langle S^\#, A^\#, \Sigma^\# \rangle$ dérive de Rtran $(\langle S, A, \Sigma \rangle)$ par inclusion d'un contrôleur (scheduler) surveillant l'exécution des programmes.

Observons en effet, que toute sémantique $\langle S, A, \Sigma \rangle$ peut être définie par un système de transition $\langle S, A, t, \varepsilon \rangle$ et un contrôleur $\langle S^c, A, t^c, \varepsilon^c \rangle$ de sorte que $\langle S, A, \Sigma \rangle$ concorde avec la sémantique engendrée par $\langle S^\#, A, t^\#, \varepsilon^\# \rangle$ (où $S^\# = S \times S^c$, $t^\#_a(\langle s, s^c \rangle, \langle s', s^{c'} \rangle) = [t_a(s, s') \wedge t^c_a(s^c, s^{c'})]$ et $\varepsilon^\#(\langle s, s^c \rangle) = [\varepsilon(s) \wedge \varepsilon^c(s^c)]$ ) à une correspondance $\rho^\# \in (S^\# \to S)$ près telle que $\rho^\#(\langle s, s^c \rangle) = s$. L'objet de l'agent extérieur $\langle S^c, A, t^c, \varepsilon^c \rangle$ est de contrôler l'initialisation et les transitions du système $\langle S, A, t, \varepsilon \rangle$. En pratique, on ne s'intéresse qu'au cas où ces fonctions et relations $t, t^c, \varepsilon, \varepsilon^c$ sont calculables.

## 2.7.3 SPECIFICATION D'UNE SEMANTIQUE NON CLOSE REDUITE PAR ELIMINATION DES TRACES PREFIXES STRICTS ET FERMEE PAR FUSIONS

Nous considérons maintenant le cas assez fréquent d'une sémantique qui n'est pas close mais sans traces préfixes stricts et fermée par fusions.

Exemple 2.7.3-1

(1) L'automate 2.2-1 engendre la sémantique 2.1.2-3 qui, réduite aux traces faiblement équitables pour $\langle\{a,b\}\rangle$ est la sémantique 2.1.2-2 qui n'est pas fermée par limites puisque sa fermeture par limites est précisément la sémantique 2.1.2-3, d'après le lemme 2.6.7v5.1. La sémantique 2.1.2-2 n'est pas close mais elle est fermée par fusions et réduite par élimination des traces préfixes stricts.

(2) Il en va de même pour les programmes parallèles avec hypothèse d'exécution faiblement équitable des processus. Ceci correspond à l'idée que les processus sont exécutés par des processeurs différents dont aucun ne tombe indéfiniment en panne de sorte que les processus peuvent s'exécuter à des vitesses différentes mais aucun processus toujours prêt ne doit attendre indéfiniment. Cette hypothèse d'équité faible a (comme le montre l'exemple 2.7.3-1.1 ci-dessus) pour conséquence que la sémantique correspondante n'est pas fermée par limites.

De plus, comme les processeurs parallèles calculent à des vitesses différentes sans priorités, tous les entremêlements possibles des opérations effectuées par chaque processeur sont réalisables de sorte que l'état suivant un état donné ne dépend pas de la façon dont cet état donné a été atteint. Formellement ceci s'exprime par le fait que la sémantique est fermée par fusions.

Enfin, si l'exécution peut globalement progresser (un processeur au moins n'étant pas bloqué) alors elle progressera fatalement (car aucun des processeurs n'a une vitesse de calcul nulle). Par conséquent, la sémantique est réduite par élimination des traces préfixes stricts.

□

Notons par rapport au cas général une simplification des méthodes (2.7.2.1 et 2.7.2.2) de spécification de sémantiques non closes à l'aide de systèmes de transitions :

### 2.7.3.1 Spécification par un système de transition et une condition sur les traces qu'il engendre

Lorsque Retps $(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma\rangle$ et Efus $(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma\rangle$ et $\langle S,A,\Sigma\rangle$ n'est pas close, on a $\Sigma \subset \Sigma\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ d'après le théorème 2.6.8 n2 et la définition 2.6.8 : 2. Dans ce cas, il est donc possible de spécifier la sémantique $\langle S,A,\Sigma\rangle$ par le système de transition $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ qu'elle engendre et une condition sur les traces engendrées par ce système de transition pour éliminer les traces parasites, c'est-à-dire celles de $(\Sigma\langle S,A,t\langle S,A,\Sigma\rangle,\varepsilon\langle S,A,\Sigma\rangle\rangle \cap \Sigma)$.

Exemples 2.7.3.1 - 1

(1) La sémantique 2.1.2-2 peut se définir comme étant la sémantique engendrée par l'automate 2.2-1 avec la restriction aux traces finies.

(2) La sémantique des programmes parallèles avec hypothèse d'exécution faiblement équitable des processus est fermée par fusions, réduite par élimination des traces préfixes strictes mais pas fermée par limites (2.7.2-1.2). En associant une action différente à chaque processus (action qui devient le nom du processus), nous pourrons spécifier cette sémantique $\langle S,A,\Sigma\rangle$ à l'aide du système de transition $\langle S,A,t\langle S,A,\Sigma\rangle,\varepsilon\langle S,A,\Sigma\rangle\rangle$ et d'une condition sur les traces de la sémantique Rtran $(\langle S,A,\Sigma\rangle)$ ainsi définie pour éliminer les traces non équitables. Cette condition s'exprime naturellement par l'opérateur de fermeture inférieure Wfair $\langle\alpha\rangle$ où $\alpha$ est l'ensemble des noms de processus.

□

Quand nous utilisons cette méthode, nous définissons d'abord un système de transition $\langle S, A, t, \varepsilon \rangle$ puis une condition C sur les traces pour obtenir la sémantique $\langle S, A, \{p \in \Sigma\langle S, A, t, \varepsilon\rangle : C(p)\}\rangle$. Pour faire des preuves, nous chercherons à faire des raisonnements utilisant un système de transition plutôt que des traces. Il est possible d'utiliser $\langle S, A, t, \varepsilon\rangle$ ou le système de transition qui engendre $R_{trans}(\langle S, A, \{p \in \Sigma\langle S, A, t, \varepsilon\rangle : C(p)\}\rangle)$. Ces deux systèmes de transition sont en général différents. Il se pose alors le problème de savoir s'il est plus facile de spécifier la sémantique $\langle S, A, \Sigma\rangle$ par un moyen quelconque et d'en déduire le système de transition $\langle S, A, t\langle S, A, \Sigma\rangle, \varepsilon\langle S, A, \Sigma\rangle\rangle$ ou si à l'inverse il vaut mieux commencer par spécifier un système de transiti en déduire $\langle S, A, \Sigma\rangle$ par restriction de la sémantique . Sans pouvoir apporter de réponse générale à ce réponse est possible ( ces deux systèmes de transition ) dans chacun des cas particuliers que nous traitons.

Exemple 2.7.3.1-2

Le théorème 2.6.8v6 montre que dans le cas de programmes parallèles avec hypothèse d'exécution faiblement ou fortement équitable, il revient au puis d'en déduire le système système de transition puis d'en onde solution est donc généralement retenue car un système de transition qui définit un pas du calcul est plus simple à spécifier qu'une sémantique qui définit une suite de pas de calcul.

□

### 2.7.3.2 Spécification par concordance avec une sémantique close

La technique est la même que dans le cas général:

Exemple 2.7.3.2-1

Nous pouvons définir la sémantique 2.1.2-2, version faiblement équitable de 2.1.2-3, en ajoutant un contrôleur à l'automate 2.2-1 comme suit :

$S^{\#} = \{0,1\} \times \omega$

$A^{\#} = \{a,b\}$

$t_a^{\#}(\langle \Delta, m\rangle, \langle \Delta', m'\rangle) = [\Delta = \Delta' = 0 \wedge m' = m-1]$, $t_b^{\#}(\langle \Delta, m\rangle, \langle \Delta', m'\rangle) = [\Delta = 0 \wedge \Delta' = 1]$

$\varepsilon^{\#}(\langle \Delta, m\rangle) = [\Delta = 0]$

et en définissant la fonction $e^{\#}(\langle \Delta, m\rangle) = \Delta$.

□

Exemple 2.7.3.2-2

Nous pouvons définir la réduction aux traces faiblement équitables $W_{fair}(\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle)$ d'une sémantique $\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle$ engendrée par un système de transition $\langle S, A, t, \varepsilon\rangle$, en ajoutant un contrôleur au système de transition $\langle S, A, t, \varepsilon\rangle$ comme suit :

$S^{\square} = (A \to \omega) \times S$

$$t^{\square}(\langle m, \Delta\rangle, \langle m', \Delta'\rangle) = [\exists k \in A.\ t_k(\Delta, \Delta') \wedge \left( \bigvee \begin{array}{l} [m_k > 0 \wedge m'_k < m_k \wedge \forall j \in (A \setminus k).\ m'_j = m_j] \\ [\forall j \in A.\ ((\forall \Delta' \in S.\ \neg t_j(\Delta, \Delta') \vee m_j = 0) \wedge m'_j > 0)] \end{array} \right)]$$

$\varepsilon^{\square}(\langle m, \Delta\rangle) = \varepsilon(\Delta)$

et par concordance à la fonction $fs^{\square}$ des états près, définie par :

$fs^{\square}(\langle m, \Delta\rangle) = \Delta$

(Observons que le contrôleur organise l'exécution en reprises. Au début de la reprise, est déterminé le nombre maximal $m_k$ de fois l'action $k \in A$ sera activée pendant la reprise. Si l'action $k$ est activée dans l'état $\Delta$,

le nombre maximum d'activations dans la reprise décroit strictement pour k et reste inchangé pour les autres actions. Une nouvelle reprise peut commencer dans l'état s si toute action $k \in A$ n'est pas activable dans l'état s ou a été activée au moins une fois dans la reprise précédente).

Lemme 2.7.3.2-2v1

$$\simeq \langle f_s^{\natural} \rangle (\langle S^{\natural}, A, \Sigma\langle S^{\natural}, A, t^{\natural}, \varepsilon^{\natural}\rangle\rangle) = \underset{\sim}{W}fair(\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle)$$

Démonstration

Montrons que si $p^{\natural} \in \Sigma\langle S^{\natural}, A, t^{\natural}, \varepsilon^{\natural}\rangle$ alors $f_s^{\natural}(p^{\natural})$ est une trace faiblement équitable de $\Sigma\langle S, A, t, \varepsilon\rangle$. C'est évident si $p^{\natural}$ est une trace finie. Sinon, il existe $m \in (\omega \to (A \to \omega))$ et $p \in (\omega \to S)$ tels que $\forall i \in \omega . (p_i^{\natural} = \langle m_i, p_i\rangle)$. Supposons que $p = f_s^{\natural}(p^{\natural})$ ne soit pas faiblement équitable pour A, de sorte que $\exists k \in A, i \in \omega . \forall j \geq i . (\exists a \in S . t_k(p_j, a) \wedge \neg t_k(p_j, p_{j+1}))$. Dans une reprise, la somme des $m_{jk}$ pour $k \in A$ décroit strictement à chaque pas, de sorte qu'aucune reprise ne peut être infinie. En particulier, la reprise à laquelle "i appartient" doit se terminer en $i' \geq i$. Au début de la reprise suivante suivante, nous avons $m_{(i'+1)k} > 0$ et les $m_{jk}$ ne sont pas modifiés pendant cette reprise puisque l'action k n'est jamais activée. Quand cette dernière reprise se termine en $i'' > i'$, nous avons $\exists a \in S . t_k(p_{i''}, a) \wedge m_{i''k} = m_{(i'+1)k} > 0$ en contradiction avec la définition de $t^{\natural}$.

Il reste à montrer que si p est une trace de $\Sigma\langle S, A, t, \varepsilon\rangle$, faiblement équitable pour A, nous pouvons construire $m \in (|p| \to (A \to \omega))$ tel que $p^{\natural}$ définie par $(|p^{\natural}| = |p| \wedge \forall i \in |p| . p_i^{\natural} = \langle m_i, p_i\rangle)$ (et donc telle que $f_s^{\natural}(p^{\natural}) = p$) appartienne à $\Sigma\langle S^{\natural}, A, t^{\natural}, \varepsilon^{\natural}\rangle$.

Si $|p| = 1$, posons $m_{0k} = 0$

Si $1 < |p| < \omega$, posons $m_{ik} = \sum_{j=i}^{|p|-2} (t_k(p_j, p_{j+1}) \to 1 | 0)$

Si $|p| = \omega$, alors grâce à l'hypothèse d'équité faible, nous pouvons définir $\gamma \in (\omega \to \omega)$ par

$\gamma_0 = 0$ et $\gamma_{i+1} = \wedge\{j \in \omega : \forall k \in m.\ [(\forall l \geq \gamma_i.\ \forall a \in S.\ \neg t_k(p_l, a)) \vee (\exists l \in \omega.\ \gamma_i \leq l < j \wedge t_k(p_l, p_{l+1}))]\}$
de sorte que toutes les actions qui ne sont pas continuellement bloquées après $\gamma_i$ sont activées au moins une fois entre $\gamma_i$ et $\gamma_{i+1}$. Nous définissons $\eta \in (\omega \to \omega)$ tel que $\eta_i$ est le $\gamma_{j+1}$ tel que $\gamma_j \leq i < \gamma_{j+1}$. Posons
$m_{i_k} = \sum_{j=i}^{\eta_i} (t_k(p_j, p_{j+1}) \to 1 | 0)$.

□

□

## 2.8 EXEMPLE DE DEFINITION DE LA SEMANTIQUE D'UN LANGAGE DE PROGRAMMATION

Nous définissons la sémantique d'un langage que nous utiliserons ultérieurement comme exemple.

ammes ($\mathcal{P}_r$) peuvent être séquentiels ($\mathcal{P}_s$), parallèles , parallèles communicant par envois de messages ($\mathcal{P}_{pc}$), parallèles itables ($\mathcal{P}_{pw}$) ou parallèles synchronisés par sémc

$$\mathcal{P}_r \rightarrow \mathcal{P}_s \mid \mathcal{P}_{pa} \mid \mathcal{P}_{pc} \mid \mathcal{P}_{pw} \mid \mathcal{P}_{ps}$$

socions à tout programme $P_r$ syntaxiquement correct un système de transition $\langle S[\![P_r]\!], A[\![P_r]\!], t[\![P_r]\!], \varepsilon[\![P_r]\!]\rangle$ qui est spécifié par induction sur la syntaxe du programme.
La sémantique des programmes séquentiels, parallèles asynchrones et parallèles communicants est close. Elle peut donc être définie comme la sémantique e de transition, (cf. 2.7.1).
La sémantique des prog slement équitables et celle des programmes c éduite par élimination ds traces préfixes stricts, fermée par fusions mais n'est pas close. Elle sera définie par élimination traces non équitables de l'ensemble des traces engendrées par ce tème de transitions (cf. 2.7.3.1) ou bien à une fonction des états près en incorporant un contrôleur dans ce système de transition (cf. 2.7.3.2).

## 2.8.1 PROGRAMMES SEQUENTIELS

Un programme séquentiel est une suite de commandes exécutées en séquence. Une commande peut être nulle, une affectation, une affectation aléatoire ou une composition alternative ou itérative de commandes. Chaque commande est précédée et suivie par une étiquette qui n'apparait qu'une seule fois dans le programme et qui sert uniquement à désigner des points de contrôle du programme. Pour simplifier, les déclarations de types et de variables sont omises.

### 2.8.1.1 Syntaxe

Soient $\mathcal{L}, \mathcal{V}, \mathcal{E}$ et $\mathcal{B}$ des ensembles non vides donnés, respectivement d'étiquettes, de variables, d'expressions et d'expressions booléennes dont nous omettrons la syntaxe pour simplifier.

