Une algèbre de processus : pour un calcul basé sur la déduction

Zineb Habbas

THESE

Présentée par

TALANTIKIT Zineb (épouse HABBAS)

Pour obtenir le grade de DOCTEUR

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 30 mars 1992)

Spécialité : INFORMATIQUE

==========

UNE ALGEBRE DE PROCESSUS

Pour Un Calcul Basé Sur La Déduction

==========

Date de soutenance : 25 septembre 1992

Composition du jury :

J. Sifakis	Président
J-M. Adamo	Rapporteur
B. Berthomieux	Rapporteur
Ph. Jorrand	Examinateur
Ph. Schnoebelen	Examinateur

Thèse préparée au sein du Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle (LIFIA)

Remerciements

Je tiens à remercier :

- Joseph SIFAKIS, Directeur de recherche au CNRS, de m'avoir fait l'honneur de présider mon jury de thèse.

- Jean Marc ADAMO, Professeur à l'Université Claude-Bernard de Lyon et Bernard BERTHOMIEU, chargé de recherche au CNRS, d'avoir accepté de juger ce travail en tant que rapporteurs, dans les délais courts qui leur étaient impartis. Je les remercie pour leurs remarques et leurs suggestions et leur témoigne toute ma reconnaissance.

Je voudrais également remercier tout particulièrement Philippe JORRAND, Directeur de Recherche au CNRS qui a dirigé cette thèse. Je lui suis reconnaissante d'avoir accepté de m'accueillir dans son laboratoire, de son entière disponibilité et de l'intérêt qu'il a tant manifesté tout au long de ce travail. Il n'a cessé de me soutenir et de m'encourager durant mes moments difficiles, combien nombreux. Qu'il trouve ici ma profonde gratitude.

Cette thèse doit certainement beaucoup à Philippe SCHNOEBELEN, chargé de recherche au CNRS (IMAG). Je le remercie profondément d'avoir accepté de diriger une bonne partie de ce travail. Je me souviendrai toujours de sa patience de sa compréhension et de ses capacités de me sortir rapidement de certaines situations délicates. Il m'a aussi fait l'honneur de faire partie de mon jury de thèse . Pour tout cela, je lui dois toute ma reconnaissance.

Mes remerciements s'adressent ausi à
- Juan-Vicente ECHAGUE, pour ses encouragements et ses critiques constructives qu'il n'a cessé d'apporter tout au long de ce travail. Toujours disponible et prêt à écouter, j'ai eu beaucoup de plaisir à travailler avec lui.

- Cyril AUTANT, Mounira BELMESK Zoubir BELMESK, Ferroudja CHERIEF, Wilfried PFISTER, Jean Michel HUFLEIN, François LAROUSSINIE, Françoise NAYROLES, Xavier PANDOLFI et Sophie PINCHINAT, anciens ou nouveaux membres de l'équipe Parallélisme du

laboratoire LIFIA avec lesquels il y a eu, de près ou de loin, des échanges. Je garderai un bon souvenir de l'ambiance qui règne au niveau de cette équipe.

- Je tiens, avant de finir, à exprimer ma profonde reconnaissance à Jacques MOSSIERE, Directeur de l'Ensimag, qui m'a permis de mener à terme ce travail.

- Je tiens à remercier ici globalement toute ma grande famille, ma soeur Chafica et mon amie Soraya qui m'ont tant soutenue, encouragée et souvent supportée.

- Enfin, je ne pourrai jamais assez exprimer mes remerciements à Mustapha, Hana et Sabrina . Sans leur patience, leurs encouragements et leur compréhension cette thèse n'aurait pas vu le jour. Je leur dédie cet humble travail.

Sommaire

2. L'Algèbre de Processus HAL : The Herbrand Agent Language

3 Un Modèle de Calcul de Processus avec Partage de Variables

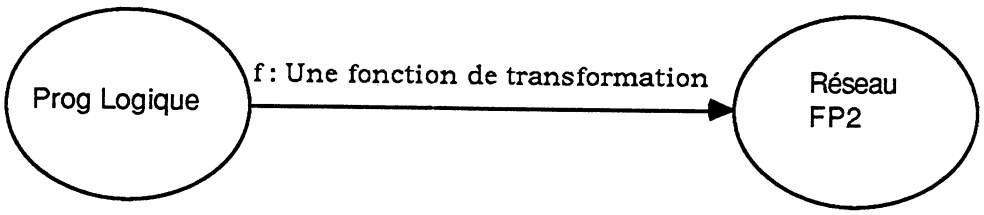
4 Logique Modale et Algèbre de Processus

Introduction

1 Bref historique

Un des sujets abordés au niveau de l'équipe parallélisme du Lifia est celui de la spécification de machines parallèles dans des domaines d'application tels que la programmation logique, la programmation fonctionnelle , la programmation équationnelle etc.
Certes, l'introduction du parallélisme dans ces domaines n'est pas un problème nouveau. Beaucoup de chercheurs s' y sont intéressés ces dernières années. La raison pour laquelle il y a eu une telle volonté de mener des recherches dans ces domaines est simple : le raisonnement logique et équationnel offre un nouveau style de programmation trés élégant et trés attrayant par ses fondements théoriques puissants. Les langages de ce type (Lisp, Prolog) ont eu un grand succés dans le domaine de la modélisation du raisonnement et de l'intelligence artificielle de façon plus générale. Par contre, les performances de ces langages sont plutôt décevantes.
Le grand espoir aujourd'hui, pour rendre intéressant ce nouveau style de programmation est d'y introduire le parallélisme. Le parallélisme peut être introduit à plusieurs niveaux (algorithmique, interprétation ...etc.) . Il existe en effet, aujourd'hui un certain nombre de versions parallèles des langages précités (parlog, parlisp) .
Mais dans le cadre de ce travail, nous nous intéressons à l'aspect spécification du parallélisme. Nous nous situons donc à un niveau abstrait et loin de tout problème d'implémentation de langages parallèles. Plus précisément, le problème posé est celui de spécifier des machines parallèles en

en utilisant comme outil de spécification le langage FP2 . FP2 : langage fonctionnel et parallèle est défini au niveau théorique comme une algèbre de processus. Il a été développé au Lifia [Jor 86, SJ 89] et possède aujourd'hui une sémantique formelle propre [Sch 90]. Un certain nombre de travaux développés autour de FP2 ont été conduits avec succès [Bel 90, Bel 92].
Dans le cadre de la spécification de machines à inférence parallèle, la première expérience avec ce problème fut menée par Ibanez dans [Iba 90]. Dans son travail, elle s'est intéressée à la spécification d'une machine Prolog parallèle à l'aide de FP2. Le problème posé peut être schématisé comme suit :



Etant donné un programme logique (ensemble de closes de Horn) , la spécification consiste en la définition de la fonction de transformation f où f(p) est un réseau de processus FP2. On dit plus communément dans la littérature que l'on construit un "mapping" d'un ensemble de programmes logiques vers un ensemble de réseaux de processus FP2. Cette transformation doit être correcte. La preuve de correction est possible car les deux langages considérés (langage logique d'une part et FP2 d'autre part) sont "proprement définis", ils ont tous deux des sémantiques formelles bien établies. Un programme logique est vu à l'issue de la transformation comme un objet FP2, c'est à dire un réseau de processus parallèles sur lequel on peut raisonner, prouver et vérifier des propriétés. Une éventuelle implémentation de ce réseau est tout à fait possible.
Mais au vu de cette expérience d'une part et du travail de Belmesk apparu dans [Bel 90] d'autre part, un certain nombre de difficultés, dûes essentiellement aux limitations de FP2, persistaient. En effet, suite à ces deux expériences précitées, il s'est avéré que pour spécifier de manière aisée et concise des machines à inférence, l'outil de spécification doit autoriser :

(1) La synchronisation n-aire
(2) La communication multi-directionnelle
(3) L'échange de termes ouverts
(4) La description de réseaux à topologie dynamique

Par exemple :
Les sous termes d'un terme sont réduits, les sous buts du corps d'une clause de Horn sont résolus en parallèle. Ceci implique qu'un ensemble de règles de réécriture ou un ensemble de clauses de Horn

est appliqué en parallèle. Lorsque l'on spécifie cela via des processus communicants, les processus chargés des opérations de réduction ou de déduction déclenchent de manière synchrone tous ceux qui représentent ces règles ou ces clauses.

Quand on résoud un but dans la logique des clauses de Horn, l'unificateur le plus général (mgu) des sous buts et tête de clause est calculé . Spécifier cela via des processus communicants implique une communication bidirectionnelle entre les processus contenant les sous buts et chaque processus contenant les têtes de clause unifiables correspondantes. En plus, quand les variables sont partagées par les sous buts, la communication devient multi-directionnelle entre les processus contenant les sous buts et ceux choisis pour les résoudre.

Les mgu (most general unifier) calculés lors d'une étape de déduction ne sont pas nécessairement fermés. Lorsque l'on spécifie cela via des processus communicants, des termes ouverts ou non clos peuvent alors être échangés entre processus.

Finalement, le nombre d'applications d'une règle de réécriture durant la réduction, le nombre d'instances d'une clause durant la déduction ne peuvent être connus statiquement. Ces instances doivent être créées dynamiquement en fonction du besoin. Si ces règles et clauses sont représentées par des processus alors la création dynamique de processus s'avére nécessaire.

Dans FP2, la synchronisation n - aire et la communication multi-directionelle sont possibles mais seuls les termes fermés peuvent être échangés entre processus . En outre, il n' y a pas de définitions récursives autorisant des descriptions de réseaux à topologie dynamique.
Outre FP2, parmi les nombreuses algèbres de processus bien connues dans la littérature, telles que CCS [Mil 80, Mil 89], SCCS[Mil 83], LOTOS [Bri 88] et d'autres algèbres plus récemment développées telle que l'algèbre anonyme définie dans [Plé 86] aucune, à notre connaissance, ne permet de satisfaire toutes ces exigences, au moins de manière concise .

A partir de là , nous avons été fortement motivés pour définir un nouvel outil de spécification : l'algèbre de processus HAL (The Herbrand Agent Language) orientée calcul basé sur la déduction.

2 Contenu de la thèse

Dans cette thèse, nous nous intéressons à la définition théorique d'une nouvelle algèbre de processus. Nous visons donc la définition de sémantiques formelles, la proposition de méthode de preuve, de vérification ... etc.

Tout d'abord, nous avons étudié divers modèles de parallélisme existant dans la littérature [Pom 86, DN 87]. Nous présentons dans le chapitre1, un certain nombre de modèles bien connus dans la

littérature. Nous insistons plus particulièrement sur la bisimulation forte et sa caractérisation par HML, logique de Hennessy et Milner [HM 85] car tous deux seront largement explorés par la suite.

Dans le chapitre 2, nous suivons la démarche suivie par Milner dans [Mil 89], pour donner une sémantique à HAL avec communication close. Nous définissons une sémantique opérationnelle à la Plotkin [Plo 81] où la communication se fait exclusivement par envoi de messages. Nous considérons ensuite la bisimulation forte comme équivalence sur les objets HAL et montrons que celle ci est une congruence relativement aux opérateurs de cette algèbre.

La sémantique proposée dans le chapitre 2 ne répond pas encore à tous les objectifs fixés au départ. En effet la communication est jusque là close, c'est à dire seuls les termes fermés peuvent être échangés entre processus. Ceci reste un handicap certain quand on veut spécifier des machines à inférence parallèle. Nous avons alors été ammenés à étudier pour HAL, une autre sémantique où la communication se fait à la fois par envoi de messages et partage de variables. Nous avons proposé pour cela une sémantique opérationnelle. Ensuite nous avons défini une équivalence de bisimulation non close sur les processus HAL et étudié les propriétés de congruence de celle ci par rapport aux opérateurs de cette algèbre. Ce dernier travail est original, car à notre connaissance le parallélisme avec partage de variables est surtout utilisé au niveau implémentation. Il n'existe pas de modèles théoriques propres pour ce dernier type de parallélisme. Dans le cadre de notre travail, il s'est avéré très utile car il permet de traiter la variable au niveau de HAL comme la variable logique. Ce dernier résultat [BH 92 a, BH 92b] fait l'objet du chapitre 3 de cette thèse.

HAL est une algèbre de processus qui contient en plus du noyau d'opérateurs de base classique (tels le préfixage, la somme non déterministe, le renommage... etc.) certains opérateurs qui lui sont spécifiques tels l'opérateur de connexion et de composition parallèle. Pour mieux comprendre, au moins au niveau théorique, ces les opérateurs de HAL, il nous a semblé intéressant de construire un système de preuve modal compositionnel correct et complet pour un noyau de HAL.

Pour construire ce système, nous considérons comme logique modale, la logique HML(Logique de Hennessy et Milner) et procédons par étapes. Nous subdivisons HAL en trois sous ensembles L_0, L_1 et L_2 et construisons trois systèmes de preuve Sys_0, Sys_1 et Sys_2 corrects et complets. Nous consacrons le chapitre 4 de cette thèse pour la présentation de ces trois systèmes ainsi que leurs résultats de correction et de complétude. Ce résultat apparait dans [Hab 91].

Chapitre 1

Introduction Aux Modèles De Parallélisme

1.1 Introduction

La sémantique permet de mieux comprendre un programme, d'énoncer la correction d'un programme, de définir si deux programmes sont équivalents ou non .
Dans le domaine de la programmation séquentielle, la sémantique d'un programme est une fonction reliant ces deux états. Deux systèmes séquentiels sont considérés équivalents si à deux états de départs identiques correspondent deux états d'arrivée identiques. Cette façon de voir la sémantique n'est guère suffisante quand on considère des programmes ou des systèmes parallèles. Les programmes parallèles sont des objets bien plus complexes que les programmes séquentiels et de manière générale deux programmes parallèles peuvent fournir le même résultat final en ayant des comportements différents dans un même environnement. En fait, il ne faut pas voir un système parallèle comme un objet complètement isolé mais comme un objet en intéraction avec un environnement extérieur. De fait il est alors important de préciser ce que l'on entend par programmes parallèles équivalents, programmes parallèles corrects. La théorie du parallélisme a pour rôle de définir des modèles de parallélisme , c'est à dire de définir formellement cette notion d'équivalence ou de préordre entre systèmes parallèles. Naturellement il faut pour cela, définir avant tout des critères d'équivalence , autrement dit qu'entendons nous par être équivalents ?

Concrètement si le même formalisme est utilisé pour décrire ce qui est exigé d'un système parallèle (sa spécification) et comment il peut être réellement construit (son implémentation), il est alors possible d'utiliser des théories basées sur les équivalences ou les préordres pour dire qu'une implémentation est correcte par rapport à sa spécification. Si une méthode de développement par étapes est utilisée, alors il est très utile d'être capable de substituer des spécifications très grandes à des spécifications plus concises équivalentes. En général, nous devons être capables d'interchanger des sous systèmes qui ont des comportements équivalents, dans le sens où un sous système peut en remplacer un autre sans perturber le comportement du système tout entier.

Un système S_1 est dit équivalent à un système S_2, quand sous certains aspects ou sous certaines considérations les deux systèmes sont compatibles. Le genre d'équivalences ou de préordres engendrés dépend étroitement de la façon dont les systèmes sont utilisés. En fait, la façon dont un système est utilisé détermine certains aspects comportementaux qui doivent être pris en compte et ceux qui doivent être ignorés. Il est donc nécessaire de connaitre pour chaque équivalence les propriétés qu'elle préserve.

Il existe dans la littérature plusieurs équivalences pour des modèles devant être utilisés pour décrire et raisonner autour des systèmes concurrents et non déterministes. Ceci est dû à un grand nombre de propriétés importantes pour l'analyse de tels systèmes. Toutes les équivalences proposées sont souvent basées sur l'idée d'observation : deux systèmes sont équivalents quand aucune observation externe ne peut les distinguer. En fait, ce qui nous intéresse ce n'est pas la structure interne du système mais son comportement par rapport au monde extérieur, c'est à dire son effet sur l'environnement et sa réaction à partir de l'environnement. On peut grossièrement regrouper les équivalences définies en deux grandes classes : Les équivalences qui réduisent le parallélisme au non déterminisme séquentiel et celles qui traitent le parallélisme comme une notion primitive.

Les équivalences qui réduisent le parallélisme au non déterminisme séquentiel sont habituellement appelées équivalences "interleaving". Plusieurs notions d'équivalences basées sur cette approche ont été proposées. Les plus largement connues sont : l'équivalence de traces [Hoa 85], l'équivalence projective [Mil 80], la bisimulation [Par 81], [Mil 83], l'équivalence de failure [BHR 84], l'équivalence de ready [BKO 88]. Souvent leurs théories sont différentes et leur seule forme d'abstraction commune est reliée au non déterminisme et à l'utilisation de systèmes de transitions étiquetés ou de graphes comme modèle de base.

Par ailleurs, plusieurs chercheurs ont souligné le fait que la sémantique basée sur la notion d'interleaving est inadéquate quand on veut prendre en compte les comportements non séquentiels, et que les sémantiques qui considèrent le parallélisme comme une notion primitive sont mieux adaptées à cet effet.

Par conséquent, différentes équivalences distinguant le parallélisme du non déterminisme ont été proposées : la bisimulation distribuée [CH 87], la pomset bisimulation [BC 87]. Une variété d'équivalences a été discutée dans [vGG 89]. Pour les théories basées sur le vrai parallélisme les

réseaux de Pétri [Rei 85], et les structures d'évènements [Win 80, 87] constituent les modèles de base.

Comme ces équivalences sont nombreuses et variées, un rôle important de la théorie du parallélisme est donc de les classifier. Par exemple, toutes ces équivalences peuvent être partiellement ordonnées par une relation " identifie plus que" ou "identifie moins que". Cette classification donne naissance à un treillis complet de relations. Des résultats de classification peuvent être trouvés dans [DN 87], [Gla 90, a] pour le cas de l'interleaving" et dans [v GG 87] et [Gla 90 b] pour le cas du vrai parallélisme. Le résultat obtenu permet de comprendre plus finement ces différentes équivalences.

Pour donner une sémantique à un langage parallèle, il faut choisir une équivalence. Mais ce choix n'est pas complètement arbitraire. Il dépend de l'utilisateur du langage mais surtout du langage lui même. En effet, l'équivalence considérée doit être une congruence relativement à tous les opérateurs de base du langage, c'est à dire, si p et q sont équivalents alors ils restent équivalents dans n'importe quel contexte .

Classer toutes les équivalences, en définir d'autres, vérifier que ces équivalences sont des congruences...etc. sont des tâches importantes dans la sémantique du parallélisme. Comme introduction aux modèles de parallélisme, nous avons choisi dans ce chapitre de nous situer dans un cadre simple de l'interleaving. Dans ce contexte, nous rappelons les équivalences basées sur les systèmes de transitions. Nous insistons particulièrement sur la théorie de la bisimulation forte (c'est à dire celle qui considère l'action interne comme une action quelconque) et ses propriétés, car c'est celle que nous retenons pour le reste de notre travail. Nous ne développons en aucun cas les sémantiques basées sur le vrai parallélisme. Néanmoins nous introduirons la notion de "step" équivalence qui est une généralisation des équivalences interleaving. Nous terminerons en rappelant une caractéristique importante de la bisimulation qui est sa caractérisation par la logique HML (Logique de Hennesy Milner) [HM 85].

1.2 Systèmes de transition

Dans le domaine de la sémantique du parallélisme, il est apparu comme une technique standard de donner la sémantique aux langages en termes de systèmes de transition étiquetés [Kel 76] ou de graphes. Les systèmes de transition sont un modèle relationnel abstrait basé sur deux notions primitives : les états et les transitions. Etant donné n'importe quel autre modèle pour lequel il est possible de définir une notion d'état et d'action causant une transition d'état, il est possible de définir pour chaque objet de ce modèle un système de transition. Cette correspondance détermine une sémantique "interleaving" pour ce modèle et plusieurs propriétés de systèmes parallèles peuvent être étudiées purement en termes de systèmes de transitions. Nous allons considérer une

classe particulière de systèmes de transitions pour modéliser des systèmes parallèles qui peuvent être observés par un agent externe. De plus, ces systèmes sont capables d'exécuter des actions internes ou non observables.

Définition 1.1 (Système de transition étiqueté)
Un système de transition étiqueté est une structure $S = (Q, Act, \longrightarrow, q_0)$ où Q est un ensemble non vide d'états ou de configurations, Act un ensemble d'actions, $\longrightarrow \subseteq Q \times Act \times Q$ est la relation de transition et $q_0 \in Q$ un état particulier ou état initial du système.

Pour donner la sémantique opérationnelle de modèles de calculs parallèles, on associe généralement à Q l'ensemble des termes du calcul, appellé dans la terminologie courante du domaine de parallélisme, processus.
Dans cette définition chacune des transitions $\xrightarrow{a}$ où $a \in Act$ décrit l'effet de l'exécution de l'action élémentaire a. Si p et q appartiennent à Q alors $p \xrightarrow{a} q$ indique que dans l'état p, le système exécute une a - action et se développe en un état q. D'après Milner [Mil 80] le symbole τ peut être utilisé pour dénoter les actions internes et $p \xrightarrow{\tau} q$ indique que dans l'état p le système exécute une étape silencieuse ou invisible et se développe en un processus q.
Un système de transition peut évidemment être déplié en un arbre. L'état initial est la racine de l'arbre et la relation de transition est représentée par les arcs étiquetés par des éléments de Act ou par τ. Les autres noeuds de l'arbre représentent les états différents de l'état initial.

1.2.1 Quelques notations et définitions de base

$Act = \{ \mu, \mu_1, \mu_2, ...etc \}$ dénote l'ensemble des actions visibles.
$Act^* = \{ \sigma, \sigma', ...etc. \}$ dénote l'ensemble des chaines sur Act et ε représente la chaine vide.
$Act_\tau = Act \cup \{\tau\} = \{a, a_1, ..., b, b_1, ...c, c_1, ... etc.\}$ dénote l'ensemble des actions visibles augmenté de l'action invisible.

Si p, q, p_1, ...etc sont des états du système de transition alors $p \xrightarrow{a_1 a_2 ... a_n} q$ est une abréviation pour dire qu'il existe $p_1, ...p_n$ tel que $p \xrightarrow{a_1} p_1 \xrightarrow{a2} p_2 ... \xrightarrow{a_n} p_n = q$.

$p \overset{\varepsilon}{\Rightarrow} q$ signifie $p \xrightarrow{\tau} .... \xrightarrow{\tau} q$

$p \overset{\sigma}{\Rightarrow}$ signifie qu'il existe q tel que $p \overset{\sigma}{\Rightarrow} q$

Soit un système de transition $S = (Q, Act, \longrightarrow, q_0)$ et soit $q \in Q$ alors nous avons les définitions suivantes:

Définition 1.2 (Actions initiales de q)
L'ensemble d'actions initiales de q est donné par l'ensemble $I(q) = \{ a \in Act \mid q \xrightarrow{a} \}$

Définition 1.3 (Traces de q)
L'ensemble des traces de q est donné par Traces(q) = { $\sigma \in Act^* | q \xrightarrow{\sigma}$ }

Définition 1.4 (Traces complètes de q)
L'ensemble des traces complètes de q est donné par Tcomp(q) = { $\sigma \in Act^* | q \xrightarrow{\sigma} q'$ et I(q') = Ø}

Définition 1.5 (Notion de processus à branchement fini)
Pour $a \in Act$, $S(q, a) = \{p \in Q | q \xrightarrow{a} p\}$. Un processus q appartenant à Q est à branchement fini quand pour tout p atteignable à partir de q et tout a appartenant à Act on a S(q, a) est fini.

1.3 Equivalence de traces

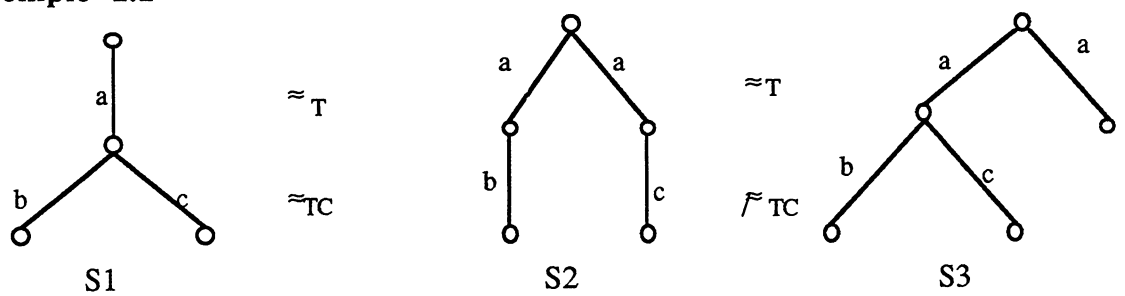
Une façon naturelle de comparer deux processus est de considérer qu'ils sont équivalents s'ils exécutent les mêmes séquences d'actions visibles [Hoa 78]. Nous considérons deux variantes de l'équivalence de traces. L'équivalence de traces proprement dite et l'équivalence de traces complètes.

Définition 1.6 (Equivalence de traces $\approx_T$)
Soient S1 et S2 deux systèmes de transition $S1 = (P, Act_\tau, \rightarrow_1, p_0)$ et $S2 = (Q, Act_\tau, \rightarrow_2, q_0)$ alors S1 et S2 sont équivalents au sens des traces et l'on note :
$S1 \approx_T S2$ ssi $\forall \sigma \in Act^* (p_0 \stackrel{\sigma}{\Rightarrow}$ ssi $q_0 \stackrel{\sigma}{\Rightarrow})$. Il est clair que $S1 \approx_T S2$ ssi Traces (p_0) = Traces(q_0) .

Définition 1.7 (Equivalence de traces complètes $\approx_{TC}$)
Soient S1 et S2 deux systèmes de transition $S1 = (P, Act, \rightarrow_1, p_0)$ et $S2 = (Q, Act, \rightarrow_2, q_0)$ alors S1 et S2 sont équivalents au sens des traces complètes et l'on note $S1 \approx_{TC} S2$ (TC pour Trace complète) ssi $Tcomp(p_0) = Tcomp(q_0)$ et $T(p_0) = T(q_0)$.

Proposition 1.8
$\approx_{TC}$ et $\approx_T$ sont des relations d'équivalence.

Exemple 1.1



Traces(S1) = {ε, a, ab, ac} = Traces(S2) = Traces(S3), alors que Tcomp(S1)={ab, ac} = Tcomp(S2) mais Tcomp(S3) = {ab, ac, a}.

L'équivalence de traces est celle qui est utilisée dans la théorie des automates et des langages comme base pour donner différentes sémantiques. Cependant quand on considère des systèmes qui ne s'exécutent pas de manière isolée mais des systèmes qui ont besoin d'échanger des informations et de se synchroniser avec d'autres systèmes, il est important de savoir si certaines communications ont lieu ou pas ou si oui ou non il y a possibilité de "deadlock". Dans l'exemple 1.1 on a vu que les systèmes S1 et S2 étaient équivalents. Voyons maintenant ces deux systèmes comme deux machines qui communiquent avec un environnement externe. Il est possible d'imaginer un environnement qui est capable de distinguer S1 de S2. En effet, si un environnement propose la séquence d'actions ab à S1, celle ci est toujours acceptée. Ce n'est pas le cas si l'environnement propose cette même séquence à S2, car ce dernier peut choisir de faire a et de passer dans un état où l'action b n'est pas possible. Un des inconvénients majeurs de l'équivalence de traces est qu'elle ne peut prendre cela en considération.

1.4 Bisimulation

Il existe plusieurs façons d'améliorer l'équivalence de traces dans le but d'être capable de différencier les systèmes S1 et S2 de l'exemple 1.1. Pour cela il faut que cette équivalence tienne compte en plus des séquences d'actions exécutées par le système, des états intermédiaires que le système traverse pour l'exécuter. Ces états intermédiaires peuvent être exploités de manière différente pour donner naissance à différentes équivalences. La première équivalence que nous présentons dans ce sens est la bisimulation. En fait la bisimulation regroupe plus d'une équivalence, elle représente plutôt une famille d'équivalences. Nous allons étudier ici en détails la bisimulation forte [Par 81] ainsi que sa variante ou équivalence projective définie pour la première fois dans [Mil 80].

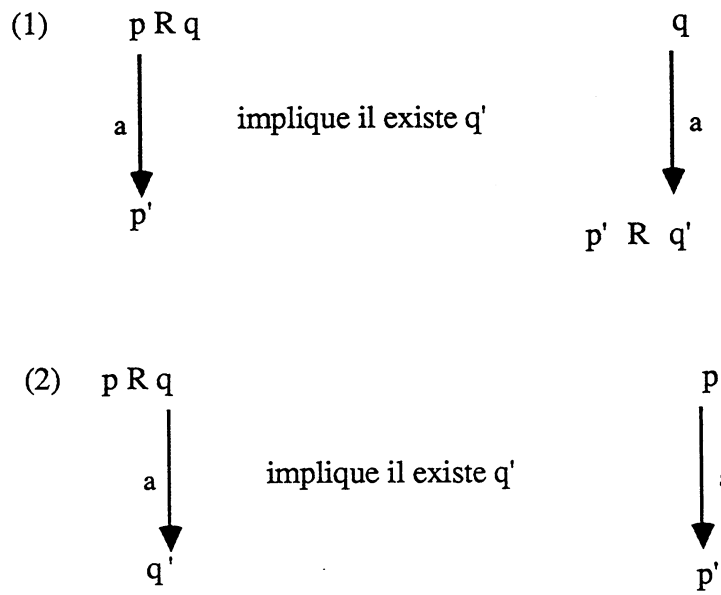
Définition 1.9 (Bisimulation forte entre deux systèmes de transition).
Etant donnés deux systèmes de transition S1 = $(P, Act_\tau, \longrightarrow_1, p_0)$ et S2= $(Q, Act_\tau, \longrightarrow_2, q_0)$ une relation $R \subseteq P \times Q$ est une bisimulation forte entre S1 et S2 notée $S1 \approx_B S2$ ssi

(1) Pour toute transition $p \xrightarrow{a}_1 p'$ de S1 et $q \in Q$ tel que $(p, q) \in R$ alors il existe $q \xrightarrow{a}_2 q'$ de S2 telle que $(p', q') \in R$.

(2) Réciproquement, pour toute transition $q \xrightarrow{a}_2 q'$ de S2 et $q \in Q$ tel que $(p, q) \in R$ alors il existe $p \xrightarrow{a}_1 p'$ de S1 telle que $(p', q') \in R$.

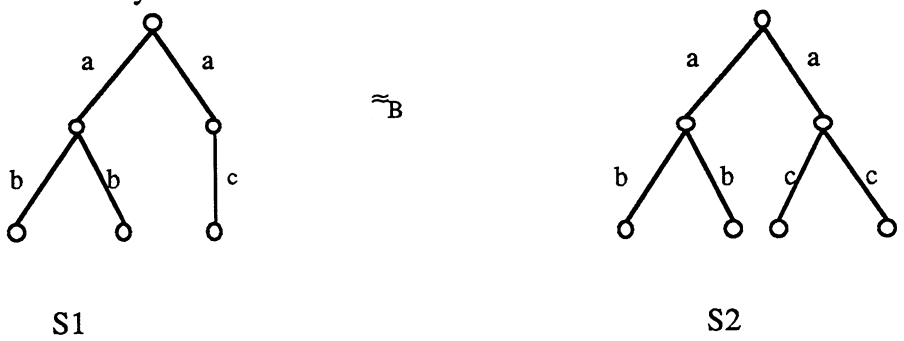
La bisimulation est autant une bisimulation entre systèmes de transition (ou graphes) qu'entre états d'un même système de transition. Si p et q sont deux états d'un même système de transition on dira qu'ils sont bisimilaires (noté $\approx_B$), si et seulement s'il existe une bisimulation R de S dans S telle que p R q.
De même on dira que deux états $p \in P$ et $q \in Q$ de deux systèmes de transition S1 et S2 respectivement sont bisimilaires ssi ils sont bisimilaires dans le système de transition S1 + S2 .

La définition de la bisimulation peut paraître plus claire à l'aide du schéma suivant :



c'est à dire p et q sont bisimilaires si chaque transition de p peut être imitée par une transition de q et que les états atteints après ces transitions sont bisimilaires et réciproquement.

Exemple 1.2 : Les deux systèmes S1 et S2 suivants sont bisimilaires.



Exemple 1.3 : S1 et S2 de l'exemple 1.1 ne sont pas bisimilaires. En effet, après avoir fait un a le système S2 peut rentrer dans un état où l'action b n'est plus possible alors que S1 ne se retrouve jamais dans un tel état.

On a vu que ces processus sont indistinguables au sens des traces. En fait, si deux processus sont bisimilaires alors ils sont équivalents au sens des traces mais l'inverse n'est pas vrai.

1.4.1 Propriétés de la bisimulation forte

La bisimulation forte étant une relation, la première propriété dont elle jouit est qu'elle est préservée par les opérations sur les relations telles que l'inverse d'une relation, la composition de relations etc.

Proposition 1.10

Soit $S = (Q, A, \longrightarrow, p_0)$ un système de transition et soient $R_1, R_2, \ldots$ des bisimulations fortes sur S alors les relations suivantes

(1) Id_P : Identité sur P.

(2) $R_{-1}^{\ i}$: L'inverse d'une relation.

(3) $R_1 \circ R_2$: La composition des relations.

(4) $\cup_{i \in I} R_i$: L'union de relations.

sont des bisimulations fortes.

Preuve

Les preuves de (1), (2) et (4) sont immédiates. Nous donnons la preuve de (3).

(3) Montrons que $R_1 \circ R_2$ est une bisimulation forte i.e. si $(p,q) \in R_1 \circ R_2$ et $p \xrightarrow{a} p'$ alors il existe $q \xrightarrow{a} q'$ avec $(p',q') \in R_1 \circ R_2$ et si $q \xrightarrow{a} q'$ alors il existe $p \xrightarrow{a} p'$ avec $(p',q') \in R_1 \circ R_2$.
Soit $(p, q) \in R_1 \circ R_2$ alors il existe r tel que $(p, r) \in R_1$ et $(r, q) \in R_2$. Si $p \xrightarrow{a} p'$ alors il existe r' tel que $r \xrightarrow{a} r'$ avec $(p', r') \in R_1$ car $(p,q) \in R_1$. Aussi puisque $(r, q) \in R_2$ alors il existe $q \xrightarrow{a} q'$ avec $(r', q') \in R_2$ donc $(p',q') \in R_1 \circ R_2$. Le cas $q \xrightarrow{a} q'$ se traite de manière similaire. ❐

Définition 1.11

p et q sont fortement bisimilaires et on note $p \approx_B q$ si $(p, q) \in R$ pour une bisimulation forte R, c'est à dire :

$$\approx_B = \cup \{ R \mid R \text{ est une bisimulation forte} \}$$

Proposition 1.12

(1) $\approx_B$ est la plus grande bisimulation forte.

(2) $\approx_B$ est une relation d'équivalence.

Preuve

(1) D'après la proposition 1.10(4) $\approx_B$ est une bisimulation forte et elle inclut toutes les autres donc c'est la plus grande.

(2) est une relation d'équivalence d'après (1), (2) et (3) de la proposition précédente. ❐

1.4.2 Caractérisation de la bisimulation forte par point fixe

Dans cette section nous donnons une caractérisation de la bisimulation forte par point fixe [Mil 88], [Mil 89].

Pour toute relation $R \subseteq Q^2$ nous pouvons définir $\mathcal{F}$ comme l'ensemble des paires (p, q) satisfaisant la définition suivante.

Définition 1.13 (Définition de $\mathcal{F}$)

Soit $< \mathcal{P}(QxQ), \subseteq >$ le treillis complet de relations sur Q nous définissons

$\mathcal{F} : \mathcal{P}(QxQ) \longrightarrow \mathcal{P}(QxQ)$

Si $R \subseteq QxQ$ alors $(p, q) \in \mathcal{F}(R)$ si

(i) $\forall\ p \xrightarrow{a} p'$ il existe q' telle que $q \xrightarrow{a} q'$ et $(p', q') \in R$.

(ii) $\forall\ q \xrightarrow{a} q'$ il existe p' telle que $p \xrightarrow{a} p'$ et $(p', q') \in R$.

Proposition 1.14

(1) $\mathcal{F}$ est monotone i.e. $R_1 \subseteq R_2$ implique $\mathcal{F}(R_1) \subseteq \mathcal{F}(R_2)$.

(2) R est une bisimulation forte ssi $R \subseteq \mathcal{F}(R)$.

Preuve

Soient $R_1 \subseteq R_2$

(1) Soit $(p, q) \in \mathcal{F}(R_1)$ donc

Si $p \xrightarrow{a} p'$ alors il existe q' telle que $q \xrightarrow{a} q'$ et $(p', q') \in R_1$.

Si $q \xrightarrow{a} q'$ alors il existe p' telle que $p \xrightarrow{a} p'$ et $(p', q') \in R_1$.

Comme $R_1 \subseteq R_2$ donc $(p', q') \in R_2$ et donc $(p, q) \in \mathcal{F}(R_2)$

(2) Pour montrer que R est une bisimulation forte si $R \subseteq \mathcal{F}(R)$ il suffit de constater que la définition de la bisimulation forte est exactement celle de $R \subseteq \mathcal{F}(R)$. ❐

Remarque : R est point fixe de $\mathcal{F}$ si $R = \mathcal{F}(R)$. R est un prépoint fixe de $\mathcal{F}$ si $R \subseteq \mathcal{F}(R)$. Donc toutes les bisimulations fortes sont des prépoints fixes de $\mathcal{F}$.

Nous montrons maintenant que le plus grand prépoint fixe de $\mathcal{F}$ est un point fixe de $\mathcal{F}$.

Proposition 1.15

La bisimulation forte est un point fixe de $\mathcal{F}$ i.e. $\approx_B = \mathcal{F}(\approx_B)$. En plus c'est le plus grand point fixe de $\mathcal{F}$.

Preuve

$\approx_B$ est une bisimulation forte donc $\approx_B \subseteq \mathcal{F}(\approx_B)$. Comme $\mathcal{F}$ est monotone donc $\mathcal{F}(\approx_B) \subseteq \mathcal{F}(\mathcal{F}(\approx_B))$ i.e. que $\mathcal{F}(\approx_B)$ est aussi un prépoint fixe de $\mathcal{F}$. Mais $\approx_B$ est le plus grand prépoint fixe de $\mathcal{F}$ car $\approx_B = \cup\{R \mid R \text{ est une bisimulation forte }\}$ et donc $\approx_B = \cup\{R \mid R \subseteq \mathcal{F}(R)\}$ et donc $\approx_B$ inclut $\mathcal{F}(\approx_B)$ i.e $\mathcal{F}(\approx_B) \subseteq \approx_B$. Finalement $\approx_B = \mathcal{F}(\approx_B)$.
$\approx_B$ est le plus grand point fixe de $\mathcal{F}$ car c'est le plus grand prépoint fixe. □

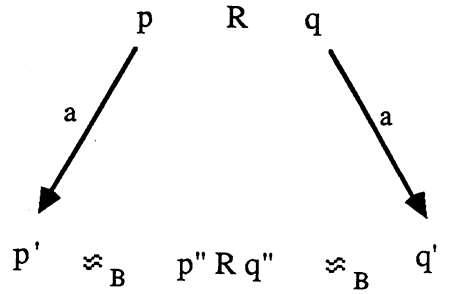
Nous complétons la définition de la bisimulation forte en introduisant la notion d'équivalence forte modulo $\approx_B$. Notons que $\approx_B R \approx_B$ signifie la composition de trois relations $\approx_B$, R et $\approx_B$.

Définition 1.16 (bisimulation forte modulo $\approx_B$)

R est une bisimulation forte modulo $\approx_B$ si p R q implique pour tout $a \in Act$:

(1) Si $p \xrightarrow{a} p'$ alors il existe q' tel que $q \xrightarrow{a} q'$ et $p \approx_B R \approx_B q$

(2) Si $q \xrightarrow{a} q'$ alors il existe p' tel que $p \xrightarrow{a} p'$ et $p \approx_B R \approx_B q$



Cette définition a été introduite par Milner pour faciliter certaines preuves. Le lemme et la proposition qui suivent montrent en effet que pour prouver que $p \approx_B q$ il suffit d'exhiber une relation forte modulo $\approx_B$ contenant la paire (p, q).

Lemme 1.17

Si R est une bisimulation forte modulo $\approx_B$ alors $\approx_B R \approx_B$ est une bisimulation forte.

Preuve

Soit $p \approx_B R \approx_B q$ et $p \xrightarrow{a} p'$. Pour montrer que $\approx_B R \approx_B$ est une bisimulation forte il faut vérifier que la propriété de transfert est satisfaite.Comme $p \approx_B R \approx_B q$ il existe donc p_1 et q_1 tels que $p \approx_B p_1$, $q \approx_B q_1$ et $p_1 R q_1$.

Puisque $p \approx_B p_1$ donc il existe $p_1 \xrightarrow{a} p'_1$ avec $p' \approx_b p'_1$ (1) . De même puisque Puisque $q \approx_B q_1$ donc il existe $q_1 \xrightarrow{a} q'_1$ avec $q' \approx_B q'_1$ (2) . Aussi comme $p_1 R q_1$ donc pour $p_1 \xrightarrow{a} p'_1$ il existe $q_1 \xrightarrow{a} q'_1$ avec $p'_1 \approx_B R \approx_B q'_1$ (3). Par composition de (1), (2) et de (3) le résultat attendu en découle. ❐

Proposition 1.18

Si R est une bisimulation forte modulo $\approx_B$ alors $R \subseteq \approx_B$.

Preuve

$\approx_B R \approx_B$ est une bisimulation forte d'après le lemme précédent donc $\approx_B R \approx_B \subseteq \approx_B$ d'après la définition de la bisimulation. Mais $Id_P \subseteq \approx_B$ donc $R \subseteq \approx_B R \approx_B$ et donc $R \subseteq \approx_B$. ❐

1.5 Simulation

Définition 1.19 (Equivalence de simulation $\approx_S$)

Etant donné un système de transition $S = (P, Act_\tau, \rightarrow p_0)$ une relation $R \in PxP$ est une simulation satisfaisant pour $a \in Act_\tau$:

Pour toute transition $p \xrightarrow{a} p'$ de S et $q \in Q$ tel $(p, q) \in R$ alors il existe $q \xrightarrow{a} q'$ de S tel que $(p',q') \in R$.

Le processus p peut être simulé par q notée $p \sqsubseteq q$ s'il existe une relation R de simulation telle que pRq. p et q sont similaires et l'on note $p \approx_S q$ ssi $p \sqsubseteq q$ et $q \sqsubseteq p$

Proposition 1.20

La simulation est une relation d'équivalence.

1.6 Equivalence projective et Equivalence observationnelle

Il existe une variante de la bisimulation ou équivalence projective : l'équivalenceobservationnelle a été introduite pour la première fois par Milner dans [Mil 80]. Il a été prouvé dans [HM 85] et [BBK87] que celle ci coïncide avec la bisimulation dans le cas de processus à branchement fini.

Définition 1.21 (Equivalence projective)
$P \approx_{pr} q$ ssi $\pi_n(p) \approx_{pr} \pi_n(q)$
où $\pi_n(p)$ représente la troncature de p à la profondeur n de l'arbre.

Définition 1.22 (Equivalence observationnelle notée $\equiv$)
Soit $S = (P, Act_\tau, \longrightarrow, p_0)$ un système de transition étiqueté et $p, q \in P$ alors
(1) $p \equiv_0 q$: toujours vraie.
(2) $p \equiv_{k+1} q$ ssi $\forall a \in Act_\tau$
(i) $\forall p \xrightarrow{a} p'$ il existe $q \xrightarrow{a} q'$ telle que $p' \equiv_k q'$
(ii) $\forall q \xrightarrow{a} q'$ il existe $p \xrightarrow{a} p'$ telle que $p' \equiv_k q'$
(3) $p \equiv q$ ssi $p \equiv_k q \quad \forall k \geq 0$

Remarque : La relation entre états d'un système peut s'étendre à une relation entre systèmes comme pour la bisimulation.

L'équivalence observationnelle est une forme particulière de l'équivalence projective.

Proposition 1.23
$\equiv$ est une relation d'équivalence.

Remarque : L'intuition qu'il y a derrière cette équivalence est que deux états p et q sont distingués s'ils sont distingués à un niveau k fini. Nous allons maintenant montrer que celle - ci coïncide avec la bisimulation forte dans le cas de processus à branchement fini.

Théorème 1.24 (Equivalence observationnelle et bisimulation forte)
Pour tous systèmes de transitions S1 et S2 on a :
(1) $S1 \approx_B S2 \Rightarrow S1 \equiv S2$
(2) $S1 \equiv S2 \Rightarrow S1 \approx_B S2$ si S1 et S2 sont à branchement fini.

Preuve
(1) On veut montrer que $S1 \approx_B S2 \Rightarrow S1 \equiv S2$. Exprimons $\equiv$ en fonction de $\mathcal{F}$. En effet on a :
$\equiv_0 = Q \times Q$, $\equiv_1 = \mathcal{F}(\equiv_0)$, ..., $\equiv_{n+1} = \mathcal{F}(\equiv_n) = \mathcal{F}^n(\equiv_0) = \mathcal{F}(Q \times Q)$ et $\equiv = \cap_{n \geq 0} \mathcal{F}^n(Q \times Q)$.
D'autre part on a $\approx_B \subseteq Q \times Q = \equiv_0$ et donc $\mathcal{F}^n(\approx_B) \subseteq \mathcal{F}^n(Q \times Q) = \equiv_n$. Comme $\approx_B$ est le plus grand point fixe de $\mathcal{F}$ donc $\approx_B = \mathcal{F}^n(\approx_B) \subseteq \mathcal{F}^n(Q \times Q) = \equiv_n$ pour tout n et donc) $\approx_B \Rightarrow \equiv$.

(2) Pour montrer que $\equiv$ implique la bisimulation forte dans le cas de systèmes de transition à branchements finis, nous reproduisons la preuve de Hennessy et Milner [HM85]. L'idée de la preuve consiste à montrer que $\equiv$ est la solution maximale de l'équation $S = \mathcal{F}(S)$ où $\mathcal{F}$ est la

définition par point fixe de la bisimulation forte. On aura ainsi montré que $\equiv$ coïncide avec la bisimulation forte.

(i) Montrons que $\equiv \subseteq \mathcal{F}(\equiv)$.

Soient $p \equiv q$ et $p \xrightarrow{a} p'$, alors pour chaque n, il existe q_n tel que $p' \equiv_n q_n$. Puisque p est à branchement fini, il existe q' tel que $q' = q_n$ où q_n apparait une infinité de fois et donc $p' \equiv_n q'$ pour tout n i.e. $p' \equiv q'$ et donc $(p, q) \in \mathcal{F}(\equiv)$. Finalement $\equiv \subseteq \mathcal{F}(\equiv)$ car p et q sont arbitraires.

(ii) Montrons que $\mathcal{F}(\equiv) \subseteq \equiv$.

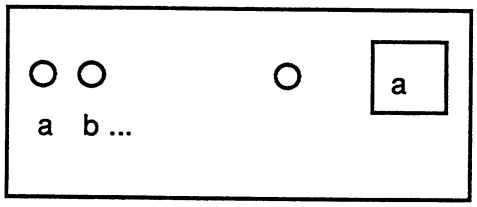
Nous prouvons par induction sur n que $(p, q) \in \mathcal{F}(\equiv) \Rightarrow p \equiv_{n+1} q$. Soit $(p,q) \in \mathcal{F}(\equiv)$ et $p \xrightarrow{a} p'$ alors il existe q' tel que $q \xrightarrow{a} q'$ avec $p' \equiv q'$. D'après (i), $(p', q') \in \mathcal{F}(\equiv)$. Par induction $p' \equiv_n q'$. De même si $q \xrightarrow{a} q'$ il existe p' tel que $p \xrightarrow{a} p'$ avec $p' \equiv_n q'$ et donc $p \equiv_{n+1} q$.

(iii) Par (i) et (ii) on a montré que $\equiv$ est solution de l'équation $S = \mathcal{F}(S)$. Montrons maintenant que si S est une solution de cette équation alors pour tout couple (p, q), si $(p, q) \in S$ alors $p \equiv q$.
Soit S telle que $S = \mathcal{F}(S)$ nous prouvons par induction sur n que $(p, q) \in S \Rightarrow p \equiv_{n+1} q$.
Soit $(p, q) \in S$ et $p \xrightarrow{a} p'$ alors $(p, q) \in \mathcal{F}(S)$. Donc il existe $q \xrightarrow{a} q'$ et $(p', q') \in S$. Par induction $p' \equiv_n q'$ et donc $p \equiv_{n+1} q$. Donc $(p, q) \in S \Rightarrow p \equiv q$. ❐

Nous avons vu jusque là comment la bisimulation pouvait améliorer l'équivalence de traces. Du point de vue observationnel, la bisimulation peut être trop restrictive. En effet pour qu'un observateur puisse distinguer les processus S1 et S2 de l'exemple 1.1, il faut qu'il observe qu'à chaque instant les séquences d'actions exécutées soient identiques et que les états atteints après exécution de ces mêmes séquences soient identiques. Cela suppose qu'à chaque instant, l'observateur est capable de faire séparément des copies des systèmes observés et de les tester séparément. Ce fait est inhérent à la nature récursive de la bisimulation. De ce point de vue la bisimulation peut être critiquée et certains auteurs ont proposé des équivalences intermédiaires, entre celle des traces et la bisimulation.
Dans ce qui suit nous étudions à titre d'exemple deux d'entres elles bien connues dans la littérature l'équivalence de failure[BHR 84] et l'équivalence de ready [BKO 88]. Toutes deux sont basées sur la notion d'observations. L'équivalence de ready identifie les processus qui acceptent les mêmes ensembles d'actions, alors que l'équivalence de failure identifie des processus qui refusent les mêmes ensembles d'actions.

1.7 Equivalence de failure.

Nous décrivons cette équivalence comme dans [Gla 90] à l'aide d'une machine hypothétique.



La machine "failure"

Cette machine contient comme interface avec le monde extérieur une fenêtre permettant de visualiser l'action courante ainsi que les boutons correspondant à chaque action de Act. Grâce à ces boutons, l'observateur peut distinguer les actions bloquées de celles qui ne le sont pas. Cette situation peut changer à tout moment durant l'exécution d'un processus. Le processus peut choisir de manière autonome un chemin d'exécution conforme à sa position dans le système de transition, mais le système ne peut que déclencher l'exécution d'une action non bloquée. Si le processus atteint un état où toutes les actions initiales de son comportement restant sont bloquées il devient inactif et la machine stagne. La fenêtre est vide dans ce cas là. Avec cette machine l'observateur peut enregistrer qu'après une certaine séquence d'actions σ, l'ensemble X d'actions est refusé par le processus. X est appelé l'ensemble de refus ou de "failure" et la paire $<\sigma,X>$ est la paire de failure. On appelle l'ensemble de toutes ces paires le comportement observable.

Définition 1.25 (Paire de failure)
$<\sigma, X> \in Act^* \times \mathcal{P}(Act)$ est appelée une paire de failure d'un processus p s'il existe un processus q tel que $p \stackrel{\sigma}{\Rightarrow} q$ et $I(q) \cap X = \emptyset$.

Définition 1.26 (Equivalence de failure).
Soit F(p) l'ensemble des paires de failure d'un processus p , deux processus p et q sont équivalents au sens de failure et l'on note $\approx_F$ cette relation ssi ils ont même ensemble de failure, c'est à dire :

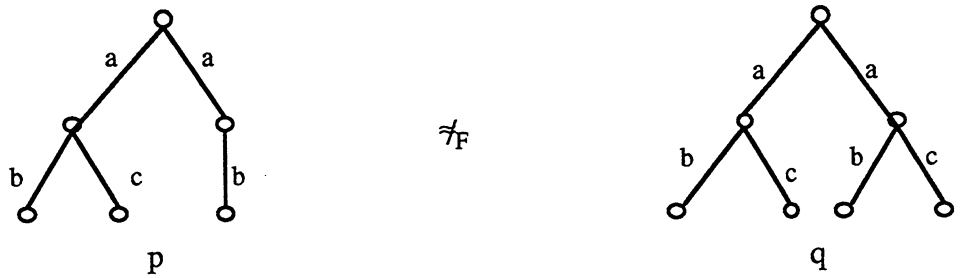
$$p \approx_F q \quad \text{ssi} \quad F(p) = F(q)$$

Proposition 1.27
$\approx_F$ est une relation d'équivalence.

Exemple 1.4 : Les deux processus suivants ne sont pas équivalents en terme de failure car F(p)={ $<\varepsilon, \{b, c\}<$, $<\varepsilon, \{b\}>$, $<\varepsilon, \{c\}>$, $<\varepsilon, \emptyset>$, $<a, \{a\}>$, $<a, \emptyset>$, $<a, \{a, c\}>$, $<a,\{ c\}>$,

<ab,Act>, <ac, Act>, <ab,Ø >, <ab, {a}>, <ab, {b}>, <ab, {c}>, <ac,Ø >, <ac, {a}>, <ac,{b}>, <ac, {c}> }

F(q) = { <ε, {b, c}>, <ε, {b}>, <ε, {c}>, <ε, Ø >, <a, {a}>, <a, Ø>, <ab, Act >, <ac, Act >, <ab, Ø >, <ab, {a}>, <ab, {b}>, <ab, {c}>, <ac,Ø >, <ac, {a}>, <ac, {b}>, <ac,{c}> }
On remarque que la paire <a,{ c}> ∈ F(p) et ∉ F(q).



Remarque : Il existe une variante de cette équivalence proposée par Hennessy dans [Hen 88], appelée équivalence d'arbres d'acceptation. Un arbre d'acceptation est un arbre à branchement fini sans τ. Cet arbre peut être représenté comme l'ensemble des paires <σ, X> ∈ Act* x $\mathcal{P}$ (Act) pour lequel il y a un processus q tel que $p \stackrel{\sigma}{\Rightarrow} q$ avec X⊆ I(q). Il s'en suit que pour de tels processus l'équivalence des arbres d'acceptation coïncide avec l'équivalence de failure.

1.8 Equivalence de ready

Cette équivalence peut être schématisée par une machine hypothétique qui a la même configuration que celle de failure. Mais dans ce cas l'observateur enregistre des observations composées des séquences d'actions exécutées et d'un ensemble d'actions ou menu que le processus est prêt à exécuter.

Définition 1.28 (Paire de ready)
<σ, X> ∈ Act* x $\mathcal{P}$ (Act) est appelée une paire de ready d'un processus p s'il existe un processus q tel que $p \stackrel{\sigma}{\Rightarrow} q$ et X =I(q).

Définition 1.29 (Equivalence de ready).
Soit R(p) l'ensemble des paires de ready d'un processus p , deux processus p et q sont équivalents au sens de ready et l'on note $\approx_R$ cette relation ssi ils ont même ensemble de ready.

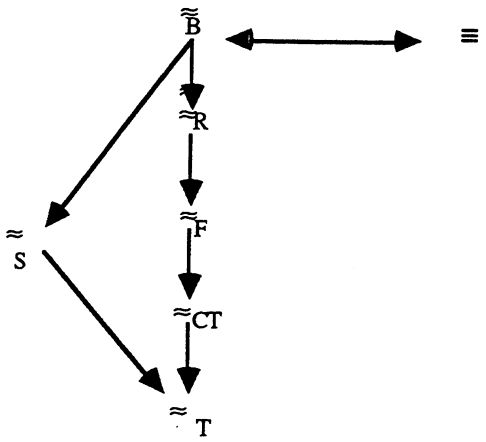
$$p \approx_R q \quad \text{ssi} \quad R(p) = R(q)$$

Proposition 1.30

$\approx_R$ est une relation d'équivalence.

1.9 Treillis des relations d'équivalence

Nous classons maintenant toutes les équivalences définies jusque là en un treillis.



Proposition 1.31

(1) $\approx_B \Rightarrow \approx_R \Rightarrow \approx_F \Rightarrow \approx_{TC} \Rightarrow \approx_T$

(2) $\approx_B \Rightarrow \approx_S \Rightarrow \approx_T$

(3) Aucune implication ne peut être renversée et aucune ne peut être rajoutée..

Preuve :

(1) $\approx_B \Rightarrow \approx_R \Rightarrow \approx_F \Rightarrow \approx_{TC} \Rightarrow \approx_T$

1.1 $\approx_B \Rightarrow \approx_R$. On suppose que $p \approx_B q$ et $<\sigma, X> \in R(p)$ et l'on montre que $<\sigma,X> \in R(q)$. Si $<\sigma, X> \in R(p)$ alors il existe p' tel que $p \overset{\sigma}{\Rightarrow} p'$ et $X = I(p')$. Comme $p \approx_B q$ il existe q' tel que $q \overset{\sigma}{\Rightarrow} q'$ et $Y = I(q')$ avec $X = Y$. Donc $<\sigma,X> \in R(q)$. De la même façon si l'on considére $<\sigma,X> \in R(q)$ on en déduit par le même raisonnement que $<\sigma,X> \in R(p)$.

1.2 $\approx_R \Rightarrow \approx_F$. Soit $p \approx_R q$ et $<\sigma, X> \in F(p)$ alors il existe $p \overset{\sigma}{\Rightarrow} p'$ tel que $I(p') \cap X = \emptyset$. Mais alors $<\sigma, I(p')> \in R(p) = R(q)$. D'où il existe $q \overset{\sigma}{\Rightarrow} q'$ avec $I(q) = I(p')$ tel que $I(p') \cap X = \emptyset$. D'où $<\sigma, X> \in F(q)$.

1.3 $\approx_F \Rightarrow \approx_{TC}$. Il suffit de constater que si $\sigma \in Tcomp(p)$ alors $<\sigma, Act> \in F(p)$.

1.4 $\approx_{TC} \Rightarrow \approx_T$: Découle directement de la définition de $\approx_{TC}$.

(2) $\approx_B \Rightarrow \approx_S \Rightarrow \approx_T$

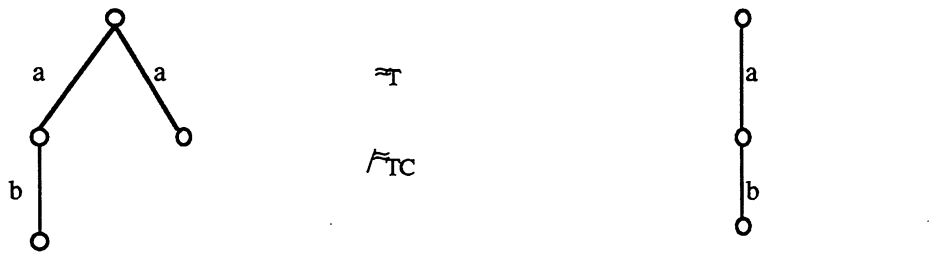
2.1 $\approx_B \Rightarrow \approx_S$: Il suffit de considérer $R = R^{-1}$ dans la définition de la simulation.

2.2 $\approx_S \Rightarrow \approx_T$. Nous considérons deux processus p et q tels que $p \approx_S q$ et $\sigma \in T(p)$ et nous montrons que $\sigma \in T(p)$.Si $\sigma \in T(p)$ alors il existe p' tel que $p \stackrel{\sigma}{\Rightarrow} p'$. Comme $p \approx_S q$ il existe q' tel $q \stackrel{\sigma}{\Rightarrow} q'$ et donc $\sigma \in T(q)$. De la même façon si l'on considére $\sigma \in T(q)$ on en déduit par le même raisonnement que $\sigma \in T(p)$.

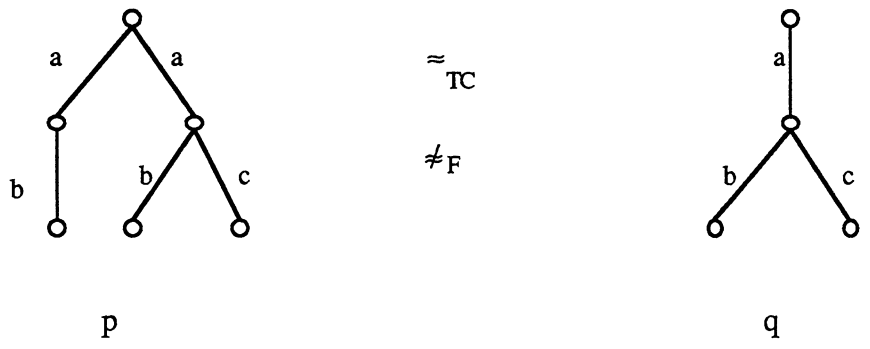
(3) Pour montrer qu'aucune flèche ne peut être renversée, il suffit de considérer les exemples suivants.

3.1 $p \approx_T q \not\Rightarrow p \approx_{TC} q$

L'ensemble des traces de p est donné par Traces(p) = { ε, a, ab} = Traces(q), alors que Tcomp(p)={a, ab} = Tcomp(q) ∪ {a}.



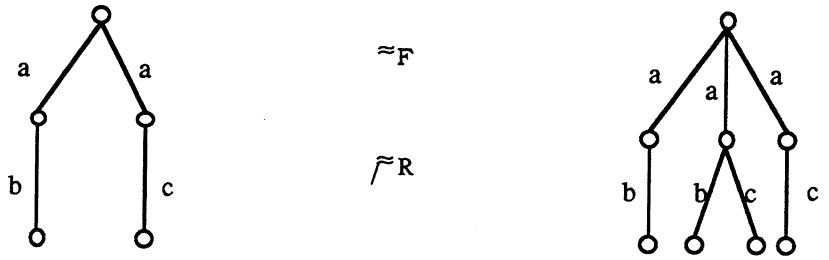
3.2 $p \approx_{TC} q \not\Rightarrow p \approx_F q$



Tcomp(p) = {ab, ac} = Tcomp(q).

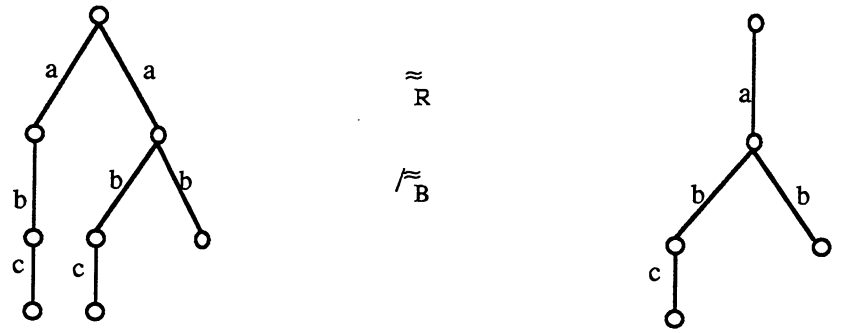
F(p) = {<ε, {b, c}>, <ε, {b }>, <ε, { c}>, <ε, Ø>, <a, {a,c}>, <a,{c}>, <a, {a }>, <a,Ø>, <ab, Act>, <ab, {a, b}>, <ab, {a, c}>, <ab, {b, c}>, <ab, {a}>, <ab, {b}>, <ab, {c}>, <ab, Ø>, <ac, Act>, <ac, {a, b}>, <ac, {a, c}>, <ac, {b, c}>, <ac, {a}>, <ac, {b}>, <ac, {c}>, <ac,Ø>} alors que F(q) = F(p)) -{<a, {a,c}>, <a,{c}>}.

3.3 $p \approx_F q \not\Rightarrow p \approx_R q$



p q

3.4 $p \approx_R q \not\Rightarrow p \approx_B q$

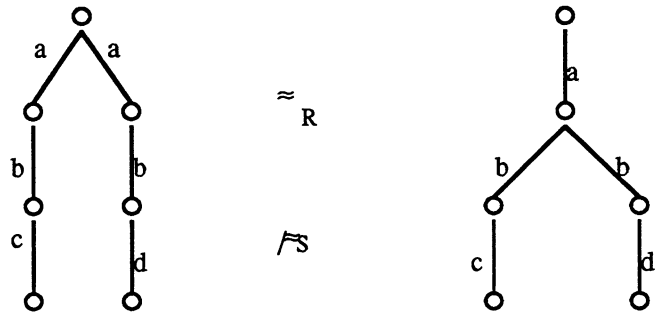


p q

3.5) $p \approx_S q \not\Rightarrow p \approx_B q$: Voir exemple 3. 1

3.6) $p \approx_T q \not\Rightarrow p \approx_S q$: Voir exemple 3.3

Pour montrer qu'on ne peut rajouter aucune flèche il suffit de constater (voir exemple suivant) que $p \approx_R q \not\Rightarrow p \approx_S q$.



De cette contre implication il en découle que $p \approx_F q \not\Rightarrow p \approx_S q$. $p \approx_{CT} q \not\Rightarrow p \approx_S q$.
De même comme $p \approx_S q \not\Rightarrow p \approx_R q$ (Voir exemple 3.1) alors

$p \approx_S q \not\Rightarrow p \approx_F q$ et

$p \approx_S q \not\Rightarrow p \approx_{CT} q$ ❐

1.10 Step équivalences

Les équivalences définies jusque là sont toutes basées sur le fait que le parallélisme est représenté par l'interleaving pur. Il est possible de les généraliser en considérant des "steps", (c'est à dire des multi - ensembles d'actions apparaissant simultanément) au lieu d'une seule action comme unité d'exécution de base. Cette vue est considérée dans des algèbres telles que SCCS [Mil 83], Meije [AB 84], FP2[Jor 86, SJ 89] et ainsi que dans HAL [BH 89, BHJ 90] . Dans la step sémantique la transition de base est de la forme $p \xrightarrow{e} q$ ou $e \subseteq$ Act au lieu de $p \xrightarrow{a} q$ avec $a \in$ Act. En utilisant ces transitions généralisées, il est possible de reconstituer le treillis précédent en redéfinissant les différentes step équivalences. Les step traces et step bisimulation sont des généralisations directes de leurs homologues dans le cas pur interleaving.

Clairement nous avons : $p \approx_{SB} q \Rightarrow p \approx_{ST} q$ où

$\approx_{SB}$ correspond à la step bisimulation et $\approx_{ST}$ à la step trace.

La step failure a été introduite par Vogler dans [TV89] pour donner la step failure sémantique à CSP [Hoa 78, Hoa 81]. Il a montré que celle ci était une généralisation du cas classique et qu'elle l'impliquait en utilisant un opérateur de déparallélisation. Dans [BH 89] une "step acceptance tree sémantique" a été proposée pour HAL.

1.11 Caractérisation logique de la bisimulation

Jusque là nous avons introduit une façon de définir des équivalences entre systèmes qui sont entièrement basées sur des considérations opérationnelles. Deux programmes sont équivalents si aucune observation ne les distingue. Une autre façon de caractériser les équivalences entre systèmes dépend des propriétés que ces derniers peuvent satisfaire. On peut dire informellement que deux processus sont équivalents si et seulement s' ils satisfont les mêmes propriétés. En d'autres termes deux processus sont distingués s'il y a une propriété satisfaite par l'un et pas par l'autre. Il reste bien évidemment à préciser ce que sont ces propriétés et à définir cette notion exprimant qu'un processus ou un système satisfait une propriété. Soit $\mathcal{A}$ l 'ensemble des propriétés, la relation entre un processus p et une propriété A de $\mathcal{A}$ est représentée par une relation de satisfaction $\models \subseteq P \times \mathcal{A}$ où $\mathcal{A}$ est l'ensemble des propriétés et P l'ensemble des processus. $p \models A$ se lit p satisfait A.
Soit $|p| = \{A \mid p \models A\}$ l'ensemble des propriétés satisfaites par un processus p. L'équivalence induite par cet ensemble de propriétés est donnée par la définition suivante :

Définition 1.32 (Equivalence induite par $\mathcal{A}$)

$$p \approx_{\mathcal{A}} q \quad \text{ssi} \quad |p| = |q|$$

L'ensemble $\mathcal{A}$ est non encore défini. Dans le domaine de la vérification et de la spécification de systèmes parallèles la logique temporelle et modale sont largement utilisées pour décrire cet ensemble $\mathcal{A}$ de propriétés de systèmes. Il existe dans la littérature un grand nombre de logiques temporelles. Le choix de telle ou telle logique dépend d'un certain nombre de critères. En particulier soit L, la logique choisie pour décrire les propriétés il est important de savoir si celle ci est compatible avec l'équivalence comportementale choisie pour comparer nos systèmes. Ceci est formalisé par la définition suivante :

Définition 1.33 (Compatibilité entre l'équivalence comportementale et logique)
Si $\approx_L$ est l'équivalence induite par la logique et $\approx_C$ est l'équivalence comportementale (Bisimulation, trace,... etc.) alors on a $p \approx_C q$ ssi $p \approx_L q$

Dans ce chapitre, nous choisissons pour le langage $\mathcal{A}$, la logique de Hennessy et Milner bien connue sous le nom de HML. Celle ci est compatible avec la bisimulation, que nous avons choisie comme sémantique pour l'algèbre HAL.
La logique HML est une logique modale très simple. Elle est définie en étendant la logique propositionnelle par des opérateurs modaux <> et [].