Les ensembles $\mathcal{P}s$ de programmes séquentiels et $\mathcal{C}$ de commandes séquentielles sont définis par la syntaxe suivante :

$\mathcal{P}s \longrightarrow \mathcal{L}_0 : \mathcal{C}_0 ; \ldots ; \mathcal{L}_{m-1} : \mathcal{C}_{m-1} ; \mathcal{L}_m :$ $\quad (m>0)$

$\mathcal{C} \longrightarrow$ skip | $\mathcal{V} := \mathcal{E}$ | $\mathcal{V} := ?$ | if $\mathcal{B}$ then $\mathcal{P}s_1$ else $\mathcal{P}s_2$ fi | while $\mathcal{B}$ do $\mathcal{P}s$ od

( if $\mathcal{B}$ then $\mathcal{P}s$ fi est l'abréviation de if $\mathcal{B}$ then $\mathcal{P}s$ else skip fi )

Soient $N$ une chaîne terminale dérivant du non-terminal $\mathcal{N}$, $\alpha_i$ des chaînes éventuellement vides et $N_i$ des chaînes terminales non vides dérivant respectivement des non-terminaux $\mathcal{N}_i$, nous utiliserons la notation $N \equiv \alpha_0 N_0 \ldots \alpha_m N_m \alpha_{m+1}$ pour $\exists \alpha_{0,\ldots,}\alpha_{m+1} \cdot (\ldots N = \alpha_0 N_0 \ldots \alpha_m N_m \alpha_{m+1} \ldots)$, le quantificateur étant convenablement placé pour lier toutes les occurences des $\alpha_{0,\ldots,}\alpha_{m+1}$ dans la formule où ils sont utilisés.

Cette notation sera utilisée pour exprimer des conditions de contexte sur les programmes comme par exemple qu'une étiquette ne peut

apparaître plus d'une fois dans un programme :

$$\forall Pr \in \mathcal{P}r.\ (Pr \equiv \alpha L_1 : \beta L_2 : \gamma) \Rightarrow (L_1 \neq L_2)$$

### 2.8.1.2 Sémantique

#### 2.8.1.2.1 Etats

Un état d'un programme séquentiel Ps est une paire <état de contrôle, état mémoire>. L'état de contrôle est similaire à un compteur de programme. Il est représenté par une étiquette L qui désigne un point unique du programme. Supposons que les variables prennent leurs valeurs dans un domaine $\mathcal{D}$ (que nous ne spécifions pas pour simplifier). L'état mémoire M est un élément de $\mathcal{M} = (\mathcal{V} \to \mathcal{D})$ c'est-à-dire une fonction totale de l'ensemble $\mathcal{V}$ des variables dans l'ensemble $\mathcal{D}$ des valeurs des variables.

Formellement, l'ensemble $S[\![Ps]\!]$ des états du programme Ps est :

$$S[\![Ps]\!] = \{L \in \mathcal{L} : Ps \equiv \alpha L : \beta\} \times \mathcal{M}$$

#### 2.8.1.2.2 Actions

N'ayant pas besoin dans la suite de nommer les actions qui sont sont exécutées par les programmes séquentiels, nous définissons :

$$A[\![Ps]\!] = \{\underline{a}\}$$

où $\{\underline{a}\}$ est l'action unique que nous omettrons dans la suite.

### 2.8.1.2.3 Etats initiaux

Les états initiaux d'un programme séquentiel Ps sont caractérisés par:

$$\varepsilon[\![Ps]\!] \in (S[\![Ps]\!] \to \{tt, ff\})$$
$$\varepsilon[\![Ps]\!](<L,M>) = [Ps \equiv L:\alpha]$$

### 2.8.1.2.4 Relation de transition

La relation de transition $t[\![Ps]\!]$ entre un état $<L,M>$ du programme séquentiel Ps et ses successeurs $<L',M'>$ (s'il en existe) est définie par:

$$t[\![Ps]\!] \in ((S[\![Ps]\!] \times S[\![Ps]\!]) \to \{tt, ff\})$$
$$t[\![Ps]\!](<L,M>,<L',M'>) = [\, \underset{\sim}{cond}[\![Ps]\!](L,L')(M) \wedge \underset{\sim}{succ}[\![Ps]\!](L)(M,M')]$$

$<L',M'>$ est l'état successeur de $<L,M>$ si et seulement si l'exécution d'un pas du programme séquentiel Ps au point de contrôle L dans l'état mémoire M a pour effet de déplacer le contrôle en L' (tel que $\underset{\sim}{cond}[\![Ps]\!](L,L')(M)$ est vrai) et de changer l'état mémoire en M' (tel que $\underset{\sim}{succ}[\![Ps]\!](L)(M,M')$ est vrai).

L'état mémoire ne peut être modifié que par une commande d'affectation. Une affectation aléatoire $V:=?$ affecte une valeur quelconque de $\mathcal{D}$ à la variable V. Une affectation $V:=E$ affecte à V la valeur $\mathbb{E}[\![E]\!](M)$ de l'expression E dans l'état mémoire M à la variable V. Pour simplifier, la sémantique $\mathbb{E} \in (\Sigma \to ((\mathcal{V} \to \mathcal{D}) \to \mathcal{D}))$ des expressions n'est pas spécifiée.

$$\underset{\sim}{succ}[\![Ps]\!](L)(M,M') = (Ps \equiv \alpha L: V:=E; \beta \Rightarrow \forall W \in (\mathcal{V} \sim V).[M'(W) = M(W)] \wedge M'(V) = \mathbb{E}[\![E]\!](M)$$
$$\mid (Ps \equiv \alpha L: V:=?; \beta \Rightarrow \forall W \in (\mathcal{V} \sim V).[M'(W) = M(W)]$$
$$\mid M' = M))$$

Pour définir le contrôle nous supposons donnée la sémantique $\mathbb{B} \in (\mathcal{B} \to ((\mathcal{V} \to \mathcal{D}) \to \{tt, ff\}))$ des expressions booléennes telle que $\mathbb{B}[\![B]\!](M)$ soit la valeur de l'expression booléenne B dans l'état mémoire M. $\mathbb{E}[\![B]\!]$ et $\mathbb{B}[\![B]\!]$ sont des fonctions partielles pour rendre compte des erreurs possibles à l'exécution. La relation de transition $\underline{cond}[\![Ps]\!](L,L')(M)$ entre l'état de contrôle L et son successeur L' dans l'état mémoire M est définie par cas comme suit :

$$
\begin{array}{l}
\underline{cond}[\![Ps]\!](L,L')(M) = \\
\quad [ \\
\quad\quad [ \quad Ps \equiv \alpha L : \underline{skip} \; ; \; L' : \beta \\
\quad\quad\quad \vee \quad Ps \equiv \alpha L : V := E ; \; L' : \beta \quad \wedge \; M \in \underline{dom}(\mathbb{E}[\![E]\!]) \\
\quad\quad\quad \vee \quad Ps \equiv \alpha L : V := ? ; \; L' : \beta \\
\quad\quad\quad \vee \quad Ps \equiv \alpha L : \underline{else} \; Ps' \; \underline{fi} \; ; \; L' : \beta \\
\quad\quad\quad \vee \quad Ps \equiv \alpha L : \underline{fi} \; ; \; L' : \beta \\
\quad\quad ] \\
\quad\quad \vee \\
\quad\quad [( \quad Ps \equiv \alpha L : \underline{if} \; B \; \underline{then} \; L' : \beta \\
\quad\quad\quad\quad \vee \quad Ps \equiv \alpha L : \underline{while} \; B \; \underline{do} \; L' : \beta \\
\quad\quad\quad\quad \vee \quad Ps \equiv \alpha \; \underline{while} \; B \; \underline{do} \; L' : I_0 ; \ldots ; L_{m-1} : I_{m-1} \; ; \; L : \underline{od} \; ; \; \beta \\
\quad\quad\quad ) \\
\quad\quad\quad \wedge \quad M \in \underline{dom}(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M) \\
\quad\quad ] \\
\quad\quad \vee \\
\quad\quad [( \quad Ps \equiv \alpha L : \underline{if} \; B \; \underline{then} \; Ps' \; \underline{else} \; L' : \beta \\
\quad\quad\quad\quad \vee \quad Ps \equiv \alpha L : \underline{while} \; B \; \underline{do} \; Ps' \; \underline{od} \; ; \; L' : \beta \\
\quad\quad\quad\quad \vee \quad Ps \equiv \alpha \; \underline{while} \; B \; \underline{do} \; L_0 : I_0 ; \ldots ; L_{m-1} : I_{m-1} \; ; \; L : \underline{od} \; ; \; L' : \beta \\
\quad\quad\quad ) \\
\quad\quad\quad \wedge \quad M \in \underline{dom}(\mathbb{B}[\![B]\!]) \wedge \neg \mathbb{B}[\![B]\!](M) \\
\quad\quad ] \\
\quad ]
\end{array}
$$

Par exemple, si le contrôle est au point L avant d'exécuter une boucle while ou après avoir exécuté son corps alors le contrôle passe au point L' qui désigne la première commande de son corps si le test est bien défini et vrai, le contrôle sort de la boucle si le test est bien défini et faux et le programme s'arrête en L (qui n'a pas de successeur) si le test n'est pas défini (ce qui produit une erreur à l'exécution).

#### 2.8.1.2.5 Traces

La sémantique d'un programme séquentiel $Ps \in \mathcal{P}s$ est :

$\langle S[\![Ps]\!], A[\![Ps]\!], \Sigma \langle S[\![Ps]\!], A[\![Ps]\!], t[\![Ps]\!], \varepsilon[\![Ps]\!] \rangle\rangle$

### 2.8.1.3 Exemple

Le programme séquentiel suivant calcule $z^m$ quand $m \geq 0$ :

```
0:
    P:=1;
1:
    while N>0 do
2:      P:=2xP;
3:      N:=N-1;
4:
    od;
5:
```

et sera utilisé pour illustrer quelques méthodes de preuve.

## 2.8.2 PROGRAMMES PARALLELES ASYNCHRONES

### 2.8.2.1 Syntaxe

Un programme parallèle asynchrone $\mathcal{P}pa$ est composé d'un prélude séquentiel $\mathcal{P}s$, suivi de processus séquentiels $[\![ \mathcal{P}ra_0 \| \ldots \| \mathcal{P}ra_{m-1} ]\!]$ partageant des variables globales communes et se déroulant en parallèle de manière asynchrone, suivis d'un postlude séquentiel $\mathcal{P}s'$ :

$$\mathcal{P}pa \longrightarrow \mathcal{P}s\ [\![ \mathcal{P}ra_0 \| \ldots \| \mathcal{P}ra_{m-1} ]\!]\ ;\ \mathcal{P}s' \qquad (m>1)$$

Les processus asynchrones $\mathcal{P}ra_i$, $i \in m$ sont des listes de commandes asynchrones étiquetées :

$$\mathcal{P}ra \longrightarrow \mathcal{L}_0 : \mathcal{C}a_0 ; \ldots ; \mathcal{L}_{m-1} : \mathcal{C}a_{m-1} ; \mathcal{L}_m : \qquad (m \geq 1)$$

Chaque commande asynchrone est une commande séquentielle (cf. 2.8.1.1) ou bien une liste de commandes séquentielles $\mathcal{P}s$ exécutées de manière indivisible, ce que nous notons $\langle \mathcal{P}s \rangle$ :

$$\mathcal{C}a \longrightarrow \underline{\text{skip}} \mid \mathcal{V} := \mathcal{E} \mid \mathcal{V} := ? \mid \underline{\text{if}}\ \mathcal{B}\ \underline{\text{then}}\ \mathcal{P}ra_1\ \underline{\text{else}}\ \mathcal{P}ra_2\ \underline{\text{fi}} \mid$$
$$\underline{\text{while}}\ \mathcal{B}\ \underline{\text{do}}\ \mathcal{P}ra\ \underline{\text{od}} \mid \langle \mathcal{P}s \rangle$$

### 2.8.2.2 Sémantique

#### 2.8.2.2.1 Etats

Un état d'un programme $\mathcal{P}pa = \mathcal{P}s\ [\![ \mathcal{P}ra_0 \| \ldots \| \mathcal{P}ra_{m-1} ]\!]\ ;\ \mathcal{P}s'$ est une paire ⟨état de contrôle, état mémoire⟩ où l'état mémoire affecte des valeurs dans $\mathcal{D}$ aux variables partagées $\mathcal{V}$. L'état de contrôle est une étiquette quand le contrôle est dans le prélude $\mathcal{P}s$ ou le postlude $\mathcal{P}s'$ ou un tuple $\langle \mathcal{L}_0, \ldots, \mathcal{L}_{m-1} \rangle$ de points de contrôle qui décrivent respectivement l'état de

contrôle des processus $Pra_0, \ldots, Pra_{m-1}$ qui se déroulent en parallèle. Formellement nous définissons :

$$S[\![Ppa]\!] = \left(\{L \in \mathscr{L} : Ps \equiv \alpha L:\beta \vee Ps' \equiv \alpha L:\beta\} \cup \prod_{i \in m} \{L \in \mathscr{L} : Pra_i \equiv \alpha L:\beta\}\right) \times \mathscr{M}$$

### 2.8.2.2.2 Actions

Les actions du programme $Ps[\![Pra_0 \| \ldots \| Pra_{m-1}]\!]; Ps'$ sont $\underline{\underline{\iota}}, 0, \ldots, m-1, \underline{\underline{\iota}}'$ correspondant respectivement à un pas dans le prélude, le processus $Pra_i$ et le postlude :

$$A[\![Ppa]\!] = \{\underline{\underline{\iota}}, \underline{\underline{\iota}}'\} \cup m$$

### 2.8.2.2.3 Etats initiaux

L'exécution du programme commence au point d'entrée du prélude dans un état mémoire quelconque :

$$\mathcal{E}[\![Ppa]\!] \in (S[\![Ppa]\!] \to \{tt, ff\})$$
$$\mathcal{E}[\![Ppa]\!](\Delta) = [\exists L \in \mathscr{L}, M \in \mathscr{M}.\ \Delta = \langle L, M\rangle \wedge Ppa \equiv L:\alpha]$$

### 2.8.2.2.4 Relation de transition

La relation de transition

$$t[\![Ppa]\!] \in (A[\![Ppa]\!] \to ((S[\![Ppa]\!] \times S[\![Ppa]\!]) \to \{tt, ff\}))$$

est définie par cas :

L'exécution du prélude et du postlude est purement séquentielle :

$$t[\![Ppa]\!]_{\sharp}(\langle L,M\rangle,\langle L',M'\rangle) = [Ppa \equiv Ps[\![Pra_0 \| \ldots \| Pra_{m-1}]\!]; Ps' \wedge t[\![Ps]\!](\langle L,M\rangle,\langle L',M'\rangle)]$$
$$t[\![Ppa]\!]_{\sharp'}(\langle L,M\rangle,\langle L',M'\rangle) = [Ppa \equiv Ps[\![Pra_0 \| \ldots \| Pra_{m-1}]\!]; Ps' \wedge t[\![Ps']\!](\langle L,M\rangle,\langle L',M'\rangle)]$$

- Les exécutions des processus parallèles commencent simultanément :

$$t[\![Ppa]\!]_{\sharp}(\langle L,M\rangle,\langle\langle L_0,\ldots,L_{m-1}\rangle,M'\rangle) = [Ppa \equiv \alpha L:[\![L_0:\alpha_0 \| \ldots \| L_{m-1}:\alpha_{m-1}]\!]; Ps' \wedge M'=M]$$

Nous rendons compte de l'exécution parallèle des processus par un mélange nondéterministe d'exécutions d'actions atomiques. Chaque pas du programme consiste donc à exécuter de manière indivisible une action atomique d'un processus $Pra_i$ tandis que les autres processus $Pra_j$, $j \in (m \setminus i)$ n'évoluent pas. L'exécution d'une action atomique indivisible correspond à l'évaluation d'une affectation ou d'un test d'une commande séquentielle (cf. 2.7.1.2.4) ou encore à l'exécution d'une liste de commandes séquentielles $\sharp Ps \flat$ en exclusion mutuelle. Formellement :

$$t[\![Ppa]\!]_i(\langle\langle L_0,\ldots,L_{m-1}\rangle,M\rangle,\langle\langle L'_0,\ldots,L'_{m-1}\rangle,M'\rangle) =$$
$$[Ppa \equiv Ps[\![Pra_0 \| \ldots \| Pra_{m-1}]\!]; Ps' \wedge \forall j \in (m \setminus i).\, L'_j = L_j \wedge$$
$$(t[\![Pra_i]\!](\langle L_i,M\rangle,\langle L'_i,M'\rangle) \vee (Pra_i \equiv \alpha L_i:\sharp Ps''\flat; L'_i\beta \wedge Ps'' \equiv L:\gamma L: \wedge t[\![Ps'']\!]^*(\langle L,M\rangle,\langle L',M'\rangle)))]$$

- L'exécution de la commande parallèle se termine quand tous les processus ont terminé leur exécution :

$$t[\![Ppa]\!]_{\sharp'}(\langle\langle L_0,\ldots,L_{m-1}\rangle,M\rangle,\langle L,M'\rangle) = [Ppa \equiv Ps[\![\alpha_0 L_0: \| \ldots \| \alpha_{m-1}; L_{m-1}:]\!]; L:\beta \wedge M'=M]$$

### 2.8.2.2.5 Traces

Nous définissons

$$\Sigma[\![Ppa]\!] = \Sigma\langle s[\![Ppa]\!], A[\![Ppa]\!], \varepsilon[\![Ppa]\!], t[\![Ppa]\!]\rangle$$

ce qui correspond à une exécution parallèle des processus sans aucune hypothèse d'équité. Par conséquent, l'exécution du programme

B:= true ; [[ B:= false || while B do skip od ]]

peut ne jamais se terminer si à chaque pas de l'exécution le second processus est activé tandis que le premier processus est toujours en attente.