1.11.1 Syntaxe de HML

L'ensemble L_{HML} de formules HML est donné par la syntaxe suivante :

$$A, B ::= Tr \mid Fal \mid A \vee B \mid A \wedge B \mid \langle a \rangle A \mid [a]A \qquad \forall a \in Act_\tau$$

1.11.2 Définition informelle de HML

Tr et Fal sont les seules formules atomiques.Tr est utilisé pour représenter "True" valeur que chaque processus satisfait toujours pendant que Fal représentant "False " n'est satisfaite par aucun processus. $p \models \langle a \rangle A$ signifie que le processus p peut exécuter une a - expérience et se retrouver dans un état satisfaisant A. De la même manière $p \models [a]A$ signifie que tout processus résultat d'une a - expérience satisfait la formule A. En particulier $p \models [a]Fal$ signifie que p est a - bloqué i.e. que p ne peut faire l'action a.

1.11.3 Sémantique de HML

Soit $S = (P, Act_\tau, \longrightarrow)$ un système de transition et L_{HML} le langage de formules HML. Si $\models$ dénote la relation de satisfaction qui est définie entre processus et formules : $\models \subseteq P \times L$ est la relation telle que pour tout $p \in \mathcal{P}$

$p \models Tr$
$p \not\models Fal$
$p \models A \vee B$ ssi $p \models A$ ou $p \models B$
$p \models A \wedge B$ ssi $p \models A$ et $p \models B$
$p \models \langle a \rangle A$ ssi $\exists$ q telle que $p \xrightarrow{a} q$ et $q \models A$
$p \models [a]A$ ssi $\forall q \; p \xrightarrow{a} q$ implique $q \models A$

1.11.4 Equivalence HML et bisimulation

Quand nous utilisons la logique temporelle pour raisonner au sujet des programmes parallèles et non déterministes, l'étude de l'adéquation de la logique considérée avec la sémantique des processus considérée pour les programmes parallèles est en général un prérequis important. Précisément, la logique ne devrait pas distinguer deux programmes qui sont sémantiquement équivalents et réciproquement .

Théorème 1.34 (Comparaison de HML et de la bisimulation).
Si p et q sont à branchement fini alors $p \approx_B q \Leftrightarrow p \approx_{HML} q$

Preuve

$\Rightarrow$ $p \approx q \Rightarrow p \approx_{HML} q$

Les cas Tr, Fal, $A \vee B$ et $A \wedge B$ sont immédiats. Il reste à prouver deux cas :

- $p \models \langle a \rangle A$ ssi $\exists$ p' telle que $p \xrightarrow{a} p'$ et $p' \models A$. Comme $p \approx q$ alors il existe $q \xrightarrow{a} q'$ avec $p' \approx q'$. Par induction on a $q' \models A$ et donc $q \models \langle a \rangle A$.

- Si $p \not\models [a]A$ alors il existe $p \xrightarrow{a} p'$ et $p' \not\models A$. Comme $p \approx_B q$ alors il existe $q \xrightarrow{a} q'$ avec $p' \approx_B q'$. Par induction on a $q' \not\models A$ et donc $q \not\models [a]A$.

$\Leftarrow$ $p \approx_{HML} q \Rightarrow p \approx q$

On veut montrer que $\approx_{HML}$ est une bisimulation. Supposons que non i.e. que $p \approx_{HML} q$ et soit a telle que $p \xrightarrow{a} p'$ et $\forall q$ si $q \xrightarrow{a} q'$ alors $p' \not\approx_{HML} q'$. Soit $|S(a, q)|$ la cardinalité de S(a,q). Si $|S(a, q)| = 0$ alors $q \models [a]$ Fal alors que $p \not\models [a]$Fal. Ceci contredit l'hypothèse, Sinon il existe une formule Φ_i $1 \leq i \leq |S(a, q)|$ tel que $q_i \models \Phi_i$ et $p1 \not\models \Phi_i$. Donc $q \models [a] \bigvee \Phi_i$ $1 \leq i \leq (a,q)|$ alors que $p \not\models [a] \bigvee \Phi_i$ pour $1 \leq i < \leq |S(a, q)|$. Ceci contredit aussi l'hypothèse. □

On a vu précédemment que l'équivalence projective coïncide avec la bisimulation dans le cas de processus à branchements finis. De ce fait et du théorème précédent nous avons alors la proposition suivante :

Proposition 1.35

$$p \simeq_{HML} q \quad \Leftrightarrow \quad p \equiv q$$

Définition 1.36 (Longueur modale d'une formule)

La longueur modale d'une formule est formellement définie comme une fonction : $L_{HML} \longrightarrow \mathbb{N}$

$m(Tr) = m(Fal) = 0$

$m(A \wedge B) = m(A \vee B) = Max(m(A), m(B))$

$m(\langle a \rangle A) = m([a]A) = 1+m(A)$

Remarque : Nous pouvons stratifier L_{HML} par rapport à la profondeur modale : $\forall n \in \mathbb{N}$

$HML_n = \{A \in HML : m(A) \leq n\}$

$HML_0 \subseteq HML_1 \subseteq \ldots \subseteq HML_i \subseteq \ldots$

Such $HML = \bigcup_{n \geq 0} HMLn$

Lemme 1.37

Soient p et q deux processus alors : $p \equiv_n q \Leftrightarrow p \simeq_{HMLn} q$

1.12 Conlusion

Nous venons de présenter les équivalences de base bien connues dans la littérature du parallélisme et leur classification est donnée dans la section 1.9. Une extension de ces équivalences , toujours dans le cadre de systèmes de transitions, mais avec la possibilité d'exécuter des ensembles d'actions synchrones, donnant lieu à des "step sémantiques " est présentée dans la section 1.10.
Ce chapitre peut être vu par le lecteur comme une base sur les modèles de parallélisme d'une part, d'autre part il contient les concepts théoriques nécessaires pour le développement des autres chapitres, en particulier la notion de bisimulation forte et sa caractérisation HML. La bisimulation forte est utilisée dans le chapitre 2 comme modèle de parallélisme pour l'algèbre HAL. Le résultat de caractérisation de la bisimulation forte par la logique HML est exploré dans le chapitre 4.

Chapitre 2

L'Algèbre De Processus HAL : The Herbrand Agent Language

2.1 Motivations (Pourquoi une nouvelle algèbre?)

Notre objectif principal est de définir un outil permettant de décrire de manière concise des calculs parallèles plutôt complexes. Les applications typiques d'un tel outil sont la spécification du parallélisme dans la réécriture de termes [Che 90a, Che 90b] et la déduction logique [Jor 91, BH 92a, BH 92b]. Pour arriver à un tel but, nous avons généralisé les notions de communication et de synchronisation existantes dans les autres algèbres de processus telles que CCS [Mil 80] et FP2 [Jor 86, SJ 89]. Dans CCS, la synchronisation met en jeu exactement deux processus. La communication est orientée d'un processus émetteur vers un processus récepteur et seuls les termes fermés peuvent être échangés entre deux processus. Dans FP2, la synchronisation n-aire et la communication multi-directionnelle sont possibles. Mais encore là, seuls les termes fermés (l'ensemble des termes de l'univers de Herbrand) peuvent être échangés entre processus. En outre, contrairement à CCS, dans FP2 il n'y a pas de définitions récursives autorisant des descriptions de réseaux à topologie dynamique. Bien évidemment ceci limite sévèrement l'expressivité de ce langage.

L'introduction du parallélisme dans la réécriture de termes et la déduction logique montre l'utilité d'avoir un outil dans lequel :

- La synchronisation n-aire
- La communication multi-directionnelle
- L'échange de termes ouverts
- La description de réseaux à topologie dynamique

sont aisément exprimables. Par exemple :

- Les sous termes d'un terme sont réduits, les sous buts du corps d'une clause de Horn sont résolus en parallèle. Ceci implique qu'un ensemble de règles de réécriture ou un ensemble de clauses de Horn est appliqué en parallèle. Lorsque l'on spécifie cela via des processus communicants, les processus chargés des opérations de réduction ou de déduction déclenchent de manière synchrone tous ceux qui représentent ces règles ou ces clauses.

- Quand on résoud un but dans la logique des clauses de Horn, le mgu(most general unifier) des sous buts et tête de clause est calculé. Spécifier cela via des processus communicants implique une communication bidirectionnelle entre les processus contenant les sous buts et chaque processus contenant les têtes de clause unifiables correspondantes. En plus, quand les variables sont partagées par les sous buts, la communication devient multi-directionnelle entre les processus contenant les sous buts et ceux choisis pour les résoudre.

- Les mgu (most general unifier) calculés lors d'une étape de déduction ne sont pas nécessairement fermés. Lorsque l'on spécifie cela via des processus communicants, des termes ouverts ou non clos peuvent alors être échangés entre processus.

- Finalement, le nombre d'applications d'une règle de réécriture durant la réduction, le nombre d'instances d'une clause durant la déduction ne peuvent être connus statiquement. Ces instances doivent être créées dynamiquement en fonction du besoin. Si ces règles et clauses sont représentées par des processus alors la création dynamique de processus s'avère nécessaire.

Parmi les nombreuses algèbres de processus bien connues dans la littérature, telles que CCS [Mil 80, Mil 89], SCCS[Mil 83], LOTOS [Bri 88] et d'autres algèbres plus récemment développées telles que FP2 [SJ 89] ainsi qu'une algèbre anonyme définie dans [Plé 86] aucune, à notre connaissance, ne permet de satisfaire toutes ces exigences, au moins de manière concise . Tout ceci

nous a fortement motivé pour définir la nouvelle algèbre de processus HAL : " The Herbrand Agent Language".
Dans ce chapitre nous présentons cette algèbre de processus . Dans la section 2.2, nous présentons quelques notions et fixons quelques notations nécessaires pour la présentation des autres sections. Dans ce chapitre, HAL est vu comme une extension de CCS (Calcul de Systèmes Communicants), c'est pour cette raison que nous faisons dans la section 2.3 une présentation brève de CCS: une algèbre de processus classique et nous dégagerons les limites de son expressivité ainsi que celles d'autres algèbres existant dans la littérature (SCCS, FP2). Ensuite, dans la section 2.4, nous introduisons l'algèbre de processus HAL par sa syntaxe, sa définition informelle et quelques exemples illustratifs. Enfin, dans la section 2.5, nous donnons la sémantique opérationnelle de HAL et montrons que la bisimulation forte est une congruence relativement aux opérateurs de HAL.

2.2 Notions préliminaires

Dans cette section, nous rappelons quelques notions de base et résultats sur les substitutions et unification. Ces notions sont essentiellement tirées de [Pal 90]

Etant donnés :
X : un ensemble non vide de variables : X = {x, y, ..., z, ... }
C : un ensemble de constructeurs avec une arité $n \geq 0$ où les constructeurs d'arité nulle sont appelés des constantes et les constucteurs d'arité $n \geq 1$ sont notés : f, g, h... etc.
L'ensemble des *termes* $T_{C,X}$ est défini de manière usuelle sur X et C. L'ensemble des variables apparaîssant dans un terme t est dénoté par Var(t). Un terme t sera dit indifféremment *fermé* ou *clos* si et seulement si $Var(t) = \emptyset$. Il sera dit ouvert ou *non clos* dans le cas contraire.
Nous avons alors les définitions suivantes :

Définition 2.1 (Substitution)
L'ensemble des *substitutions* $\Sigma = \{ \sigma, \theta, ... \}$ consiste en toutes les applications $\sigma : X \longrightarrow T_{C,X}$ telles que le domaine de σ : $dom(\sigma) = \{ x \in X : \sigma(x) \neq x \}$ est fini.

Nous utiliserons la notation : $\{ x \leftarrow t : x \in dom(\sigma) \text{ et } \sigma(x) = t \}$ pour représenter une substitution σ.
L'application $t\sigma$ de σ à un terme t est définie comme le terme obtenu en remplaçant chaque variable x dans t par $\sigma(x)$.
L'ensemble *ran(σ)* est défini comme $\bigcup_{x \in dom(\sigma)} Var(\sigma(x))$.
La substitution σ est dite *close* ou *fermée* si et seulement si $ran(\sigma) = \emptyset$. Elle sera dite *non close* dans le cas contraire.

Définition 2.2 (Composition de substitution et substitution idempotente)
La composition $\sigma\sigma'$ de σ et σ' est définie par $(\sigma\sigma')(x) = (\sigma(x))\sigma'$. Nous rappelons que la composition est associative, la substitution vide est l'élément neutre et que pour tout terme t : $t\ (\sigma\sigma') = (t\sigma)\sigma'$.
Une substitution σ est *idempotente* si et seulement si $\sigma\sigma = \sigma$ c-à-d si $dom(\sigma) \cap ran(\sigma) = \emptyset$.
Une substitution σ *est plus générale* qu'une substitution σ' et l'on note cela $\sigma \leq \sigma'$ si et seulement si il existe une substitution θ telle que : $\sigma\theta = \sigma'$.

Définition 2.3 (Unification et calcul du mgu) :
Une équation est une expression E de la forme t = u où t et u sont des termes . Ces deux termes sont *unifiables* si et seulement s' il existe une substitution σ telle que $t\sigma = u\sigma$. σ est appelé un unificateur de l'équation et on dénote par unif(E), l'ensemble de tous les unificateurs de E. *L'unificateur le plus général* ou *mgu* est dénoté par : $mgu(E) = \{ \sigma \in Unif(E) : \forall \sigma' \in Unif(E)\ \sigma \leq \sigma' \}$. Il est bien connu que tous ces mgu sont équivalents et ceci explique pourquoi on parle plutôt du mgu et non d'ensemble de mgu.

2.3 Algèbre de processus classique

Les algèbres de processus sont des systèmes formels décrivant de manière abstraite des systèmes de calculs organisés sous forme de collections de composants appelés "processus" ou "agents" pouvant s'exécuter en parallèle et coopérer entre eux par communication ou envoi de messages.
Etant donné un ensemble d'agents primitifs, des agents plus complexes peuvent être construits de manière hiérarchique par un ensemble d'opérateurs de composition d'agents.
Cette notion n'est pas nouvelle : CSP[Hoa 78, Hoa 85], CCS[Mil80, Mil 89], ACP [BK85] sont des exemples classiques de telles algèbres. Certaines d'entres elles ont donné naissance à des langages pratiques pour la programmation parallèle : Occam [May 87] a ses origines dans CSP et Lotos [Bri88] dans CCS.

2.3.1 Introduction informelle à CCS

CCS (Calcul de Systèmes Communicants, introduit par Milner dans [Mil 80]) peut être considéré comme un exemple typique d'algèbre d'agents classique.
Pour décrire syntaxiquement CCS, nous disposons :

- d'un ensemble A de noms : $A = \{ a, b, \ldots \}$
- d'un ensemble $\overline{A}$ de co - noms : $\overline{A} = \{ \overline{a}, \overline{b}, \ldots \}$

A et $\overline{A}$ sont des ensembles disjoints et en bijection via ($\overline{\ }$). Nous affirmons que $\overline{\overline{a}}$ = a.

L'ensemble $\mathcal{L} = A \cup \overline{A}$ est l'ensemble des "labels" ou l'ensemble des noms de portes. K, L, U, ... etc. sont utilisés pour dénoter des sous ensembles de $\mathcal{L}$.

Comme nous présentons CCS avec passage de valeurs, nous supposons que toutes les valeurs appartiennent à un ensemble fini de valeurs $V = \{ v_1, v_2, ,\ldots v_n\}$. Naturellement, en pratique il faut considérer différents types de valeurs : des entiers, des booléens, des séquences de valeurs ... Mais ici, nous simplifions la présentation en ne considérant que des entiers pour illustrer nos exemples.

L'ensemble des actions visibles est défini comme le produit cartésien de $\mathcal{L}$ et de V noté $\mathcal{L}$ x V. On dénote alors par Act l'ensemble de toutes les actions i.e. $\mathcal{L}$ x V $\cup \{\tau\}$. α, β, ...etc. sont utilisées pour représenter des éléments de Act.

Si α est une action alors $\mathcal{L}(\alpha)$ indique la porte utilisée par cette action . On a $\mathcal{L}(a(v)) = a$, $\mathcal{L}(\overline{a}(v)) = \overline{a}$, $\mathcal{L}(\tau) = \emptyset$.

Nous supposons pour finir une fonction de renommage f : Act —> Act telle que :

- $f(a(x)) = f(a)(x)$

- $f(\overline{a}(x)) = \overline{f(a)}(x)$

- $f(\tau) = \tau$.

Nous allons maintenant définir $\mathcal{P}$: l'ensemble d'agents CCS dénotés par p, q... comme le plus petit ensemble contenant les expressions suivantes :

$$p, q \quad ::= \quad nil \mid a(x).p \mid \overline{a}(e) \mid \tau.p \mid p+q \mid p \| q \mid p/U \mid p[f] \mid id$$

où id $\in$ ID (l'ensemble d'identificateurs d'agents), U est un ensemble de noms de portes (sous ensemble de $\mathcal{L}$), f une fonction de renommage et e une expression définissant des entiers.
a(x) représente l'entrée d'une valeur quelconque à une porte a. Dans a(x).p la variable x est liée. Par opposition à cela, on appellera variable libre toute variable x ne se trouvant pas dans une sous expression de la forme a(x).p'. Cette notion de variables liées est exactement celle qu'on trouve dans le λ - calcul, excepté que dans celui ci seul λ peut lier des variables alors qu'ici toutes les portes a, b,... etc. peuvent lier des variables.

N'importe quel terme généré par la grammaire précédente est appelé expression . *CCS* est l'ensemble des expressions. Pour toute expression p, VL(p) est l'ensemble des variables apparaissant libres dans p. Les agents CCS sont des expressions fermées i.e. des expressions sans aucune variable libre. $\mathcal{P} = \{p, q, \ldots, r, \ldots\}$ est l'ensemble des agents.

L'interprétation opérationnelle des agents CCS est donnée par la sémantique opérationnelle.
L'idée de base est que : Quand un agent p exécute une action α il devient un nouvel agent q, capable à son tour d'exécuter de nouvelles actions. La possibilité pour p d'exécuter une telle étape est appelée une transition et est dénotée par : $p \xrightarrow{\alpha} q$ ($\alpha \in$ Act) dans la définition de la sémantique.

La sémantique opérationnelle consiste donc à associer un système de transition défini de manière inductive par un ensemble de règles d'inférence obtenu par les techniques proposées par Plotkin [Plo81].

nil est l'agent inactif, il n'y a aucune action α et aucun processus q tel que : nil $\xrightarrow{\alpha}$ q .

Le préfixage (noté α.p)

$$\frac{}{a(x).p \xrightarrow{a(v)} p\{x \leftarrow v\}} \qquad \frac{}{\overline{a}(v).p \xrightarrow{\overline{a}(v)} p} \qquad \frac{}{\tau.p \xrightarrow{\tau} p}$$

où $\{x \leftarrow v\}$ dénote la substitution syntaxique de toutes les occurences libres de x dans p.

Exemple 2.1 : $p = a(x).\overline{b}(x+1).$ nil .

Cet agent est capable de recevoir à travers la porte a une valeur de x. Si par exemple il reçoit l'entier 3 alors il renvoie l'entier 4 par la porte $\overline{b}$. On peut schématiser p comme une boîte noire avec deux portes a et $\overline{b}$ sur lesquelles circulent des entiers .

La somme non déterministe (p+q)

$$\frac{p \xrightarrow{\alpha} r}{p+q \xrightarrow{\alpha} r} \qquad \frac{q \xrightarrow{\alpha} s}{p+q \xrightarrow{\alpha} s}$$

La composition parallèle d'agents (p||q)

$$\frac{p \xrightarrow{\alpha} r}{p\|q \xrightarrow{\alpha} r \| q} \qquad \frac{q \xrightarrow{\alpha} s}{p\|q \xrightarrow{\alpha} p \| s}$$

$$\frac{p \xrightarrow{a(v)} r \quad q \xrightarrow{\overline{a}(v)} s}{p \| q \xrightarrow{\tau} r \| s} \qquad \frac{p \xrightarrow{\overline{a}(v)} r \quad q \xrightarrow{a(v)} s}{p \| q \xrightarrow{\tau} r \| s}$$

La restriction d'agent p / U

$$\frac{p \xrightarrow{\alpha} q}{p / U \xrightarrow{\alpha} q / U} \quad \text{Si } \mathcal{L}(\alpha) \notin U$$

Le renommage d'agents $p[f]$

$$\frac{p \xrightarrow{\alpha} q}{p[f] \xrightarrow{f(\alpha)} q[f]}$$

Deux règles permettent de décrire l'agent p+q. La première distingue des deux agents, celui qui est actif et en exclut l'autre. Pour l'agent parallèle p||q, il y a quatre règles possibles. Les deux premières montrent le comportement asynchrone de p et q. Les deux dernières montrent que p et q peuvent communiquer de manière synchrone quand tous deux sont prêts à exécuter des actions sur des portes complémentaires : la composition d'une action d'entrée a(v) et d'une action de sortie $\overline{a}$ (v) engendre une action interne notée τ. L'agent p/U peut exécuter toutes les actions de p n'utilisant pas les portes de U. Pour le renommage p[f] se comporte comme p où tous les noms de porte sont modifiés par f.

En plus de ces opérateurs, CCS autorise des définitions de noms d'agents. Si id est un identificateur et p une expression d'agent alors id $\Leftarrow$ p définit un nom d'agent id qui peut être à son tour utilisé dans les expressions d'agents et qui s'interprète de manière évidente comme suit :

Définition d'agent (id$\Leftarrow$ p)

$$\frac{p \xrightarrow{\alpha} q}{id \xrightarrow{\alpha} q}$$

Remarque : Nous avons présenté seulement un sous ensemble de CCS avec Valeur. Par exemple dans CCS on autorise des définitions plus générales de la forme $Id(x_1, x_2, ..., x_n) \Leftarrow p$.

Exemple 2.2 : Buffer $\Leftarrow$ w(x). $\overline{r}$ (x).Buffer

Zéro $\Leftarrow$ $\overline{r}$ (0). Buffer

Add $\Leftarrow$ a(m).b(n).($\overline{c}$ (m+n).Add + $\overline{d}$ (m+n).Nil)

Sum $\Leftarrow$ (Add || Zéro [w : c, r : a]) / {a, c}

où :

- Buffer : Reçoit une valeur de x par la porte w et l'envoie par la porte $\overline{r}$, puis redevient de nouveau Buffer. On peut représenter un comportement possible de la manière suivante :

Buffer $\xrightarrow{W(a)}$ $\overline{r}$ (a).Buffer $\xrightarrow{\overline{r}(a)}$ Buffer $\xrightarrow{W(b)}$ $\overline{r}$ (b).Buffer $\xrightarrow{\overline{r}(b)}$ Buffer ...

- Zéro : Se comporte comme Buffer après avoir émis un zéro par sa porte $\overline{r}$.
- Add : Reçoit deux valeurs m et n à travers ses portes a et b respectivement, puis se comporte comme un nouveau processus ayant le choix entre : envoyer le résultat de la somme par sa porte $\overline{c}$ et redevenir comme Add ou bien renvoyer le résultat de son calcul par la porte $\overline{d}$ et s'arrêter.

- Sum : qui est la mise en parallèle des deux processus Add et Zéro (où les portes w et $\overline{r}$ sont renommées respectivement en c et $\overline{a}$) est un processus avec seulement deux portes b et $\overline{d}$ visibles de l'extérieur. Il est aisé de se convaincre que le processus Sum a pour effet de calculer Σmi pour tous les mi traversant la porte b et renvoie le résultat par $\overline{d}$.

2.3.2 Expressivité des algèbres classiques

Dans ces algèbres, les agents exécutent des séquences d'actions. Une communication entre agents est une composition d'actions que les partenaires de celle ci sont prêts à exécuter. Dans des algèbres telles que CCS, une action transporte une valeur dans une direction prédéfinie("entrée" ou "sortie") à travers une porte. Exactement deux actions sur deux portes comportant des étiquettes complémentaires peuvent être composées pour fournir une action silencieuse τ.

De fait, la communication dans de telles algèbres est réduite à sa plus simple et naïve expression. Elle met en jeu exactement deux partenaires (Un émetteur et un récepteur) et une valeur est conduite **unidirectionnellement** du premier vers le dernier.

La communication est **synchrone** : Cette idée qui était déjà présente dans CSP[Hoa 78] est conceptuellement très utile car le cas d'un message en cours de transmission n'est pas à considérer. Ceci implique que la communication prend le sens de synchronisation entre agents et par conséquent seuls deux agents peuvent se synchroniser.

Dans certaines algèbres : SCCS [Mil 83] et Meije [Bou 85], une vue plus générale de la synchronisation a été considérée. Les actions restent atomiques. Mais l'ensemble des actions forment un groupe muni d'une opération produit commutative et associative où $\alpha\beta$ signifie l'occurrence simultanée des actions α et β. Une action peut avoir une structure complexe $\alpha\beta\chi\delta$ mais elle reste atomique car elle a lieu en un seul pas d'exécution. La composition parallèle dans SCCS(notée $\otimes$) est définie opérationnellement par la seule règle suivante :

$$\frac{p \xrightarrow{\alpha} r \qquad q \xrightarrow{\beta} s}{p \otimes q \xrightarrow{\alpha\beta} r \otimes s}$$

Cependant dans SCCS, la communication reste orientée.

2.4 L'algèbre de processus HAL

La communication dans les algèbres de processus classiques consiste en une paire d'offres émanant de deux processus mis en parallèle avec une variable dans une offre et une valeur dans l'autre . Cette asymétrie n'a aucune raison apparente d'exister. En effet, il est possible de rendre la communication symétrique en considérant dans les deux offres proposées de part et d'autre des termes de T_C où C est un ensemble de constructeurs. (T_C : l'ensemble des termes clos est appelé univers de Herbrand sur C). La composition des deux offres a(t) et $\overline{a}$ (t) avec t appartenant à T_C donne naissance à l'action silencieuse τ.

Cette généralisation a un grand nombre de conséquences intéressantes :

1. Il est clair que la composition de CCS peut être réalisée grâce à ce type de communication : il suffit de considérer t = u = v.
2. La synchronisation sans communication est aussi tout à fait possible: il suffit de considérer a(t) et $\overline{a}$ (t) où t est n'importe quel terme clos.
3. La communication n'est plus unidirectionnelle. Les valeurs échangées circulent dans les deux directions entre les processus concernés par la composition.
4. La communication de termes non complètement spécifiés (communication non close), très utile pour spécifier le mécanisme d'inférence parallèle, devient alors possible à condition d'autoriser dans les offres des termes t et t' de $T_{C,X}$ et la communication n'a lieu que s'il existe une substitution non close σ telle que $t\sigma = t'\sigma$.

Cette idée a déjà été explorée par certains auteurs. Une algèbre décrite dans [Plé 86] fait une jonction opérationnelle entre les types de données algébriques [Ref] et CCS. Elle généralise le mécanisme de communication par l'utilisation de l'unification. Mais elle est restreinte à la composition d'actions de CCS car seulement deux processus sont concernés par la communication. De plus la communication est close. L'algèbre FP2 [Jor 86, SJ 89] contient aussi cette idée avec une approche style SCCS pour la composition d'actions. FP2 autorise la communication n-aire et la synchronisation multi directionnelle. Cependant FP2 est aussi restreint à la communication close et à une vue statique de la composition d'agents.

2.4.1 Introduction de HAL

2.4.1.1 Quelques définitions de base

- *Portes* : $\mathcal{K}$ est un ensemble d'identificateurs représentant des noms de portes.

- *Offres* : Soit k un nom de porte et t un terme de T_C , on définit alors deux types d'offres:
  - l'offre d'entrée par k?(t)
  - l'offre de sortie par k!(t)

Nous utilisons comme dans CSP les symboles ? et ! pour indiquer la direction de la communication.

- *Ensemble d'offres* : Soient $k_1, k_2, ..., k_n$: n noms de portes distincts et $t_1, t_2, ..., t_m$ m termes. $a = \{k_1?(t_1),..., k_n?(t_n), k_{n+1}!(t_{n+1}), ... k_m!(t_m)\}$ dénote l'ensemble de m offres synchrones de ti sur ki (i=1..m). L'ensemble vide d'offres est dénoté par τ. Formellement, un ensemble d'offres a est une fonction partielle de $\mathcal{K}$ vers $\{?, !\} \times T_{C,X}$..

- *Evénement* : Soient $k_1, k_2, ..., k_n$: n noms de portes et $v_1, v_2, ..., v_n$ n valeurs de T_C , $e = \{k_1(v_1), k_1(v_1),..., k_n(v_n)\}$ est un événement. Un événement est donc toute instance close d'un ensemble d'offres a, privé des symboles d'orientation de la communication (!, ?). On note a* l'ensemble a sans ces symboles . Formellement un événement est défini comme une fonction partielle de $\mathcal{K}$ vers T_C . On dénote par E l'ensemble des événements.

2.4.1.2 Agents HAL

Les agents HAL exécutent des transitions atomiques étiquetées par des événements .

L'ensemble d'agents HAL est le plus petit ensemble généré par la grammaire suivante :

$$p,q ::= nil \mid a.p \mid p+q \mid p \| q \mid p[\![k,l]\!] \mid p\lceil U \mid p[f] \mid id \qquad (*)$$

où a est un ensemble d'offres, k et l des noms de portes, id un nom d'agent et f une fonction de renommage.

Cette grammaire engendre une famille de termes. Un terme engendré par cette grammaire est syntaxiquement bien formé s'il existe un arbre de dérivation qui lui corresponde et que tous les noms d'agents qu'il utilise sont définis. Notons que des noms d'agents sont introduits via la notion de définition et l'on suppose sans entrer dans les détails qu'un environnement de définitions est construit au fur et à mesure que celles ci sont définies.

2.4.1.3 Notions d'agents et d'expressions d'agents

Soit Buffer ⇐I?(x).O!(x).nil une définition d'agent. Le comportement décrit par cette expression est fini. Buffer reçoit une valeur de x par I et l'émet par O. La seule variable utilisée par Buffer est dite liée car elle est introduite par le préfixe I?. La portée de cette variable est celle de l'expression O!(x).nil.
Si au lieu de I?(x).O!(x).nil on a I?(x).O!(z).nil la variable z est dite libre. Dans HAL les portes suivies de ? jouent exactement le même rôle que la notation λ dans le λ - calcul, et la notion de variables libres et liées est exactement celle qui existe en logique. En particulier nous avons les mêmes notions d'expressions fermées et ouvertes.

Notation : Soit a un ensemble d'offres. On décomposera a en deux parties :
a_{in} : La partie entrée de a ou ensemble d'offres de la forme k?(t).
a_{out} : La partie sortie de l'ensemble d'offres ou l'ensemble d'offres de la forme k!(t).

- $a = a_{in} + a_{out}$
- $Var(a_{in}) = \cup_{ki?(ti) \in a_{in}} Var(ti)$ et $Var(a_{out}) = \cup_{ki!(ti) \in a_{out}} Var(ti)$.

Définition 2.4 (Calcul des variables libres)
L'ensemble des variables libres d'un terme HAL est donné par la spécification suivante.

$VL(nil) = \emptyset$
$VL(p+q) = VL(p \| q) = VL(p) \cup VL(q)$.
$VL(p[\![k,l]\!]) = VL(p\lceil U) = VL(p)$.
$VL(id) = \emptyset$
$VL(a.p) = (VL(p) \cup Var(a_{out})) - Var(a_{in})$

Remarque : La définition précédente signifie que x est libre dans p s'il n'est pas contenu dans une expression de la forme {...k?(x)...}p' . Une expression p est dite fermée si $VL(p) = \emptyset$.

Définition 2.5 (Agent et expression).
Agent : On appelle agent tout terme fermé i.e. tout terme p engendré par *. On note $\mathcal{P}$ l'ensemble des agents et p, q, ...r etc. dénotent des éléments de $\mathcal{P}$.
Expression : On appelle expression tout terme ouvert engendré par *. On note $\mathcal{HAL}$ lensembles des expressions. Evidemment cet ensemble contient $\mathcal{P}$.

Remarque : Un agent peut être vu comme un programme et une expression comme un schéma de programme.
Pour autoriser l'utilisation de valeurs reçues nous avons besoin de formaliser la substitution de ces valeurs aux variables qui les utilisent.

Définition 2.6: (Remplacement dans p)
Soit $\sigma : X \longrightarrow T_C$, Le remplacement $p\sigma$ est défini formellement de manière structurelle comme suit:

$(nil)\sigma = nil$

$(a.p)\sigma = (a_{in}+(a_{out})\sigma')p\sigma'$ où $\sigma' = \sigma \mid_{VL(a.p)}$

$(p+q)\sigma = p\sigma+q\sigma$

$(p\|q)\sigma = p\sigma \|q\sigma$

$p[\![k,l]\!]\sigma = p\sigma [\![k,l]\!]$

$(p\lceil U)\sigma = p\sigma \lceil U$

$(id)\sigma = id$

Lemme 2.7
Si p est un processus et $\sigma : X \longrightarrow T_C$ avec $VL(p) \subseteq X$ alors $p\sigma$ est aussi un agent.
Remarque : lorsqu'on remplace par une substitution σ ouverte, il faut généraliser la définition précédente en renommant les variables liées pour éviter les captures. on a alors :
$(a.p)\sigma = (a_{in})\mu+(a_{out})\mu\sigma')p\mu\sigma'$ où μ est la substitution renommant les variables liées.