### 2.8.2.3 Exemple

Le programme parallèle asynchrone suivant calcule $2^n$ pour $n \geq 0$ :

```
0:  [[
        11:
            P1 := 1;
        12:
            while  N>1  do
        13:
                ⟨ 131:
                        N:=N-1;
                  132:
                        P1:= 2×P1;
                  133: ⟩ ;
        14:
            od;
        15:
    ||
        21:
            P2: 1;
        22:
            while  N>1  do
        23:
                ⟨ 231:
                        N:= N-1;
                  232:
                        P2:=2×P2;
                  233: ⟩;
        24:
            od;
        25:
    ]];
1:  if  N=0  then  P:=P1×P2  else  P:= 2×P1×P2  fi ;
2:
```

(Remarquons que seule la commande d'affectation N:=N-1 doit être atomique. Aucune condition d'équité n'est nécessaire sur l'exécution des processus car le calcul peut être entièrement fait par l'un des processus).

## 2.8.3 PROGRAMMES PARALLELES COMMUNICANTS

### 2.8.3.1 Syntaxe

Pour décrire des programmes parallèles distribués où les processus se synchronisent et communiquent par des commandes d'envoi et de réception de messages sur rendez-vous, nous utilisons une variante de CSP (Hoare[78]) où les commandes de communication utilisent des canaux au lieu du nom des autres processus. Par simplicité également nous n'utiliserons pas de variables locales aux processus mais des variables globales (qui ne peuvent être ni consultées ni modifiées par les autres processus) :

$$\mathcal{P}pc \longrightarrow \mathcal{P}s \,[\![\, \mathcal{P}rc_0 \,\|\, \ldots \,\|\, \mathcal{P}rc_{m-1} \,]\!]\, ; \mathcal{P}s' \qquad (m>1)$$

Un processus est une liste séquentielle de commandes :

$$\mathcal{P}rc \longrightarrow \mathcal{L}_0 : \mathcal{C}c_0 \,;\, \ldots \,;\, \mathcal{L}_{m-1} : \mathcal{C}c_{m-1} \,;\, \mathcal{L}_m : \qquad (m \geq 1)$$

chaque commande étant une commande asynchrone (cf. 2.8.2.1) ou un envoi de message sur rendez-vous, une reception de message sur rendez-vous ou encore une commande alternative qui permet de selectionner une communication parmi plusieurs alternatives. Soit $\mathcal{C}h$ un ensemble de canaux, nous avons :

$$\begin{aligned} \mathcal{C}c \longrightarrow\ & \underline{\text{skip}} \mid \mathcal{V} := \mathcal{E} \mid \mathcal{V} := ? \mid \underline{\text{if}}\ \mathcal{B}\ \underline{\text{then}}\ \mathcal{P}rc_1\ \underline{\text{else}}\ \mathcal{P}rc_2\ \underline{\text{fi}} \\ & \text{while } \mathcal{B} \text{ do } \mathcal{P}rc \text{ od} \mid \not\langle\, \mathcal{P}s \,\not\rangle \mid \mathcal{C}h\,!\,\mathcal{E} \mid \mathcal{C}h\,?\,\mathcal{V} \mid \\ & \underline{\text{se}}\ \mathcal{A}lt_0\ \underline{\text{or}}\ \ldots\ \underline{\text{or}}\ \mathcal{A}lt_\ell\ \underline{\text{es}} \end{aligned}$$

$$\mathcal{A}lt \longrightarrow \mathcal{B}\,;\, \mathcal{C}h\,!\,\mathcal{E}\ \text{then}\ \mathcal{P}rc \mid \mathcal{B}\,;\, \mathcal{C}h\,?\,\mathcal{V}\ \underline{\text{then}}\ \mathcal{P}rc$$

### 2.8.3.2 Sémantique

Hoare n'ayant fait aucune hypothèse d'équité sur l'exécution des programmes CSP, la sémantique des programmes parallèles communicants est similaire à celle des programmes asynchrones sauf en ce qui concerne les commandes de communication. Dans la suite du paragraphe, nous avons :

$Ppc \equiv Ps \,[\![ Prc_0 \| \ldots \| Prc_{m-1} ]\!]\, ; Ps'$.

#### 2.8.3.2.1 Etats

Comme en 2.8.2.2.1, nous définissons :

$$S[\![Ppc]\!] = (\{L \in \mathcal{L} : Ps \equiv \alpha L:\beta \ \vee\ Ps' \equiv \alpha L:\beta\} \cup \bigcup_{i \in m} \{L \in \mathcal{L} : Prc_i \equiv \alpha L:\beta\}) \times \mathcal{M}$$

#### 2.8.3.2.2 Actions

Les actions du programme $Ps\,[\![ Prc_0 \| \ldots \| Prc_{m-1} ]\!]\,; Ps'$ sont $\sharp, 0, \ldots, m-1, \sharp'$ correspondant respectivement à un pas dans le prélude, le processus $Prc_i$ et le postlude comme en 2.8.2.2.2, plus l'ensemble des canaux intervenant dans le programme

$$A[\![Ppc]\!] = \{\sharp, \sharp'\} \cup m \cup \{\underline{Ch} \in \mathcal{C}h : Ppc \equiv \alpha Ch \beta\}$$

#### 2.8.3.2.3 Etats initiaux

Comme en 2.8.2.2.3, nous définissons :

$$\mathcal{E}[\![Ppc]\!] \in (S[\![Ppc]\!] \to \{tt, ff\})$$
$$\mathcal{E}[\![Ppc]\!](A) = [\exists L \in \mathcal{L}, M \in \mathcal{M}.\ A = \langle L, M\rangle \wedge Ppc \equiv L:\alpha]$$

### 2.8.3.2.4 Relation de transition

L'exécution d'un pas d'un programme parallèle communicant est similaire à celle d'un programme asynchrone sauf en ce qui concerne les commandes de communication. Par rapport à 2.8.2.2.4, nous rajoutons donc à la relation de transition un cas qui correspond à l'exécution d'une communication :

$$
\begin{aligned}
&t[\![Ppc]\!]_{\underline{Ch}}(\ll L_0,\dots,L_{m-1}\gg, M\gg, \ll L'_0,\dots,L'_{m-1}\gg, M'\gg) = \\
&\quad [\, Ppc \equiv Ps[\![Prc_0 \| \dots \| Prc_{m-1}]\!]; Ps' \ \wedge \\
&\quad \exists i,j \in m.\ \Big( Prc_i \equiv \alpha L_i : Ch!E\,;\ L'_i : \beta \\
&\qquad \vee \big(Prc_i \equiv \alpha L_i : \underline{as}\ Alt_0\ \underline{or}\ \dots\ \underline{or}\ Alt_{\ell-1}\ \underline{es} \ \wedge\ \exists k \in \ell. \\
&\qquad\quad Alt_k \equiv B_1\,;\ Ch!E\ \underline{then}\ L'_i : \eta \ \wedge\ M \in \underline{dom}(\mathbb{B}[\![B_1]\!]) \wedge \mathbb{B}[\![B_1]\!](M)\big)\Big) \\
&\qquad \wedge\ M \in \underline{dom}(\mathbb{E}[\![E]\!]) \\
&\qquad \wedge\ \Big( Prc_j \equiv \gamma L_j : Ch?V\,;\ L'_j : \delta \\
&\qquad\quad \vee \big(Prc_j \equiv \gamma L_j : \underline{as}\ Alt'_0\ \underline{or}\ \dots\ \underline{or}\ Alt'_p\ \underline{es} \ \wedge\ \exists q \in p. \\
&\qquad\quad\quad Alt'_q \equiv B_2\,;\ Ch?V\ \underline{then}\ L'_j : \xi \ \wedge\ M \in \underline{dom}(\mathbb{B}[\![B_2]\!]) \wedge \mathbb{B}[\![B_2]\!](M)\big)\Big) \\
&\qquad \wedge\ \forall k \in (m \setminus \{i,j\}).\ L'_k = L_k \\
&\qquad \wedge\ M'(V) = \mathbb{E}[\![E]\!](M) \ \wedge\ \forall W \in (\mathcal{V} \setminus V).\ (M'(W) = M(W))\ ]
\end{aligned}
$$

Nous rajoutons également le cas correspondant à la sortie d'une commande alternative :

$$
\begin{aligned}
&t[\![Ppc]\!]_i(\ll L_0,\dots,L_{m-1}\gg, M\gg, \ll L'_0,\dots,L'_{m-1}\gg, M'\gg) = \\
&\quad [Ppc \equiv Ps[\![Prc_0 \| \dots \| Prc_{m-1}]\!]; Ps' \ \wedge\ \forall j \in (m \setminus i).\ L'_j = L_j \ \wedge \\
&\quad \Big( t[\![Prc_i]\!](\langle L_i, M\rangle, \langle L'_i, M'\rangle) \\
&\qquad \vee \big(Prc_i \equiv \alpha L_i : \not\langle Ps'' \not\rangle\,;\ L'_i : \beta \ \wedge\ Ps'' \equiv L : \gamma\,;\ L' : \ \wedge\ t[\![Ps'']\!]^*(\langle L, M\rangle, \langle L', M'\rangle)\big) \\
&\qquad \vee \big(Prc_i \equiv \alpha\ \underline{as}\ Alt_0\ \underline{or}\ \dots\ \underline{or}\ \beta L_i :\ \underline{or}\ \dots\ \underline{or}\ Alt_{\ell-1}\ \underline{es}\,;\ L'_i : \gamma \ \wedge\ M' = M\big)\Big)\ ]
\end{aligned}
$$

#### 2.8.3.2.5 Traces

La sémantique d'un programme parallèle communicant Ppc est :

$\langle s[\![Ppc]\!], A[\![Ppc]\!], \Sigma < s[\![Ppc]\!], A[\![Ppc]\!], t[\![Ppc]\!], \in[\![Ppc]\!] >>$

### 2.8.3.3 Exemples

Exemple 2.8.3.3-1

Le programme parallèle suivant réalise une section critique. Remarquons que le troisième processus joue le rôle de contrôleur. (Ayant défini (avec les notations Pascal) le type signal = (any) et la la variable Any : signal, nous écrivons P!Any (respectivement P?Any) pour la transmission (respectivement reception) d'un signal).

```
0:
  [[ 11:
        while true do
     12:
        P! any ;
     13:
        V! any ;
     14:
     od ;
     15:
  ||
     21:
     while true do
     22:
        P! any ;
     23:
        V! any ;
     24:
     od;
     25;
  ||
     31:
     while true do
     32:
        P? Any ;
     33:
        V? Any;
     34:
     od;
     35:
  ];
1:
```

Remarquons que l'entrée dans une section critique donnée n'est pas fatale.

Par contre, elle l'est dans la version suivante où l'hypothèse d'équité faible est nécessaire. (Nous codons la valeur du sémaphore et le contenu de la file d'attente comme suit :

- $s=0$ le sémaphore est ouvert
- $s=i$ le sémaphore est fermé. Il a été franchi par le processus $i$, $i=1,2$ et ensuite le processus $\tilde{i}$ n'a pas demandé à le franchir (où nous notons $\tilde{1}=2$ et $\tilde{2}=1$)
- $s=i\tilde{i}$ le sémaphore est fermé. Il a été franchi par le processus $i$ et ensuite le processus $\tilde{i}$ a demandé à le franchir).

```
0:
   S := 0;
1:
   [[ 11:
        while true do
   12:
            if S=0 then S:=1 else S:=12;   {demande d'entrée en section critique}
   13:
            P1 ! Any;                      {autorisation d'entrée en section critique}
   14:
            V ! Any;                       {sortie de la section critique}
   15:
        od;
   16:
   ||
   21:
        while true do
   22:
            if S=0 then S:=2 else S:=21;
   23:
            P2 ! Any;
   24:
            V ! Any;
   25:
        od;
   26:
   ||
   31:
        while true do
   32:
            se  (S=1); P1?Any then skip;
   33:
              or (S=21); V?Any then S:=2; P2?Any;
   34:
              or (S=2); P2?Any then skip;
   35:
              or (S=12); V?Any then S:=1; P1?Any;
   36:
              or ((S=1) v (S=2)); V?Any then S:=0;
   37:
            es;
   38:
        od;
   39:
   ]];
2:
```

Exemple 2.8.3.3.2

Le programme suivant (Hoare [73]) réalise la transmission de messages d'un producteur vers un consommateur via un tampon pour régulariser les flux ; les variables sont du type suivant :

```
type signal = (any);
N : 1.. maxint;
Output, Input : array [0.. N-1] of type-messages;
Buffer        : array [0.. 9] of type-messages;
I, Im, Out, J : integer;
Pe, Ft, Te    : boolean;
PTm, TCm      : channel type-messages;
PTs, TCs      : channel signal;
Any           : signal;
```

```
0:
[[
{Producteur :}  11:      I := 0;
                12:      while I<>N do
                13:          PTm ! Output [I];
                14:          I := I+1;
                15:      od;
                16:      PTs ! any;
                17:
```

```
||
{Tampon}  201:
                Im:=0;  Out:=0;  Pe:= false;   Ft := false;
          202:
                while  not Ft  do
          203:
                   as  ¬Pe ∧ Im<Out+10 ; PTm? Buffer[Im mod 10]  then
          204:
                       Im := Im+1;
          205:
                   or  Out<Im;  TCm ! Buffer [Out mod 10] then
          206:
                       Out := Out+1 ;
          207:
                   or  ¬Pe ;  PTA? Any  then
          208:
                       Pe := true ;
          209:
                   or   Pe ∧ (Im=Out) ;  TCA ! any  then
          210:
                       Ft := true ;
          211:
                   sa;
          212:
                od;
          213:

||
{Consommateur:}  31:
                J:=0;  Te := false;
          32:
                while ¬Te  do
          33:
                   as  true ;  TCm ? Imput [J]  then
          34:
                       J := J+1 ;
          35:
                   or  true ;  TCA ? Any  then
          36:
                       Te := true ;
          37:
                   sa ;
          38:
                od ;
          39:

||;
1:
```

## 2.8.4 PROGRAMMES PARALLELES FAIBLEMENT EQUITABLES

### 2.8.4.1 Syntaxe

Pour simplifier nous adoptons la même syntaxe que pour les programmes asynchrones :

$\mathcal{P}pw \longrightarrow \mathcal{P}pa$

### 2.8.4.2 Sémantique

La sémantique d'un programme parallèle faiblement équitable peut être définie de manière équivalente (cf. 2.7.3.2)

- comme la restriction $\underset{\sim}{W}fair(\langle S[\![Ppw]\!], A[\![Ppw]\!], \Sigma\langle S[\![Ppw]\!], A[\![Ppw]\!], t[\![Ppw]\!], \varepsilon[\![Ppw]\!]\rangle\rangle$ aux traces faiblement équitables de la sémantique des programmes parallèles
- par concordance avec la sémantique close $\langle S[\![Ppw]\!]^{\not\Box}, A[\![Ppw]\!], \Sigma\langle S[\![Ppw]\!]^{\not\Box}, A[\![Ppw]\!], t[\![Ppw]\!]^{\not\Box}, \varepsilon[\![Ppw]\!]^{\not\Box}\rangle\rangle$ à la fonction $fs^{\not\Box}$ entre états près.

### 2.8.4.3 Exemple

Soit $F \in (\mathbb{Z} \to \mathbb{Z})$ un opérateur sur les entiers. S'il possède un point fixe alors l'exécution du programme parallèle faiblement équitable suivant se termine avec une valeur finale $p$ de $P$ telle que $F(p)=p$.
Le premier processus cherche un point fixe parmi les entiers strictement négatifs et le deuxième processus cherche parmi les entiers positifs ou nuls. L'hypothèse d'équité faible est donc nécessaire pour garantir la terminaison :

```
0:
   S1 := true ;  S2 := true
1:
   [[
      10:
         L := -1;
      11:
         while S1 ∧ S2 do
      12:
            if F(L) = L then
      13:
               S1 := false;
      14:
            else
      15:
               L := L-1;
      16:
            fi;
      17:
         od;
      18:
   ||
      20:
         H := 0;
      21:
         while S1 ∧ S2 do
      22:
            if F(H) = H then
      23:
               S2 := false;
      24:
            else
      25:
               H := H+1;
      26:
            fi;
      27:
         od;
      28:
   ]];
2:
   if ¬S1 then
3:
      P := L;
4:
   else if ¬S2 then
5:
      P := H;
6:
   fi;
7:
```

## 2.8.5 PROGRAMMES PARALLELES SYNCHRONES

De nombreuses solutions ont été proposées pour synchroniser des processus parallèles partageant des données communes. En général, elles peuvent s'implanter à l'aide de la notion primitive de sémaphores (Dijkstra [68]). C'est la solution que nous avons retenue comme exemple en raison de sa généralité et de sa relative simplicité.

### 2.8.5.1 Syntaxe

La syntaxe des programmes parallèles synchrones

$$\mathcal{P}ps \longrightarrow \mathcal{P}s \,[\![\, \mathcal{P}rs_0 \parallel \ldots \parallel \mathcal{P}rs_{m-1} \,]\!]\, ; \mathcal{P}s' \qquad (m>1)$$

ne diffère de la syntaxe des programmes parallèles asynchrones que par le fait que les processus synchrones

$$\mathcal{P}rs \longrightarrow \mathcal{L}_0 : \mathcal{C}s_0 \; ; \ldots ; \mathcal{L}_{m-1} : \mathcal{C}s_{m-1} \; ; \mathcal{L}_m : \qquad (m \geq 1)$$

sont des listes de commandes synchrones étiquetées. Ces commandes synchrones peuvent être une commande séquentielle ou bien une opération $\underline{p}$ ou $\underline{v}$ sur un sémaphore. Soit $\mathcal{S}e \subseteq \mathcal{V}$ un ensemble non vide de variables sémaphores, nous avons

$$\mathcal{C}s \longrightarrow \underline{\text{skip}} \mid \mathcal{V} := \mathcal{E} \mid \mathcal{V} := ? \mid \underline{\text{if}}\ \mathcal{B}\ \underline{\text{then}}\ \mathcal{P}rs_1\ \underline{\text{else}}\ \mathcal{P}rs_2\ \underline{\text{fi}} \mid$$
$$\underline{\text{while}}\ \mathcal{B}\ \underline{\text{do}}\ \mathcal{P}rs\ \underline{\text{od}} \mid \underline{p}(\mathcal{S}e) \mid \underline{v}(\mathcal{S}e)$$

Comme pour les autres variables, les déclarations de sémaphores ($\underline{\text{Sem}}$ Se $\underline{\text{init}}$ $\Lambda_0$) sont laissées implicites. Nous supposerons cependant qu'elles associent à tout sémaphore $\text{Se} \in \mathcal{S}e$, une valeur initiale entière $\underline{\text{Isem}}(\text{Se})$ (notée $\Lambda_0$ ci-dessus).

Les conditions de contexte que les sémaphores n'apparaissent pas ailleurs que dans les commandes $\underline{p}$ et $\underline{v}$ et que les autres variables ne peuvent pas apparaître dans ces instructions sont habituelles :

$$\wedge \begin{array}{l} ([(Pps \equiv \alpha \in \beta \wedge E \equiv \alpha' v \beta') v (Pps \equiv \alpha V := \beta) v (Pps \equiv \alpha B \beta \wedge B \equiv \alpha' v \beta')] \Rightarrow V \notin Se) \\ ([(Pps \equiv \alpha \underline{p}(V) \beta) v (Pps \equiv \alpha \underline{v}(V) \beta)] \Rightarrow V \in Se) \end{array}$$

### 2.8.5.2 Sémantique

De très nombreuses définitions opérationnelles de la sémantique des sémaphores ont été proposées. Des différences souvent légères entre ces définitions entraînent parfois de subtiles différences de comportement pour les programmes qui les utilisent. La définition que nous proposons se veut fidèle à la définition originale de Dijkstra [68]. D'un point de vue pratique, on pourra stratégies de gestion des processus e d'attente sont utiles, mais Haberm de Dijkstra avec files d'attente st autres stratégies.

Dans la suite du paragraphe, nous posons $Pps \equiv Ps \,[\![ Prs_0 \| \ldots \| Prs_{n-1} ]\!]; Ps'$.

#### 2.8.5.2.1 Etats

La différence avec les états des programmes parallèles asynchrones (cf. 2.8.2.2.1) est qu'à tout sémaphore $Se \in \mathcal{S}e \subseteq \mathcal{V}$ est associé une valeur entière et une file d'attente de processus. Pour simplifier, nous supposons $\mathbb{Z} \subseteq \mathcal{D}$ et représentons la file d'attente par une séquence (éventuellement vide) d'éléments de $m$ :

$$C[\![Pps]\!] = \{L \in \mathcal{L} : Ps \equiv \alpha L:\beta \vee Ps' \equiv \alpha L:\beta\} \cup \underset{i \in m}{\pi} \{L \in \mathcal{L} : Prs_i \equiv \alpha L:\beta\}$$
$$S[\![Pps]\!] = C[\![Pps]\!] \times \mathcal{M} \times (\mathcal{S}e \to n^{<\omega})$$

### 2.8.5.2.2 Actions

Les actions du programme parallèle synchrone $Pps \equiv Ps \,[\![\, Prs_0 \,\|\, \ldots \,\|\, Prs_{m-1} \,]\!]; Ps'$ sont :

- $\underline{\sharp}$ correspondant à l'exécution d'un pas du prélude Ps
- $\underline{\sharp}'$ correspondant à l'exécution d'un pas du postlude Ps'
- $i$ correspondant à l'exécution d'un pas du processus $Prs_i$, $i \in m$ (autre qu'une opération sur un sémaphore)

La commande $\underline{\sharp}(Se)$ donne lieu à deux actions possibles, à savoir la mise en attente (généralement suivie d'un réveil par un autre processus exécutant un $\underline{v}(Se)$) ou le passage du sémaphore :

- $\underline{w}(Se,i)$ correspond à l'exécution de la commande $\underline{\sharp}(Se)$ par le processus $Prs_i$ qui provoque la mise en attente de ce processus devant le sémaphore Se.
- $\underline{\sharp}(Se,i)$ correspond à l'exécution de la commande $\underline{\sharp}(Se)$ par le processus $Prs_i$ et au passage de ce sémaphore par ce processus

La commande $\underline{v}(Se)$ donne lieu à deux actions possibles selon qu'il y ait ou non un processus en attente qu'il faut réveiller :

- $\underline{v}(Se,i)$ le processus $Prs_i$ exécute une commande $\underline{v}(Se)$ qui libère le sémaphore alors qu'aucun autre processus n'était en attente sur ce sémaphore
- $\underline{v\sharp}(Se,i,j)$ le processus $Prs_i$ exécute une commande $\underline{v}(Se)$ qui libère le sémaphore et permet au processus $Prs_j$ qui était en attente de le passer.

Nous avons donc :

$$A[\![Pps]\!] = \{\underline{\sharp}, \underline{\sharp}'\} \cup m \cup \{\underline{w}(Se,i), \underline{\sharp}(Se,i), \underline{v}(Se,i), \underline{v\sharp}(Se,i,j) : Se \in \mathcal{S}e \wedge i,j \in m \wedge i \neq j\}$$

### 2.8.5.2.3 Etats initiaux

Dans un état initial le contrôle est au début du programme. Les sémaphores $Se \in \mathcal{Se}$ ont la valeur initiale $\underline{Inem}(Se)$ (définie par une déclaration, ...) et la file d'attente associée est vide :

$$\varepsilon[\![Pps]\!](\Delta) = [\exists L \in \mathcal{L}, M \in \mathcal{M}, Q \in (\mathcal{Se} \to m^{<\omega}).\ (\Delta = \langle L, M, Q\rangle \wedge Pps \equiv L:\alpha \wedge \forall Se \in \mathcal{Se}.(M(Se) = \underline{Inem}(Se) \wedge Q(Se) = \langle\rangle))]$$

### 2.8.5.2.4 Relation de transition

Pour les actions $\sharp, \sharp'$ et $0, \ldots, m-1$ la relation de transition est similaire à celle des programmes parallèles asynchrones (cf. 2.8.2.2.4, la valeur et la file d'attente des sémaphores n'étant pas modifiées par ces actions). Il convient d'ajouter quatre cas correspondant à l'exécution d'une commande sur un sémaphore :

$$t[\![Pps]\!]_{\underline{\omega}(Se,i)}(\langle\langle L_0, \ldots, L_{m-1}\rangle, M, Q\rangle, \langle\langle L'_0, \ldots, L'_{m-1}\rangle, M', Q'\rangle) =$$

$$[Pps \equiv Ps[\![Prs_0 \| \ldots \| Prs_{m-1}]\!]; Ps' \wedge Prs_i \equiv \alpha L_i : \sharp(Se); \beta \wedge M(Se) \leq 0 \wedge \forall l \in |Q(Se)|.\ Q(Se)_l \neq i \wedge \forall k \in m.\ L'_k = L_k \wedge M' = M \wedge Q'(Se) = (Q(Se)^\frown\langle i\rangle) \wedge \forall Sm \in (\mathcal{Se} \smallsetminus Se).\ Q'(Sm) = Q(Sm).]$$

(Le processus $Prs_i$ exécute $\sharp(Se)$ alors que le sémaphore a une valeur négative ou nulle et que le processus $Prs_i$ n'est pas dans la file d'attente. Le processus $Prs_i$ entre dans la file d'attente et le contrôle reste aux mêmes points du programme).

$$t[\![Pps]\!]_{\sharp(Se,i)}(\langle\langle L_0, \ldots, L_{m-1}\rangle, M, Q\rangle, \langle\langle L'_0, \ldots, L'_{m-1}\rangle, M', Q'\rangle) =$$

$$[Pps \equiv Ps[\![Prs_0 \| \ldots \| Prs_{m-1}]\!]; Ps' \wedge Prs_i \equiv \alpha L_i : \sharp(Se); L'_i : \beta \wedge M(Se) > 0 \wedge \forall k \in (m \smallsetminus i).\ L'_k = L_k \wedge M'(Se) = M(Se) - 1 \wedge \forall V \in (\mathcal{V} \smallsetminus Se).M'(V) = M(V) \wedge Q' = Q]$$

(Le processus $Prs_i$ exécute $\sharp(Se)$ alors que le sémaphore a une valeur strictement positive. Le contrôle du processus passe donc la commande $\sharp(Se)$ et la valeur du sémaphore diminue de 1).

$$t[\![Pps]\!]_{\underline{v}(Se,i)}(\ll L_0,\dots,L_{m-1}\gg, M, Q\gg, \ll L'_0,\dots,L'_{m-1}\gg, M', Q'\gg) =$$

$$[\ Pps \equiv Ps[\![Prs_0 \| \dots \| Prs_{m-1}]\!]; Ps' \ \wedge\ Prs_i \equiv \alpha L_i : \underline{v}(Se); L'_i : \beta \ \wedge\ Q(Se) = \langle\rangle \ \wedge$$
$$\forall k \in (m \setminus i).\ L'_k = L_k \ \wedge\ M'(Se) = M(Se)+1 \ \wedge\ \forall v \in (\mathcal{V} \setminus Se).\ M'(v) = M(v) \ \wedge\ Q' = Q\ ]$$

(Le processus $Prs_i$ exécute la commande $\underline{v}(Se)$ alors qu'aucun autre processus n'est en attente sur ce sémaphore. Le contrôle de $Prs_i$ franchit la commande et la valeur du sémaphore est augmentée de 1).

$$t[\![Pps]\!]_{\sharp(Se,i,j)}(\ll L_0,\dots,L_{m-1}\gg, M, Q\gg, \ll L'_0,\dots,L'_{m-1}\gg, M', Q'\gg) =$$

$$[\ Pps \equiv Ps[\![Prs_0 \| \dots \| Prs_{m-1}]\!]; Ps' \ \wedge\ Prs_i \equiv \alpha L_i : \underline{v}(Se); L'_i : \beta \ \wedge\ Q(Se) = (\langle j\rangle \hat{}\, Q'(Se)) \ \wedge$$
$$Prs_j \equiv \alpha' L_j : \sharp(Se); L'_j : \beta' \ \wedge\ \forall k \in (m \setminus \{i,j\}).\ L'_k = L_k \ \wedge\ M' = M \ \wedge\ \forall Sm \in (\mathcal{S}e \setminus Se).\ Q'(Sm) = Q(Sm)\ ]$$

(Le processus $Prs_i$ exécute la commande $\underline{v}(Se)$ alors que le processus $Prs_j$ est en tête de la file d'attente sur ce sémaphore en attente d'exécution de $\sharp(Se)$. De manière atomique le processus $Prs_j$ quitte la file d'attente et les processus $Prs_i$ et $Prs_j$ franchissent simultanément les commandes respectives $\underline{v}(Se)$ et $\sharp(Se)$).

### 2.8.5.2.5 Traces

Dans une implantation de programmes parallèles avec sémaphores sur monoprocesseur, le contrôleur ("scheduler") synchronisant les opérations sur les sémaphores a la main régulièrement au moyen d'interruptions d'horloge, quand aucune opération sur les sémaphores n'est exécutée.