2.4.1.4 Notion de sorte d'un agent.

Nous avons délibérément choisi de considérer que nos agents communiquent via des portes. Dans une implémentation éventuelle d'un système décrit par HAL il serait important d'avoir des informations sur ces portes, connaitre exactement les portes utilisées par un agent. En particulier on doit pouvoir vérifier que deux agents p et q mis en parallèle ont des ensembles de noms de portes disjoints. Cela étant, nous avons besoin alors de définir formellement pour chaque processus l'ensemble des portes qu'il utilise. C'est ce que nous allons formaliser dans cette section par la notion de sorte.

Définition 2.8 (Successeur d' un agent)
Soient p et q deux agents et p $\xrightarrow{e}$q une transition . La paire <e, q> est un successeur immédiat de p , e est un événement de p et q est un e - successeur de p.

Définition 2.9 (Sorte d'un agent)
Soit K $\subseteq \mathcal{K}$, si les événements de p et de tous ses successeurs immédiats utilisent des portes dans K alors K est une sorte de p. On dit que p est de sorte K et l'on note $\models$ p : K.

Proposition 2.10 (Propriétés d'une sorte)
Soit p un agent et K $\subseteq \mathcal{K}$, K est une sorte de p ssi quand p $\xrightarrow{e}$q alors
(1) dom(e) $\subseteq$K
(2) K est la sorte de q.

Définition 2.11 (Calcul de la sorte)
K : HAL —> 2^K est la plus petite solution de la définition récursive suivante :
$\mathcal{K}(a.p) = \mathcal{K}(p) \cup dom(a)$
$\mathcal{K}(p+q) = \mathcal{K}(p||q) = \mathcal{K}(p) \cup \mathcal{K}(q)$.
$\mathcal{K}(p[\![k,l]\!]) = \mathcal{K}(p)$.
$\mathcal{K}(p\lceil U) = \mathcal{K}(p) - U$
$\mathcal{K}(id) = \mathcal{K}(p)$
$\mathcal{K}(p[f]) = f(\mathcal{K}(p))$.

Le plus petit point fixe existe car cette définition n'utilise que des opérateurs monotones.

Théorème 2.12 : $\models$ p : $\mathcal{K}(p)$

2.4.1.5 Définition informelle d'agents HAL

- *L'agent inactif* : nil est l'agent qui ne fait aucune action.

- *Le préfixage* : a.p est l'agent préfixé par a où a est un ensemble d'offres. L'agent a.p peut exécuter des transitions étiquetées par e où e est de la forme $\sigma(a^*)$ (σ est une substitution sur les variables de a), puis se comporte comme l'agent pσ.

- *Le choix non déterministe* : p+q se comporte soit comme p, soit comme q compte tenu des offres proposées par l'environnement de p+q.

- *La composition parallèle* : Chaque événement de p||q est un événement e de p ou un événement f de q ou l'union e+f (somme de deux fonctions avec des domaines disjoints) de deux événements: un de p et un de q. Ceci signifie qu'à chaque étape, p||q se comporte soit comme p ou q de manière asynchrone ou comme p et q de manière synchrone.

- *La connexion* : p[[k,l]] est un agent p dont les portes k et l ont été connectées ou liées. Cet agent peut exécuter toutes les actions atomiques de p. En outre, pour toutes les transitions de p étiquetées par des événements de la forme e + {k(v), l(v)} où v est un terme de T_C alors p[[k,l]] peut exécuter des actions décrites par e. Typiquement, la connexion est utilisée pour connecter deux portes d'agents parallèles, mais sa définition est plus générale puisque dans p[[k,l]] l'agent p n'est pas nécessairement une composition parallèle.

- *La restriction* : Cette opération s'applique aux connecteurs. Elle permet de les cacher c.-à-d. d'en interdire l'utilisation. p⌈U est un agent sans portes dans U. Il peut exécuter toutes les transitions de p étiquetées par e à condition que $U \cap dom(e) = \emptyset$.

- *Le renommage* : Cette opération s'applique aussi aux connecteurs. p[f] se comporte exactement comme p. Mais dans p[f] les ensembles d'offres e ont été remplacés par f(e). f(e) est définie de la manière suivante :
  - $f(k(t)) = f(k)(t)$
  - $f(k(t)) = k(t)$ si $k \notin dom(e)$.
  - $f(\{k_1(t_1), k_2(t_2), ..., k_n(t_n)\} = \{f(k_1(t_1), f(k_2(t_2), ..., f(k_n(t_n)\}$

Avec les opérateurs définis jusque là, seuls les comportements finis sont autorisés. Pour décrire les comportements infinis, introduisons la notion de définition d'agents .

- *Définition ,d'agent* : Soit ID l'ensemble des noms d'agents ID = {A, B,..., etc.} Une définition d'agent s'écrit de la manière suivante :
  A ⇐ p, où A est un nom d'agent et p un agent .

2.4.1.6 Exemples

Exemple 2.3

Soit X = { x, y, z } un ensemble de variables et 0, s(0), ... des entiers, nous définissons :

- Buffer, un "one place buffer", avec les ports I et O (pour "Input "et "output") : Voir figure1, sa définition en HAL est : Buffer ⇐I?(z).O!(z).Buffer
  Les comportements possibles de Buffer : Si I?(s(0)) est sa première action, il devient O!(s(0)Buffer, où la seule action possible est O!(s(0)). Il devient de nouveau buffer et il est prêt à accepter une nouvelle valeur sur I, etc.
  Notons que l'agent Buffer a une "mémoire" : lorsqu'il accepte s(0) sur I, il "l'enregistre" dans O!(s(0)). Buffer pour l'utiliser dans le prochain comportement.

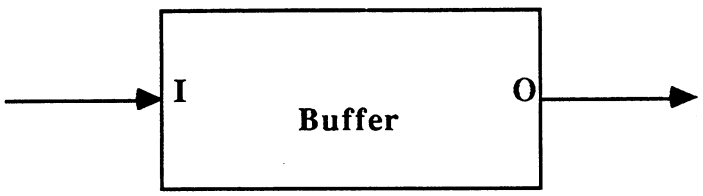


figure1

Exemple 2.4

Pour les exemples suivants, nous utilisons en plus l'ensemble de constructeurs : C={0, s, p},pour zéro successeur et prédécesseur. Nous allons calculer les prédécesseurs d'entiers naturels, en tenant compte du faite que le prédécesseur de 0 est 0. Ce qui impliquerait que :

$$p(s(0)) = 0$$
$$s(p(0)) = s(0)$$

- Remove - one -p : Un agent plutôt naïf avec les portes E et R (pour "Expression" et"Résult").
  Il accepte un terme de T_C sur sa porte E et, à l'intérieur de la même action, si ce terme est de la forme p(s(x)) ou p(0), l'agent offre respectivement x ou 0 sur sa porte R sinon il retourne le terme:

Remove-one-p ⇐ E?(0) R!(0) . nil
+ E?(p(s(x))) R!(x) . nil
+ E?(p(0)) R!(0) . nil
+ E?(s(x)) R!(s(x)) . nil

Exemple 2.5

On termine avec l'exemple de l'agent "plus", qui après avoir reçu deux termes $s^k(0)$ et $s^l(0)$ $(k, l \geq 0)$ par ses portes E et F, exécute un certain nombre d'étapes de calcul et renvoie $s^{k+l}(0)$ par sa porte U. Son expression HAL serait la suivante :

$$\begin{aligned} Plus \Leftarrow\ & \{E(0), F(u)\}.nil + \\ & (\{E(s(t), F(u)\}.\{G(t),H(u)\}.\{V(v)\}.\{U(s(v)\}.nil \parallel Plus[E:E',F: F', U:U']) \\ & [\![G,E']\!] [\![H, F']\!] [\![V, U']\!] \lceil G, H, V, E', F', U' \end{aligned}$$

Une séquence possible d'action de ce processus prenant comme entrées deux entiers s(s(0)) et s(0) est : $Plus \xrightarrow{\{E(s(s(0))),(s(0))\}} q \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\{U(s(s(0)))\}} nil$

2.5 Sémantique de HAL

Pour donner une sémantique à une algèbre de processus, il y a plusieurs voies et plusieurs critères à considérer. Suivant les voies choisies et les critères considérés plusieurs sémantiques peuvent être engendrées. On a vu dans le chapitre1 divers modèles de parallélisme.

Parmi les voies à explorer on peut citer : Les sémantiques "pur interleaving", c'est à dire celles qui ne distinguent pas le parallélisme du non déterminisme, les "step sémantiques" et les sémantiques "vrai parallélisme".

Parmi les critères à considérer on peut citer :

- Le cas où les actions ne sont pas interprétées. On parle dans ce cas d'algèbre de synchronisation.
- Le cas où les actions sont interprétées. On parle alors d'algèbre de communication où d'algèbre avec
  passage de valeurs.
- Le cas où l'action interne τ est visible.
- Le cas où l'action interne τ est invisible.

Dans cette thèse, nous nous sommes situés dans le cadre "step sémantique" avec passage de valeurs et avons considéré l'action interne comme une action visible.

2.5.1 Sémantique opérationnelle

Pour donner un sens à nos agents, nous utilisons la notion générale de systèmes de transitions étiquetés T = (States , Act , —>) qui consiste en un ensemble States d'états , un ensemble Act

d'actions et une relation de transition $\longrightarrow \subseteq$ States x Act x States . Dans notre système nous considérons $\mathcal{P}$ pour States, E pour Act .
Nous dénotons par $p \xrightarrow{e} r$ un élément de $<p, e, r>$ dans $\longrightarrow$.

La sémantique des agents consiste en la définition de chaque transition $\xrightarrow{e}$ $e \in E$. Cette définition découle de la structure des agents. Dans la définition qui suit Pref, Sum, Par, Con, Res, Ren et Def sont utilisés pour indiquer respectivement les opérateurs de préfixage, somme non déterministe, composition parallèle, restriction, renommage et définition d'agents.

Définition 2.13 (Les règles de transition)

Pref $\dfrac{}{a.p \xrightarrow{\sigma(a^*)} p\sigma}$ où σ est une substitution close $\sigma : Var(a) \longrightarrow T_C$.

Sum1 $\dfrac{p \xrightarrow{e} r}{p+q \xrightarrow{e} r}$ **Sum2** $\dfrac{q \xrightarrow{e} s}{p+q \xrightarrow{e} s}$

Par1 $\dfrac{p \xrightarrow{e1} r}{p \| q \xrightarrow{e1} r \| q}$ **Par2** $\dfrac{q \xrightarrow{e2} s}{p \| q \xrightarrow{e2} p \| s}$

Par3 $\dfrac{p \xrightarrow{e1} r \quad q \xrightarrow{e2} s}{p \| q \xrightarrow{e1+e2} r \| s}$ **Ren** $\dfrac{p \xrightarrow{e} r}{p[f] \xrightarrow{f(e)} r[f]}$

Con1 $\dfrac{p \xrightarrow{e} r}{p[\![k,l]\!] \xrightarrow{e} r[\![k,l]\!]}$ **Con2** $\dfrac{p \xrightarrow{e+\{k(v),\,l(v)\}} r}{p[\![k,l]\!] \xrightarrow{e} r[\![k,l]\!]}$

Res $\dfrac{p \xrightarrow{e} r}{p \lceil U \xrightarrow{e} r \lceil U}$ Si $U \cap dom(e) = \emptyset$ **Def** $\dfrac{p \xrightarrow{e} r}{id \xrightarrow{e} r}$ S'il existe une définition $id \Leftarrow p$.

Maintenant que nous avons défini nos transitions nous allons vérifier qu'elles sont bien formées, c'est à dire que tout successeur d'un agent est un agent, que la sorte d'un agent q successeur de p est contenue dans celle de p ...

Lemme 2.14
Si p est un agent et $p \xrightarrow{e} q$ alors q est un agent.

Preuve : Par induction sur la dérivation

Cas de base : Soit a.p un agent i.e VL(a.p) = Ø. Si $a.p \xrightarrow{a^*\sigma} p\sigma$ alors d'après la règle Pref σ : Var(a) —>T_C. Toutes les variables libres de p sont instanciées par σ car a.p est fermée et donc VL(p) ⊆ Var(a_{in}). Les autres cas sont obtenus par induction sur la longueur de la dérivation.de Nous traitons juste la composition parallèle. Les autres cas sont similaires.

Si $p\|q \xrightarrow{e} r$ et p||q un agent alors il y a trois possibilités.

1) $p \xrightarrow{e} p_1$ et $r = p_1\|q$. Par induction p_1 est fermé. Comme q est fermé par hypothèse donc $p_1\|q$ est aussi fermé.

2) $q \xrightarrow{e} q_1$ et $r = p\|q_1$. Par induction q_1 est fermé. Comme p est fermé par hypothèse donc $p\|q_1$ est aussi fermé.

3) $p \xrightarrow{e_1} p_1$, $q \xrightarrow{e_2} q_1$ avec $e = e_1 + e_2$ et $r = p_1\|q_1$. Par induction p_1 et q_1 sont fermés. Donc $p_1\|q_1$ est aussi fermé. □

Lemme 2.15

Soit $p \xrightarrow{e} q$ alors

- (1) dom(e) ⊆ $\mathcal{K}(p)$.
- (2) $\mathcal{K}(q)$ ⊆ $\mathcal{K}(p)$

Preuve

Par induction sur la dérivationn nous traitons deux cas : le préfixage et la somme non déterministe.

Cas de base : Par Pref avec p = a.q Soit $a.q \xrightarrow{a^*\sigma} p\sigma$. D'après la définition sur le calcul des sortes $\mathcal{K}(a.q) = \mathcal{K}(q) \cup dom(a)$ donc (1) et (2) sont vérifiées.

Les autres cas s'obtiennent par induction sur la dérivation.

Par Sum1 avec p= q+r . On a donc soit $q \xrightarrow{e} q_1$, soit $r \xrightarrow{e} r_1$. Comme $\mathcal{K}(q+r) = \mathcal{K}(q) \cup \mathcal{K}(r)$ donc (1) et (2) sont vérifiées. □

2.6 Bisimulation forte et agents HAL

Pour capter le comportement opérationnel des agents capables d'envoyer et de recevoir des ensembles de valeurs , nous utilisons la notion de step bisimulation introduite dans le chapitre 1 que nous

appellons simplement bisimulation. Nous montrons que c'est une congruence relativement aux opérateurs de notre algèbre.

Cette équivalence est souvent choisie pour comparer des processus parallèles pour les raisons suivantes :

- Elle a un sens intuitif suffisamment clair.
- Elle est suffisamment fine.
- Elle admet une technique de preuve élégante.

Cette dernière caractéristique est trés utile et nous l'explorerons pour montrer les proprités de congruence de la bisimulation forte. Mais avant tout remarquons que le renommage de variables liées respecte la bisimulation forte. Si nous définissons par $E \approx_\alpha F$ pour exprimer que E et F sont équivalents modulo l'α - conversion alors on a la proposition suivante :

Proposition 2.16

$p \approx_\alpha q \Rightarrow p \approx_B q$

Preuve :

Il suffit de montrer que $\approx_\alpha$ est une bisimulation. ❐

Proposition 2.17 (Propriétés de congruence)

La relation d'équivalence $\approx_B$ est une congruence relativement aux opérateurs de HAL sur les agents, c'est à dire : avec les agents p, q et r

$a.p \approx_B a.q$ si $p \sim q$	
$p+r \approx_B q+r$ et $r+p \approx_B r+q$	Pour tout p et tout r.
$p\|r \approx_B q\|r$ et $r\|p \approx_B r\|q$	Pour tout p, tout q et tout r.
$p[\![k,l]\!] \approx_B q[\![k,l]\!]$	Où k et l sont des noms de portes.
$p\lceil U \approx_B q\lceil U$	Où U est un sous ensemble de K
$p[f] \approx_B q[f]$	Où f est une fonction de renommage.

Jusque là nous avons défini la bisimulation sur les agents HAL. Il est possible de l'étendre aux expressions. Nous définissons une equivalence notée ~ sur les expressions et nous montrerons que c'est une congruence relativement<<n aux opérateurs de notre algèbre. Notons que ~ est déjà utilisée dans la définition précédente pour l'opérateur de préfixage.

Définition 2.18: (~ sur les expressions)

Soient p et q deux expressions alors

$p \sim q$ ssi $p\sigma \sim_B q\sigma \quad \forall \sigma \quad X \longrightarrow T_C$ et telle que $VL(p) \cup VL(q) \subseteq X$

Proposition 2.19

Soient p et q deux expressions HAL telles que $p \approx q$ alors

$a.p \sim a.q$	
$p+r \sim q+r$ et $r+p \sim r+q$	Pour tout p et tout r.
$p\|r \sim q\|r$ et $r\|p \sim r\|q$	Pour tout p, tout q et tout r.
$p[\![k,l]\!] \sim q[\![k,l]\!]$	Où k et l sont des noms de portes.
$p\lceil U \sim q\lceil U$	Où U est un sous ensemble de $\mathcal{K}$.
$p[f] \sim q[f]$	Où f est une fonction de renommage.

Preuve

On traite l'opérateur de base ou préfixage et l'opérateur de composition parallèle. Les autres cas se traitent de manière similaire.

1) Préfixage. Nous avons $p \sim q$ et nous voulons montrer que $a.p \sim a.q$ i.e que pour tout σ $X \rightarrow T_C$ et telle que $VL(p) \cup VL(q) \subseteq dom(\sigma)$, $p\sigma \approx_B q\sigma$. Considérons donc une substitution σ telle que $(a.p)\sigma$ et $(a.q)\sigma$ soient deux agents et montrons que $(a.p)\sigma \approx_B (a.q)\sigma$
Soit $\sigma : VL(p) \cup VL(q) \rightarrow T_C$ une substitution fermant a.p et a.q.
On a $(a.p)\sigma = (a_{in}+a_{out}\sigma').p\sigma'$ où $\sigma' = \sigma|_{V L (a.p).}$ (Les variables libres restreintes à celles de a.p)
$(a.p)\sigma = (a_{in}+a_{out}\sigma').p\sigma' \xrightarrow{(a_{in}+a_{out}\sigma')*\theta} p\sigma\theta$ et $(a.p)\sigma = a\sigma'.q\sigma' \xrightarrow{(a_{in}+a_{out}\sigma')*\theta} q\sigma\theta$. Comme par hypothèse $p \sim q$ donc $p\sigma\theta \approx_b q\sigma\theta$ et $(a_{in}+a_{out}\sigma')\theta.p\sigma\theta \approx_b (a_{in}+a_{out}\sigma')\theta.q\sigma\theta$ et finalement $(a.p)\sigma \approx_B (a.q)\sigma$. Ceci est vrai pour tout σ donc $a.p \sim a.q$.
2) Composition parallèle. Nous considérons $p \sim q$ et nous voulons montrer que $p\|r \sim q\|r$. Pour cela considérons une substitution σ fermant p q et r, donc fermant $p\|r$ et $q\|r$ et montrons que $(p\|r)\sigma \approx_B (q\|r)\sigma$.
Soit $(p\|r)\sigma \xrightarrow{e} s$ donc $p\sigma\|r\sigma \xrightarrow{e} s$ car $(p\|r)\sigma = p\sigma\|r\sigma$. On se retrouve dans le cas de la bisimulation sur les termes fermés et bien sûr $(p\|r)\sigma \approx_B (q\|r)\sigma$. Comme ceci est vrai pour tout σ donc $p\|r \sim q\|r$. □

Preuve (de la proposition 2.17) : Corollaire de 2.19 □

Proposition 2 .20

Modulo $\approx_B$ nous pouvons établir les propriétés suivantes.

(1) $p+q \approx_B q+p$
(2) $p+(q+r) \approx_B (p+q)+r$
(3) $p+nil \approx_B p$

(4) $p+p \approx_B p$

Lemme 2.21

Si l'on dénote pas Succ(p) l'ensemble de tous les suuccesseurs de p i.e.

$Succ(p) = \{<e, p> \mid p \xrightarrow{e} p'\}$ alors on a $Succ(p) = Succ(q) \Rightarrow p \approx_B q$

Nous allons utiliser ce lemme à chaque fois que l'on doit établir une relation de bisimulation entre deux processus ayant les mêmes successeurs.

Preuve (de la proposition 2.20)

La preuve découle directement du lemme 2.21 car en effet

$Succ(p+q) = Succ(q+p)$, $Succ(p+p) = Succ(p)$, $Succ(p+nil) = Succ(p)$ et $Succ((p+q)+r) = Succ(p+(q+r))$. □

Proposition 2.22

(1) $p \| q \approx_B q \| p$

(2) $p \| (q \| r) \approx_B (p \| q) \| r$

(3) $p \| nil \approx_B p$

Preuve

On prouve la propriété (2) en montrant que la relation $R2 = \{(p \| (q \| r), (p \| q) \| r)\}$ est une bisimulation.

Soit $(p \| (q \| r), (p \| q) \| r) \in R2$ et $(p \| (q \| r)) \xrightarrow{e} s$ alors trois cas sont possibles :

1) $p \xrightarrow{e} p_1$ et $s = p_1 \| (q \| r)$. Par Par1 deux fois on obtient $(p \| q) \| r) \xrightarrow{e} (p_1 \| q) \| r)$ et clairement $(p_1 \| (q \| r), (p1 \| q) \| r)) \in R2$.

2) Soit $(q \| r) \xrightarrow{e} s_1$ et $s = p \| s_1$. Il y'a à ce niveau là trois sous cas.

   2.1) $q \xrightarrow{e} q_1$ et $s_1 = (q_1 \| r)$ alors par Par1 $(p \| q) \xrightarrow{e} (p \| q_1)$. En appliquant une deuxième fois Par1 on a $((p \| q) \| r) \xrightarrow{e} ((p \| q_1) \| r))$et bien sur $(p \| (q_1 \| r), (p \| q_1) \| r) \in R2$.

   2.2) Si . Si $r \xrightarrow{e} r_1$: Similaire au cas précédent.

   2.3) $q \xrightarrow{e} q_1$ et $r \xrightarrow{e} r_1$ avec $e = e_1 + e_2$ et $s_1 = (q_1 \| r_1)$: Par Par1 deux fois on a $((p \| q) \| r) \xrightarrow{e} ((p \| q_1) \| r_1))$et bien sur $(p \| (q_1 \| r_1), (p \| q_1) \| r_1) \in R2$.

3) Soit $p \xrightarrow{e_1} p_1$ et $(q \| r) \xrightarrow{e_2} s1$ avec $e = e_1 + e_2$ et $s = (p_1 \| s_1)$. En décomposant $(q \| r) \xrightarrow{e_2} s_1$ et en appliquant Par2 on obient le résultat attendu.

La propriété de transfert est satisfaite dans tous les cas donc R2 est une bisimulation.

Pour montrer les propriétés (1) et (3) il suffit de prouver que les relations R1 et R3 sont des bisimulations où :
R1 = {(p||q, q||p}et R3 = {(p||nil , p} ❐

La proposition suivante donne certaines propriétés satisfaites par l'opérateur des restriction. Cet opérateur est distributif par rapport à la somme et à la composition d'agents. L'ordre dans lequel les portes sont cachées n'a pas d'importance. Enfin quand on veut cacher un ensemble de portes U d'un agent a.p cela revient à cacher ces portes dans p si aucune porte de a n'appartient à U. Dans le cas négatif l'opération retourne l'agent nil.

Proposition 2.23

(1) $(p+q)\lceil U \approx_B p\lceil U+q\lceil U$
(2) $(p\|q)\lceil U \approx_B p\lceil U\| q\lceil U$
(3) $p\lceil U\lceil V \approx_B p\lceil U\cup V$
(4) $p\lceil U\lceil V \approx_B p\lceil V\lceil U$
(5) $(a.p)\lceil U \approx_B a.p\lceil U$ si $dom(a)\cap U = \emptyset$.
(6) $(a.p)\lceil U \approx_B$ nil sinon.

Preuve

Pour montrer (1), (3), (5) et (6) il suffit de constater que Succ($(p+q)\lceil U$) = Succ($p\lceil U+q$) , Succ($p\lceil U\lceil V$) = Succ($p\lceil U\lceil V$) , Succ($(a.p)\lceil U$) = Succ($a.p\lceil U$). (4) est un corollaire de (3). Pour montrer (2) il faut montrer que la relation R2 est une bisimulation forte où :
R2= {($(p\|q)\lceil U$, $p\lceil U\|q\lceil U$)} ❐

Comme pour la restriction, le renommage est distributif par rapport à l'opérateur + et à l'opérateur parallèle.

Proposition 2.24

(1) $(p+q)[f] \approx_B p[f] + q[f]$
(2) $(p\|q)[f] \approx_B p[f] \| q[f]$
(3) $p[Id] \approx_B p$
(4) $p[f][g] \approx_B p[g\circ f]$
(5) $a.p[f] \approx_B f(a).p[f]$

Preuve :
Pour montrer (2) il faut prouver que la relation R2 suivante est une bisimulation forte où :
R2 = { ((p||q)[f], p[f] || q[f]) } .
Pour les autres cas il faut se rammener à l'utilisation du lemme 2.21. ❐

La proposition suivante établit des propriétés satisfaites par l'opérateur de connexion. On remarque que l'opérateur de connexion n'est pas distributif par rapport à l'opérateur de composition d'agents.

Proposition 2.25

(1) $(p+q)[\![k,l]\!] \approx_B p[\![k,l]\!] + q[\![k,l]\!]$

(2) $p\lceil U[\![k,l]\!] \approx_B p[\![k,l]\!]\lceil U$ si $\{k,l\} \cap U = \emptyset$

Il reste à montrer que la définition récursive d'agents préserve la bisimulation forte.Avant de montrer cela, nous allons d'abord situer le cadre de travail en fixant quelques notations.

Soient $A \Leftarrow p$ et $B \Leftarrow q$ deux définitions d'agents. De manière générale p peut utiliser A et q peut utiliser B.

Exemple : $A \Leftarrow k_1?(s(y)).\ (k_2!(y).nil \parallel k_3?(z).A)$

$B \Leftarrow k'_1?(y).\ k'_2!(s(y);nil + k'_3?(s(s(y)).B$

Nous écrivons, dans ce qui suit p comme une expression E contenant la variable X et q comme une expression F contenant la variable X. X étant mis à la place de A et B respectivement dans p et q.

Si nous revenons à l'exemple précédent nous avons :

$A \Leftarrow k_1?(s(y)).\ (k_2!(y).nil \parallel k_3?(z).X)$

$B \Leftarrow k'_1?(y).\ k'_2!(s(y);nil + k'_3?(s(s(y)).X.$

On note de manière générale : $A \Leftarrow E\{A\backslash X\}$ et $B \Leftarrow F\{B\backslash X\}$ (*)

Dans cette section nous voulons montrer que si A et B sont deux définitions ayant la forme * et si E et F sont bisimilaires alors A et B le sont aussi.
Mais E et F contiennent la variable X où X peut être n'importe quel agent. Nous définissons alors la bisimulation entre E et F de la manière suivante :

Définition 2.26 : (≈)
Soient E et F contenant la variable X au plus alors $E \approx F$ ssi pour tout $p \in P$, $E\{p/X\} \approx F\{p/X\}$.

Maintenant nous sommes en mesure d'établir la proposition assurant que la définition récursive préserve la bisimulation forte.

Proposition 2.27

Soient E et F contenant la variable libre X au plus et soient A $\Leftarrow$E{A/X} et B $\Leftarrow$ F{B/X} et E$\approx$F donc A $\approx_B$ B.

Preuve

Pour prouver cette proposition il suffit de montrer que S est une bisimulation forte modulo $\approx_B$ où S = { G{A/X} , G{B/X} : G contient au plus la variable X}

c.à-d. il faut montrer que Si G{A /X} $\xrightarrow{a}$ p' alors il existe q' et q" tel que G{B /X} $\xrightarrow{a}$ q" $\approx_B$ q' et (p', q') $\in$ S.

La preuve se fait par induction sur la dérivation et par cas sur la stucture de G.

1) G= X donc G{A/X} = A . Soit A $\xrightarrow{e}$p' alors par Def il existe E{A / X} $\xrightarrow{e}$p' . Par induction on a E{B / X} $\xrightarrow{e}$q " $\approx_B$ q' et (p', q') $\in$ S. Mais E $\approx$ F donc F{B / X} $\xrightarrow{e}$ q'" et q'"$\approx_B$ q' et (p', q') $\in$ S. Comme B $\Leftarrow$ F{B/X} donc G{b/X} =B —>q " $\approx_B$ q' et (p',q') $\in$ S et donc la propriété de transfert est vérifiée.

2) G= a.G'. G{A/X}= a.G'{A/X} car X ne peut être dans a. Soit G{A / X} $\xrightarrow{a*\sigma}$ p' donc p' = G'{AX}σ = G"{A / X} où G" $\equiv$ G'σ où σ est une substitution close σ : Var(a) —>T_C. Or G{b / X} $\xrightarrow{a*\sigma}$ q' donc q' = G'{B / X}σ $\equiv$ G"{B / X} où G" $\equiv$ G'σ .
Clairement (G"{A / X}, G"{B / X})$\in$ S . Donc la propriété de transfert est vérifiée.

3) G= G1+G2. La preuve est trés simple. Il suffit de considérer que (G1+G2){A/X} = G1{A/X} + G2{A/X} .

4) G=G1|| G2. On veut montrer que ((G1|| G2){A/X}, (G1|| G2){B/X}) $\in$ S. Remarquons que (G1|| G2){A/X} $\equiv$ G1{A/X} || G2{A/X}.
Soit (G1|| G2){A/X} $\xrightarrow{e}$p' alors G1{A/X} || G2){A/X} $\xrightarrow{e}$p' . Il y'a trois cas possibles.
(i) G1{A/X} $\xrightarrow{e}$p'$_1$ et p' = p'$_1$ || G2{A/X}.Par induction il existe G1{B/X} $\xrightarrow{e}$ q"$_1\approx_B$ q'1 et tel que (p'$_1$, q'$_1$) $\in$ S. Par Par1 on a G1{B/X} || G2{B/X} —>q "$_1$|| G2{B/X} $\approx_B$ q'1||G2{B/X}.

Il reste à montrer que (p', q') ∈ S.
Soit H1 tel que $p'_1 \equiv H1\{A/X\}$ et $q'_1 \equiv H1\{B/X\}$. Notons par H = H1 ‖ G2. Nous avons alors (p', q') = (H{A/X} , H{B/X}) ∈ S.

(ii) $G2\{A/X\} \xrightarrow{e} q'_1$: Se traite comme le cas précédent.

(iii) $G1\{A/X\} \xrightarrow{e_1} p'_1$ et $G2\{A/X\} \xrightarrow{e_2} q'_1$ avec $e = e_1 + e_2$. et $p' = p'_1 \| q'_1$. Par induction il existe $G1\{B/X\} \xrightarrow{e1} p''_2 \approx_B p'_2$ avec $(p'_1, p'_2) \in S$ et $G2\{B/X\} \xrightarrow{e2} q'''_2 \approx_B q'_2$ avec $(q'_1,q'_2) \in S$. Par Par3 on a $(G1 \| G2)\{A/X\} \xrightarrow{e} q'$ avec $q' = p'_2 \| q'_2$. Il reste à montrer que $((p'_1 \| q'_1),(p'_2 \| q'_2)) \in S$.
Soit $P'_i = H_i\{A/X\}$ et $q'_i = Hi\{B/X\}$et $H = H_1 \| H_2$. Nous avons alors
(p',q') = (H{A/X},H{A/X}) ∈ S. Le cas $(G1 \| G2)\{B/X\} \xrightarrow{e} q'$ est similaire car la relation S est symétrique.
La propriété de transfert est vérifiée

4) G = G'[[k,l]]. G{A/X} = G'[[k,l]] {A/X} = G'{A/X} [[k,l]] . On veut donc montrer que (G'[[k,l]] {A/X}, G'[[k,l]] {A/X}) ∈ S .
Soit $G'\{A/X\}$ [[k,l]] $\xrightarrow{e} p'$ (p' = p [[k,l]]pour un processus p) alors il y'a deux cas possibles.
(i) $G'\{A/X\} \xrightarrow{e} p$. Par induction $G'\{B/X\} \xrightarrow{e} q' \approx_B q$ et$(p,q) \in$ S.Par Con1 on obtient $G'\{B/X\}[[k,l]] \xrightarrow{e} q'$ [[k,l]] $\approx_b q[[k,l]]$. Il reste à montrer que (p [[k,l]], q [[k,l]]) ∈ S. Comme $(p,q) \in S$ alors il existe un processus H tel que p ≡ H{A/X} et q ≡ H{B/X} alors p [[k,l]] = H{A/X} [[k,l]] doncp [[k,l]]= H [[k,l]]{A/X} ≡ H' {A/X} où H' ≡ H [[k,l]]. De la même façon on a q [[k,l]] = H'{B/X} et d'après la définition de S on a (H' {A/X} , H' {B/X})∈ S.
(ii) $G'\{A/X\} \xrightarrow{eU\ \{k(v),\ l(v)\}} p'$. par induction $G'\{B/X\} \xrightarrow{eU\ \{k(v),\ l(v)\}} q' \approx_B q$ et(p,q)∈ S.
Par Con2 , on obtient $G'\{B/X\}[[k,l]] \xrightarrow{e} q'$ [[k,l]] $\approx_B q[[k,l]]$. Par le même raisonnement que dans (i) on a (p[[k,l]] , q[[k,l]]) ∈ S.

La propriété de transfert étant partout vérifiée donc S est une bisimulation forte modulo $\approx_B$ et donc une bisimulation forte. ❐

Chapitre 3

Un Modèle de calcul de Processus avec Partage de Variables

3.1 Introduction

Dans le chapitre précédent, nous avons fait une étude de l'algèbre de processus HAL avec l'hypothèse que la communication se fait par envoi de messages. Ceci nous a ammené à traiter la variable comme dans CCS et toutes les autres algèbres de processus classiques.

Dans ce chapitre nous considérons une deuxième interprétation de HAL en supposant que la communication se fait à la fois par envoi de messages et par partage de variables [DMM 81, Wol 89, GL 90]]. Nous traitons la variable dans ce cas comme une variable logique. Nous allons suivre la même approche que dans le chapitre 2 pour définir ce modèle.
Dans la section 3.2, nous rappelons la syntaxe utilisée pour écrire les processus. Le fait d'interpréter la variable comme variable globale modifie quelque peu la syntaxe présentée dans le chapitre 2. Dans la section 3.3 nous proposons une sémantique opérationnelle non close. La sémantique close en est un cas particulier . Ensuite dans la section 3.4 nous définissons une bisimulation non close sur les agents HAL et étudions ses propriétés de congruence relativement aux opérateurs de HAL.

3.2 Le langage considéré

3.2.1 Quelques notions de base

- *Portes* : $\mathcal{K}$ est un ensemble d'identificateurs représentant des noms de portes.

- *Offres* : Soit k un nom de porte et un terme de T_C , on définit alors une offre d'un terme t sur une porte k comme k(t)

- *Ensemble d'offres* : Soient $k_1, k_2, ..., k_n$ n noms de portes distincts et $t_1, t_2, ..., t_n$ n termes. $a = \{k_1(t_1), ..., k_n(t_n)\}$ dénote l'ensemble de n offres synchrones de ti sur k_i (i=1..n). L'ensemble vide d'offres est dénoté par τ. Formellement, un ensemble d'offres a est une fonction partielle de $\mathcal{K}$ vers $T_{C,X}$. On note par $\mathcal{A} = \{ a, b, ...c, ... \}$ l'ensemble des ensembles d'offres.

- *Evénement* . Soient $k_1, k_2, ..., k_n$: n noms de portes et $v_1, v_2, ..., v_n$ n valeurs de T_C , $e = \{k_1(v_1), k_2(v_2), ..., k_n(v_n)\}$ est un événement. Un événement est donc toute instance close d'un ensemble d'offres a. Formellement un événement e est défini comme une fonction partielle de $\mathcal{K}$ vers T_C. On note par $E = \{e, e_1, ... \}$ l'ensemble des événements.

3.2.2 Agents HAL

Les agents HAL exécutent des transitions atomiques étiquetées par des événements e appartenant à E. L'ensemble d'agents HAL est le plus petit ensemble généré par la grammaire suivante :

$$p,q ::= nil \mid a.p \mid p+q \mid p \| q \mid p[\![k,l]\!] \mid p\lceil U \mid p[f] \mid id \qquad (*)$$

où a est un ensemble d'offres, k et l des noms de portes, id un nom d'agent et f une fonction de renommage.

Il n' y a pas dans ce cas de notion d'agents et de processus puiqu'il n'y a pas de notions de variables libres et et liéees. On appellera alors indifféremment dans ce chapitre agent ou processus tout terme bien formé engendré par (*) . La notion de terme bien formé est déjà défini dans le chapitre 2. La notion de sorte est aussi la même que celle définie dans le chapitre 2.
Avant de passer à l'aspect sémantique de ce modèle, nous donnons un petit exemple permettant de mieux comprendre les différences entre ce dernier et celui que nous avons présenté dans le précédent chapitre.

Soit p = {A(x)}.{B(x)}.nil

- Ce processus est syntaxiquement correct dans HAL avec partage de variables. La variable x qui se trouve dans l'offre A(x) est la même que celle qui se trouve dans l'offre B(x).

- Dans HAL avec envoi de messages, ce processus est syntaxiquement incorrect, car les ensembles d'offres ne sont pas orientées. En orientant les ensembles d'offres, nous avons alors deux façons possibles d'écrire p comme un agent correct dans ce modèle.
  . p = {A?(x)}.{B!(x)}. nil : La variable x est la même dans les deux ensembles d'offres car elle est liée par la porte A?.

  . p = {A?(x)}.{B?(x)}. nil : La variable x dans ce cas n'est plus la même dans les deux ensembles d'offres et on aurait pu tout aussi bien écrire p = {A?(x)}.{B?(y)}. nil où x et y sont des variables quelconques.

3.3 Sémantique du modèle avec partage de variables

3.3.1 Introduction intuitive

Avant de donner une sémantique formelle, nous allons essayer de donner l'idée intuitive du comportement de nos processus. Les variables apparaissant dans les ensembles d'offres sont toutes supposées être des variables partagées ou globales. Le comportement d'un processus est alors défini seulement dans un environnement σ ou environnement global. σ est une substitution idempotente permettant d'enregistrer la façon dont les variables de X ont été instanciées, de la même manière qu'un environnement Prolog par exemple .
Les transitions de base de nos processus ont la forme $<\sigma,p> \xrightarrow[\theta]{e} <\sigma', q>$. Celle ci se lit : Dans l'environnement σ le processus p peut effectuer les instanciations décrites par l'assignation θ donnant alors naissance à l'action visible e et se transforme lui même en q dans l'environnement σ'.Puisque σ' n'est rien d'autre que $\sigma\theta$ on peut alors écrire la transition de base plus simplement σ: $p \xrightarrow[\theta]{e} q$.

3.3.2 Définition informelle des agents

Nous donnons le sens informel pour l'opérateur de préfixage. L'opérateur de préfixage est l'opérateur de base pour la construction des processus. Tous les changements qui peuvent être apportés au niveau sémantique se rapportent essentiellement à cet opérateur. Les autres opérateurs sont définis informellent comme précédemment.

- *Préfixage* : Dans l'environnement σ, le processus a.p propose la communication $a\sigma$. Toute instance de celle ci est possible à condition qu'elle n'introduise pas et qu'elle n'instancie pas des variables déjà instanciées par σ. Nous formalisons ce qui précède par la notion de *compatibilité*. Formellement la règle de préfixage est :

$$\textbf{Pref} \quad \frac{}{\sigma : a.p \xrightarrow[\theta]{a\sigma\theta} p} \qquad \text{Si } \theta \textit{ compatible } \text{avec } \sigma.$$

Remarque : L'environnement σ dans lequel sont interprétés les agents représente un état mémoire. Mais il ne faut pas voir cet état mémoire comme dans le cas des langages impératifs (ex pascal) où la variable est représentée par une cellule dans laquelle on peut lire, écrire mais aussi écraser son ancien contenu par une nouvelle écriture. Il faut plutôt l'imaginer comme une substitution Prolog :i.e. une variable x est introduite une fois et elle ne peut plus être réinstanciée. Elle peut bien sur être raffinée. Par exemple : si x a reçu la valeur f(y) et que y reçoit plus tard la valeur g(g(z)) alors x est raffinée en f(g(g(z))).

Définition 3.1 (Définition formelle de la notion de compatibilité)
Soient σ et θ deux substitutions idempotentes. θ est dite compatible avec σ si et seulement si $dom(\sigma) \cap dom(\theta) = \emptyset$ et $dom(\sigma) \cap ran(\theta) = \emptyset$.

Lemme 3.2
Si θ est compatible avec σ alors $\sigma\theta$ est une substitution idempotente .

Preuve
Nous allons montrer que $dom(\sigma\theta) \cap ran(\sigma\theta) = \emptyset$.
Considérons $x \in dom(\sigma\theta)$ alors il existe deux possibilités.

- Si $x \in dom(\sigma)$ alors $x \notin ran(\sigma)$ et $x \notin ran(\theta)$ et donc $x \notin ran(\sigma\theta)$ puisque $ran(\sigma\theta) \subseteq ran(\sigma) \cup ran(\theta)$.

- Si $x \notin dom(\sigma)$ alors $x \in dom(\theta)$ impliquant $x \notin ran(\sigma\theta)$. □

Remarque : Ce lemme prouve que dans **Pref**, après avoir exécuté une transition , on atteint bien un environnement correct i.e. une substitution idempotente.

Lemme 3.3
Si μ est compatible avec σ et θ compatible avec $\sigma\mu$ alors $\mu\theta$ est compatible avec σ.

Preuve :
Nous allons montrer que $dom(\sigma) \cap dom(\mu\theta) = \emptyset$ et $dom(\sigma) \cap ran(\mu\theta) = \emptyset$.

- Si $x \in dom(\sigma)$ alors $x \notin dom(\mu)$. Mais $x \in dom(\sigma\mu)$ d'où $x \notin dom(\theta)$ et alors $x \notin dom(\mu\theta)$.
- Si $x \in dom(\mu\theta)$ alors soit $x \in dom(\mu)$ soit $x \in dom(\theta)$. Si $x \in dom(\mu)$ alors $x \notin dom(\sigma)$. Si $x \in dom(\theta)$ alors $x \notin dom(\sigma\mu)$ et alors $x \notin dom(\sigma)$.
- Si $x \in dom(\sigma)$ alors $x \notin ran(\mu)$. Mais $x \in dom(\sigma\mu)$ d'où $x \notin ran(\theta)$ et alors $x \notin ran(\mu\theta)$. Si $x \in ran(\mu\theta)$ alors soit $x \in ran(\mu)$ ou $x \in ran(\theta)$. Si $x \in ran(\mu)$ alors $x \notin dom(\sigma)$. Si $x \in ran(\theta)$ alors $x \notin dom(\sigma\mu)$ et alors $x \notin dom(\sigma)$. □

3.3.3 Sémantique opérationnelle

Nous allons maintenant définir de manière usuelle la sémantique opérationnelle en termes de systèmes de transition étiqueté.

Le système de transition considéré dans ce cas est de la forme $S = (Q, E, \Sigma_i, \longrightarrow)$ où :

- Q est l'ensemble des paires de la forme $<\sigma, p>$ avec $p \in \mathcal{P}$ et $\sigma \in \Sigma_i$.
- E est l'ensemble des événements.
- Σ_i est l'ensemble des substitutions idempotentes.
- $\longrightarrow \subseteq Q \times E \times \Sigma_i \times \mathcal{P}$ est la relation de transition.

La relation de transition est définie par un ensemble de règles. Nous appelons dans la suite une spécification un tel ensemble. Nous proposons deux spécifications S_1 et S_2 pour HAL et nous montrerons leur équivalence.

Définition 3.4 (Spécification S_1)
L'ensemble S_1 des règles de transition est donné par figure 3.1.

Pref $\dfrac{}{\sigma : a.p \xrightarrow[\theta]{a\sigma\theta} p}$ Si θ compatible avec σ

Sum1 $\dfrac{\sigma: p \xrightarrow[\theta]{e} r}{\sigma: p+q \xrightarrow[\theta]{e} r}$ **Sum2** $\dfrac{\sigma: q \xrightarrow[\theta]{e} r}{\sigma: p+q \xrightarrow[\theta]{e} r}$

Par1 $\dfrac{\sigma: p \xrightarrow[\theta]{e} r}{\sigma: p \parallel q \xrightarrow[\theta]{e} r \parallel q}$

Par2 $\dfrac{\sigma: q \xrightarrow[\theta]{e} r}{\sigma: p \parallel q \xrightarrow[\theta]{e} p \parallel r}$

Par3 $\dfrac{\sigma: p \xrightarrow[\theta]{e_1} r \quad \sigma: q \xrightarrow[\theta]{e_2} s}{\sigma: p \parallel q \xrightarrow[\theta]{e_1 \cup e_2} r \parallel s}$

Res $\dfrac{\sigma: p \xrightarrow[\theta]{e} r}{\sigma: p\lceil U \xrightarrow[\theta]{e} r\lceil U}$ Si $U \cap dom(e) = \emptyset$.

Con 1 $\dfrac{\sigma : p \xrightarrow[\theta]{e} q}{\sigma: p[\![k,l]\!] \xrightarrow[\theta]{e} q[\![k,l]\!]}$

Con 2 $\dfrac{\sigma : p \xrightarrow[\theta]{e \cup \{k(t),\, l(t)\}} q}{\sigma: p[\![k,l]\!] \xrightarrow[\theta]{e} q[\![k,l]\!]}$

Rel $\dfrac{\sigma : p \xrightarrow[\theta]{e} q}{\sigma : p[f] \xrightarrow[\theta]{f(e)} q[f]}$

Def $\dfrac{\sigma : p \xrightarrow[\theta]{e} q}{\sigma : Id \xrightarrow[\theta]{e} q}$ (1)

(1) Si $Id \Leftarrow p$ est une définition de p.

figure3.1

Remarque : Les règles **Par3** et **Con2** peuvent paraitre trop restrictives. **Par3** exige que, pour se synchroniser deux processus doivent avoir le même assignement θ, pendant que **Con2** exige que, pour communiquer, deux processus doivent être prêts à échanger le même terme. Nous verrons plus loin que l'utilisation de ces règles ne limite en aucun cas la puissance de notre spécification. Dans ce qui suit nous allons d'abord contrôler la bonne formation de nos transitions par la proposition suivante :

Proposition 3.5

Si $\sigma : p \xrightarrow[\theta]{e} q$ alors θ est compatible avec σ et $\sigma\theta$ (le nouvel environnement) est une substitution idempotente.

Preuve

La preuve est faite par induction sur la dérivation de $\sigma : p \xrightarrow[\theta]{e} q$. Le cas de base est obtenu par le lemme 3.2 (la compatibilité étant assurée par la condition de **Pref**) et les étapes inductives sont immédiates. ❐

Nous avons déjà mentionné que **Par3** et **Con2** pouvaient paraitre trop restrictives. Des règles plus générales pour la composition synchrone et la communication sont données par la spécification S_2 suivante :

Définition 3.6 (Spécification S_2)
L'ensemble des règles S2 est donné par S_1 - { Par3, Con2} ∪ {Par3', Con2'}

$$\textbf{Par3'}\quad \frac{\sigma: p \xrightarrow[\theta_1]{e_1} r \qquad \sigma: q \xrightarrow[\theta_2]{e_2} s}{\sigma: p \parallel q \xrightarrow[\theta_1 \cup \theta_2]{e_1\theta_2 \cup e_2\theta_1} r \parallel s} \quad \text{Si } \theta 1 \text{ et } \theta 2 \text{ sont } \textit{cohérentes}$$

$$\textbf{Con2'}\quad \frac{\sigma: p \xrightarrow[\theta]{e \cup \{k(t1), l(t2)\}} q}{\sigma: p[\![k,l]\!] \xrightarrow[\theta\mu]{e\mu} q[\![k,l]\!]} \quad \mu = mgu(t_1, t_2)$$

figure 3.2

Remarque :
- **Par3'** exprime que dans p ∥ q, p et q peuvent s'exécuter simultanément en proposant des assignements θ1 et θ2 *cohérents* (i.e. θ_1 et θ_2 coïncident sur leur domaine commun, $dom(\theta_1) \cap ran(\theta_2) = \emptyset$ et $dom(\theta_2) \cap ran(\theta_1) = \emptyset$).

- **Con2'** autorise la communication entre deux processus ssi ils sont prêts à échanger des termes unifiables. Nous montrons dans ce qui suit que effectivement les spécifications S_1 et S_2 sont équivalentes comme le prouve la proposition suivante.

Proposition 3.7 (Equivalence des spécifications S_1 et S_2)
S_1 et S_2 sont équivalentes i.e. $S_1 \vdash \sigma : p \xrightarrow[\theta]{e} q$ ssi $S_2 \vdash \sigma : p \xrightarrow[\theta]{e} q$

La preuve de cette proposition fait appel aux lemme et propositions suivants.

Proposition 3.8
Si $S_1 \vdash \sigma : p \xrightarrow[\theta]{e} q$ et μ une substitution telle que θμ est compatible avec σ alors
$S_1 \vdash \sigma: p \xrightarrow[\theta\mu]{e\mu} q$.

Preuve

La preuve est faite par induction sur la dérivation de $\sigma : p \xrightarrow[\theta]{e} r$ (ici tout est dans S_1).

- **Pref.** Si σ: a.p $\xrightarrow[\theta]{a\sigma\theta}$ p par **Pref** et μ tel que $\theta\mu$ idempotente et compatible avec σ alors $dom(\sigma) \cap dom(\theta\mu) = \emptyset$ et $dom(\sigma) \cap ran(\theta\sigma) = \emptyset$ et donc σ: a.p $\xrightarrow[\theta\mu]{a\sigma\theta\mu}$ q .

- **Sum1.** Si $\sigma : p+q \xrightarrow[\theta]{e} r$ par **Sum1** alors $\sigma : p \xrightarrow[\theta]{e} r$. Par hypothèse d' induction nous déduisons que σ: p $\xrightarrow[\theta\mu]{e\mu}$ r et $\sigma : p+q \xrightarrow[\theta\mu]{e\mu} r$ en utilisant **Sum1**. Le cas **Sum2** est similaire.

- **Par1.** Si $\sigma : p \| q \xrightarrow[\theta]{e} r \| s$ par **Par1** alors $\sigma : p \xrightarrow[\theta]{e} r$ et s = q. Par hypothèse d'induction nous avons $\sigma : p \xrightarrow[\theta\mu]{e\mu} r$ et $\sigma : p \| q \xrightarrow[\theta\mu]{e\mu} r \| q$ par **Par1**. Le cas **Par2** est similaire.

- **Par3.** Si $\sigma : p \| q \xrightarrow[\theta]{e} r \| s$ par **Par3** alors $\sigma : p \xrightarrow[\theta]{e_1} r$ et σ: q $\xrightarrow[\theta]{e_2}$ s. Par hypothèse d'inductuion $\sigma : p \xrightarrow[\theta\mu]{e_1\mu} r$ et $\sigma : q \xrightarrow[\theta\mu]{e_2\mu} s$ et par **Par3** $\sigma : p \| q \xrightarrow[\theta\mu]{(e_1 \cup e_2)\mu} r \| s$.

- **Res.** Si $\sigma : p \lceil U \xrightarrow[\theta]{e} q \lceil U$ par **Res** alors $\sigma : p \xrightarrow[\theta]{e} q$ avec $U \cap dom(e) = \emptyset$. Par hypothèse d'induction $\sigma : p \xrightarrow[\theta\mu]{e\mu} q$ et alors $\sigma : p \lceil U \xrightarrow[\theta\mu]{e\mu} q \lceil U$ par **Res** car $dom(e\mu) = dom(e)$.

- **Con1.** Si $\sigma : p[\![k, l]\!] \xrightarrow[\theta]{e} q[\![k, l]\!]$ par **Con1** alors $\sigma : p \xrightarrow[\theta]{e} q$ et par hypothèse d'induction on a $\sigma : p \xrightarrow[\theta\mu]{e\mu} q$ et alors $p[\![k, l]\!] \xrightarrow[\theta\mu]{e\mu} q[\![k, l]\!]$ en utilisant **Con1**.

- **Con2.** Si $\sigma : p[\![k, l]\!] \xrightarrow[\theta]{e} q[\![k, l]\!]$ par **Con2** alors $\sigma : p \xrightarrow[\theta]{(e \cup \{k(t), l(t)\})} q$ et par hypothèse d'induction $\sigma : p \xrightarrow[\theta\mu]{(e \cup k(t), l(t))\mu} q$. Comme $(e \cup \{k(t), l(t)\})\mu = e\mu \cup \{k(t\mu), l(t\mu)\}$ alors par **Con2** $\sigma : p[\![k, l]\!] \xrightarrow[\theta\mu]{e\mu} q[\![k, l]\!]$.

- **Def** . Si $\sigma : Id \xrightarrow[\theta]{e} q$ par **Def** alors $\sigma : p \xrightarrow[\theta]{e} q$ avec $Id \Leftarrow p$. Par hypothèse d'induction, on a $\sigma : p \xrightarrow[\theta\mu]{e\mu} q$ et par **Def** on obtient alors $\sigma : Id \xrightarrow[\theta\mu]{e\mu} q$.
- **Ren**. Si $\sigma : p[f] \xrightarrow[\theta]{e} q[f]$ par **Ren** alors $\sigma : p \xrightarrow[\theta]{e'} q$ avec $e = f(e')$. Par hypothése d'induction on a $\sigma : p \xrightarrow[\theta\mu]{e'\mu} q$. Par **Ren** on obtient finalement $\sigma : p[f] \xrightarrow[\theta\mu]{e\mu} q[f]$ car $f(e'\mu) = f(e')\mu = e\mu$. ❐

Proposition 3.9

Si $\sigma : p \xrightarrow[\theta]{e} q$ alors $Var(e) \cap dom(\sigma\theta) = \emptyset$.

Preuve

La preuve est faite par induction sur la dérivation de $\sigma : p \xrightarrow[\theta]{e} q$.

- Cas de base. Si $\sigma: a.p \xrightarrow[\theta]{a\sigma\theta} p$ par **Pref** alors $Var(a\sigma\theta) \cap dom(\sigma\theta) = \emptyset$ car $Var(a\sigma\theta) \subseteq ran(\sigma\theta)$ et $dom(\sigma\theta) \cap ran(\sigma\theta) = \emptyset$ par hypothèse.

- Induction . Les étapes inductives sont immédiates. Nous prouvons uniquement le cas de l'opérateur de composition parallèle.

  Soit $\sigma : p \| q \xrightarrow[\theta]{e} r$ alors il y a trois cas possibles.

  1) $\sigma : p \xrightarrow[\theta]{e} p'$ et $r = p' \| q$. Par hypothèse d'induction nous avons $Var(e) \cap dom(\sigma\theta) = \emptyset$.

  2) $\sigma : q \xrightarrow[\theta]{e} q'$ et $r = p \| q'$. Par hypothèse d'induction nous avons $Var(e) \cap dom(\sigma\theta) = \emptyset$.

  3) $\sigma : p \xrightarrow[\theta]{e_1} p'$, $\sigma : q \xrightarrow[\theta]{e_2} q'$ et $r = p' \| q'$.Par hypothèse d'induction nous avons $Var(e_1) \cap dom(\sigma\theta) = \emptyset$ et $Var(e_2) \cap dom(\sigma\theta) = \emptyset$ et donc $Var(e) \cap dom(\sigma\theta) = \emptyset$ car $Var(e) = Var(e_1) \cup Var(e_2)$. ❐

Lemme 3.10

Si σ est une substitution idempotente et θ_1 et θ_2 sont cohérentes et compatibles avec σ alors

1. $\theta_1 \cup \theta_2$ est une substitution idempotente .

2. $\theta_1 \cup \theta_2$ est compatible avec σ.
3. $\theta_1 \cup \theta_2 = \theta_1\theta_2 = \theta_2\theta_1$.
4. $\sigma(\theta_1 \cup \theta_2)$ est une substitution idempotente.

Preuve

1. Il y a trois possibilités.

   - Si $x \in dom(\theta_1)$ alors $x(\theta_1 \cup \theta_2) = x(\theta_1)$ et $x(\theta_1 \cup \theta_2)(\theta_1 \cup \theta_2) = x(\theta_1)(\theta_1)$ car $ran(\theta_1) \cap dom(\theta_2) = \emptyset$ et alors $x(\theta_1 \cup \theta_2)(\theta_1 \cup \theta_2) = x(\theta_1)$ puisque θ_1 est idempotente.

   - Si $x \in dom(\theta_2)$: La preuve est similaire.

   - Si $x \notin dom(\theta_1 \cup \theta_2)$ alors $x(\theta_1 \cup \theta_2)(\theta_1 \cup \theta_2) = x(\theta_1 \cup \theta_2) = x$.

2. $dom(\theta_1 \cup \theta_2) \cap dom(\sigma) = \emptyset$ et $dom(\theta_1 \cup \theta_2) \cap ran(\sigma) = \emptyset$ car θ_1 et θ_2 sont compatibles avec σ.

3 $x(\theta_2) = x$ et $x(\theta_2\theta_1) = x(\theta_1)$. Le cas $x \in dom(\theta_2)$ est similaire. Si $x \notin dom(\theta_1 \cup \theta_2)$ alors $x(\theta_1 \cup \theta_2)(\theta_1 \cup \theta_2) = x(\theta_1 \cup \theta_2) = x(\theta_1\theta_2) = x(\theta_2\theta_1) = x$.

4. $\sigma(\theta_1 \cup \theta_2)$ est une substitution idempotente car $\theta_1 \cup \theta_2$ est une substitution idempotente compatible avec σ et σ est idempotente. ❐

Preuve (de la Proposition 3.7)

($\Rightarrow$) Evident car **Par3'** et **Con2'** sont plus générales que **Par3** et **Con2** .

($\Leftarrow$) La preuve est faite par induction sur la structure de la dérivation $S_2 \vdash \sigma : p \xrightarrow[\theta]{e} q$. Comme S_1 et S_2 diffèrent seulement par **Par3'** et **Con2'** nous considérons alors seulement ces deux règles dans la preuve.

1. Soit $S_2 \vdash \sigma:\ p \parallel q \xrightarrow[\theta_1\cup\theta_2]{e_1\theta_2 \cup e_2\theta_1} r \parallel s$ une dérivation obtenue par **Par3'**. Alors il existe deux dérivations $S_2 \vdash \sigma: p \xrightarrow[\theta_1]{e_1} r$ et $S_2 \vdash \sigma: q \xrightarrow[\theta_2]{e_2} s$. Par hypothèse d'induction nous avons $S_1 \vdash \sigma: p \xrightarrow[\theta_1]{e_1} r$ et $S_1 \vdash \sigma: q \xrightarrow[\theta_2]{e_2} s$. De la Proposition 3.8 et le lemme 3.9 nous déduisons $S_1 \vdash \sigma: p \xrightarrow[\theta_1\theta_2]{e_1\theta_2} r$ et $S_1 \vdash \sigma : q \xrightarrow[\theta_2\theta_1]{e_2\theta_1} s$ et en utilisant **Par3 on obtient :** $\sigma: p \parallel q \xrightarrow[\theta1\cup\theta2]{e1\theta2\cup e2\theta1} r \parallel s$ car $\theta_1 \cup \theta_2 = \theta_1\theta_2 = \theta_2\theta_1$.

2. Soit $S_2 \vdash \sigma: p[\![k,l]\!] \xrightarrow[\theta_1]{e_1} q[\![k,l]\!]$ obtenue par **Con2'** alors il existe une dérivation $S_2 \vdash \sigma: p \xrightarrow[\theta]{e\cup\{k(t1),l(t2)\}} q$ tel que $e_1 = e\mu$ et $\theta_1 = \theta\mu$ and $\mu = mgu(t_1, t_2)$. Par hypothèse d'induction $S_1 \vdash \sigma: p \xrightarrow[\theta]{e\cup\{k(t1),l(t2)\}} q$ et $S_1 \vdash \sigma: p \xrightarrow[\theta\mu]{e\mu\cup\{k(t),l(t)\}} q$ où $t = t_1\mu = t_2\mu$. Par la Proposition 3.7 et le lemme 3.9 (car $\theta\mu$ compatible avec σ) on a :
$S_1 \vdash \sigma: p[\![k,l]\!] \xrightarrow[\theta_1]{e_1} q[\![k,l]\!]$ par **Con2.** □

Jusque là nous avons proposé deux spécifications S_1 et S_2 pour nos processus. Nous avons montré qu'elles étaient équivalentes i.e. que toute transition engendrée par S_1 peut aussi être engendrée par S_2 et réciproquement. Les règles de S_1 sont plus aisées à manipuler mais celles de S_2 mettent plus en évidence certains concepts importants (telle que l'unification dans la communication par exemple). Nous pouvons dire que S_1 est une version plus simple et implémentable de S_2.

Maintenant que nous avons prouvé la bonne formation de nos spécifications, nous allons définir une relation d'équivalence sur les processus . Cette relation permettra d'identifier modulo certains critères, deux processus. Les états de nos systèmes de transitions étant des paires de la forme $<\sigma,p>$ avec σ une substitution et p un terme HAL, la relation que nous allons considérer ne peut être directement la bisimulation [Mil 80, Par 81] sur les processus. Nous exhibons une première relation de bisimulation sur les états notée $\cong$ et puis nous définissons une relation sur les processus que nous notons $\cong_{NG}$. Nous allons montrer que cette relation sur les processus est une congruence relativement aux opérateurs HAL. C'est à dire que $\cong_{NG}$ peut être choisie comme sémantique pour la comparaison de processus HAL.

3.4 Equivalence sur les processus

Définition 3.11 (Equivalence sur les états)
Soit States l'ensemble des paires $(\sigma, p) \subseteq \Sigma_i \times \mathcal{P}$ et $R \subseteq$ States x States, nous disons que R est une bisimulation ssi $\forall$ (σ, p) R (σ', q)

i : $\forall\, \sigma : p \xrightarrow[\theta]{e} r \quad \exists\ \sigma' : p \xrightarrow[\theta]{e} s$ telle que $(\sigma\theta, r)\, R\, (\sigma'\theta, s)$

ii : Inversement .

et nous disons que p et q sont *équivalents* dans σ que l'on note $\sigma : p \cong \sigma : q$ ssi (σ, p) R (σ, q) pour une bisimulation R.

Definition 3.12 (Equivalence sur les processus)
Soient p et q deux expressions d'agent HAL, nous disons que p et q sont équivalents et nous écrivons $p \cong_{NG} q$ ssi $\forall \sigma$ dans Σ_i $(\sigma,p) \cong (\sigma, q)$. Clairement cette relation est une relation d'équivalence.

Théorème 3.13 (Propriétés de congruence de $\cong_{NG}$)
Si p et q sont deux agents HAL vérifiant $p \cong_{NG} q$ alors

$a.p \cong_{NG} a.q$	pour tout $a \in \mathcal{A}$
$p+r \cong_{NG} q+r$ et $r+p \cong_{NG} r+q$	pour tout $r \in \mathcal{P}$
$p\|\|r \cong_{NG} q\|\|r$ et $r\|\|p \cong_{NG} r\|\|q$	pour tout $r \in \mathcal{P}$
$p[[k,l]] \cong_{NG} q[[k,l]]$	pour tous k, l des noms de portes
$p\lceil U \cong_{NG} q\lceil U$	pour tout U est est un ensemble de noms de portes.
$p[f] \cong q[f]$	où f est une fonction de renommage.

Maintenant nous allons donner quelques lemmes utiles pour la preuve de ce théorème.

Lemme 3.14

Si θ, μ compatibles avec σ et $\sigma\mu : p \xrightarrow[\theta]{e} r$ alors $\sigma : p \xrightarrow[\mu\theta]{e} r$.

Preuve
Directe (en utilisant le lemme 3.3) . □

Lemme 3.15

Si θ, μ compatibles avec σ et θ compatible avec $\sigma\mu$ et $\sigma : p \xrightarrow[\mu\theta]{e} r$ alors $\sigma\mu : p \xrightarrow[\theta]{e} r$.

Preuve
découle directement des hypothèses du lemme. □

Lemme 3.16
Si R est une relation de bisimulation alors $R' = \{((\sigma\mu, p), (\sigma\mu, q)) \mid (\sigma, p)\ R\ (\sigma, q)$ et μ compatible avec $\sigma\}$ est une bisimulation.

Une telle bisimulation est appelée une CBI -bisimulation (Closed By Instantiation).

Preuve

Soit $(\sigma\mu, p)\ R'\ (\sigma\mu, q)$ et soit $\sigma\mu : p \xrightarrow[\theta]{e} r$ alors $\sigma : p \xrightarrow[\mu\theta]{e} r$, par le lemme 3.14. Comme R est une bisimulation avec $(\sigma, p)\ R\ (\sigma, q)$ alors il existe $\sigma : q \xrightarrow[\mu\theta]{e} s$ et donc $\sigma\mu : q \xrightarrow[\theta]{e} s$, en utilisant le lemme 3.15. Ceci établit la propriété de transfert pour R' car $((\sigma\mu\theta,r), (\sigma\mu\theta, s))$ est dans R et donc dans R'. ❐

Corollaire 3.17

$p \cong_{NG} q$ ssi $(Id, p) \equiv (Id, q)$, où Id est une substitution identité. .

Preuve : ($\Rightarrow$) découle aisément de la définition de $\cong_{NG}$ et ($\Leftarrow$) découle du lemme 3.15. ❐

Preuve (du théorème 3.11)

Nous donnons les preuves pour le préfixage et la composition parallèle. Les autres preuves sont similaires.

- *Préfixage.* : Soit $p \cong_{NG} q$. Montrons que $a.p \cong_{NG} a.q$.

  $p \cong_{NG} q \Rightarrow \exists$ R une CBI- bisimulation telle que : $(Id,p)\ R\ (Id, q)$ par le lemme 3.16.

  Soit $R' = R \cup \{((\sigma,a.p),(\sigma, a.q)) \mid \sigma \in \Sigma_i\}$ .Nous allons montrer que R' est une bisimulation

  Supposons $(\sigma,p)\ R\ (\sigma, q)$ alors la propriété de transfert est directe car R est une bisimulation et est contenue dans R'.

  Maintenant supposons $(\sigma, a.p)\ R\ (\sigma, a.q)$ et $\sigma : a.p \xrightarrow[\theta]{a\sigma\theta} p$ donc $\sigma : a.q \xrightarrow[\theta]{a\sigma\theta} q$. Comme par hypothèse $(Id,p)R(Id, q)$ alors $(\sigma\theta, p)\ R\ (\sigma\theta, q)$ et $(\sigma\theta, p)\ R'\ (\sigma\theta, q)$.

- *Composition parallèle* : Soit $p \cong_{NG} q$. Montrons que $p\|s \approx_{NG} q\|s$ pour s dans P(l'ensemble des agents HAL).

  $p \cong_{NG} q \Leftrightarrow \exists$ R une CBI- bisimulation telle que : $(Id,p)\ R\ (Id, q)$ par le lemme 3.14.

  Définissons d'abord $R' = \{(\sigma,p\|s)\ (\sigma, q\|s) \mid (\sigma,p)R(\sigma, q), s \in P\}$ et montrons que R' est une bisimulation.

  Supposons $(\sigma, p\|r)\ R'\ (\sigma, q\|r)$

  Soit $\sigma : p\|r \xrightarrow[\theta]{e} t$ alors il y'a trois cas possibles.