De manière plus abstraite, cette hypothèse d'équité faible se traduit par le fait que si le contrôle est devant une commande $\underline{\sharp}(se)$ (auquel cas l'une des deux actions $\underline{w}(se,i)$ ou $\underline{\sharp}(se,i)$ est toujours exécutable) alors fatalement le processus correspondant sera mis dans la file d'attente (par exécution de l'action $\underline{w}(se,i)$) ou franchira la commande $\underline{\sharp}(se)$ (par exécution de l'action $\underline{\sharp}(se,i)$). De même, si le contrôle d'un processus est devant une commande $\underline{v}(se)$ (auquel cas l'une des actions $\underline{v}(se,i)$ ou $\underline{v\sharp}(se,i,j)$, $i \neq j$ est toujours exécutable) alors fatalement le processus franchira la commande $\underline{v}(se)$ (par exécution de l'une des actions $\underline{v}(se,i)$ ou $\underline{v\sharp}(se,i,j)$).

Plus formellement, nous définissons l'ensemble réduit d'actions

$$Ar[\![Pps]\!] = \{\sharp, \sharp'\} \cup m \cup \{\underline{\sharp}(se,i), \underline{v}(se,i) : se \in \mathcal{Se} \wedge i \in m\}$$

et la correspondance $fa \in (A[\![Pps]\!] \to Ar[\![Pps]\!])$ par cas :

$fa(\underline{w}(se,i)) = fa(\underline{\sharp}(se,i)) = \underline{\sharp}(se,i)$ si $se \in \mathcal{Se}$, $i \in m$

$fa(\underline{v}(se,i)) = fa(\underline{v\sharp}(se,i,j)) = \underline{v}(se,i)$ si $se \in \mathcal{Se}$, $i,j \in m$

$fa(x) = x$ si $x$ est $\sharp, \sharp'$ ou $i \in m$

Les traces $p$ de $\Sigma[\![Pps]\!]$ sont les traces engendrées par le système de transition associé à Pps (et appartenant donc à $\Sigma<S[\![Pps]\!], A[\![Pps]\!], t[\![Pps]\!], \varepsilon[\![Pps]\!]>$) dont, l'image $\simeq<fa>(p)$, après avoir renommé les actions par $fa$, est faiblement équitable pour $Ar[\![Pps]\!]$ :

$$\Sigma[\![Pps]\!] = \{p \in \Sigma<S[\![Pps]\!], A[\![Pps]\!], t[\![Pps]\!], \varepsilon[\![Pps]\!]> : \simeq<fa>(p) \in \underset{\sim}{Wfair}<Ar[\![Pps]\!]>(\simeq<fa>(\Sigma<S[\![Pps]\!], A[\![Pps]\!], t[\![Pps]\!], \varepsilon[\![Pps]\!]>))\}$$

### 2.8.5.2.6 Propriétés de la sémantique des sémaphores

Pour nous convaincre que cette définition de la sémantique des sémaphores correspond bien à l'idée intuitive que nous en avons, nous démontrons des propriétés classiques d'invariance et d'équité des sémaphores. Ces propriétés sont fréquemment utilisées mais tenues pour vraies sans

démonstration ou en utilisant des arguments informels.

Nous commençons par démontrer des lemmes préliminaires :

Le premier lemme exprime que si la valeur initiale du sémaphore est positive ou nulle alors il est toujours vrai en cours d'exécution du programme que si la valeur du sémaphore est positive ou nulle alors la file est vide et que si la valeur du sémaphore est nulle alors la file contient (au plus) n processus deux à deux différents :

Lemme 2.8.5.2.6 v1

$$\forall p \in \Sigma[\![Pps]\!], j \in |p|, L \in C[\![Pps]\!], M \in \mathcal{M}, Q \in (Se \to n^{<\omega}), se \in Se.$$
$$[p_j = \langle L, M, Q\rangle \wedge \underline{Isem}(se) \geq 0]$$
$$\Rightarrow [(|Q(se)| = 0 \wedge M(se) \geq 0) \vee (0 \leq |Q(se)| \leq n \wedge M(se) = 0 \wedge \forall k, k' \in |Q(se)|.(k \neq k' \Rightarrow Q(se)_k \neq Q(se)_{k'}))]$$

Démonstration

Par induction sur $j \in |p|$. Si $j=0$ alors $\in[\![Pps]\!](p_0)$, $p_0 = \langle L, M, Q\rangle$ et $\underline{Isem}(se) \geq 0$ entraînent $Q(se) = \langle\rangle$ donc $|Q(se)| = 0$ et $M(se) = \underline{Isem}(se) \geq 0$. Si la propriété est vraie pour $p_j = \langle L, M, Q\rangle$ et $j+1 \in |p|$, démontrons que par définition de $t[\![Pps]\!]_{p_j}$, la propriété reste vraie pour $p_{j+1} = \langle L', M', Q'\rangle$. Si l'action $p_j$ est $\underline{p}_m$, $\underline{p}'_m$ ou $i \in m$ c'est évident car $M'(se) = M(se)$ et $Q'(se) = Q(se)$. Si $p_j = \underline{w}(se, i)$ alors nous avons $M(se) \leq 0$ et $\forall \ell \in |Q(se)|. Q(se)_\ell \neq i$ et donc par hypothèse d'induction $0 \leq |Q(se)| < n$ et $M(se) = 0$ et les éléments de $Q(se)$ sont deux à deux différents. Comme $Q'(se) = Q(se)^\frown\langle i\rangle$ et $M'(se) = M(se)$, nous en déduisons que $1 \leq Q'(se) \leq n$, $M'(se) = 0$ et les éléments de $Q'(se)$ sont deux à deux différents. Si $p_j = \underline{p}(se, i)$ alors $M(se) > 0$ et donc par hypothèse d'induction $|Q(se)| = 0$. Nous avons également $M'(se) = M(se) - 1$ et $Q'(se) = Q(se)$ donc nous en déduisons $M'(se) \geq 0$ et $|Q'(se)| = 0$. Si $p_j = \underline{v}(se, i)$ nous avons $|Q(se)| = 0$, $M'(se) = M(se) + 1$ et $Q'(se) = Q(se)$ et donc $|Q'(se)| = 0$ et $M'(se) > 0$. Si $p_j = \underline{vp}(se, i, k)$ alors $|Q(se)| > 0$ et donc $M(se) = 0$. Comme $M'(se) = M(se)$ et $0 \leq |Q'(se)| < |Q(se)|$, nous en déduisons $0 \leq |Q'(se)| < n$, $M'(se) = 0$ et les éléments de $Q'(se)$ sont

deux à deux différents puisque ceux de $\varphi(se)$ le sont par hypothèse d'induction.
□

Le lemme suivant montre que si un processus est dans la file d'attente d'un sémaphore $se$ alors son point de contrôle est devant une commande $\sharp(se)$:

Lemme 2.8.5.2.6 v2

$\forall Pps \in \mathcal{P}ps,\ m \in \omega,\ p \in \Sigma[\![Pps]\!],\ \langle L_0,\dots,L_{m-1}\rangle \in \mathcal{L}^m,\ M \in \mathcal{M},\ \varphi \in (\mathcal{S}e \to m^{<\omega}),\ se \in \mathcal{S}e.$
$m \in m,\ j \in |p|.$

$$\Big[Pps \equiv Ps[\![Pre_0 \| \dots \| Pre_{m-1}]\!]; Ps' \ \wedge\ \sharp_j = \ll L_0,\dots,L_{m-1}\rangle, M, \varphi\rangle \ \wedge\ \exists i \in |\varphi(se)|.\ \varphi(se)_i = m\Big]$$
$$\Rightarrow$$
$$\Big[Pre_m \equiv \alpha L_m : \sharp(se); \beta\Big]$$

Démonstration

Par induction sur $j \in |p|$. Le lemme est vrai pour $j=0$ car $\in[\![Pps]\!](p_0)$ implique $\varphi(se)=0$. Si le lemme est vrai pour $\sharp_j = \ll L_0,\dots,L_{m-1}\rangle, M, \varphi\rangle$ et $j+1 \in |p|$ alors il reste vrai pour $\sharp_{j+1} = \ll L'_0,\dots,L'_{m-1}\rangle, M', \varphi'\rangle$.

- Si $\forall l \in |\varphi(se)|.\ \varphi(se)_l \neq m$ et $\exists k \in |\varphi'(se)|.\ \varphi'(se)_k = m$ alors par définition de $t[\![Pps]\!]$, nous avons $\sharp_j = \underline{\omega}(se,m)$ et $Pre_m \equiv \alpha L_m : \sharp(se); \beta$
- Si $\exists l \in |\varphi(se)|.\ \varphi(se)_l = m$ alors par hypothèse d'induction, nous avons $Pre_m \equiv \alpha L_m : \sharp(se); \beta$. Par définition de $t[\![Pps]\!]$, $\sharp_j$ n'est pas $\sharp$, $\sharp'$, $\underline{\omega}(se,m)$, $\sharp(se,m)$ (car nous aurions $M(se)>0$ et d'après le lemme 2.8.5.2.6 v1, ceci impliquerait $\varphi(se)=\langle\rangle$), $\underline{\nu}(se,m)$ ou $\underline{\nu\sharp}(se,m,i')$ (car nous aurions $Pre_m \equiv \alpha' L_m : \underline{\nu}(se); \beta'$ contraire à l'hypothèse que $L_m$ n'apparaît qu'une fois dans $Pre_m$). Si $\sharp_j$ est $i \in m$, $\underline{\omega}(sm,i)$, $\sharp(sm,i)$, $\underline{\nu}(sm,i)$, $\underline{\nu\sharp}(sm,i,i')$ avec $sm \in (\mathcal{S}e \vee se)$, nous avons $i \neq m$ et $i' \neq m$ et donc $L'_m = L_m$ par définition de $t[\![Pps]\!]$. Appliquant l'hypothèse d'induction, nous en déduisons que $Pre_m = \alpha L'_m : \sharp(se); \beta$. Enfin si $\sharp_j = \underline{\nu\sharp}(se,i,m)$, par définition de $t[\![Pps]\!]$ nous avons $\varphi(se) = m\,\varphi'(se)$ et donc d'après 2.8.5.2.6 v1, $\forall k \in |\varphi'(se)|.\ \varphi'(se)_k \neq m$, ce qui implique le lemme de façon triviale.
□

Le lemme suivant exprime que le nombre d'exécutions d'une commande $\sharp(se)$ conduisant à la mise en attente du processus qui l'exécute est égal au nombre d'exécutions d'une commande $\underline{v}(se)$ conduisant au réveil d'un processus en attente plus le nombre de processus en attente dans la file d'attente du sémaphore $se$ :

Etant donnée une sémantique $\langle S, A, \Sigma\rangle$, nous définissons

$\sigma \in (\wp^A \to ((\Sigma \times \omega) \to \omega))$

$\sigma\alpha(p,j) = \sum_{k \in (j \cap |p|)} (\sharp_k \in \alpha \to 1 | 0)$ (avec $\Sigma\, \phi = 0$)

de sorte que $\sigma\alpha(p,j)$ est le nombre de fois qu'une action appartenant à $\alpha$ est exécutée entre $p_0$ et $p_j$.

Lemme 3.8.5.2.6 N 3

$p \in \Sigma[\![Pps]\!]$, $j \in |p|$, $L \in C[\![Pps]\!]$, $M \in \mathcal{M}$, $Q \in (\mathcal{S}e \to m^{<\omega})$, $se \in \mathcal{S}e$.

$[p_j = \langle L, M, Q\rangle] \Rightarrow [\sigma\{\underline{w}(se,i) : i \in m\}(p,j) = \sigma\{\underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j) + |Q(se)|]$

Démonstration

Par induction sur $j \in |p|$. Si $j=0$, nous avons $\in [\![Pps]\!](p_j)$ et donc $|Q(se)| = 0$ de sorte que $\sigma\{\underline{w}(se,i) : i \in m\}(p,j) = \sigma\{\underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j) = |Q(se)| = 0$. Si par hypothèse d'induction la propriété est vraie pour $p_j = \langle L, M, Q\rangle$, $j+1 \in |p|$ et $p_{j+1} = \langle L', M', Q'\rangle$ alors si $\sharp_j$ est $\sharp_i$, $\sharp'_i$, $i \in m$, $\underline{w}(sm,i)$, $\underline{\sharp}(sm,i)$, $\underline{v}(sm,i)$, $\underline{v\sharp}(sm,i,k)$ pour $sm \in (\mathcal{S}e \vee se)$, $\underline{\sharp}(se,i)$ ou $\underline{v}(se,i)$, nous avons $\sigma\{\underline{w}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{w}(se,i) : i \in m\}(p,j)$, $\sigma\{\underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j+1) = \sigma\{\underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j)$ et $Q' = Q$. Si $\sharp_j = \underline{w}(se,k)$ alors $\sigma\{\underline{w}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{w}(se,i) : i \in m\}(p,j) + 1$, $\sigma\{\underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j+1) = \sigma\{\underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j)$ et $|Q'(se)| = |Q(se)| + 1$. Si $\sharp_j = \underline{v\sharp}(se,k,k')$ alors $\sigma\{\underline{w}(se,i) : i \in m\}(p,j) = \sigma\{\underline{w}(se,i) : i \in m\}(p,j+1)$, $\sigma\{\underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j+1) = \sigma\{\underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j) + 1$ et $|Q'(se)| = |Q(se)| - 1$.

□

Le lemme suivant exprime que le nombre d'exécutions d'une commande $\underline{\mathfrak{p}}(se)$ sans attente du processus qui l'exécute est égal au nombre d'exécutions de la commande $\underline{v}(se)$ alors qu'aucun processus n'était en attente dans la file de se plus la valeur initiale moins la valeur courante du sémaphore se :

Lemme 3.8.5.2.6 ~4

$$\forall p \in \Sigma[\![Pps]\!],\ j \in |p|,\ L \in C[\![Pps]\!],\ M \in \mathcal{M},\ Q \in (\mathcal{S}e \to m^{<\omega}),\ se \in \mathcal{S}e.$$
$$[p_j = \langle L, M, Q\rangle]$$
$$\Rightarrow [\sigma\{\underline{\mathfrak{p}}(se,i) : i \in m\}(p,j) = \sigma\{\underline{v}(se,i) : i \in m\}(p,j) + I_{se \in m}(se) - M(se)]$$

Démonstration

Par induction sur $j \in |p|$. Pour $j=0$, nous avons $\sigma\{\underline{\mathfrak{p}}(se,i) : i \in m\}(p,0) = \sigma\{\underline{v}(se,i) : i \in m\}(p,j) = 0$ et $\varepsilon[\![Pps]\!](p_0)$ entraîne que $M(se) = I_{se \in m}(se)$. Si la propriété est vraie pour $p_j = \langle L, M, Q\rangle$, $j+1 \in |p|$ et $p_{j+1} = \langle L', M', Q'\rangle$ alors si $p_j$ est $\underline{\mathfrak{p}}$, $\underline{\mathfrak{p}}'$, $i \in m$, $\underline{\omega}(sm,i)$, $\underline{\mathfrak{p}}(sm,i)$, $\underline{v}(sm,i)$, $\underline{\mathfrak{p}}\mathfrak{p}(sm,i,i')$ avec $sm \neq se$, $\underline{\omega}(se,i)$ ou $\underline{\mathfrak{p}}\mathfrak{p}(se,i,i')$ alors $M' = M$ et $\sigma\{\underline{\mathfrak{p}}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{\mathfrak{p}}(se,i) : i \in m\}(p,j)$ et $\sigma\{\underline{v}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{v}(se,i) : i \in m\}(p,j)$. Si $p_j$ est $\underline{\mathfrak{p}}(se,i)$ alors $\sigma\{\underline{\mathfrak{p}}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{\mathfrak{p}}(se,i) : i \in m\}(p,j) + 1$, $\sigma\{\underline{v}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{v}(se,i) : i \in m\}(p,j)$ et $M'(se) = M(se) - 1$. Si $p_j$ est $\underline{v}(se,i)$ alors $\sigma\{\underline{\mathfrak{p}}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{\mathfrak{p}}(se,i) : i \in m\}(p,j)$, $\sigma\{\underline{v}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{v}(se,i) : i \in m\}(p,j) + 1$ et $M'(se) = M(se) + 1$ et dans les deux cas, nous déduisons de l'hypothèse d'induction que $\sigma\{p(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{v}(se,i) : i \in m\}(p,j+1) + I_{se \in m}(se) - M'(se)$.