  1. $\sigma : p \xrightarrow[\theta]{e} p_1$ et $t = p_1\|s$. Il existe dans ce cas par hypothèse d'induction une transition $\sigma : q \xrightarrow[\theta]{e} q_1$ avec $(\sigma\theta,p_1)\ R\ (\sigma\theta,q_1)$ et par **Par1** $\sigma : q\|s \xrightarrow[\theta]{e} q_1\|s$. D'après la définition de R' on $a(\sigma\theta,p_1\|s)R'(\sigma\theta,q_1\|s)$.

2. $\sigma : q \xrightarrow[\theta]{e} q_1$ et $t = p \| q_1$. Comme le cas précédent.

3. $\sigma : p \xrightarrow[\theta]{e1} p_1$ et $\sigma : s \xrightarrow[\theta]{e2} s_1$ et $t = p_1 \| s_1$ alors par hypothèse $\sigma : q \xrightarrow[\theta]{e1} q_1$ avec $(\sigma\theta,p1)R(\sigma\theta,q1)$. Par **Par3** $\sigma : q\|s \xrightarrow[\theta]{e1Ue2} q_1\|r_1$. Par R' nous avons $(\sigma\theta,p_1\|r_1)R'$ $(\sigma\theta, q_1\|r_1)$. □

Pour les agents p, q, r de $\mathcal{P}$, nous pouvons dire que modulo la bisimulation non close + est idempotente, commutative, associative and admet nil comme élément neutre.

Proposition 3.18 (Propriétés de l'opérateur + par rapport à ≅)
Soient p, q et r des processus HAL alors nous avons :

(1) $p+q \cong_{NG} q+p$
(2) $p+(q+r) \cong_{NG} (p+q)+r$
(3) $p+nil \cong_{NG} p$
(4) $p+p \cong_{NG} p$

Proposition 3.19 (Propriétés de l'opérateur de composition parallèle par rapport à ≅)

(1) $p\|q \cong_{NG} q\|p$
(2) $p \|(q\|r) \cong_{NG} (p\|q)\| r$
(3) $p\|nil \cong_{NG} p$

Proposition 3.20 (Propriétés de l'opérateur de restriction par rapport à ≅)

(1) $(p+q)\lceil U \cong_{NG} p\lceil U+q\lceil U$
(2) $(p\|q)\lceil U \cong_{NG} p\lceil U\| q\lceil U$
(3) $p\lceil U\lceil V \cong_{NG} p\lceil U\cup V$
(4) $p\lceil U\lceil V \cong_{NG} p\lceil V\lceil U$
(5) $(a.p)\lceil U \cong_{NG} a.p\lceil U$ si $dom(a)\cap U = \emptyset$.
(6) $(a.p)\lceil U \cong_{NG} nil$ sinon.

Proposition 3.21 (Propriétés de l'opérateur de renommage par rapport à ≅)

(1) $(p+q)[f] \cong_{NG} p[f] + q[f]$
(2) $(p\|q)[f] \cong_{NG} p[f] \| q[f]$
(3) $p[Id] \cong_{NG} p$
(4) $p[f][g] \cong_{NG} p[g\circ f]$
(5) $a.p[f) \cong_{NG} f(a).p[f]$

La proposition suivante établit des propriétés satisfaites par l'opérateur de connexion. On remarque que l'opérateur de connexion n'est pas distributif par rapport à l'opérateur de composition d'agents.

Proposition 3.22 (Propriétés de l'opérateur de connexion par rapport à $\cong$)

(1) $(p+q)[\![k,l]\!] \cong_{NG} p[\![k,l]\!] + q\,[\![k,l]\!]$

(2) $p\lceil U[\![k,l]\!] \cong_{NG} p[\![k,l]\!] \lceil U$ si $\{k,l\} \cap U = \emptyset$

Remarque : Les preuves des propositions 3.19 à 3.23 sont omises. Mais elles sont trés similaires aux preuves 2.20 à 2.24.

3.5 Application

HAL est un langage expressif pour la description de machines parallèles interprétant les langages logiques et fonctionnels.De telles applications sont présentées dans [BH 92a , BH 92b, Jor 91]. Dans cette section nous illustrons la sémantique opérationnelle que nous venons de présenter par un exemple trés simple.

Soient t_1 et t_2 deux termes de $T_{C,X}$ définis comme suit :

- $t_1 = f(x, z)$
- $t_2 = f(g(y), y)$.

t_1 et t_2 sont unifiables , c'est à dire il existe une substitution θ telle que : $t_1\theta = t_2\theta$.

Maintenant montrons qu'il est possible de traduire t_1 et t_2 en des agents HAL dont les comportements laissent observer une substitution θ telle que $t1\theta = t2\theta$. En d'autres termes l'unification de t1 et t2 peut être accomplie par un processus HAL.

- T1 $\Leftarrow$ {A(f(x, z))} . nil
- T2 $\Leftarrow$ {B(f(g(y), y))}. nil
- Unif $\Leftarrow$ (T1‖ T2) $[\![A,B]\!]\lceil$ {A, B}

En utilisant les règles de figure1 nous obtenons :

Id: $\{A(f(x, z))\}.nil \xrightarrow[\theta]{e_1}> nil$, Par **Pref**

Id : $\{B(f(g(y), y)\}\ .nil \xrightarrow[\theta]{e_2}> nil$ Par **Pref**

où $e_1 = \{A(f(g(u), u)\}$, $e_2 = \{B(f(g(u), u)\}$ et $\theta = \{x \leftarrow f(g(u)), z \leftarrow u , y \leftarrow u \}$

Id : $(T1\| T2) \xrightarrow[\theta]{e_1 \cup e_2}> nil$ Par **Par3**

Id : $(T1\| T2)[\![A,B]\!] \xrightarrow[\theta]{\tau}> (nil \| nil)\ [\![A,B]\!]$ Par **Con2**

Id : $(T1 \| T2)[\![A,B]\!] \Gamma \{A, B\} \xrightarrow[\theta]{\tau} (nil \| nil) \quad [\![A,B]\!] \Gamma \{A, B\}$ Par **Res.**

3.6 Conclusion

Dans le but de spécifier des machines parallèles, plus particulièrement des machines à inférence parallèle l'algèbre HAL a été proposée. Dans ce chapitre nous avons donné une sémantique à HAL où la communication entre processus se fait via le partage de variables et envoi de message.Une caractéristique importante de HAL est l'utilisation de l'unification de termes échangés entre deux processus pour réaliser la synchronisation de ces deux derniers. Quand l'unification est close, on parle de sémantique close sinon on parle de sémantique non close. Dans ce chapitre nous avons considéré la sémantique non close. Dans la section 3.4, nous avons illustré ce modèle par un exemple montrant comment l'unification s'exprimait en HAL. Cette application peut être généralisée à la spécification de machines parallèles pour des programmes logiques basées sur les clauses de Horn [Bel 92, BH 92] .

Chapitre 4

Logique Modale et Algèbre de Processus

4.1 Motivations

La vérification de programmes parallèles consiste à comparer un programme avec sa spécification. Quand le programme et la spécification sont représentés à l'aide de systèmes de transitions, de graphes ou de termes définis sur une algèbre donnée, alors les différentes équivalences vues au chapitre1 peuvent être utilisées comme outil de vérification. Par exemple, on peut dire qu'un programme est correct par rapport à sa spécification si et seulement si il existe une bisimulation reliant les graphes représentant le programme d'une part et sa spécification d'autre part. Mais les systèmes de transitions, les graphes ... etc. ne permettent pas de spécifier toutes les propriétés sur les programmes. Certaines propriétés s'expriment dans d'autres formalismes.
Parmi les formalismes possibles celui de la logique temporelle et modale s'est avéré bien adapté.
En effet ces dernières années ce formalisme a été intensivement utilisé. C'est un outil puissant permettant d'exprimer et de vérifier un large spectre de propriétés de programmes parallèles. Cependant, il a étè sévèrement critiqué pour être global, non modulaire et non compositionnel. Nous voulons dire par là que dans le but de formuler et vérifier une propriété temporelle, nous

devons considérer le programme entier. La raison essentielle est que nous ne connaissions pas à priori d'opérateurs dans la logique, soit * cet opérateur, tel que si un processus p satisfait une formule A et qu'un processus q satisfait une formule B alors le processus p||q (où p et q sont mis en parallèle) satisfait une formule A*B. En d'autres termes nous ne savons pas dériver des propriétés logiques à partir de (p||q) connaissant celles qui sont dérivées séparément à partir de p et q.
Par conséquent, même si la logique temporelle est un bon outil de spécification et même si les modèles de vérification tels que le "modèle checking" sont rigoureux, ils restent tous les deux de faibles supports quand on travaille avec des structures de réseaux à topologie dynamique. C'est pour cette raison que des travaux sur la compositionalité dans le domaine de la spécification et la vérification font l'objet d'importants sujets de recherche .
Dans ce chapitre nous construisons un système de preuve modal compositionnel pour une algèbre de processus $\mathcal{L}_p$. La logique modale que nous considérons est HML (Logique de Hennessy et Milner) que nous avons déja présentée au chapitre 1.

Nous organisons le reste de ce chapitre comme suit : Dans la section 4.2 nous définissons le langage $\mathcal{L}_p$(sous ensemble de HAL) par sa syntaxe et sa sémantique opérationnelle. Dans la section 4.3 nous proposons de trois systèmes de preuve Sys_0, Sys_1 et Sys_2 issus de la décomposition de $\mathcal{L}_p$ en trois sous langages L_0, L_1 et L_2 . Pour chaque système proposé, nous établissons les résultats de correction et de complétude.

4.2 Un langage pour décrire les processus

4.2.1 Notions de base

Dans cette section, nous définissons le langage pour décrire les processus par sa syntaxe et sa sémantique. celui ci est syntaxiquement proche de HAL. Il est plus simple car nous ne considérons pas le passage de valeurs.

Pour définir la syntaxe du langage nous introduisons un certain nombre de notions et notations.

- L'ensemble d'actions élémentaires : $Act_e = \{a, a_1, \ldots, b, b_1, \ldots\}$
- L'ensemble d'actions ou évènements : $E_M = \mathcal{P}_M(Act_e)$ où $\mathcal{P}_M$ est l'ensemble des multi-ensembles de Act_e.
- Un connecteur : Un connecteur est un multi-ensemble de paires d'actions élémentaires. $\{<a_i,b_i> : a_i, b_i \in Act_e\}$. On utilise K, K'...etc. pour dénoter les différents connecteurs.

Nous considérons le langage $\mathcal{L}_p$ pour décrire des processus comme une Σ - algèbre libre T_Σ définie sur une signature Σ contenant comme opérateurs l'ensemble E d'évènements, X un ensemble de variables ainsi que les opérateurs pour le non déterminisme et la communication.

Soit $\Sigma = E_M \cup \{nil, +, \|, [\![]\!], \lceil, rec\} \cup X$. Σ est l'ensemble d'opérateurs où nil, + et rec ont exactement le même sens que dans CCS.

4.2.2 Syntaxe du langage considéré

L'ensemble d'agents $\mathcal{L}_p$ est généré par la grammaire suivante :

$$p, q ::= nil \mid e.p \mid p+q \mid p\|q \mid p[\![K]\!] \mid p\lceil U \mid rec\ x.p \mid x$$

où $e \in E_M$, K est un connecteur, U est un sous ensemble de Act_e, $x \in X$.

4.2.3 Définition informelle des agents de $\mathcal{L}_p$

Les agents de $\mathcal{L}_p$ sont pour la plupart définis de manière classique . Les agents nil, p+q, $p\lceil U$ et $p\|q$ sont définis exactement comme dans le chapitre 2. L'agent e.p exécute une transition étiquetée par e ou une e - transition et devient l'agent p. L'agent $p[\![K]\!]$ exécute des transitions de la forme $p[\![K]\!] \xrightarrow{e\pm s} q[\![K]\!]$ où s est un multi-ensemble obtenu à partir de K par la fontion φ que nous définissons formellement comme suit :

φ: $K \longrightarrow \mathcal{P}_M(K)$

- $\emptyset \in \varphi(K)$
- Si $<a, b> \in K$ alors $\{a, b\} \in \varphi(K)$
- Si $<ai, bi> \in K$ et $s \in \varphi(K - <ai, bi>)$ alors $\{ai, bi\} \pm s \in \varphi(K)$.

4.2.4 Sémantique opérationnelle de $\mathcal{L}_p$

Définition 4.1 (la définition de $\longrightarrow$)

$$E_M \quad \frac{}{e.p \xrightarrow{e} p} \qquad Sum_1 \quad \frac{p \xrightarrow{e} r}{p+q \xrightarrow{e} r} \qquad Sum_2 \quad \frac{q \xrightarrow{e} s}{p+q \xrightarrow{e} s}$$

$$Con_1 \quad \frac{p \xrightarrow{e} r}{p[\![<a, b>]\!] \xrightarrow{e} r[\![<a, b>]\!]} \qquad Con_2 \quad \frac{p \xrightarrow{e} r}{p[\![<a, b>]\!] \xrightarrow{e+\{a, b\}} r[\![<a, b>]\!]}$$

$$Par_1 \quad \frac{p \xrightarrow{e} r}{p\|q \xrightarrow{e} r \| q} \qquad Par_2 \quad \frac{q \xrightarrow{e} s}{p\|q \xrightarrow{e} p \| s} \qquad Par_3 \quad \frac{p \xrightarrow{e_1} r \quad q \xrightarrow{e_2} s}{p\|q \xrightarrow{e_1 \pm e_2} r\|s}$$

$$\text{Res}\quad \frac{p \xrightarrow{e} r}{p\lceil U \xrightarrow{e} r \lceil U}\ (1) \qquad \text{Rec}\quad \frac{p[x := \text{rec } x.p] \xrightarrow{e} r}{\text{rec } x.\ p \xrightarrow{e} r}$$

Remarque : Le comportement de rec x.p est complètement déterminé par des dépliages répétés de p.Si p = rec x.q alors p^0 = nil et $p^{n+1} = p[\ \text{recx.q})^n :=x]$. On a alors la relation d'ordre < sur les processus.

Définition 4.2 (Relation d'ordre sur les processus)
p < q si et seulement si p est un sous terme de q et le nombre de rec imbriqué dans p est inférieur au nombre de rec imbriqué dans q.

4.2.5 Propriétés de congruence de $\mathcal{L}_p$

Aprés avoir donné la sémantique opérationnelle à $\mathcal{L}_p$, nous considérons la bisimulation forte comme équivalence sur $\mathcal{L}_p$ et montrons que celle ci peut être considérée comme sémantique pour ce langage. C'est ce que nous établissons par la proposition suivante :

Proposition 4.3
$\approx_B$ est une congruence relativement aux opérateurs de $\mathcal{L}_p$.

Preuve
Nous illustrons la preuve par le cas de l'opérateur de connexion seulement.
Nous voulons montrer que si p $\approx_B$ q alors p⟦K⟧ $\approx_B$ q⟦K⟧ . Pour cela montrons que la relation R = { (p⟦K⟧ , q⟦K⟧) : p $\approx_B$ q pour tous p, q ∈ $\mathcal{L}_p$} est une bisimulation forte.
Soit (p⟦K⟧ , q⟦K⟧) ∈ R et p⟦K⟧ $\xrightarrow{e}$ r. D'après Con il existe p $\xrightarrow{e\pm s}$ p' pour s ∈ φ(K) avec r=p'⟦K⟧.Comme p $\approx_B$ q alors il existe une transition q $\xrightarrow{e\pm s}$ q ' pour s ∈ φ(K) avec p' $\approx_B$ q'.
D'après la règle Con, on a q⟦K⟧ $\xrightarrow{e}$ q'⟦K⟧. D'après la définition de R le couple (p'⟦K⟧ , q'⟦K⟧) appartient à R. La propriété de transfert est vérifiée. par conséquent R est une bisimulation forte. □

Lemme 4.4 : $(\text{recx.p})^n \equiv_n \text{recx.p}$
Preuve (Par induction sur n.)
Cas (n =0) : $(\text{rec x.p})^0$ = nil et nil $\equiv_0$ rec x.p .
Cas (n > 0) : $(\text{rec x.p})^{n+1} = p[\ \text{recx.p})^n :=x]$. Par hypothèse d'induction $(\text{recx.p})^n \equiv_n$ rec x.p donc $p[\ \text{recx.p})^n :=x] \equiv_n p[\text{recx.p} :=x] = \text{rec x.p}$ □

(1) Si e ∩ U = Ø

Lemme 4.5

$\equiv_n$ est une congruence relativement aux opérateurs de composition parallèle, de connexion et de restriction , c'est à dire si p et q appartiennent à $\mathcal{L}_p$ et $p \equiv_n q$ alors

(1) $p \| r \equiv_n q \| r$ pour tout $r \in \mathcal{L}_p$

(2) $p[\![K]\!] \equiv_n q[\![K]\!]$ pour tout K

(3) $p\lceil K \equiv_n q\lceil K$ pour tout K.

4.3 Vers un système de preuve modal correct et complet pour $\mathcal{L}_p$

Dans cette section, nous allons construire un système de preuve modal compositionnel correct et complet pour $\mathcal{L}_p$. Les règles du système seront du style des règles d'introduction de Gentzen [Gen69].

En plus des règles d'introduction des opérateurs de la logique, le système comportera des règles d'introduction des opérateurs de $\mathcal{L}_p$.

Pour construire ce système, nous procédons par étapes. Nous considérons trois subdivisions de $\mathcal{L}_p$:

- L_0 est le sous ensemble de $\mathcal{L}_p$ défini sur la signature $\Sigma_0 = \{nil, ., +\}$
- L_1 est le sous ensemble de $\mathcal{L}_p$ défini sur la signature $\Sigma_1 = \Sigma_0 \cup \{\|, rec\}$
- $L_2 = \mathcal{L}_p$

Nous construisons trois systèmes de preuve Sys_0, Sys_1 et Sys_2 correspondant respectivement à L_0, L_1 et L_2 et établirons les résultats de correction et de complétude pour chacun d'eux.

4.3.1 Le système de preuve Sys_0

La construction de ce système de preuve résulte de l'introduction d'une relation de preuve $\vdash$ qui coïncide avec la relation sémantique $\models$.

Remarque : Les axiomes sont préfixés par Ax_i où i est le numéro de l'axiome. Les règles sont préfixées par le nom du connecteur ou de l'opérateur qu'elles introduisent . Quand il y a plusieurs règles pour un même connecteur ou opérateur elles sont supposées être numérotées de gauche à droite et de haut en bas. La numérotation a été volontairement omise pour ne pas surcharger le système. Cette remarque est valable pour les autres systèmes.

4.3.1.1 Le système Sys_0

Axiomes

$Ax_1 \quad \dfrac{}{p \vdash Tr} \qquad Ax_2 \quad \dfrac{}{nil \vdash [e]A} \qquad Ax_3 \quad \dfrac{}{e_1.p \vdash [e]A}$ (2)

(2) si $e_1 \neq e$

Règles

$\vee$ I $\dfrac{p \vdash A}{p \vdash A \vee B}$ $\dfrac{p \vdash B}{p \vdash A \vee B}$ $\wedge$ I $\dfrac{p \vdash A \quad p \vdash B}{p \vdash A \wedge B}$

<e>I $\dfrac{p \vdash A}{e.p \vdash \langle e \rangle A}$ [e]I $\dfrac{p \vdash A}{e.p \vdash [e]A}$

+<e>I $\dfrac{p \vdash \langle e \rangle A}{p+q \vdash \langle e \rangle A}$ $\dfrac{q \vdash \langle e \rangle A}{p+q \vdash \langle e \rangle A}$ +[e]I $\dfrac{p \vdash [e]A \quad p \vdash [e]A}{p+q \vdash [e]A}$

Remarque : Les règles et axiomes de Sys_0 sont relativement aisés à comprendre . nil $\vdash$ [e]A car nil ne peut faire aucune action. De la même manière, $e_1.p \vdash [e]A$ si $e_1 \neq e$ car $e_1.p$ peut seulement exécuter e_1. Les règles d'introduction pour les connectives de la logique propositionnelle sont classiques. Les règles d'introduction relatives à l'opérateur de préfixage coïncident avec celles des opérateurs <e> et [e] de la logique considérée. Les règles d'introduction restreintes pour le cas + dépendent de la forme de A et B.

4.3.1.2 Correction et complétude de Sys_0

Dans cette section, nous établissons les résultats de correction et de complétude de Sys_0.

Théorème 4.6 (Correction de Sys_0)
Le système Sys_0 est correct, c'est à dire : Si $p \vdash B$ alors $p \models B$

Preuve
Nous montrons que tous les axiomes sont valides et que les règles préservent la validité.

- Ax_1 Evident car $p \models Tr$ est toujours vraie.
- Ax_2 Il n'y a aucun p et aucun e tels que nil $\xrightarrow{e}$ p donc forcément nil $\models$ [e]A .
- Ax_3 Si $q = e_1.p$ alors il n' y a aucun r et aucun e différent de e_1 tels que $e_1.p \xrightarrow{e_1} r$ et donc $e_1.p \models [e]A$ si $e \neq e_1$.
- $\vee$ I Supposons $p \models A$ alors clairement $p \models A \vee B$. Il en est de même si $p \models B$.
- $\wedge$ I Supposons $p \models A$ et $p \models B$ alors $p \models A \wedge B$.
- + <e> I Si $p \models \langle e \rangle A$ alors il existe q tel que $p \xrightarrow{e} p'$ et $p' \models A$. Par Sum_1, si $p \xrightarrow{e} p'$ alors $p+q \xrightarrow{e} p'$ et donc $p+q \models \langle e \rangle A$.
- + [e] I Supposons que $p \models [e]A$ et $q \models [e]A$ donc $\forall p'$ $p \xrightarrow{e} p' \Rightarrow p' \models A$ et $\forall q'$ $q \xrightarrow{e} q' \Rightarrow q' \models A$. Donc $\forall r$ $p+q \xrightarrow{e} r \Rightarrow r \models A$ et donc $p+q \models [e]A$. □

Théorème 4.7 (Complétude de Sys_0)
Le système Sys_0 est complet, c'est à dire :

Si $p \models B$ alors $p \vdash B$.

Preuve : (Par induction sur la structure de B).

- Cas(B= Tr) Donc $p \vdash Tr$ par Ax_1.
- Cas (B = Fal) On n'a jamais $p \models Fal$
- Cas (A= $B1 \vee B2$) Si $P \models B1 \vee B2$ alors soit $p \models B1$ soit $p \models B2$. Supposons que $p \models B1$ alors par hypothèse d'induction, $p \vdash B1$ et par $\vee_1 I$, $p \vdash B1 \vee B2$. Il en est de même si $p \models B2$.
- Cas (A =$B1 \wedge B2$) Si $p \models B1 \wedge B2$ alors $p \models B1$ et $p \models B2$. Par hypothèse d'induction on a $p \vdash B1$ et $p \vdash B2$ e t par $\wedge I$, $p \vdash B1 \wedge B2$.
- Cas (B=<e>B1) On fait une induction sur la structure de p.

  Sous cas 1 : (p=nil) . Impossible.

  Sous cas 2 : ($p = e_1.q$) . Si $e_1.q \models <e>B1$ alors $e_1 = e$ et $q \models B1$. D'où par induction $q \vdash B1$ et par <e> I, on obtient $e.q \vdash <e>B1$.

  Sous cas3 : ($p = q+r$) On a soit $q \models <e>B1$ soit $r \models <e>B1$. Si $q \models <e>B1$ alors $q \vdash <e>B1$ et par $+<e>I$, $q+r \vdash <e>B1$. Si $r \models <e>B1$ alors par le même raisonnement on a $q+r \vdash <e>B1$.

- Cas (A = [e]B1) On fait une induction sur la structure de p.

  Sous casl (p = nil). Si nil $\models$ [e]B1 alors nil $\vdash$ [e]B1 (d'après Ax_2).

  Sous cas2 ($p = e_1.q$). Si $e_1.q \models$ [e]B1 alors soit $e_1 \neq e$ et donc $e_1.q \vdash$ [e]B1 d'après Ax_3.
  Soit $e_1 = e$ et donc $q \models B$. Par hypothèse d'induction , on a $q \vdash B$ et par [e] I, $e.q \vdash [e]B$.
  Sous cas3 : ($p = q+r$). Si $q+r \models$ [e]B1 alors $q \models$ [e]B1 et $r \models$ [e]B1.
  Par hypothèse d'induction $q \vdash$[e]B1 et $r \vdash$ [e]B1 et par +[e]I, $q+r \vdash$[e]B1. □

4.3.1.3 Exemple : $e_1.p + e_2.q \vdash <e_1>Tr \wedge <e_2>Tr \wedge [e]Fal$

$$\dfrac{\dfrac{\dfrac{\dfrac{Ax_1}{p \vdash Tr}}{e_1.p \vdash <e_1>Tr}\,<e>I}{e_1.p + e_2.q \vdash <e_1>}\,+I \qquad \dfrac{\dfrac{\dfrac{Ax_1}{q \vdash Tr}}{e_2.p \vdash <e_2>Tr}\,<e>I}{e_1.p + e_2.q \vdash <e_2>Tr}\,+I}{e_1.p + e_2.q \vdash <e_1>Tr \wedge <e_2>Tr}\,\wedge I \qquad \dfrac{Ax_3}{e_1.p \vdash [e]Fal} \qquad \dfrac{Ax_4}{e_2.p \vdash [e]Fal}$$

$$\dfrac{e_1.p + e_2.q \vdash <e_1>Tr \wedge <e_2>Tr \qquad e_1.p \vdash [e]Fal \qquad e_2.p \vdash [e]Fal}{e_1.p + e_2.q \vdash <e_1>Tr \wedge <e_2>Tr \wedge [e]Fal}\,\wedge I$$

4.3.2 Le système de preuve Sys_1

Sys_1 doit traiter en plus de Sys_0, l'opérateur de composition parallèle et l'opérateur de récursion. Mais l'introduction de l'opérateur de composition parallèle complique considérablement le problème. Habituellement, les difficultés pour traiter cet opétateur sont contournées en le mettant "à plat " grâce à l'utilisation du théorème d'expansion .Cette technique est développée dans les modèles checking [Wol 89]. Mais nous ne voulons suivre cette approche puisque nous souhaitons un système de preuve compositionnel. Autrement dit si un processus p satisfait une formule A et qu'un processus q satisfait une formule B alors nous devons pouvoir déduire que le processus op(p, q) satisfait une formule $op_L(A, B)$ où op est un opérateur de combinaison de processus et op_L un opérateur logique qui soit la contrepartie de l'opérateur op.
Le système Sys_0 est un exemple de ce type d'approche. Pour les opérateurs de L_0, la construction de Sys_0 a été relativement aisée. Malheureusement la prise en compte de l'opérateur de composition parallèle n'est pas aussi évidente. La difficulté majeure est dûe à l'absence d'un opérateur de la logique qui soit la contrepartie de l'opérateur de composition parallèle. En d'autres termes, il n'existe pas un opérateur * dans la logique tel que si p satisfait A et q satisfait B alors p||q satisfait A*B.

Notons que cela n'était pas possible avec l'opérateur + non plus. Mais particulièrement pour cet opérateur, il a été possible de contourner la difficulté en considérant des formes restreintes pour A. Pour l'opérateur de composition parallèle, même des formes restreintes ne suffisent pas. Divers travaux basés sur la résolution de ce problème ont été développés ces dernières années. Nous citons particulièrement les travaux de Pnueli [Pnu 85], De Winskel[Win 85,a] et [Win 85b]], Larsen [Lar90b] et Stirling[Sti 85a] , [Sti 85b] et [Sti 87].

Dans ce chapitre nous développons l'approche similaire à celle développée par C.Stirling pour notre algèbre. Cette approche consiste à introduire une nouvelle relation sémantique que nous définissons de la manière suivante :

Définition 4.8 (la relation sémantique relativisée)
Si p est un agent et A et B sont deux formules alors on a :

$$B, p \models A \text{ si } \forall q \quad q \models B \Rightarrow q \| p \models A$$

On note aussi cette relation $p \models_B A$ pour pouvoir parler de $\models_B$.

Remarque : Du point de vue sémantique la formule B dans $B, p \models A$ peut être considérée comme la description de l'environnement, un résumé partiel de tout processus et peut se lire : si on plonge p dans un environnement qui satisfait une formule B alors tout processus q satisfaisant B mis en parallèle avec q satisfait A.

En considérant deux relations de preuve $\vdash_B$ et $\vdash$ coïncidant avec les deux relations sémantiques $\models_B$ et $\models$ respectivement , les règles d'introduction pour l'opérateur parallèle en découlent alors aisément :

$$\frac{p \vdash A \; A,q \vdash B}{p \parallel q \vdash B} \qquad \frac{A, p \vdash B \quad B, q \vdash C}{A, p \parallel q \vdash C}$$

Ces règles sont analogues à la règle cut de Gentzen [69] et la deuxième règle est similaire à la règle d'introduction de Hoare pour la composition parallèle.

Mais cette relation relativisée pose des problèmes pour le traitement de la somme non déterministe. En effet les règles suivantes ne sont pas valides:

- $A, p \models \langle e \rangle B$ ou $A, q \models \langle e \rangle B \Rightarrow A, p+q \models \langle e \rangle B$
- $A, p \models [e] B$ et $A, q \models [e]B \Rightarrow A, p+q \models [e]B$

Pour le montrer observons l'exemple suivant :
$r = e.nil$, $p = e_1.\, nil$, $q = e_2.\, nil$, $A = \langle e \rangle([e_1]Fal \wedge [e_2]Fal)$ et $B = ([e_1]Fal \vee [e_2]Fal)$.
Par le biais de cet exemple on a exhibé un agent r tel que :
$r \models A$ et $r \parallel p \models \langle e \rangle B$ alors que $r \parallel (p+q) \not\models \langle e \rangle B$.

Ceci est dû de manière plus générale au fait que le parallèle est asynchrone. L'environnement peut répondre à une e - expérience indépendemment de p, q ou p+q.

Une solution possible consiste à étendre la logique HML par deux nouvelles connectives $\langle \underline{e} \rangle$ et $[\underline{e}]$. Evidemment la sémantique des deux connectives dépend de la semantique de l'opérateur parallèle de HAL,et nous la définissons de la manière suivante :

Définition 4.9 (Sémantique des nouvelles connectives)

- $p \parallel q \models \langle \underline{e} \rangle B$ ssi $\exists\, q \xrightarrow{e} q_1$ tel que $p \parallel q_1 \models B$ ou $\exists\, p \xrightarrow{e_1} p_1$ $q \xrightarrow{e_2} q_1$ tel que $e = e_1 \pm e_2$ et $p_1 \parallel q_1 \models B$.
- $p \parallel q \models [\underline{e}]\, B$ ssi $\forall\; q \xrightarrow{e} q_1 : p \parallel q_1 \models B$ et $\forall p \xrightarrow{e_1} p_1, \forall\; q \xrightarrow{e_2} q_1$ tel que $e = e_1 \pm e_2$ nous avons $p_1 \parallel q_1 \models B$.
- $A,p \models \langle \underline{e} \rangle B$ ssi $\forall\; r \models A \quad r \parallel p \models \langle \underline{e} \rangle\; B$.
- $A,p \models [\underline{e}]\, B$ ssi $\forall\; r \models A : r \parallel p \models [\underline{e}]\; B$.

Nous dénotons par L : l'ensemble $L_{HML} \cup \{ \langle \underline{e} \rangle A$: pour $A \in L_{HML}\} \cup \{ [\underline{e}]A$: pour $A \in L_{HML}\}$

4.3.2.1 Le système Sys_1

Axiomes

$Ax_1 \dfrac{}{p \vdash Tr}$ $Ax_2 \dfrac{}{nil \vdash [e]A}$ $Ax3 \dfrac{}{A, p \vdash Tr}$

$Ax_4 \dfrac{}{Fal,p \vdash B}$ $Ax_5 \dfrac{}{A, nil \vdash [\underline{e}]B}$ $Ax_6 \dfrac{}{e_1.p \vdash [e]B}$ (3)

$Ax_7 \dfrac{}{[e_1]A, e_2.p \vdash [\underline{e}]B}$ (4)

Règles

$\vee$ I $\dfrac{p \vdash A}{p \vdash A \vee B}$ $\dfrac{p \vdash B}{p \vdash A \vee B}$ $\dfrac{A,p \vdash B}{A, p \vdash B \vee C}$ $\dfrac{A, p \vdash C}{A, p \vdash B \vee C}$ $\dfrac{A, p \vdash C \quad B, p \vdash C}{A \vee B, p \vdash C}$

$\wedge$ I $\dfrac{p \vdash A \quad p \vdash B}{p \vdash A \wedge B}$ $\dfrac{A,p \vdash C}{A \wedge B, p \vdash C}$ $\dfrac{B,p \vdash C}{A \wedge B, p \vdash C}$ $\dfrac{A,p \vdash B \quad A,p \vdash C}{A, p \vdash B \wedge C}$

<e>I $\dfrac{p \vdash A}{e.p \vdash \langle e\rangle A}$ $\dfrac{A,p \vdash \langle \underline{e}\rangle B}{A,p \vdash \langle e\rangle B}$ $\dfrac{A, p \vdash B}{\langle e\rangle A, p \vdash \langle e\rangle B}$

<$\underline{e}$> I $\dfrac{A, p \vdash B}{A,e.p \vdash \langle \underline{e}\rangle B}$ $\dfrac{A, p \vdash B}{\langle e_1\rangle A, e_2.p \vdash \langle \underline{e}\rangle B}$ (5)

[e] I $\dfrac{p \vdash A}{e.p \vdash [e]A}$ $\dfrac{[e]D \wedge E, p \vdash [\underline{e}]B \quad D, p \vdash B}{[e]D \wedge E, p \vdash [e]B}$

[$\underline{e}$] I $\dfrac{A,p \vdash B \quad C,p \vdash B}{A \wedge [\tau]C, e.p \vdash [\underline{e}]B}$ $\dfrac{A,p \vdash B}{[e_1]A, e_2.p \vdash [\underline{e}]B}$ (6)

+ I (<>) $\dfrac{p \vdash \langle e\rangle A}{p+q \vdash \langle e\rangle A}$ $\dfrac{q \vdash \langle e\rangle A}{p+q \vdash \langle e\rangle A}$ $\dfrac{A,p \vdash \langle \underline{e}\rangle B}{A,p+q \vdash \langle \underline{e}\rangle B}$ $\dfrac{A,q \vdash \langle \underline{e}\rangle B}{A,p+q \vdash \langle \underline{e}\rangle B}$

+ I ([]) $\dfrac{p \vdash [e]A \quad q \vdash [e]A}{p+q \vdash [e]A}$ $\dfrac{A,p \vdash [\underline{e}]B \quad A,q \vdash [\underline{e}]B}{A, p+q \vdash [\underline{e}]}$

---

(3) Si $e1 \neq e$ (4) Si $e1 \pm e2 \neq e$ et $e2 \neq e$

(5) $e = e1 \pm e2$

(6) $e = e1 \pm e2$ et $e1 \neq \tau$

rec I $\dfrac{(\text{rec } x.p)^{m(A)} \vdash A}{\text{rec } x.p \vdash A}$ $\dfrac{A,(\text{rec } x.p)^{m(A)} \vdash B}{A, \text{rec } x.p \vdash B}$

||I $\dfrac{p \vdash A \quad A,q \vdash B}{p \,\|\, q \vdash B}$ $\dfrac{p \vdash A \quad A,q \vdash B}{q \| p \vdash B}$ $\dfrac{A, p \vdash B \quad B, q \vdash C}{A, p \| q \vdash C}$

$\dfrac{A, p \vdash B \quad B, q \vdash C}{A, q \| p \vdash C}$ (7) $\dfrac{A,p \vdash B \quad B, q \vdash [e]\, C \quad A,q \vdash D \quad D, p \vdash [e]\, C}{A, p \| q \vdash [e] C}$

Remarques : Avant d'établir les résultats de correction et de complétude de Sys1, nous allons d'abord prouver un certain nombre de lemmes et de propositions utiles .

Lemme 4.10
Si $p \| q \models A$ alors il existe C tel que $p \models C$ et $C, q \models A$

Preuve
Par induction sur la structure de A.

- (A = Tr) C = Tr convient.
- (A = Fal) On n'a jamais $p \| q \models$ Fal .
- (A= $A1 \vee A2$) Si $p \| q \models A1 \vee A2$ alors soit $p \| q \models A1$ soit $p \| q \models A2$. Si $p \| q \models A1$ alors par hypothèse d'induction il existe C1 tel que $p \models C1$ et $C1, q \models A1$ donc $C1, q \models A1 \vee A2$. Si $p \| q \models A2$ alors par hypothèse d'induction il existe C2 tel que $p \models C2$ et $C2,q \models A2$ donc $C2, q \models A1 \vee A2$. C = C1 ou C = C2 convient.
- (A =$A1 \wedge A2$) : Si $p \| q \models A1 \wedge A2$ alors $p \| q \models A1$ et $p \| q \models A2$. Par hypothèse d'induction il existe C1 tel que $p \models C1$ et $C1,q \models A1$ et il existe C2 tel que $p \models C2$ et $C2, q \models A2$. Donc $p \models C1 \wedge C2$ et $C1 \wedge C2, q \models A1 \wedge A2$. $C = C1 \wedge C2$ convient.
- (A= <e>A1). Trois sous cas sont à considérer :

  Sous cas1 : $p \xrightarrow{e} p'$ avec $p' \| q \models A1$. Par hypothèse d' induction , il existe C' tel que $p \models C'$ et $C',q \models A1$ donc $p \models \langle e \rangle C'$ et $\langle e \rangle C', q \models \langle e \rangle A1$. D'où C= <e>C' convient.

  Sous cas2 : $q \xrightarrow{e} q'$ avec $p \| q' \models A1$.Par hypothèse d' induction, il existe C tels que $p \models C$ et $C,q' \models A1$. Donc $C, q \models \langle e \rangle A1$. D'où C convient.

(7) Si c n'est pas de la forme [e]b

Sous cas 3 : $p \xrightarrow{e_1} p'$, $q \xrightarrow{e_2} q'$ avec $e = e_1 \pm e_2$ et $p' \| q' \models A1$: Par hypothèse d'induction, il existe C' tel que $p' \models C'$ et $C', q' \models A1$. Donc $p \models \langle e_1 \rangle C'$ et $\langle e_1 \rangle C', q \models \langle e \rangle A1$. D'où $C = \langle e_1 \rangle C'$convient.

- $(A = [e]A1)$. $p \| q \models [e]A1 \Rightarrow \forall p \xrightarrow{e} p_i$ $p_i \| q \models A1$ et $\forall q \xrightarrow{e} q_j$ $p \| q_j \models A1$ et $\forall p \xrightarrow{e_1} p_k$ $\forall q \xrightarrow{e_2} q_l$ tel que $e = e_1 \pm e_2$ $p_k \| q_l \models A$.
  Si $p_i \| q \models A1$ alors il existe C_i tel que $p_i \models C_i$ et $C_i, q \models A1 \Rightarrow p \models [e] \bigvee_i C_i$ et $\bigvee_i C_i, q \models A1$.
  Si $p \| q_j \models A1$ alors il existe D_j tel que $p \models D_j$ et $D_j, q_j \models A1$. Donc $p \models \bigwedge_j D_j$ et $\bigwedge_j D_j, q_j \models A1$.
  Si $p_k \| q_l \models A1$ alors il existe R_{kl} tel que $p_k \models R_{kl}$ et $R_{kl}, q_l \models A1$.
  $C = [e] \bigvee_i C_i) \wedge (\bigwedge_j D_j) \wedge (\bigwedge_{e1+e2=e} [e_1] \bigwedge_k \bigvee_l R_{kl})$ convient. C est une formule HML et vérifie $p \models C$ et $C, q \models [e]A1$.

- $(A = \langle \underline{e} \rangle A1)$ Similaire aux ous cas 2 et 3 de $\langle e \rangle A1$.
- $(A = [\underline{e}]A1)$ A partir du cas [e] A1 on a $C = (\bigwedge_j D_j) \wedge (\bigwedge_{e1+e2=e} [e1] \bigwedge_k \bigvee_l R_{kl})$ qui convient. □

Remarque : Le lemme précédent exhibe pour p, q et A fixés une formule C vérifiant $p \models C$ et $C, q \models A$. On note dans la suite $C(p, q, A)$ la formule C obtenue par ce lemme.

Proposition 4.11
Pour tout A fixé l'ensemble $C(A) = \{ C(p, q, A)\ p \models C \text{ et } C, q \models A \}$ est fini.

Preuve : (par induction sur la structure de A)

- $C(Tr) = \{Tr\}$
- $C(Fal) = \{Fal\}$
- $C(B1 \vee B2) \subseteq C(B1) \cup C(B2)$
- $C(B1 \wedge B2) \subseteq \{ C1 \wedge C2 \mid Ci \in (Bi) \}$
- $C(\langle e \rangle B) \subseteq \{ \langle e_1 \rangle C1 \mid e_1 \subseteq e \text{ et } C1 \in C(B) \}$
- $C(\langle \underline{e} \rangle B)$ similaire au cas cas précédent.
- $C[e]B) \subseteq \{ C1 \wedge C2 \wedge C3 \mid C1 = [e] \bigvee_{i \in I} D_i \quad Di \in C(B)$, $C2 = \bigwedge_{j \in J} D_j \quad Dj \in C(B)$ et $C3 = \bigwedge_{e1 \pm e2 = e} [e_1] \bigwedge_k \bigwedge_l R_{kl}, \quad R_{kl} \in (B) \}$.
- $C[\underline{e}]B)$: Similaire au cas précédent. □

Lemme 4.12

Si A, p||q $\models$ B et B n'est pas de la forme [e] C alors il existe D tel que A,p $\models$ D et D, q $\models$ B.

Preuve

Si A, p ||q $\models$ B alors $\forall$ r $\models$A (r ||p)||q $\models$ B. Par le lemme 4. 10 il existe Cr dépendant de r tel que r|| p$\models$ Cr et Cr, q$\models$ B. Donc $\bigvee_r C_r$,q$\models$ B. A, p $\models \bigvee_r C_r$ puisque $\forall$ r $\models$ A r||p $\models$ C_r donc r||p $\models \bigvee_r C_r$. C = $\bigvee_r$ Cr convient. □

Lemme 4.13

Si A , p|| q $\models$ [e]B alors il existe D1, D2 telles que :

C1, q $\models$D1 D1, q $\models$ [e]B

C2, p $\models$ D2 D2, q $\models$ [e]B

Preuve

Si A, p||q$\models$ [e]B alors $\forall$ r $\models$ A (r||p)||q $\models$ [e]B et (r||q)||p $\models$[e]B. Par le lemme 4.10 nous avons $\forall$ r$\models$A il existe Cr, Dr tels que r||p$\models$ Cr et Cr, q$\models$ [e]B et r||q$\models$ Dr et Dr, p$\models$ [e]B.

D'où $\bigvee_r C_r$, q$\models$ [e]B et $\bigvee_r D_r$, p$\models$ [e]B. A, p $\models \bigvee_r C_r$ et A,q $\models \bigvee_r D_r$. Donc D1= $\bigvee_r C_r$ et D2= $\bigvee_r D_r$ conviennent. □

Lemme 4.14

(i) Si rec x.p $\models$ A alors il existe n telle que (rec x.p)^n $\models$ A.

(ii) Si A, rec x.p $\models$ B alors il existe n telle que A, (rec x.p)^n $\models$ B.

Preuve

(i) Par le lemme 4.4, (rec x.p)^n $\equiv_n$ rec x.p, donc par lemme 1.35 . Si rec x.p $\models$ A alors il existe n telle que (Rec x.p)^n $\models$ A.

(ii) Si A, rec x.p $\models$ B alors par définition, $\forall$ r $\models$A $\exists$ n tel que r ||rec x.p. Comme par le lemme 4.4, (rec x.p)^n $\equiv_n$ rec x.p donc par le lemme 4.5 $\forall$ r $\models$ A $\exists$ n tel que r ||(rec x.p)^n .

D'où A, (rec x.p)^n $\models$ B □

4. 3.2.2 Correction et complétude de Sys_1

Théorème 4.15 (Correction de Sys_1)

Le système Sys1 est correct , c'est à dire :

(i) Si p$\vdash$ B alors p$\models$ B

(ii) Si A,p$\vdash$ B alors A, p$\models$ B

Preuve : Standard. □

Théorème 4.16 (Complétude de Sys_1)
Le système de preuve Sys_1 est complet , c'est à dire $p \in \mathcal{L}_p$, $A \in L_{HML}$ et $B \in L$,

(i) Si $p \models B$ alors $p \vdash B$

(ii) Si $A,p \models B$ alorsil existe C tel que $|C| = |A|$ et $C, q \vdash B$.

Preuve

La preuve de (i) et (ii) se fait par induction sur p puis sur A. Pour la clarté de la présentation de la preuve, nous regroupons les cas qui ne dépendent pas de p afin d'éviter la répétition de certaines preuves identiques.

Preuve de (i)

Les cas où B appartient à HML et p = nil, q+r, e.q sont traités par le théorème 4.7. Il reste le traitement de $p = q \| r$ et $p = rec\ x.q$ qui ne dépendent pas de A.

Cas ($p = q \| r$) Si $q \| r \models A$ alors par le lemme 4.10, il existe C tel que $q \models C$ et $C, r \models A$. Par hypothèse d'induction $q \vdash C$ et il existe D avec $|D| = |C|$ tel que $D, r \vdash A$. D'où par $\|I$, $q \| r \vdash A$.

Cas($p = rec\ x.q$) Si $rec\ x.q \models A$ alors d'après le lemme 4.14, (i) , il existe n tel que $recx.q^n \models A$. Donc par hypothèse d' induction $(rec\ x.q)^n \vdash A$. D'où par rec_1 I, $(rec\ x.q) \vdash A$.

Preuve de (ii)

Cas (B de la forme Tr, Fal ou $B1 \wedge B2$) : Immédiats. Ces cas ne dépendent pas de la forme de p. La preuve est donc la même pour tout $p \in \mathcal{L}_p$.

Cas ($B = B1 \vee B2$) On a $A, p \models B1 \vee B2$ d'où $\forall r \models A$ $r \| p \models B1$ ou $r \| p \models B2$.
On note $R1 = \{r \models A : r \| p \models B1\}$ et $R2 = \{r \models A : r \| p \models B2\}$.
Soient $A1 = A \wedge \bigvee_{r \in R1} C(r, p, B1)$ et $A2 = A \wedge \bigvee_{r \in R2} C(r, p, B2)$. On a $A1, p \models B1$ et $A2,p \models B2$. Si $A1,p \models B1$ alors par induction il existe D1 tel que $|A1| = |D1|$ et $D1, p \vdash B1$ et par $\vee 3$I, $D1,p \vdash B1 \vee B2$. Si $A2,p \models B2$ alors par hypothèse d'induction $\exists$ D2 tel que $|A2| = |D2|$ et $D2, p \vdash B2$ et par $\vee$I on a $D2,p \vdash B1 \vee B2$. Par la règle $\vee$I on a $D1 \vee D2,p \vdash B1 \vee B2$. Il reste à vérifier que $|A1 \vee A2| = |A|$. Evidemment $|A1 \vee A2| \subseteq |A|$. Réciproquement si $r \in |A1|$ alors $r \| p \models B$ par exemple $r \| p \models B1$ donc $r \models C(r, p, B1)$ donc $r \in |A1|$ etc.

Cas (B de la forme <e>B1 .

p = nil. Si A, nil $\models$ <e> B1 alors $\forall$ r $\models$A r $\|$ nil $\models$ <e> B1 . Or r $\|$ nil $\approx_B$ r d'où, d'après le théorème 1.33 , r $\models$ <e> B1. Donc $|A| = |A \wedge \langle e\rangle B1|$. Par ailleurs : on a B1,nil $\models$ B1 et donc par hypothèse d'induction il existe B' tels que $|B1| = |B'|$ et B', nil $\vdash$ B1 .
D'où A $\wedge$ <e>B', nil$\models$<e>B1

p = e_1.q. soit e_1 = e ou $e_1 \neq e$.
Sous cas1 (e_1 = e) A,e.q $\models$ <e>B1 $\Rightarrow$ $\forall$ r $\models$A r $\|$ e.q$\models$ <e>B1. D'après la sémantique opérationnelle de l'opérateur de composition parallèle on a :
$\forall$ r $\models$A r $\|$q$\models$ B1 ou $\exists$ r$\xrightarrow{e}$r' r' $\|$ e.q$\models$ B1 ou $\exists$ r$\xrightarrow{\tau}$r' r' $\|$ q$\models$ B1
On note R1= {r $\models$A : r $\|$ q $\models$ B1} , R2 = {r $\models$ A : $\exists$r$\xrightarrow{e}$r': r' $\|$ e.q$\models$ B1} et
R3= { r $\models$ A : $\exists$ r$\xrightarrow{\tau}$r' : r' $\|$q$\models$ B1} et soient C1 = A $\wedge$ <e>$\bigvee_{r \in R1} C(r, q, B1)$,
C2 = A $\wedge$ $\bigvee_{r \in R2} C(r, e.q, B1)$ et C3 = A $\wedge$ <e>$\bigvee_{r \in R3} C(r', q, B1)$.
$\bigvee_{r \in R1} C(r, q, B1)$, q $\models$ B1 alors par hypothèse d' induction il existe D1 tels que $|D1| =$

$|\bigvee_{r \in R1} C(r, q, B1)|$ et D1, q$\vdash$ B1. D'où par <e>I, D1, e.q$\vdash$ <e>B1 (1)

$\bigvee_{r \in R2} C(r, e.q, B1)$, e.q$\models$ B1 alors par hypothèse d' induction il existe D2 tel que $|D2| = |\bigvee_{r \in R2} C(r, e.q, B1)|$. Par <e>I , <e>D2, e.q$\vdash$<e> B1 et par la règle $\wedge$I, on a
A$\wedge$<e>D2,e.q$\vdash$<e>B1. (2)
$\bigvee_{r \in R3} C(r', q, B1)$, q $\models$ B alors par hypothèse d' induction il existe D3 tel que
$|\bigvee_{r \in R3} C(r',q,B)| = |D3|$. Par <e>I on a <$\tau$ >D3, e.q$\vdash$<e> B1 et par $\wedge$I ,
A$\wedge$ <τ >D3,e.q$\vdash$<e> B (3)

A partir de (1), (2) et (3) et par $\vee$I on a D1$\vee$ (A $\wedge$ <e>D2) $\vee$ (A $\wedge$ <τ>D3) , e.q$\vdash$ <e>B1.
Comme $|A| = |C1 \vee C2 \vee C3| = |D1 \vee (A \wedge \langle e\rangle D2) \vee (A \wedge \langle\tau\rangle D3)|$ donc
C =D1$\vee$ (A $\wedge$ <e>D2)$\vee$ (A$\wedge$<τ>D3) convient.

(*Sous cas $e_1 \neq e$*) . Si A,e_1.q$\models$<e>B1 alors $\forall$ r $\models$A r $\|$ e_1.q $\models$ <e>B1. D'après la sémantique opérationnelle de l'opérateur de composition parallèle, on a : $\forall$ r $\models$A : $\exists$ r $\xrightarrow{e_2}$ r' tel que
e =$e_1 \pm e_2$ et r' $\|$ e1.q$\models$ B1 ou $\exists$ r $\xrightarrow{e}$r' r' $\|$ e_1. q$\models$ B1. On note :
R1= {r $\models$A : $\exists$ r $\xrightarrow{e}$r' : r' $\|$ e_1. q$\models$ B1} et R2 = { r $\models$A $\exists$ r $\xrightarrow{e_2}$ r' : e = $e_1 \pm e_2$ et
r' $\|$ q$\models$ B1} et soient C1 = C1 = A $\wedge$ <e>$\bigvee_{r \in R1} C(r', e.q, B1)$ et

C2 = A $\wedge \bigwedge_{e1 \pm e2 = e}$ <e_2> $\bigvee_{r \in R2} C(r', e_2.q, B1)$. On a alors :

$\bigvee_{r \in R1} C(r', e.q, B1), e_1.q \models B1$. Par hypothèse d'induction il existe D1 tel que $D1, e_1.q \vdash B1$ avec $|\bigvee_{r \in R1} C(r', e.q, B1)| = |D1|$. Par <e>I, $<e>D1, e_1.q \vdash <e>B1$ et par $\wedge$I, $A \wedge <e>D1, e_1.q \vdash <e>B1$. (1)

$\bigvee_{r \in R2} C(r', e_2.q, B1), q \models B1$. Par hypothèse d'nduction, il existe D2 tel que $|D2| = \bigvee_{r \in R2} C(r', e_2.q, B1)$ et $D2, q \vdash B1$. Par <e>I, $<e_2>D1, e_1.q \vdash <e>B1$ et par $\wedge_3$I, $\Lambda_{e1 \pm e2 = e} <e_2> \bigvee_{r \in R2} C(r', e_2.q, B1)$, $e_1.q \vdash <e>B1$ et par $\wedge_3$I une deuxième fois $A \wedge <e_2>D1, e_1.q \vdash <e>B1$ (2)

A partir de (1) et (2) et par $\vee$I, on a $(A \wedge <e>D1) \vee (A \wedge <e_1>D2), q \vdash <e>B1$.
$C = (A \wedge <e>D1) \vee (A \wedge <e_1>D2)$ convient.

p = q+s

On note $R1 = \{r \models A : r \| q \models <\underline{e}>B1\}$, $R2 = \{r \models A : r \| s \models <\underline{e}>B1\}$

$R3 = \{r \models A : \exists r \xrightarrow{e} r'$ et $r' \| q+s \models B1\}$ et soient C1, C2 et C3 telles que $|A| = |C1 \vee C2 \vee C3|$ où $C1 = A = A \wedge <e>\bigvee_{r \in R1} C(r, q, <\underline{e}>B)$, $C2 = A \wedge \bigvee_{r \in R2} C(r, s, <\underline{e}>B)$ et $C3 = A \wedge \bigvee_{r \in R3} C(r', q+s, B)$.

$\bigvee_{r \in R1} C(r, q, <\underline{e}>B,, q \models <\underline{e}>B1$ alors par hypothèse d' induction il existe D1 tels que $D1, q \vdash <\underline{e}>B1$ et $|C1| = |D1|$. Par +I, $D1, q+s \vdash <\underline{e}>B1$. Donc $<e>D1, q+s \vdash <e>B1$ par <e>I et $A \wedge <e>D1, q+s \vdash <e>B1$

$\bigvee_{r \in R2} C(r, s, <\underline{e}>B1), q \models <\underline{e}>B1$ alors par hypothèse d' induction il existe D2 tels que $D2, q \vdash <\underline{e}>B1$ et $|C2| = |D2|$ et par +I, $D2, q+s \vdash <\underline{e}>B1$. Donc $D2, q+s \vdash <\underline{e}>B1$ par <e>I.

$\bigvee_{r \in R3} C(r', q+s, B1), q+s \models B$ donc par hypothèse d'induction, il existe D3 tels que $D3, q+s \vdash B1$ et $|D3| = |\bigvee_{r \in R3} C(r', q+s, B1)|$. Par <e>I, $<e>D3, q+s \vdash <e>B1$ et par $\wedge$ I, $A \wedge <e>D3, q+s \vdash <e>B$. Par la règle d'introduction du $\vee$, on a :
$D1 \vee D2 \vee (A \wedge <e>D3), q+s \vdash <e>B1$. $C = D1 \vee D2 \vee (A \wedge <e>D3)$ convient.

p = q||r . D'après le lemme 4.10 il existe D tel que $A, q \models D$ et $D, r \models B1$. Par hypothèse d'nduction il existe C vérifiant $|C| = |A|$ et tel que $C, q \vdash D$ et $D, r \vdash B1$. Par ||I $C, q||r \vdash B1$.

p = rec x.q. D'après le lemme 4.14 (ii) , il existe n tel que $A, (rec\ x.q)^n \models B1$. Par hypothèse d'induction il existe C tel que $|C| = |A|$ et $C, (rec\ x.q)^n \vdash B1$. Par rec_2 I , $C, rec\ x.q \vdash B1$.

Les cas où A est de la forme [e]B1, <e>B1,ou [e]B se traitent de manièresimilaire. □

4.3.2.3 exemple : $\{a, b\}.nil \parallel \{c\}nil \vdash \langle\{a, b, c\}\rangle Tr$

$$\dfrac{\dfrac{Ax_1\ \dfrac{}{nil \vdash Tr}}{\langle\{a, b\}\rangle I\ \{a, b\}nil \vdash \langle\{a, b\}\rangle Tr} \qquad \dfrac{\dfrac{Ax_2\ \dfrac{}{Tr, nil \vdash Tr}}{\langle\{a, b, c\}\rangle I\ \langle\{a, b\}\rangle Tr, \{c\}nil \vdash \langle\{a, b, c\}\rangle Tr}}{\langle\{a, b, c\}\rangle I\ \langle\{a, b\}\rangle Tr, \{c\}nil \vdash \langle\{a, b, c\}\rangle Tr}}{\parallel I\ \{a, b\}.nil \parallel \{c\}nil \vdash \langle\{a, b, c\}\rangle Tr}$$

4.3.3 Le système de preuve Sys2

4.3.3.1 Motivation

Après avoir proposé un système de preuve Sys_1 correct et complet pour un sous ensemble de l'algèbre considérée, nous allons dans cette section traiter les opérateurs de connexion et de restriction.

La relation sémantique étendue ne nous permet pas d'avoir la correction des règles d'introduction de l'opérateur de composition parallèle lorsque l'on a en plus dans le contexte l'opérateur de connexion car $(p\|q)\,[\![K]\!]\Gamma U$ n'est pas bisimilaire à $(p\,[\![K]\!]\Gamma U \| q)\,[\![K]\!]\Gamma U$.

Exemple : $p=\{a\}nil+\{b\}.Nil$, $q=\{c\}.nil$, $K=\{<a, c>\}$, $U=\{a, c\}$.
En effet $(p\|q)\,[\![K]\!]\Gamma U = \{b\}nil + \tau.nil$ mais $(p[\![K]\!]\Gamma U\|q)\,[\![K]\!]\Gamma U = \{b\}nil$
Ceci est dû au fait que l'opérateur de composition parallèle est global alors que celui de connexion est local. Pour résoudre ce problème nous introduisons les relations relativisées suivantes:

Définition 4.17
$A, B \models_{K, U} C$ ssi $\forall\, p \models A\ \ \forall\, q \models B\ \ \ (p\|q)[\![K]\!]\Gamma U \models C$
$A \models_{K, U} C$ ssi $\forall\, p \models A\ \ p\,[\![K]\!]\Gamma U \models C$
$p \models_{K, U} C$ ssi $p[\![K]\!]\Gamma U \models C$

L'introduction des trois règles de preuve coïncidant avec les trois relations sémantiques nous permet de construire un système de preuve correct et complet Sys_2.

4.3.3.2 Le Système de preuve

Axiomes

$$Ax_1\ \frac{}{p \vdash_{K,U} Tr} \qquad Ax_2\ \frac{}{nil \vdash_{K,U} [e]A} \qquad Ax_3\ \frac{}{A, B \vdash_{K,U} Tr} \qquad Ax_4\ \frac{}{e_1 p \vdash_{K,U} [e]B} \quad (8)$$

(8) si $e1 \neq e \pm s$ pout tout $s \in \varphi(K)$ ou $e \cap U \neq \emptyset$

$Ax_5\ \dfrac{}{Fal, A \vdash_{K,U} B}$ $Ax_6\ \dfrac{}{A, Fal \vdash_{K,U} B}$ $Ax_7\ \dfrac{}{[e_1]A, [e_2]B \vdash_{K,U} [e]C}$ (9)

$Ax_8\ \dfrac{}{A \vdash_{K,U} Tr}$ $Ax_9\ \dfrac{}{Fal \vdash_{K,U} B}$ $Ax_{10}\ \dfrac{}{[e_1]A \vdash_{K,U} [e]B}$ (10)

Règles

$\vee I\ \dfrac{p \vdash_{K,U} A}{p \vdash_{K,U} A \vee B}$ $\dfrac{p \vdash_{K,U} B}{p \vdash_{K,U} A \vee B}$ $\dfrac{A \vdash_{K,U} C}{A \vdash_{K,U} B \vee C}$

$\dfrac{A \vdash_{K,U} B}{A \vdash_{K,U} B \vee C}$ $\dfrac{A \vdash_{K,U} B \quad C \vdash_{K,U} B}{A \vee C \vdash_{K,U} B}$ $\dfrac{A, B \vdash_{K,U} C}{A, B \vdash_{K,U} C \vee D}$

$\dfrac{A, B \vdash_{K,U} D}{A, B \vdash_{K,U} C \vee D}$ $\dfrac{A, B \vdash_{K,U} C \quad D, B \vdash_{K,U} C}{A \vee D, B \vdash_{K,U} C}$ $\dfrac{A, B \vdash_{K,U} C \quad A, D \vdash_{K,U} C}{A, B \vee D \vdash_{K,U} C}$

$\wedge I\ \dfrac{p \vdash_{K,U} A \quad p \vdash_{K,U} B}{p \vdash_{K,U} A \wedge B}$ $\dfrac{A \vdash_{K,U} B \quad A \vdash_{K,U} C}{A \vdash_{K,U} B \wedge C}$ $\dfrac{A \vdash_{K,U} B}{A \wedge C \vdash_{K,U} B}$

$\dfrac{A \vdash_{K,U} B}{C \wedge A \vdash_{K,U} B}$ $\dfrac{A, B \vdash_{K,U} C \quad A, B \vdash_{K,U} D}{A, B \vdash_{K,U} C \wedge D}$ $\dfrac{A, B \vdash_{K,U} C}{A \wedge D, B \vdash_{K,U} C}$

$\dfrac{A, B \vdash_{K,U} C}{A, B \wedge D \vdash_{K,U} C}$ $\dfrac{A, B \vdash_{K,U} C}{D \wedge A, B \vdash_{K,U} C}$ $\dfrac{A, B \vdash_{K,U} C}{A, D \wedge B \vdash_{K,U} C}$

$.I\ \dfrac{p \vdash_{K,U} A}{e.p \vdash_{K,U} \langle e-s \rangle A}$ (11) $\dfrac{p \vdash_{K,U} A}{e.p \vdash_{K,U} [e-s]A}$ (12)

---

(9) si $(e_2 \neq e \pm s$ et $e_1 + c \neq e \pm s$ et $e_1 + e_2 \neq e \pm s)$ pour tout $s \in \varphi(K)$ ou $e \cap U \neq \emptyset$

(10) Si $e_1 \neq e+s$ pour tout $s \in \varphi(K)$ ou $e \cap U \neq \emptyset$

(11) Si $s \in \varphi(K)$ et $(e-s) \cap U = \emptyset$ (12) Si $s \in f(K)$

$[\![]\!]$ I $\dfrac{p \vdash_{K\cup K',U} A}{p[\![K']\!] \vdash_{K,U} A}$ $\lceil$ I $\dfrac{p \vdash_K \lceil V,\ U\cup V}{p \lceil V \vdash_{K,U} B}$ (13) rec I $\dfrac{rec\ x.p \vdash_{K,U} A}{(rec\ x.p)^{m(A)} \vdash_{K,U} A}$

\+ I $\dfrac{p \vdash_{K,U} <e>A}{p+q \vdash_{K,U} <e>A}$ $\dfrac{p \vdash_{K,U} [e]A \quad q \vdash_{K,U} [e]A}{p+q \vdash_{K,U} [e]A}$

<e>I $\dfrac{A \vdash_{K,U} B}{<e>A \vdash_{K,U} <e\text{-}s>B}$ $\dfrac{A, B \vdash_{K,U} C}{A, <e>B \vdash_{K,U} <e\text{-}s>C}$ (11)

$\dfrac{A, B \vdash_{K,U} C}{<e>A, B \vdash_{K,U} <e\text{-}s>C}$ (11) $\dfrac{A, B \vdash_{K,U} C}{<e_1>A, <e_2>B \vdash_{K,U} <e_1 \pm e_2\text{-}s>C}$ (14)

[e] I $\dfrac{A \vdash_{K,U} B}{[e] \vdash_{K,U} [e\text{-}s]B}$ $\dfrac{A,B' \vdash C \quad B, A' \vdash C \quad Bi, B'i \vdash C}{[e]B \wedge \Lambda_\iota [e1_i]Bi\ , [e]B' \wedge \Lambda_\iota [e2_i]B'_i \wedge \vdash [e\ ]C}$

$\|$I $\dfrac{p \vdash A \quad q \vdash B \quad A, B \vdash_{K,U} C}{p \parallel q \vdash_{K,U} C}$ Cut $\dfrac{A,B \vdash_{K,U} C \quad C \vdash_{K',U} D}{A,B \vdash_{K+K',U} D}$

4.3.3.3 Correction et complétude de Sys_2

Lemme 4.18

Pour tout C, tout K tout U il existe C' tel que pour tout p $p \models [\![K]\!] \lceil U \models C$ ssi $p \models C'$

Preuve (par induction sur la structure de C).

- (C = Tr) : C' = Tr convient.
- (C= Fal) : C' = Fal convient.
- (C= C1 $\wedge$ C2) : C = C'1 $\wedge$ C'2 convient.
- (C= C1 $\vee$ C2) : C = C'1 $\vee$ C'2 convient.
- (C= <e>C1) : Si $p \models [\![K]\!] \lceil U \models <e>C1$ alors il existe $p \xrightarrow{e} p' \models [\![K]\!] \lceil U \models C1$.
  Par hypothèse d'induction $p' \models C'1$ et donc il existe s tel que $p \models <e+s> C'1$ et $e \cap U = \emptyset$.

  Donc $p \models \bigvee_{s \in \varphi(K)} <e+s> C'1$. Donc $C' = \bigvee_{s \in \varphi(K)} <e+s> C'1$ si $e \cap U = \emptyset$ et Fal sinon.
- (C =[e]C1) : Si $\cap U \neq \emptyset$ alors C' = Tr convient sinon $C' = \bigwedge_{s \in \varphi(K)} [e+s]$ C'1 convient. □

---

(13) remarque : K $\lceil$ V représente les connexions où les actions de V ne participent pas.

(14) si $s \in \varphi(k)$ et $(e1 \pm e2) \cap U = \emptyset$

Corollaire 4.19
A, B $\models_{K+K',U} C$ alors il existe C' tel que $p \models_{K,U} C'$ et $C' \models_{K',U} C$

Preuve : Découle immédiatement du lemme 4.18. □

Corollaire 4.20
$p \| q \models_{K,U} A$ alors il existe B tel que $p \| q \models B$ et $B \models_{K,U} A$

Preuve : Découle immédiatement du lemme 4.18. □

Lemme 4.21
Si rec x.p $\models_{K,U}$ A alors il existe n tel que $(\text{rec x.p})^n \models_{K,U} A$

Preuve
rec x.p $\equiv_n (\text{rec x.p})^n$ par le lemme 4.4 et (rec x.p) $[\![K]\!]\Gamma U \equiv_n (\text{rec x.p})^n [\![K]\!]\Gamma U$ par le lemme 4.5.Donc d'après le lemme 1.35, si (rec x.p) $[\![K]\!]\Gamma U \models A$ alors il existe n tel que $(\text{recx.p})^n [\![K]\!]\Gamma U \models A$ □

Théorème 4.22 : (Correction de Sys_2)
le système de preuve de Sys_2 est correct i.e

(i). Si $p \vdash_{K,U} B$ alors $p \models_{K,U} B$

(ii). Si $A, B \vdash_{K,U} C$ alors $A, B \models_{K,U} C$

Preuve : Standard. □

Lemme 4.23 (Complétude de $A \models_{K,U} C$)
Si alors $A \models_{K,U} C$ alors il existe B tel que $|B| \models |A|$ et $B \vdash_{K,U} C$.