□

Nous en déduisons le théorème d'Habermann [72] qui s'énonce informellement comme suit :

"Si la valeur initiale du sémaphore se est positive ou nulle alors le nombre de fois où la commande $\underline{\mathfrak{p}}(se)$ a été franchie est égal au minimum du nombre de fois où la commande $\underline{\mathfrak{p}}(se)$ a été exécutée et du nombre de fois où la commande $\underline{v}(se)$ a été exécutée plus la valeur initiale du sémaphore".

Théorème 2.8.5.2.6∨5

$\forall p \in \Sigma[\![ppss]\!],\ j \in |p|,\ se \in \mathcal{Se}.$

$\underset{\sim}{I}_{sem}(se) \geqslant 0$

$\Longrightarrow$

$$\sigma\{\underset{\sim}{\sharp}(se,i), \underset{\sim}{\forall\sharp}(se,i,i') : i,i' \in m\}(p,j) = \cap\{\sigma\{\underset{\sim}{\omega}(se,i), \underset{\sim}{\sharp}(se,i) : i \in m\}(p,j),\ \sigma\{\underset{\sim}{\nu}(se,i), \underset{\sim}{\forall\sharp}(se,i) : i,i' \in m\}(p,j) + \underset{\sim}{I}_{sem}(se)\}$$

Démonstration

Posons $p_j = \langle L,M,Q\rangle$. D'après le lemme 2.8.5.2.6∨1 deux cas sont à considérer :

- Si $|Q(se)|=0$ alors $M(se) \geqslant 0$ et donc 2.8.5.2.6∨3 entraîne que $\sigma\{\underset{\sim}{\sharp}(se,i), \underset{\sim}{\forall\sharp}(se,i,i') : i,i' \in m\}(p,j) = \sigma\{\underset{\sim}{\sharp}(se,i) : i \in m\}(p,j) + \sigma\{\underset{\sim}{\forall\sharp}(se,i,i') : i,i' \in m\}(p,j) = \sigma\{\underset{\sim}{\sharp}(se,i) : i \in m\}(p,j) + \sigma\{\underset{\sim}{\omega}(se,i) : i \in m\}(p,j) = \sigma\{\underset{\sim}{\sharp}(se,i), \underset{\sim}{\omega}(se,i) : i \in m\}(p,j)$. D'autre part 2.8.5.2.6∨4 entraîne que $\sigma\{\underset{\sim}{\sharp}(se,i), \underset{\sim}{\forall\sharp}(se,i,i') : i,i' \in m\}(p,j) \leqslant \sigma\{\underset{\sim}{\nu}(se,i), \underset{\sim}{\forall\sharp}(se,i,i') : i,i' \in m\}(p,j) + \underset{\sim}{I}_{sem}(se)$ et donc $\sigma\{\underset{\sim}{\sharp}(se,i), \underset{\sim}{\forall\sharp}(se,i,i') : i,i' \in m\}(p,j)$ est égal à l'infimum de ces deux quantités.
- Si $|Q(se)|>0$ alors $M(se)=0$ et donc 2.8.5.2.6∨3 entraîne que $\sigma\{\underset{\sim}{\sharp}(se,i), \underset{\sim}{\forall\sharp}(se,i,i') : i,i' \in m\}(p,j) < \sigma\{\underset{\sim}{\sharp}(se,i), \underset{\sim}{\omega}(se,i) : i \in m\}(p,j)$ tandisque 2.8.5.2.6∨4 entraîne que $\sigma\{\underset{\sim}{\sharp}(se,i), \underset{\sim}{\forall\sharp}(se,i,i') : i,i' \in m\}(p,j) = \sigma\{\underset{\sim}{\nu}(se,i), \underset{\sim}{\forall\sharp}(se,i,i') : i,i' \in m\}(p,j) + \underset{\sim}{I}_{sem}(se)$ et donc $\sigma\{\underset{\sim}{\sharp}(se,i), \underset{\sim}{\forall\sharp}(se,i,i') : i,i' \in m\}(p,j)$ est égal à l'infimum de ces deux quantités.

□

La première des propriétés d'équité des sémaphores est qu'aucun processus ne peut être bloqué devant une commande $\underset{\sim}{\nu}(se)$. En termes d'actions et pour des traces infinies, nous avons que si une des actions $\underset{\sim}{\nu}(se,m)$ ou $\underset{\sim}{\forall\sharp}(se,m,i)$, $i \in (m \vee m)$ est exécutable alors une de ces actions sera fatalement exécutée.

Théorème 2.8.5.2.6 v 6

$$\forall p \in \Sigma[\![Pps]\!],\ Se \in Se^*,\ m \in m,\ i \in |p| \cdot$$
$$\left[ |p| = \omega \ \wedge\ \exists \Delta \in S[\![Pps]\!] \cdot \left( t[\![Pps]\!]_{\underline{v}(se,m)}(p_i, \Delta) \ \vee\ \exists m' \in m \cdot t[\![Pps]\!]_{\underline{vp}(se,m,m')}(p_i, \Delta) \right) \right]$$
$$\rightarrow \left[ \exists j \geq i \cdot \left( p_j = \underline{v}(se,m) \ \vee\ \exists m' \in m \cdot p_j = \underline{vp}(se,m,m') \right) \right]$$

Démonstration

Si $\underline{v}(se,m)$ ou $\underline{vp}(se,m,m')$ est exécutable en $p_j$ alors par définition de $t[\![Pps]\!]$, nous avons $Pros_m \equiv \alpha L_m : \underline{v}(se) ; L'_m : \beta$. Si $\underline{v}(se,m)$ et $\underline{vp}(se,m,m')$, $m' \in (m \vee m)$ ne sont jamais exécutées alors le contrôle de $Pros_m$ reste en $L_m$ pour tout $p_j$, $j \geq i$ (car par définition de $t[\![Pps]\!]$, seule l'exécution de l'une de ces actions peut le déplacer (en $L'_m$)). Posant $p_j = \langle\langle L_0, \ldots, L_{n-1}\rangle, M, Q\rangle$, nous observons que deux cas sont possibles :

Si $Q(se) = \langle\rangle$ alors l'action $\underline{v}(se,m)$ est exécutable,

Si $Q(se) = k\,Q'(se)$ alors d'après le lemme 2.8.5.2.6 v 2, nous avons $Pros_k \equiv \alpha' L_k : \underline{p}(se) ; \beta'$ et donc d'après la syntaxe $\alpha' L_k : \underline{p}(se) ; L'_k : \beta''$ de sorte que l'action $\underline{vp}(se,m,k)$ est exécutable.

Nous avons démontré que l'une des actions $\underline{v}(se,m)$ ou $\underline{vp}(se,m,m')$, $m' \in (m \vee m)$ est toujours exécutable en $p_j$ pour $j \geq i$. D'après l'hypothèse d'équité faible énoncé en 2.8.5.2.5 il n'est pas possible que les deux actions ne soient jamais exécutées au delà de $p_i$.

□

D'autre part, si la commande $\underline{p}(se)$ est exécutable infiniment souvent alors elle sera franchie. Cette propriété s'énonce plus précisément en termes d'actions et pour des traces infinies comme suit :

Si infiniment souvent une des actions $\underline{w}(se,m)$, $\underline{p}(se,m)$ ou $\underline{vp}(se,i,m)$ $i \in (m \vee m)$ est exécutable alors une des actions $\underline{p}(se,m)$ ou $\underline{vp}(se,i,m)$, $i \in (m \vee m)$ sera fatalement exécutée :

Théorème 2.8.5, 2.6 v 7

$$\forall p \in [\![Pps]\!],\ se \in \mathcal{S}e,\ m \in m,\ i \in |p|.$$
$$[\,|p| = \omega\,]$$
$$\Rightarrow [\,[\,\forall j \geq i.\ \exists k \geq j,\ s \in S[\![Pps]\!].\ (t[\![Pps]\!]_{\underline{w}(se,m)}(p_k, s) \vee t[\![Pps]\!]_{\underline{p}(se,m)}(p_k, s) \vee \exists m' \in m.\ t[\![Pps]\!]_{\underline{vp}(se,m',m)}(p_k, s))\,]$$
$$\Rightarrow [\,\exists j \geq i.\ (p_j = \underline{p}(se,m) \quad \vee \quad \exists m' \in m.\ p_j = \underline{vp}(se,m',m))\,]$$

Démonstration

Supposons $|p| = \omega$ et posons $p_k = \langle L^k, M^k, Q^k \rangle$ avec $L^k = \langle L^k_0, \ldots, L^k_{m-1} \rangle$ quand le contrôle est dans la commande parallèle de $Pps \equiv Ps[\![Pro_0 \| \ldots \| Pro_{m-1}]\!]; Ps'$. Supposons que infiniment souvent les actions $\underline{w}(se,m)$, $\underline{p}(se,m)$ ou $\underline{vp}(se,m',m)$ avec $m' \in m$ sont exécutables au delà de $p_i$.

- Si $\exists m' \in m.\ \underline{vp}(se,m',m)$ est exécutable (c'est-à-dire $\exists j \geq i, s' \in S[\![Pps]\!].\ t[\![Pps]\!]_{\underline{vp}(se,m',m)}(p_j, s')$) alors il faut (d'après la définition de $t[\![Pps]\!]$) que le processus $Pro_m$ soit en tête de la file d'attente du sémaphore se ($Q^i(se) = m\,Q'(se)$ où $s' = \langle L', M', Q' \rangle$). Si $\forall m'' \in m$, $\underline{vp}(se,m'',m)$ n'était jamais exécuté ultérieurement et que seule cette action permet à $Pro_m$ de sortir de la file de se, $Pro_m$ restera toujours en tête de la file de $Q^k(se)$, $k \geq j$. D'autre part le théorème 2.8.5, 2.6 v 6 montre que $\underline{vp}(se,m',m)$ étant exécutable en $p_j$, l'action $\underline{v}(se,m')$ ou $\underline{vp}(se,m',m'')$ sera fatalement exécutée en $p_k$, $k \geq j$. Pour que $\underline{v}(se,m')$ puisse être exécutée il faudrait par définition de $t[\![Pps]\!]$ que la file de se soit vide ($Q^k(se) = \langle\rangle$) contraire au fait que $Pro_m$ est en tête ($Q^k(se)_0 = m$). C'est donc $\underline{vp}(se,m',m'')$ qui est exécutée et donc par définition de $t[\![Pps]\!]_{\underline{vp}(se,m',m'')}$, $m''$ est en tête de la file de se ($Q^k(se)_0 = m''$) et donc $m'' = m$, ce qui montre que $\underline{vp}(se,m',m)$ est fatalement exécutée.

- Si $\forall m' \in m.\ \underline{vp}(se,m',m)$ n'est pas exécutable (c'est-à-dire $\forall j \geq i, s' \in S[\![Pps]\!].\ \neg t[\![Pps]\!]_{\underline{vp}(se,m',m)}(p_j, s')$) alors infiniment souvent l'une des actions $\underline{w}(se,m)$ ou $\underline{p}(se,m)$ est exécutable au delà de $p_i$. Il existe un $p_j$, $j \geq i$ tel que l'une des actions $\underline{w}(se,m)$ ou $\underline{p}(se,m)$ est exécutable. En ce point, par définition

de $t[\![Pps]\!]_{\underline{w}(se,m)}$ et $t[\![Pps]\!]_{\underline{p}(se,m)}$ le processus $Pro_m$ n'est pas dans la file du sémaphore se ou cette file est vide, $\forall l \in |\varphi^j(se)|.\ \varphi^j(se)_l \neq m$. Deux cas sont alors possibles :

. Si $Pro_m$ entre ultérieurement dans la file d'attente du sémaphore se, c'est-à-dire $\exists k > j,\ l \in |\varphi^k(se)|.\ \varphi^k(se)_l = m$; tant que m reste dans la file d'attente, les actions $\underline{w}(se,m)$ et $\underline{p}(se,m)$ ne sont pas exécutables et comme infiniment souvent l'une d'elle doit l'être, il faut que $Pro_m$ sorte de la file d'attente de se après y être entré. Par définition de $t[\![Pps]\!]$, seule une action $\underline{wp}(se,m',m)$ peut faire sortir m de la file de se et donc une action de ce type est fatalement exécutée.

. Si $Pro_m$ n'entre jamais dans la file d'attente du sémaphore se, c'est-à-dire $\forall k \geq j,\ l \in |\varphi^k(se)|.\ \varphi^k(se)_l \neq m$. Par définition de $t[\![Pps]\!]$, il n'est pas possible que les actions $\underline{w}(se,m)$ et $\underline{wp}(se,m',m)$, n'en soient exécutées car elles auraient pour effet de faire entrer ou sortir le processus $Pro_m$ de la file d'attente de se. Comme $\underline{w}(se,m)$ ou $\underline{p}(se,m)$ est exécutable en $p_j$, nous avons $Pro_m \equiv \alpha L^i_m : \underline{p}(se); \beta$. Pourque le contrôle de $Pro_m$ se déplace de $L^i_m$ au de delà de $p_j$, il faudrait qu'une action $\underline{wp}(se,m',m)$ ou $p(se,m)$ soit exécutée. La première alternative n'est pas possible, la seconde termine la preuve. Supposons que l'action $\underline{p}(se,m)$ ne soit jamais exécutée au dela de $p_j$. Nous observons alors que $\forall k \geq j.\ Pro_m \equiv \alpha L^k_m : \underline{p}(se); \beta$ avec $L^k_m = L^i_m$. Par conséquent, $\forall k \geq j$, si $M^k(se) \leq 0$ alors $\underline{w}(se,m)$ est exécutable tandis que si $M^k(se) > 0$ alors $\underline{p}(se,m)$ est exécutable. Au delà de $p_j$, l'une des deux actions $\underline{w}(se,m)$ ou $\underline{p}(se,m)$ est toujours exécutable et d'après la définition des traces $\Sigma[\![Pps]\!]$ donnée en 2.8.5.2.5, l'une de ces actions sera fatalement exécutée. Comme il ne peut s'agir de $\underline{w}(se,m)$ (car $Pro_m$ n'entre jamais dans la file d'attente de se), il s'agit de $\underline{p}(se,m)$.

□

Les recherches actuelles sur les preuves de propriétés de fatalité de programmes parallèles avec sémaphores comme Lehman, Pnueli-Stavi [81] sont basées sur des hypothèses d'équité des sémaphores similaires à celle donnée dans les théorèmes 2.8.5.2.6∿6 et 2.8.5.2.6∿7. Ces méthodes ne peuvent pas être complètes pour la raison que certaines traces satisfaisant ces hypothèses d'équité ne sont pas des traces de $\Sigma[\![Pps]\!]$. C'est le cas en particulier parce que les propriétés d'équité énoncées dans les théorèmes 2.8.5.2.6∿6 et 2.8.5.2.6∿7 n'interdisent pas que l'attente pour passer un sémaphore ne soit pas bornée.

Au contraire, pour la sémantique de Dijkstra [68], nous avons la propriété suivante, que nous démontrons :

"Si un processus $Pro_m$ est en attente devant une commande $\natural(se)$ en position $r$ dans la file d'attente du sémaphore $se$, alors $r+1$ commandes $\overset{\sim}{\nu}(se)$ et un nombre fini borné de commandes $\natural(se)$ s'exécutent avant que l'attente ne prenne fin."

Théorème 2.8.5.2.6∿8

$$\forall Pps \in \mathcal{P}ps,\ n\in\omega,\ p\in\Sigma[\![Pps]\!],\ Q\in(|p|\to(\mathcal{S}e\to n^{<\omega})),\ j,k\in|p|,\ se\in\mathcal{S}e,\ r\in\omega.$$

$$\Big[Pps \equiv Ps[\![Pro_0\|\ldots\|Pro_{n-1}]\!];Ps' \wedge j<k \wedge \exists m\in n.\forall l\in|p|.((j\le l\le k)\Rightarrow$$
$$(\exists <L^l_0,\ldots,L^l_{m-1}>\in\mathcal{L}^m,\ M^l\in(\mathcal{V}\to\mathcal{D}).\ p_l=\ll L^l_0,\ldots,L^l_{m-1}>,M^l,Q^l>\ \wedge$$
$$Pro_m\equiv\alpha L^l_m:\natural(se);L^k_m:\beta\ \wedge\ Q^l(se)_r=m\ ]$$

$$\Big[\ \sigma\{\overset{\sim}{\nu}(se,i):i\in m\}(p,k)-\sigma\{\overset{\sim}{\nu}(se,i):i\in m\}(p,j)=0$$
$$\wedge\ \sigma\{\overset{\sim}{\nu\natural}(se,i,i'):i,i'\in m\}(p,k)-\sigma\{\overset{\sim}{\nu\natural}(se,i,i'):i,i'\in m\}(p,j)=r+1$$
$$\wedge\ \sigma\{\natural(se,i):i\in m\}(p,k)-\sigma\{\natural(se,i):i\in m\}(p,j)=0$$
$$\wedge\ \sigma\{\underline{\omega}(se,i):i\in m\}(p,k)-\sigma\{\underline{\omega}(se,i):i\in m\}(p,j)\le n-|Q^j(se)|+r\ \Big]$$

Démonstration

Nous construisons $R \in (|p| \to \omega)$ tel que $R_j = r$ et $\forall l \in |p|.\ j \le l \le k$ nous ayons $\varphi^l(se)_{R_l} = m$, $\sigma\{\psi\!\!\!\psi(se,i,i') : i,i' \in m\}(p,l) - \sigma\{\psi\!\!\!\psi(se,i,i') : i,i' \in m\}(p,j) = R_j - R_l$, $\sigma\{\underline{\psi}(se,i) : i \in m\}(p,l) - \sigma\{\underline{\psi}(se,i) : i \in m\}(p,j) = 0$ et $\sigma\{\psi\!\!\!\psi(se,i) : i \in m\}(p,l) - \sigma\{\psi\!\!\!\psi(se,i) : i \in m\}(p,j) = 0$

Par hypothèse, nous avons $\varphi^j(se)_r = m$ de sorte que les propriétés ci-dessus sont vraies pour $j = l$ en posant $R_j = r$. Ayant construit $R_l$ pour $l < k$, nous définissons $R_{l+1}$ comme suit : si $\$_l = \psi\!\!\!\psi(se,i,i')$ alors par définition de $t[\![Pps]\!]_{\psi\!\!\!\psi(se,i,i')}$ nous avons $\varphi^l(se) = i'\varphi^{l+1}(se)$ avec $Pres_{i'} \equiv \alpha' L^l_{i'} : \psi\!\!\!\psi(se) ; L^{l+1}_{i'} : \beta'$. D'après l'hypothèse que les étiquettes ne figurent qu'une seule fois dans le programme Pps, nous avons $L^l_{i'} \neq L^{l+1}_{i'}$ et donc $m \neq i'$ car $L^l_m = L^{l+1}_m$. Posons $R_{l+1} = R_l - 1$ et donc $\varphi^{l+1}(se)_{R_{l+1}} = \varphi^l(se)_{R_{l+1}+1} = \varphi^l(se)_{R_l} = m$. De plus $\sigma\{\psi\!\!\!\psi(se,i,i') : i,i' \in m\}(p,l+1) = \sigma\{\psi\!\!\!\psi(se,i,i') : i,i' \in m\}(p,l) + 1$ et donc $\sigma\{\psi\!\!\!\psi(se,i,i') : i,i' \in m\}(p,l+1) - \sigma\{\psi\!\!\!\psi(se,i,i') : i,i' \in m\}(p,j) = R_j - R_{l+1} = R_j - R_{l+1}$. Nous ne pouvons pas avoir $\$_l = \underline{\psi}(se,i)$ (car il faudrait $\varphi^l(se) = \langle\rangle$) et donc $\sigma\{\underline{\psi}(se,i) : i \in m\}(p,l+1) = \sigma\{\underline{\psi}(se,i) : i \in m\}(p,l)$. De même, nous ne pouvons pas avoir $\$_l = \psi\!\!\!\psi(se,i)$ (car il faudrait $M^l(se) > 0$ et donc d'après le lemme 2.8.5.2.6 v 1, nous aurions $|\varphi(se)| = 0$ contraire à $\varphi^l(se)_{R_l} = m$) et donc $\sigma\{\psi\!\!\!\psi(se,i) : i \in m\}(p,l+1) = \sigma\{\psi\!\!\!\psi(se,i) : i \in m\}(p,l)$. Si $\$_l$ est $\underline{\omega}(se,i)$, $\underline{\omega}(sm,i)$, $\psi\!\!\!\psi(sm,i)$, $\underline{\psi}(sm,i)$, $\psi\!\!\!\psi(sm,i,i')$ avec $sm \in (Sel \vee se)$ nous avons $\varphi^{l+1}(se) = \varphi^l(se)$ par définition de $t[\![Pps]\!]$ et la propriété reste vraie en posant $R_{l+1} = R_l$.

Observons que pour $l = k-1$, nous avons $\$_{k-1} = \psi\!\!\!\psi(se,i,m)$. En effet par définition de $t[\![Pps]\!]$ et le fait que $Pres_m \equiv \alpha L^{k-1}_m : \psi\!\!\!\psi(se) ; L^k_m : \beta$ nous ne pouvons avoir que $\$_{k-1} \in \{\psi\!\!\!\psi(se,m), \psi\!\!\!\psi(se,i,m)\}$. Mais $\$_{k-1} = \psi\!\!\!\psi(se,m)$ est impossible car il faudrait $M^{k-1}(se) > 0$ et donc d'après 2.8.5.2.6 v 1 $\varphi^{k-1}(se) = \langle\rangle$ en contradiction avec $\varphi^{k-1}(se)_{R_{k-1}} = m$. Par définition de $t[\![Pps]\!]_{\psi\!\!\!\psi(se,i,m)}$ et le lemme 2.8.5.2.6 v 1 (qui implique que m ne peut pas figurer deux fois dans la file $\varphi^{k-1}(se)$), nous avons $R_{k-1} = 0$. Il vient donc $\sigma\{\psi\!\!\!\psi(se,i,i') : i,i' \in m\}(p,k) - \sigma\{\psi\!\!\!\psi(se,i,i') : i,i' \in m\}(p,j) = \sigma\{\psi\!\!\!\psi(se,i,i') : i,i' \in m\}(p,k-1) + 1 - \sigma\{\psi\!\!\!\psi(se,i,i') : i,i' \in m\}(p,j) = R_j - R_{k-1} + 1 = R_j + 1 = r+1$.

Observons qu'entre $p_j$ et $p_{k-1}$ (et donc $p_k$) les processus qui ont pu exécuter la commande $\sharp(se)$ sont ceux qui n'étaient pas dans la file d'attente $q^i(se)$ en $p_j$ ainsi que ceux qui étaient devant $Prs_m$ dans la file d'attente $q^i(se)$ et qui sont sortis de cette file avant $Prs_m$. Nous avons vu que l'exécution d'une telle commande ne peut correspondre qu'à l'action $\underline{\omega}(se,i)$ qui a pour effet de ranger le processus $Prs_i$ après $Prs_m$ dans la file d'attente de se. Comme $Prs_m$ ne sort de la file qu'en $p_k$ le processus ne peut rentrer dans la file qu'au plus une fois, nous en déduisons $\sigma\{\underline{\omega}(se,i) : i \in m\}(p,k) - \sigma\{\underline{\omega}(se,i) : i \in m\}(p,j) = m - |q^i(se)| + r$.

□

### 2.8.5.3 Sémantique libérale

Pour démontrer certaines propriétés des programmes parallèles avec sémaphores, nous pourrons quelquefois utiliser une sémantique libérale qui ne prend pas en compte la file d'attente et correspond à une attente active.

$$Sl[\![Pps]\!] = C[\![Pps]\!] \times \mathcal{M}$$

$$el[\![Pps]\!] = [\exists L \in \mathcal{L}, M \in \mathcal{M}.\ (s = \langle L, M\rangle \wedge Pps \equiv L:\alpha \wedge \forall se \in \mathcal{S}e.\ M(se) = \underline{I_{sem}}(se))]$$

$$tl[\![Pps]\!]_{\underline{\omega}(se,i)}(\langle\langle L_0,\dots,L_{m-1}\rangle, M\rangle, \langle\langle L'_0,\dots,L'_{m-1}\rangle, M'\rangle) =$$
$$[Pps \equiv Ps[\![Prs_0 \| \dots \| Prs_{m-1}]\!]; Ps' \wedge Prs_i \equiv \alpha L_i : \sharp(se); \beta \wedge M(se) \leq 0 \wedge$$
$$\forall k \in m.\ L'_k = L_k \wedge M' = M]$$

$$tl[\![Pps]\!]_{\sharp(se,i)}(\langle\langle L_0,\dots,L_{m-1}\rangle, M\rangle, \langle\langle L'_0,\dots,L'_{m-1}\rangle, M'\rangle) =$$
$$[Pps \equiv Ps[\![Prs_0 \| \dots \| Prs_{m-1}]\!]; Ps' \wedge Prs_i \equiv \alpha L_i : \sharp(se); L'_i : \beta \wedge M(se) > 0 \wedge$$
$$\forall k \in (m \setminus i).\ L'_k = L_k \wedge M'(se) = M(se) - 1 \wedge \forall v \in (\mathcal{V} \setminus Se).\ M'(v) = M(v)]$$

$$tl[\![Pps]\!]_{\underline{\nu}(se,i)}(\langle\langle L_0,\ldots,L_{m-1}\rangle,M\rangle,\langle\langle L'_0,\ldots,L'_{m-1}\rangle,M'\rangle) =$$
$$[Pps \equiv Ps[\![Pra_0\|\ldots\|Pra_{m-1}]\!];Ps' \wedge Pra_i \equiv \alpha L_i:\underline{\nu}(se);L'_i:\beta \wedge$$
$$\forall k\in(m\setminus i).L'_k=L_k \wedge M'(se)=M(se)+1 \wedge \forall v\in(\mathcal{V}\setminus Se).M'(v)=M(v)]$$

$$tl[\![Pps]\!]_{\underline{\underline{\nu}}(se,i,j)}(\langle\langle L_0,\ldots,L_{m-1}\rangle,M\rangle,\langle\langle L'_0,\ldots,L'_{m-1}\rangle,M'\rangle)$$
$$[Pps \equiv Ps[\![Pra_0\|\ldots\|Pra_{m-1}]\!];Ps' \wedge Pra_i \equiv \ :\underline{\nu}(se);L'_i:\beta \wedge$$
$$Pra_j \equiv \alpha' L_j:\underline{\underline{\nu}}(se);L'_j:\beta' \wedge \forall k\in m\setminus\{i,j\}.L'_k=L_k \wedge M'=M]$$

Cette sémantique étant utilisée pour raisonner sur des ensembles d'états accessibles, nous remarquons que l'action d'attente $\underline{\omega}(se,i)$ peut être supprimée puisqu'elle ne change pas l'état courant. Nous observons également que si l'exécution de l'action $\underline{\underline{\nu}}(se,i,j)$ conduit de l'état $s$ à l'état $s'$ alors il est également possible que les actions $\underline{\nu}(se,i)$ puis $\underline{\nu}(se,j)$ soient exécutées, ce qui conduirait également de $s$ à $s'$. Pour raisonner sur l'ensemble des états accessibles par la sémantique libérale, les actions $\underline{\omega}(se,i)$ et $\underline{\underline{\nu}}(se,i,j)$ peuvent donc être supprimées :

$$Al[\![Pps]\!] = \{\natural,\natural'\} \cup m \cup \{\underline{\nu}(se,i),\underline{\underline{\nu}}(se,i) : se\in\mathcal{S}e \wedge i\in m\}$$

Aucune propriété d'équité n'étant à prendre en compte dans cette sémantique libérale, nous définissons :

$$\Sigma l[\![Pps]\!] = \Sigma<Sl[\![Pps]\!],Al[\![Pps]\!],tl[\![Pps]\!],\varepsilon l[\![Pps]\!]>$$

La propriété évidente de cette sémantique est qu'à une fonction des états près, l'ensemble des états accessibles d'après la sémantique (exacte) est inclus dans l'ensemble des états accessibles d'après la sémantique libérale :

Théorème 2.8.5.3 ∨1

Si

$fl \in (S[\![Pps]\!] \to Sl[\![Pps]\!])$ est défini par $fl(\langle L, M, \varphi\rangle) = \langle L, M\rangle$

alors

$$\{\Delta \in Sl[\![Pps]\!] : \exists p \in \simeq\langle fl\rangle(\Sigma[\![Pps]\!]), i \in |p| \cdot p_i = \Delta\}$$
$$\subseteq \{\Delta \in Sl[\![Pps]\!] : \exists p \in \Sigma l[\![Pps]\!], i \in |p| \cdot p_i = \Delta\}$$

### 2.8.5.4 Exemple

Le programme parallèle suivant réalise une section critique :

```
Sem Se init 1;
0:
   [[ 10:
         while true do
      11:
            P(Se);
      12:
            V(Se);
      13:
         od;
      14:
   ||
      20:
         while true do
      21:
            P(Se);
      22:
            V(Se);
      23:
         od;
      24:
   ]];
1:
```

La sémantique exacte aussi bien que la sémantique libérale garantissent que 12 et 22 sont en exclusion mutuelle. La sémantique exacte garantit l'entrée en section critique mais pas la sémantique libérale.

## 2.9 REFERENCES

ABRAHAMSON K. [80], "Expressiveness and decidability of logics of process", Ph.D. Thesis, Univ. of Washington, Seattle, USA, (1980).

COUSOT P. [78], "Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes", Thèse d'Etat, USMG Grenoble, (1978).

COUSOT P. [79], "Analysis of the behavior of dynamic discrete systems", Rapport de Recherche n°161, IMAG, USMG Grenoble, (Jan. 1979).

COUSOT P. [81], "Semantic foundations of program analysis", dans "Program flow analysis, theory and applications", S.S. Muchnick & N.J Jones (Eds.), Prentice-Hall, (1981), 303-342.

COUSOT P., COUSOT R. [79], "A constructive characterization of the lattices of all retractions, preclosure, quasi-closure and closure operators on a complete lattice", Portugaliae Mathematica, Vol.38, Fasc. 1-2, (1979), 185-198.

DIJKSTRA E.W.D. [68], "Cooperating sequential processes", dans "Programming Languages", F. Genuys (Ed.), Academic Press, N.Y., (1968), 43-112.

EMERSON E.A. [81], "Alternative semantics for temporal logics", Research Report TR-182, Dept. of C. Sci., U. of Texas at Austin, (Oct. 1981), 16 p.

HABERMANN A.N. [72], "Synchronization of communicating processes", CACM 13, 3 (1972), 171-176.

HOARE C.A.R. [78], "Communicating sequential processes", CACM 21, 8 (1978), 666-677.

KELLER R.M. [76], "Formal verification of parallel programs", CACM 19, 7 (1976), 371-384.

LAMPORT L. [80], "Sometime" is sometimes "not never", 7th Annual ACM Symp. on Principles of Programming Languages, (1980), 174-185.

LEHMANN D., PNUELI A., STAVI J. [81], "Impartiality, justice and fairness: the ethics of concurrent termination", Proc. 8th Coll. on Automata, Languages and Programming, Lect. Notes in Comp. Sci. 115, Springer Verlag, (1981), 264-277.

MILNE R., STRACHEY C. [76], "A theory of programming language semantics", Chapman & Hall (Londres) & Wiley (New York), (1976).

PRATT V.R. [79], "Process logic", Proc. 6th ACM symp. on Principles of Programming Languages, (1979), 93-100.

# 3. PROPRIETES D'INVARIANCE ET DE FATALITE DES PROGRAMMES

## 3.1 SPECIFICATION DE PROGRAMMES

Une preuve de programme consiste à démontrer une relation entre une sémantique (définissant "ce que fait l'exécution du programme") et une spécification (définissant "ce que devrait faire l'exécution du programme").

Pour qu'une étude des méthodes de preuve ne dépende pas des techniques choisies pour spécifier les programmes il faut donner une définition abstraite et générale de la spécification de programmes. Nous proposons de définir la spécification d'un programme Pr comme étant une sémantique $\langle S, A, \Sigma \rangle$. De cette façon une preuve de programme consiste à démontrer qu'une relation est vraie entre deux sémantiques : la spécification $\langle S, A, \Sigma \rangle$ et la sémantique opérationnelle $\langle S[\![Pr]\!], A[\![Pr]\!], \Sigma[\![Pr]\!] \rangle$.

Exemple 3.1-1

Etant donnés des ensembles S d'états et A d'actions et une assertion $\phi \in (S \to \{tt, ff\})$ sur les états, l'ensemble des traces pour lesquelles $\phi$ est tout le temps vraie en cours d'exécution est $\Sigma = \{p \in \Sigma\langle S, A\rangle : \forall i \in |p|.\ \phi(i)\}$.

(1) La preuve de l'invariance de $\phi$ pour un programme Pr consiste à démontrer que $\langle S[\![Pr]\!], A[\![Pr]\!], \Sigma[\![Pr]\!] \rangle \subseteq \langle S, A, \Sigma \rangle$ c'est-à-dire essentiellement $\forall p \in \Sigma[\![Pr]\!], i \in |p|.\ \phi(p_i)$.

(2) La preuve de préservation de $\phi$ pour un programme Pr consiste à démontrer que $S[\![Pr]\!] \subseteq S$, $A[\![Pr]\!] \subseteq A$ et $\Sigma[\![Pr]\!] \cap \Sigma \neq \emptyset$ c'est-à-dire essentiellement $\exists p \in \Sigma[\![Pr]\!].\ \forall i \in |p|.\ \phi(p_i)$.

Etant donnés des ensembles S d'états et A d'actions et une assertion $\psi \in (S \to \{tt, ff\})$ sur les états, l'ensemble des traces pour lesquelles $\psi$ est inévitablement vraie en cours d'exécution est $\Sigma = \{p \in \Sigma\langle S,A\rangle : \exists i \in |p| . \psi(p_i)\}$.

(3) La preuve de fatalité de $\psi$ pour un programme Pr consiste à démontrer que $\langle S[\![Pr]\!], A[\![Pr]\!], \Sigma[\![Pr]\!]\rangle \subseteq \langle S,A,\Sigma\rangle$ c'est-à-dire essentiellement $\forall p \in \Sigma[\![Pr]\!] . \exists i \in |p| . \psi(p_i)$.

(4) La preuve de possibilité de $\psi$ pour un programme Pr consiste à démontrer que $S[\![Pr]\!] \subseteq S$, $A[\![Pr]\!] \subseteq A$ et $\Sigma[\![Pr]\!] \cap \Sigma \neq \emptyset$ c'est-à-dire essentiellement $\exists p \in \Sigma[\![Pr]\!], i \in |p| . \psi(p_i)$.

□

Les spécifications de programmes les plus souvent utilisées concernent les propriétés d'invariance et de fatalité dont nous donnons maintenant des exemples.

## 3.2 INVARIANCE

Nous donnons une définition de l'invariance conditionnelle qui généralise une notion introduite par Lamport[80]. Nous obtenons comme cas particulier la notion classique d'invariance dont la correction partielle est un cas particulier. Nous distinguons l'invariance relationnelle qui permet d'exprimer une relation entre un état initial et un état courant sur une trace et l'invariance assertionnelle qui permet d'exprimer une assertion sur l'état courant d'une trace de la sémantique du programme. Nous donnons les définitions et quelques exemples

### 3.2.1 INVARIANCE CONDITIONNELLE

Soient $\langle S, A, \Sigma\rangle$ une sémantique opérationnelle et $\phi, \psi \in (S \times S \to \{tt, ff\})$ des relations entre états. $\psi$ est invariante sous condition $\phi$ pour $\langle S, A, \Sigma\rangle$ si et seulement si

$$\forall p \in \Sigma,\ i \in |p|.\ [\forall j \in i.\ \phi(p_0, p_j)] \Rightarrow \psi(p_0, p_i)$$

Exemple

Considérons un programme parallèle asynchrone $Ppa \equiv Ps[\![Pra_0 \| Pra_1]\!]; Ps'$. L'assertion $\underline{Terminé}_i(s) = [\exists L_0, L_1 \in \mathcal{L}^\omega, M \in \mathcal{M}_b \cdot s = \langle\langle L_0, L_1\rangle, M\rangle \wedge Pra_i \equiv \alpha L_i :]$ exprime que l'exécution du processus $Pra_i$, $i = 0,1$ s'est terminée correctement. L'invariance de $\psi(s,s') = \neg \underline{Terminé}_1(s')$ sous condition $\phi(s,s') = \neg \underline{Terminé}_0(s')$ pour Ppa exprime que tant que l'exécution du processus $Pra_0$ n'est pas terminée correctement, l'exécution du processus $Pra_1$ ne peut pas se terminer.

□

Nous pouvons imaginer de très nombreuses variantes de cette définition comme celle-ci :

$\Psi \in (S \times S \to \{tt, ff\})$ est invariante sous condition $\phi \in (S \times S \to \{tt, ff\})$ à partir de $\varepsilon \in (S \to \{tt, ff\})$ si et seulement si

$\forall p \in \Sigma, j \in |\sharp|, i \in j.\ [\varepsilon(p_i) \wedge \forall l \in (j \dots i).\ \phi(p_l, p_{l+1})] \Rightarrow \Psi(p_j, p_{j+1})$

Exemple

Un certain nombre de propriétés du schéma producteur-consommateur suivant peuvent s'exprimer sous la forme ci-dessus :

```
0:
   [[ 10:
          while true do
      11:
              produire(PX);
      12:
              c! PX ;
      13:
          od;
      14:
   ||
      20:
          while true do
      21:
              c? CX;
      22:
              consommer(CX);
      23:
          od;
      24:
   ];
1:
```

Un message est produit dans l'état $\langle\langle c1, c2\rangle, M\rangle$ si $c1 = 11$ et l'état successeur est de la forme $\langle\langle 12, c2'\rangle, M'\rangle$ :

$\pi(\langle\langle c1, c2\rangle, M\rangle, \langle\langle c1', c2'\rangle, M'\rangle) = [c1 = 11 \wedge c1' = 12]$

Un message est consommé dans l'état $\langle\langle c1, c2\rangle, M\rangle$ si $c2 = 22$ et l'état successeur est de la forme $\langle\langle c1', 23\rangle, M'\rangle$ :

$\gamma(\langle\langle c1, c2\rangle, M\rangle, \langle\langle c1', c2'\rangle, M'\rangle) = [c2 = 22 \wedge c2' = 23]$

En choisissant $\epsilon(s) = tt$, $\phi = \neg\pi$ et $\psi = \neg\delta$ dans la formule ci-dessus, nous exprimons qu'à partir du moment où l'exécution du programme est commencée, il n'est pas possible de consommer tant qu'il n'y a pas eu production.

En choisissant $\epsilon(\langle\langle c_1, c_2\rangle, M\rangle) = [c_2 = \lambda_3]$, $\phi = \neg\pi$ et $\psi = \neg\delta$ dans la formule ci-dessus, nous exprimons qu'après une consommation il n'est pas possible de consommer à nouveau sans qu'il y ait eu production entre temps.

### 3.2.2 INVARIANCE RELATIONNELLE

Le cas particulier où $\phi(s,s') = tt$ correspond à l'invariance relationnelle. $\psi \in (S \times S \to \{tt, ff\})$ est *invariante* pour $\langle S, A, \Sigma\rangle$ si et seulement si

$$\forall p \in \Sigma,\ i \in |p|.\ \psi(p_0, p_i)$$

Exemple

La *correction partielle* est une propriété d'invariance relationnelle. Etant données une spécification $\Phi \in (\mathcal{M} \to \{tt, ff\})$ de l'état mémoire d'entrée et une spécification de sortie $\Psi \in (\mathcal{M} \times \mathcal{M} \to \{tt, ff\})$ liant l'état mémoire final à l'état mémoire initial, la correction partielle d'un programme $Pr \in \mathcal{P}r$ pour $\Phi, \Psi$ exprime que si l'exécution commence dans un état $M$ satisfaisant $\Phi$ et atteint un état de sortie $M'$ alors $\Psi$ lie $M$ et $M'$. Autrement dit

$$\psi(s,s') = \left([\exists L, L', M, M'.\ s = \langle L, M\rangle \wedge \Phi(M) \wedge s' = \langle L', M'\rangle \wedge Pr \equiv \alpha L' : ] \Rightarrow \Psi(M, M')\right)$$

c'est-à-dire qu'il est toujours vrai en cours d'exécution que si l'état initial satisfait $\Phi$ et l'état courant est un état final alors $\Psi$ lie l'état final à l'état initial.

□

### 3.2.3 INVARIANCE ASSERTIONNELLE

Le cas particulier de l'invariance conditionnelle où $\phi(s,s') = tt$ et $\psi(s,s') = \psi(s')$ correspond à l'invariance assertionnelle.

$\psi \in (S \to \{tt, ff\})$ est invariante pour $\langle S, A, \Sigma \rangle$ si et seulement si $\forall p \in \Sigma, i \in |p| . \psi(p_i)$

Exemple

L'exclusion mutuelle est une propriété d'invariance assertionnelle. Par exemple, nous exprimons que les deux processus du programme 2.8.5.4 ne peuvent jamais être simultanément en section critique par l'invariance de

$$\psi(s) = [\exists L_0, L_1, M \cdot s = \langle\langle L_0, L_1\rangle, M\rangle \wedge \neg(L_0 = 12 \wedge L_1 = 22)]$$

□

La non-terminaison, l'absence d'erreurs à l'exécution, l'absence d'interblocages globaux permanents sont d'autres exemples de propriétés de programmes qui peuvent s'exprimer par l'invariance assertionnelle.

## 3.3 FATALITE

Nous donnons la définition de la fatalité sous invariance, Gabbay-Pnueli-Shelah-Stavi[80], avec les cas particuliers de fatalité relationnelle et fatalité assertionnelle.

### 3.3.1 FATALITE SOUS INVARIANCE

Soient $\langle S, A, \Sigma \rangle$ une sémantique opérationnelle et $\phi, \psi \in (S \times S \to \{tt, ff\})$ des relations entre états.

$\psi \in (S \times S \to \{tt, ff\})$ est fatale sous invariance de $\phi$ pour $\langle S, A, \Sigma \rangle$ si et seulement si

$$\forall p \in \Sigma.\ \exists i \in |p|.\ [\forall j \in i.\ \phi(p_0, p_j) \wedge \psi(p_0, p_i)]$$

Exemple

Nous pouvons exprimer que la valeur d'une variable entière d'un programme reste strictement positive ou négative avant d'atteindre la valeur zéro par la fatalité de

$$\psi(s, s') = [\exists C', M'.\ s' = \langle C', M' \rangle \wedge M'(X) = 0]$$

sous invariance de

$$\phi(s, s') = [\exists C, C', M, M'.\ s = \langle C, M \rangle \wedge s' = \langle C', M' \rangle \wedge M(X) \times M'(X) \geq 0]$$

□

### 3.3.2 FATALITE RELATIONNELLE

La fatalité relationnelle correspond au cas particulier de la fatalité sous invariance avec $\phi(s, s') = tt$.

$\psi \in (S \times S \to \{tt, ff\})$ est fatale pour $\langle S, A, \Sigma \rangle$ si et seulement si

$$\forall p \in \Sigma.\ \exists i \in |p|.\ \psi(p_0, p_i)$$

Exemple

La correction totale est une propriété de fatalité relationnelle. Etant données une spécification d'entrée $\Phi \in (\mathcal{M} \to \{tt, ff\})$ décrivant l'état mémoire initial et une spécification de sortie $\Psi \in (\mathcal{M} \times \mathcal{M} \to \{tt, ff\})$ liant l'état mémoire final à l'état mémoire initial, la correction totale d'un programme $Pr \in \mathcal{P}r$ s'exprime par la fatalité de

$$\Psi(\Delta, \Delta') = ( [\exists L, M.\ \Delta = \langle L, M \rangle \wedge \Phi(M)] \Rightarrow [\exists L, M, L', M'.\ \Delta = \langle L, M \rangle \wedge \Delta' = \langle L', M' \rangle \wedge Pr \equiv \alpha L' : \wedge \Psi(M, M')] )$$

pour la sémantique $\langle S[\![Pr]\!], A[\![Pr]\!], \Sigma[\![Pr]\!] \rangle$.

□

### 3.3.3 FATALITE ASSERTIONNELLE

La fatalité assertionnelle correspond au cas particulier de la fatalité sous invariance avec $\phi(\Delta, \Delta') = tt$ et $\Psi(\Delta, \Delta') = \Psi(\Delta')$.

$\Psi \in (S \to \{tt, ff\})$ est fatale pour $\langle S, A, \Sigma \rangle$ si et seulement si

$$\forall p \in \Sigma.\ \exists i \in |p|.\ \Psi(p_i)$$

Exemple

L'absence de famine est un exemple de fatalité assertionnelle. Par exemple, nous exprimons que le premier processus du programme 2.8.5.4 rentre fatalement en section critique s'il en fait la demande par la fatalité de

$$\Psi(\Delta) = [\exists L_0, L_1, M.\ (\Delta = \langle\langle L_0, L_1 \rangle, M \rangle \wedge L_0 = 12)]$$

pour la sémantique

$$\langle S, A, \{ p \in \underset{\sim}{Suff}(\Sigma) : \exists L_0, L_1, M.(\Delta = \langle\langle L_0, L_1 \rangle, M \rangle \wedge L_0 = 11) \} \rangle$$

□

La terminaison, la garantie de réponse à un signal peuvent également s'exprimer par la fatalité assertionnelle.

The first part of Radhia Cousot's thesis is available at
https://pcousot.github.io/publications/RadhiaCousotTheseEsSciences.PDF

The second part of Radhia Cousot's thesis is available at
https://pcousot.github.io/publications/RadhiaCousotTheseEsSciences2.PDF

The third part of Radhia Cousot's thesis is available at
https://pcousot.github.io/publications/RadhiaCousotTheseEsSciences3.PDF