Preuve (Par induction sur la structure de C)

- (C = Tr) : $A \vdash_{K,U} Tr$ par Ax_8
- (C=Fal) : On n'a jamais $A \models_{K,U} Fal$.
- ($C = A \vee B$, $C = A \wedge B$) : Immédiats.
- ($C = \langle e \rangle B$) Par définition $\forall\ p \models A$ $p[\![K]\!]\Gamma U \models \langle e \rangle B$. Donc $\forall\ p \models A$ Il existe $p[\![K]\!]\Gamma U \xrightarrow{e} p'[\![K]\!]$et $p'[\![K]\!]\Gamma U \models B$. D'après le lemme 4.18 il existe A' tel que $p' \models A'$. Donc $A' \models_{K,U} B$. Par hypothèse d'induction $A' \vdash_{K,U} B$.
  Par $\langle e \rangle I, \langle e+s \rangle$ $A' \vdash_{K,U} \langle e \rangle B$ pour tout $s \in \varphi(K)$.
  Donc $\bigvee_{s \in \varphi(K)} \langle e+s \rangle A' \vdash_{K,U} \langle e \rangle B$. Par $\wedge I$, $A \wedge \bigvee_{s \in \varphi(K)} \langle e+s \rangle \rangle A' \vdash_{K,U} \langle e \rangle B$.

- (C = [e]B) Par définition $\forall$ p$\models$ A p[[K]]ΓU$\models$ [e]B. Donc $\forall$ p$\models$ A $\forall$ p[[K]]ΓU $\xrightarrow{e}$ p'[[K]] p'[[K]]ΓU $\models$ B. D'après le lemme 4.18 il existe A' tel que p' $\models$ A' . Donc A' $\models_{K,U}$ B. Par hypothèse d'induction A' $\vdash_{K,U}$ B. Par [e>],[e+s] A' $\vdash_{K,U}$ [e]B pour tout s $\in \varphi(K)$. Donc $\Lambda_{s \in \varphi(K)}$ [e+s] A' $\vdash_{K,U}$ [e]B. Par $\wedge$I, A $\wedge$ $\Lambda_{s \in \varphi(K)}$ [e+s]A' $\vdash_{K,U}$ [e]B. □

Théorème 4.24 (Complétude de Sys_2)
Le système de preuve Sys_2 est complet , c'est à dire Si p$\models_{K,U}$ B alors p$\vdash_{K,U}$B.

Remarque :
La preuve utilise les lemmes et proposition que nous allons définir et prouver ci aprés.

Soit L_{HML} l'ensemble des formules HML , FN(pour Forme Normale) est un sous ensemble de L_{HML} de formules en forme normale et qui est inductivement défini :

D ::= C | C$\vee$D
C ::= E | G | E $\wedge$G | Tr | Fal
E ::= <e_1>D $\wedge$<e_2>D $\wedge$...<e_n>D
G ::= [e_1]D $\wedge$... [e_n]D $\quad e_i \neq e_j$ pour $i \neq j$

Proposition 4.25
Si A$\in L_{HML}$ alors il existe B$\in$ FN tel que m(B) $\leq$ m(A) et |A| = |B|

Maintenant, définissons une nouvelle relation de preuve $\Vdash$ sur A, B, C où A, B $\in$ FN et C $\in$ L_{HML}.

Définition 4.26 (Définition de la relation de preuve $\Vdash_{K,U}$)
D, D' $\Vdash_{K,U}$ A si $\forall$ C $\in$ D, $\forall$ C'$\in$ D' C, C' $\Vdash_{K,U}$ A
C, C' $\Vdash_{K,U}$ Tr
C, C' $\Vdash_{K,U}$ A si C = Fal ou C' = Fal
C, C' $\Vdash_{K,U}$ A$\vee$ B si C, C' $\Vdash_{K,U}$ A ou C, C' $\Vdash_{K,U}$ B
C, C' $\Vdash_{K,U}$ A $\wedge$ B si C, C' $\Vdash_{K,U}$ A et C, C' $\Vdash_{K,U}$ B
C, C' $\Vdash_{K,U}$ <e>B si e $\cap$ U =$\emptyset$ et soit <e>D $\in$ C et D, C' $\Vdash_{K,U}$ B, soit <e>D'$\in$ C' et C, D' $\Vdash_{K,U}$ B, soit <e_1>D1 $\in$ C et <e_2>D2 $\in$ C' avec e =$e_1 \pm e_2$ et D1, D2 $\Vdash_{K,U}$ B.
C, C' $\Vdash$ [e]B si e $\cap$ U $\neq \emptyset$ et $\forall$ [e]D1 $\in$ C D1, D'$\Vdash$ B et $\forall$ [e]D2 $\in$C' C, D2 $\Vdash$ B et $\forall$ [e]D'1 $\in$ C , $\forall$ [e]D'2 $\in$ C' D'1, D'2 $\Vdash$ B.

Lemme 4.27

$p \| q \models_{K,U} A \Rightarrow \exists D, D'$ tel que $m(D \wedge D') \leq m(A)$ et $p \models D1$ $q \models D2$ et $D1, D2 \| \models_{K,U} A$.

Preuve

Par induction sur la structure de A.

- (A=Tr) Clairement $p \models Tr$ $q \models Tr$ et $Tr, Tr \| \models_{K,U} Tr$.
- (A = Fal) On n'a jamais $p \| q \models_{K,U} Fal$.
- ($A = B \vee C$) Si $p \| q \models_{K,U} B \vee C$ alors $p \| q \models_{K,U} B$ ou $p \| q \models_{K,U} C$. Si $p \| q \models_{K,U} B$ alors il existe D1 et D2 telles que que $p \models D1$, $q \models D2$ et $D1, D2 \| \models_{K,U} B$.
  Donc $D1, D2 \models_{K,U} B \vee C$.
  D = D1 et D'=D2 conviennent. Si $p \| q \models_{K,U} C$ alors il existe D3 et D4 telles que $p \models D3$, $q \models D4$ et D3, D4 $\| \models_{K,U} B$. Donc D3, D4 $\| \models_{K,U} B \vee C$. D = D3 et D'=D4 conviennent.
- ($A = B \wedge C$) Si $p \| q \models_{K,U} B \wedge C$ alors $p \| q \models_{K,U} B$ et $p \| q \models_{K,U} C$. Donc $\exists$ D1, D2, D3, D4 telles que $p \models D1$ $q \models D2$ et $D1,D2 \| \models_{K,U} B$ (1)
  $p \models D3$ $q \models D4$ et $D3,D4 \| \models_{K,U} C$ (2)

  avec $m(D1 \wedge D2) \leq m(B)$ et $m(D3 \wedge D4) \leq m(C)$.
  De (1) et (2) nous obtenons $p \models D1 \wedge D3$ $q \models D2 \wedge D4$ $D1 \wedge D3 , D2 \wedge D4 \| \models_{K,U} B \wedge C$ et $m(D1 \wedge D2 \wedge D3 \wedge D4) \leq m(B \wedge C)$. Donc $D = D1 \wedge D3$ et $D' = D2 \wedge D4$.
- (A= <e>B) Trois sous cas cas sont à considérer.

  Sous cas 1 : Il existe $q \xrightarrow{e} q'$ avec $p \| q' \models_{K,U} B$. Par hypothèse d'induction, il existe D1 et D2 telles que $p \models D1$ $q' \models D2$ $D1,D2 \| \models_{K,U} B$ et $m(D1 \wedge D2) \leq m(B)$. Donc $p \models D1$ $q \models <e>D2$.
  $D1, <e>D2 \| \models_{K,U} <e>B$ et $m(D1 \wedge <e>D2) \leq m(B)$. D'où D= D1 et D'= <e>D2 conviennent.

  Sous cas 2: : Il existe $p \xrightarrow{e} p'$ avec $p' \| q \models_{K,U} B$. Par induction , il existe D1, D2 telles que $p' \models D1$ $q \models D2$ $D1, D2 \| \models_{K,U} B$ et $m(D1 \wedge D2) \leq m(B)$. Donc $p \models <e>D1$ $q \models D2$ et $D1, <e>D2 \| \models_{K,U} <e>B$ et $m(D1 \wedge <e>D2) \leq m(B)$. D'où D= <e>D1 et D'=D2 conviennent.

  Sous cas 3 : $\exists p \xrightarrow{e_1} p'$, $q \xrightarrow{e_2} q'$ avec $p' \| q' \models_{K,U} B$. Par hypothèse d' induction, il existe D1, D2 telles que $p' \models D1$ $q' \models D2$ $D1, D2 \| \models_{K,U} B$ et $m(D1 \wedge D2) \leq m(B)$ alors $p \models <e_1>D1$ $q \models <e_2>D2$ $<e_1> D1, <e_2>D2 \| \models_{K,U} <e>B$ et $m(<e_1>D1 \wedge <e_2>D2) \leq$ m(B). Donc D= $<e_1>D1$ et D'= $<e_2>D2$ conviennent

- (A= [e]B)

  Sous cas 1 ($e \cap U \neq \emptyset$) : $D = D' = Tr$

  Sous cas 2 ($e \cap U = \emptyset$) : $\forall\ q \xrightarrow{e} q_i$ $p \| q_i \models_{K,U} B$ et $\forall\ p \xrightarrow{e} p_j$ $p_j \| q \models_{K,U} B$ et $\forall\ p \xrightarrow{e_1} p_k$ $\forall\ q \xrightarrow{e_2} q_l$ $p_k \| q_l \models_{K,U} B$.

  $p \| q_i \models_{K,U} B$ donc il existe par hypothèse d'induction Di et D'i telles que :
  $p \models Di$ $q \models D'i$ $Di, D'i \models_{K,U} B$ et $m(Di \wedge D'i) \leq m(B)$.

  Soit $S = \Lambda_i Di$ et $S' = \vee_i D'_i$ alors $p \models S$ $q \models [e]S'$ $S, S' \models_{K,U} B$ et $m(S \wedge S') \leq m(B)$. (1)

  $p_j \| q \models_{K,U} B$ donc il existe E'_j et E_j telles que $p \models E'_j$ $q \models E_j$ $E'_j, E_j \models_{K,U} B$ et $m(E'_j \wedge E_j) \leq m(B)$. Supposons que $E' = \vee_j E'_j$ et $E = \Lambda_j E_j$ alors
  $p \models [e]E'$ $q \models E'$ $E', E \models_{K,U} B$ et $m(S \wedge S') \leq m(B)$. (2)

  $p_k \| q_l \models_{K,U} B \Rightarrow \exists R_k, R'_l$ telles que $p_k \models R_k$ $q_l \models R'_l$ $R_k, R'_l \models_{K,U} B$ et $m(R_k \wedge R'_l) \leq m(B)$. Supposons $R = \vee_k R_{k,l}$ et $R' = \vee_l R'_l$ alors

  $p \models \Lambda_{e_1 \pm e_2 = e} [e_1]R$ $q \models \Lambda_{e_1 \pm e_2 = e} [e_2]R'$ $R, R' \models_{K,U} B$ (3)

  De (1), (2) et (3) on a $p \models S \wedge [e]E' \wedge \Lambda_{e_1 \pm e_2 = e} [e_2]R$

  $q \models [e]S' \wedge E \wedge \Lambda_{e_1 \pm e_2 = e} [e_2]R'$

  $S \wedge [e]E' \wedge \Lambda_{e1 \pm e2 = e} [e2]R$, $[e]S' \wedge E \wedge \Lambda_{e1 \pm e2 = e} [e_2]R' \models_{K,U} [e]B$.

  D'où $D = S \wedge [e]E' \wedge \Lambda_{e_1 \pm e_2 = e} [e_1]R$ et $D' = [e]S' \wedge E \wedge \Lambda_{e_1 \pm e_2 = e} [e_2]R$ conviennent. □

Lemme 4.28
$D, D' \models_{K,U} A \Rightarrow$ $D, D' \vdash_{K,U} A$

Preuve (par induction sur la structure de A)

- (A= Tr) : $D, D' \models_{K,U} Tr \Rightarrow \forall\ C \in D, \forall\ C' \in D'$ $C, C' \models_{K,U} Tr$. Donc $\forall\ C \in D$ $\forall\ C' \in D'$ $C, C' \vdash_{K,U} Tr$ (Ax_3).D'où $D, D' \vdash_{K,U} Tr$.
- (A = Fal) : $D, D' \models_{K,U} Fal \Rightarrow \forall\ C \in D, \forall\ C' \in D'$ $C, C' \models_{K,U} Fal$. C'est à dire $\forall\ C \in D$, $\forall\ C' \in D$ $C = Fal$ ou $C' = Fal$. Donc $\forall\ C \in D$, $\forall\ C' \in D'$ $C, C' \vdash_{K,U} Fal$ et finalement $D, D' \vdash_{K,U} Fal$.
- (A = A1 ∨ A2) : $D, D' \models_{K,U} A1 \vee A2 \Rightarrow \forall\ C \in D, \forall\ C' \in D'$ $C, C' \models_{K,U} A1 \vee A2 \Rightarrow \forall$ $C' \in D'$ $C, C' \models_{K,U} A1$ ou $C, C' \models_{K,U} A2$. Par hypothèse d'induction $\forall\ C' \in D'$ C, $C' \vdash_{K,U} A1$ ou $C, C' \vdash_{K,U} A2$. D'où $\forall\ C \in D$, $\forall\ C' \in D'$ $C, C' \vdash_{K,U} A1 \vee A2$. Donc D, $D' \vdash_{K,U} A1 \vee A2$.

• (A=A1 ∧ A2) : D, D' $\models_{K,U}$ A1 ∧ A2 ⇒ ∀ C ∈ D, ∀ C'∈ D' C, C' $\models_{K,U}$ A1 ∧ A2, ∀ C' ∈ D' C, C' $\models_{K,U}$ A1 et C, C' $\models_{K,U}$ A2. Par induction ∀ C'∈ D' C, C' $\vdash_{K,U}$ A1 et C, C' $\vdash_{K,U}$ A2. Donc ∀ C ∈ D, ∀ C'∈ D' C, C' $\vdash_{K,U}$ A1 ∧ A2 . Donc D, D' $\vdash_{K,U}$ A1 ∧ A2.

• (A= <e>B) : D, D' $\models_{K,U}$ <e>B ⇒ ∀ C ∈ D,∀ C'∈ D' C, C' $\models_{K,U}$ <e>B.
Donc ∀ C ∈ D, ∀ C' ∈ D' : <e>D1∈ C et D1, C' $\models_{K,U}$ B ou <e>D2 ∈ C' et C, D2 $\models_{K,U}$ B ou <e1>D3 ∈ C, <e2>D4 ∈ C' et D3, D4 $\models_{K,U}$ B et e =e1±e2.
Comme <e>D1∈ C et $<e_1>$D3 ∈ C alors <e>D1 ∧ $<e_1>$D3 ∈ C. Par ailleurs, <e>D2∈ C' et $<e_2>$D4 ∈ C' donc <e>D2 . Par ailleurs, <e>D2∈ C' et $<e_2>$D4 ∈ C' donc <e>D2. Par ailleurs, <e>D2 ∈ C' et $<e_2>$D4 ∈ C' donc <e>D2 ∧ $<e_2>$D4 ∈ C' .
Soient D1, C' $\vdash_{K,U}$ B C, D2 $\vdash_{K,U}$ B D3, D4 $\vdash_{K,U}$ B . Donc <e>D1, C' $\vdash_{K,U}$ B C, <e>D2 $\vdash_{K,U}$ B $<e_1>$D3, $<e_2>$D4 $\vdash_{K,U}$ B si e = $e_1 \pm e_2$.
D'où <e>D1 ∧ C ∧ $<e_1>$D3, C' ∧ <e>D2 ∧ $<e_2>$D4 $\vdash_{K,U}$ <e>B. Comme <e>D1 ∧ $<e_1>$D3∈ C et <e>D2∧ $<e_2>$D4 ∈ C' alors C, C' $\vdash_{K,U}$ <e>B et finalement D, D' $\vdash_{K,U}$ <e>B.

• (A= [e]B): D, D' $\models_{K,U}$ [e]B ⇒∀ C ∈ D, ∀ C'∈ D' C, C' $\models_{K,U}$ [e]B. Si e∩U≠ Ø alors $[e_1]$E ∈C et $[e_2]$E' ∈ C' et $[e_1]$E, $[e_2]$E' $\vdash_{K,U}$ [e] B (Par Ax_9) sinon K = U = Ø et ∀ [e] D1 ∈ C D1,C' $\models$ B, ∀ [e] D2 ∈ C' C, D2 $\models$ B et ∀ $[e_1]$ D1 ∈ C ∀ $[e_2]$ D2 ∈ C' D1, D2 $\models$ B. Par hypothèse d'induction, ∀ [e] D1 ∈ C D1, C' $\vdash$ B, ∀ [e] D2 ∈ C' C, D2 $\vdash$ B et ∀ $[e_1]$ Di ∈ C ∀ $[e_2]$ D'i ∈ C' pour e = $e_1 \pm e_2$,D1, D2 $\vdash$ B.Par $[e]_2$ I ,

[e]D1 ∧ $\bigwedge_i$[e_{1i}] Di , [e]D2 ∧ $\bigwedge_i$[e_{2i}] D'i $\vdash$ [e]B □

Preuve du théorème 4.24 (Complétude de Sys_2)

Nous voulons montrer que si p $\models_{K,U}$ A alors p $\vdash_{K,U}$ A. La preuve se fait par induction sur A puis p.

Cas (A de la forme Tr, Fal , B ∨ C et B ∧ C }. Immédiats.

Cas (A de la forme <e>B) On fait une induction sur p.

Sous cas1 (p = nil) On n'a jamais nil $\models_{K,U}$ <e>B

Sous cas 2 (p = e_1.q). On a donc e_1=e±s où s ∈φ(K) et q $\models_{K,U}$ B. Par hypothèse d'induction q $\vdash_{K,U}$ B et par <e>I, e.q $\vdash_{K,U}$ <e>B.

Sous cas 3 (p=q+r). Soit q $\models_{K,U}$ <e>B soit r $\models_{K,U}$ <e>B.Si q $\models_{K,U}$ <e>B alors par hypothèse d'induction q $\vdash_{K,U}$ <e>B et +I, q+r $\vdash_{K,U}$ <e>B.
Si q $\models_{K,U}$ <e>B alors par hypothèse d'induction q $\vdash_{K,U}$ <e>B et par +I, q+r $\vdash_{K,U}$ <e>B

Sous cas 4 (p = rec x.q) . Par le lemme 4.21 il existe n tel que $(\text{rec } x.q)^n \models_{K,U} A$. Par hypothèse d'induction $(\text{rec } x.q)^n \vdash A$ et par rec I, $\text{rec } x.q \vdash_{K,U} A$.

Sous cas 5 (p = q|| r). Par le lemme 4.27 il existe D1 et D2 telles que $q \models D1$ $r \models D2$ et $D1, D2 \models C$. Par hypothèse d'induction $q \vdash D1$ $r \vdash D2$ et par le lemme 4.29, $D1,D2 \vdash A$ et finalement par || I, $q||r \vdash_{K,U} A$.

Sous cas 6 (p= q [[K']]). Si $q[[K']] \models_{K,U} A$ alors $q \models_{K \cup K',U} A$. Par hypothèse d'induction, $q \vdash_{K \cup K',U} A$. Par [[]] I, $q [[K']] \vdash_{K,U} A$.

Sous cas7 (p = q⌈V) : Si $q⌈V \models_{K,U} A$ alors $q⌈V [[K]]⌈U \models A$.
Doncq $[[K⌈V]]⌈U⌈V \models A$, c-à-d. $q \models_{[K⌈V],U \pm V} A$. Par hypothèse d' induction, $q \vdash_{K⌈V,U \pm V} A$ et par ⌈ I, $q⌈V \vdash_{K,U} A$.

Cas (A de la forme [e]B) On fait une induction sur p.

Sous cas 1 (p= nil) : On a $\text{nil} \vdash_{K,U} [e]B$ par Ax_2.

Sous cas2 (p = e_1.q) . Si $e \cap U \neq \emptyset$ alors $e_1.q \vdash_{K,U} [e]B$ par Ax_4 sinon $e_1 = e \pm s$ pour $s \in \varphi(K)$ et $q \models_{K,U} B$.Par induction, $q \vdash_{K,U} B$ et par . I, $e_1.q \vdash_{K,U} [e]B$.

Sous cas 3 (p = q+r). On a $q \models_{K,U} [e]B$ et $r \models_{K,U} [e]B$. Par hypothèse d' induction $q \vdash_{K,U} [e]B$ et $r \vdash_{K,U} [e]B$. par +I, $q+r \vdash_{K,U} [e]B$.

Sous cas 4 (p = rec x.q) similaire au sous cas 4 avec A = <e>B.

Sous cas 5 : (p =q|| r) : D'après le corollaire 4.20 il existe C tel que $q||r \models C$ et $C \models_{K,U} A$. Par le lemme 4.27, il existe D1, D2 tel que $m(D1 \wedge D2) \leq m(A)$ et $q \models D1$ et $r \models D2$ et $D1,D2 \models A$. Par hypothèse d' induction , $q \vdash D1$ et $r \vdash D2$ et par le lemme 4.28 $D1,D2 \vdash C$. Par le lemme 4.23 , $C \vdash_{K,U} A$. Par la règle cut $D1,D2 \vdash_{K,U} A$ et enfin par ||I, $q||r \vdash_{K,U} A$

Les *sous cas 6 à 7* pour A de la forme [e]B se traitent exactement comme les *sous cas 6 à 7* pour A de la forme <e>B. □

4.3.3.4 Exemple : $\{a, b\}.nil \,\|\, \{c\}nil \vdash_{\{\langle a, c\rangle\}, \{a, c\}} \langle b\rangle Tr$

$$\dfrac{\dfrac{Ax_1\ \dfrac{}{nil \vdash \{\langle a, c\rangle\}, \{a, c\}\ Tr}}{\{a, b\}.nil \vdash \{\langle a, c\rangle\}, \{a, c\}\ \langle b\rangle Tr}\langle e\rangle_5 I \quad \dfrac{Ax_1\ \dfrac{}{nil \vdash \{\langle a, c\rangle\}, \{a, c\}\ Tr}}{\{c\}.nil \vdash \{\langle a, c\rangle\}, \{a, c\}\ Tr}\langle e\rangle_5 I \quad Ax_3\ \dfrac{}{Tr, Tr \vdash \{\langle a, c\rangle\}, \{a, c\}\ Tr}}{\{a, b\}.nil \,\|\, \{c\}nil \vdash_{\{\langle a, c\rangle\}, \{a, c\}} \langle b\rangle Tr)}\,\|I$$

4.4 Conclusion

Dans ce chapitre, nous avons proposé un système de preuve correct et complet pour un sous ensemble de HAL. Ce système est compositionnel, c'est à dire les règles du système tiennent compte de la structure de processus. La construction de ce système nous a permis de mieux comprendre, au moins au niveau théorique, certains opérateurs de HAL. Il nous a permis de bien saisir la notion de bisimulation et sa caractérisation par HML. L'aspect compositionnalité est d'un intérêt certain : Les réseaux de processus ne doivent pas être obligatoirement définis de manière statique. Une suite intéressante à ce travail serait de développer de tels systèmes en considérant le même langage de processus et des logiques plus expressives que HML.

Conclusion

Résultats

Dans ce travail, nous avons proposé une nouvelle algèbre de processus HAL (The Herbrand Agent Language) orientée spécification de machines à inférence parallèle. HAL est une généralisation de FP2. C'est une algèbre de processus où , comme dans FP2 une action atomique est un ensemble de communications synchrones. Contrairement à FP2, où il n'est possible de construire que des combinaisons statiques de processus, HAL permet de définir des définitions récursives de processus et grâce à un jeu d'opérateurs plus riche, ne fait pas de distinction entre les processus élémentaires et combinaisons de processus. Dans HAL, les processus offrent des termes sur leurs portes et les communications peuvent avoir lieu si les termes offerts sur les portes connectées sont unifiables. En général, un évènement, puisqu'il comporte un ensemble de telles communications, résoud donc un ensemble d'équations entre termes. La substitution , solution de ces équations est alors appliquée à la suite du comportement des partenaires de l'évènement.

Nous avons défini HAL formellement. Nous avons à cet effet proposé deux sémantiques.

1) Nous nous sommes basés sur la démarche de Milner dans [Mil 89] pour donner une sémantique close à HAL. Après avoir défini la sémantique opérationnelle, nous avons montré que la notion fondamentale de bisimulation se comportait bien vis à vis de cette nouvelle algèbre. En d'autres termes, la bisimulation forte est bien une congruence relativement aux opérateurs de HAL. Nous avons élargi ce résultat de congruence à la notion de bisimulation sur les termes ouverts. Nous avons ensuite établi un certain nombre de lois algébriques satisfaites par les opérateurs de HAL, quand on se situe dans le contexte de bisimulation forte. Par exemple modulo la bisimulation forte les processus p+q et q+p sont égaux, p|| q et q|| p sont égaux ... etc.

2) le modèle précédent ne permet pas encore de répondre à tous les objectifs fixés au départ. Pour traiter la communication non close, nous avons considéré la variable dans HAL comme variable logique et avons proposé une sémantique avec partage de variable. Comme dans le premier modèle, nous avons proposé une sémantique opérationnelle. Ensuite nous avons défini une nouvelle relation d'équivalence dite bisimulation non close et avons prouvé que c'est une congruence relativement aux opérateurs de HAL.
Nous avons aussi établi un certain nombre de lois algébriques vérifiées dans ce nouveau contexte.

Des applications de HAL au calcul de fonctions définies par réécriture [Che 90 a, Che 90 b] et à l'inférence de clauses de Horn [Jor 91 , BHJ 91 a, BHJ 91 b, BH 92 a, BH 92 b] ont été décrites. Ces applications ont permis de mettre en évidence un paradigme non classique pour la programmation, où tout calcul peut être décrit entièrement au moyen de communications synchrones entre processus sans mémoire.

Enfin , pour mieux comprendre les opérateurs de HAL, particulièrement l'opérateur de connexion, nous avons proposé un système de preuve compositionnel correct et complet pour HAL.

Perspectives

1) Au niveau sémantique :
Nous nous sommes surtout attachés à donner une sémantique " propre " pour HAL. Nous nous sommes placés alors dans le cadre le plus simple basé sur les modèles de graphes. La seule équivalence que nous avons regardé est la bisimulation forte. Tout en restant dans le modèle de graphe, on peut envisager d'étudier les propriétés de congruence de la "weak bisimulation" [Mil 89] par rapport aux opérateurs de HAL. Nous avons cité celle ci à titre d'exemple et aussi parcequ'elle considère l'action interne τ comme une action invisible. Dans [BH 90 a], nous avons proposé une sémantique dénotationnelle basée sur les arbres d'acceptation [Hen 88]. Mais toutes ces équivalences confondent le parallélisme avec le non déterminisme. Une perspective intéressante serait d'étudier une sémantique pour HAL dans le cadre de vrai parallélisme. Une sémantique , définie dans ce dernier contexte a été proposée dans [Bel 92] pour FP2. Une démarche similaire à cette dernière pourrait être suivie pour HAL.

Nous avons élaboré un modèle avec partage de variables . Cette façon de faire qui semble trés prometteuse dans le futur pour l'implémentation de machines à inférence parallèle mérite d'être explorée davantage.

Pour le système de preuve, une perspective intéressante est de l'étendre à une logique plus expressive que HML. Notre premier soucis était la maîtrise des opérateurs de l'algèbtre HAL. Cela étant fait, on peut envisager de voir si l'on peut s'assurer de la complétude du système quand on introduit par exemple la récursivité dans HML [Lar 90b]

2 Au niveau implémentation

On ne peut avoir une expérience complète avec un langage sans l'avoir implémenté. En FP2, les processus peuvent être aisément compilés car la description de réseaux est statique. Pour HAL et HAL_C défini dans [Jor 91], la compilation ne peut se faire aisément. L'interprétation de HAL nécessite une forme générale du théorème d'expansion de Milner [Mil 80]. Une façon d'implémenter directement HAL est possible en traduisant HAL dans un " Net Clause Language " de Markov [Mar 89, MD 90] où les schémas de calculs distribués peuvent être décrits en utilisant l'unification comme paradigme de base.

Bibliographie

[AB 84] D. Austry and G. Boudol. *" Algèbre de Processus et Synchronisation "*. Theoretical Computer Sciences 30(1), pp 91 - 131, 1984.

[BBK 87] J. C. M . Baeten , J. A. Bergstra, and J. W. Klop. *" On the Consistency of Koomen's Fair Abstraction Rule"*. Theoretical Computer Science, 51(1), pp 129 - 176, 1987.

[BC 87] G. Boudol and I. Castellani. " *On the semantics of concurrency. Partial orders and transition systems*" , In Proc. TAPSOFT 87, LNCS 249, pp 122 - 137, Springer Verlag, 1987.

[Bel 90] M. Belmesk. " *An executing model for exploting AND / OR parallelism in logic programs*". In Proc.ACM Int. Symp. Symbolic and Algebraic Computation (ISSAC' 0O), Tokyo, pp 228, August 1990.

[Bel 89] Z. Belmesk. " *Le Langage FP2 pour une méthodologie de spécification et de programmation des protocoles de communication* ". In Proc. 11 th TFSCS. Tunis, mai 1989.

[Bel 90] Z. Belmesk. " *Sémantique du parallélisme et langages de Protocoles de Communication, Application au langage FP*2 ". Thèse de doctorat d'état, Alger, Février 1992.

[BGW] L. Beckman, R. Gustavsson and A. Waern. *"An Algebraic Model of Parallel Execution of Logic Programs"*. In Proc. Symposium on Logic in Computer Science, Cambridge, Mass. pp 50 - 77, June 1986.

[BH 90 a] M.Belmesk and Z.Habbas, *"A New Process Algebra For The Specification of Parallel Machine: HAL"*. The Fifth International Symposium on Computer and Information Sciences. Cappadocia, 1990.

[BH 90 b] M.Belmesk and Z.Habbas. " Step Accreptance Tree Semantics fot the process algebra HAL, The Fifth International Symposium on Computer and Information Sciences. Cappadocia, 1990.

[BH 92 a] M. Belmesk and Z. Habbas. *" A Process Calculus with shared variables "*. Journal of Computers and Artificial Intelligence, Vol 11, No. 4, 335 - 350, 1992.

[BH 92 b] M. Belmesk and Z. Habbas. *"Une algèbre de processus pour la programmation logique "*. Journées Francophones de la programmation logique, JFPL, Lille ,1992.

[BHJ 90 b] M. Belmesk, Z. Habbas, Ph . Jorrand. *"A Process Algebra over The Herbrand Universe : Application toParallelism in Automated Deduction"*. In U.Furbach, Ch.Suttner and B.Fronhöfer, editors. Parallelizations in Inference Systems. LNAI Sub series of LNCS, Springer Verlag 1991.

[BHJ 90 a] M. Belmesk, Z. Habbas, Ph . Jorrand. *" A Process Algebra over The Herbrand Universe : Application to Computation and Deduction Deduction"*. Fourth Workshop On Computer Science Logic, Heidelbezrg, October 1990.

[BHR 84] S. D. Brookes, C. A. R. Hoare and A . W . Roscoe. *" A theory of Communicating Sequential Processes "*. Journal of the ACM, 31(3), pp 560 - 599, July 1984.

[BKO 88] J. A. Bergstra, J. W. Klop and E. - R . Olderog. *" readies and Failures in the Algebra of Communicating Processes"*. SIAM Journal of Computing, 17(6), pp 109 - 137, 1984.

[BKP 85] H. Barringer , R. Kuiper and A. Pnueli,. *"Now you may compose temporal logic specifications"*. Proc ACM Symp on theory of computing , pp1-13, 1985.

[Bou 85] G. Boudol." *Notes on Algebraic Calculi of Processes*". I n K.R.Apt, editor, Logics and Models of Concurrent Systems, NATO ASI Series f13, pp 261 - 303, 1985.

[Br 86] S. D. Brookes and W. C. Rounds. " *Behavioural Equivalence Relations Induced by Programming Logics*". In Proc. 10 th ICALP, Barcelona, LNCS 154, pp 97 - 108, Springer Verlag, July 1986.

[Bri 88] E. Brinksma. " *Information Processing Systems,* Open Systems Interconnection, LOTOS.A Formal Description Based upon the temporal Ordering of Observational Behavior ". Draft International Standard ISO 8807, 1988.

[CH 87] I. Castellani and M. Hennessy. " *Distributed Bisimulations* ". Report 5 87, University of Sussex, July 1987.

[Che 90] F. Cherief. *"A Communication-Based Interpretation of Equationally Defined Functions"*. In Proc of 5 th Int .Symp on Computer and Information Sciences, Cappadocia, pp 33 - 42, November 1990.

[Che 90 b] F. Cherief. " *An Algebraic model for parallel interpretation of equationnally defined functions*". In Proc. ACM Int. Symbolic and Algebraic computation (ISSAC' 9O), Tokyo, pp 285, August 1990.

[DNH 84] R. De Nicola and M. Hennessy. *"Testing equivalences for processes"* .Theoretical Computer Science (34), 1, pp 83-134, 1984.

[DMM 81] R. De Nicola, A. Martelli and U. Montanari. *" Communication through message passing or shared memory : A formal comparison"*. Proc. 2nd Int conf on Distributed Computing Systems. Paris, France, April, 1981.

[DN 87] R. De.Nicola. " *Extensional Equivalences for Transition Systems*". ACTA INFORMATICA 24, pp 211 - 237, 1987.

[Gen 69] G. Gentzen. " *Investigations into logic deduction*". In the collected Works of Gerhard Gentzen, ed. Szabo, North Holland, 1969.

[Gla 90, a] R. J. Van -Glabekk ." *The linear time - Branching time spectrum*". Concur 90, LNCS 458, pp 279 - 297, Amsterdam, The Netherlands, August 1990.

[Gla 90, b] R. J. Van -Glabekk . *"Comparative Semantics and Refinement of Actions"*. Thesis, Centrum VoorWiskunde en Informatica, VRIJE Universteit AMSTERDAM (1990).

[GL 90] R. Gerber and I. Lee. " A Calculus for Communicating shared ressources". Concur 90, LNCS 458, pp 263 - 277, Amsterdam, The Netherlands, August 1990.

[[Hab 91] Z.Habbas. *"A complete and Compositional Modal Proof System For the Process Algebra : HAL"*. 5ième Symposium Franco Soviétique, INFORMATIKA, pp 185 - 206, Grenoble, 1991. To appear in Théor Comp Science , 1992.

[Hen 88] M. Hennessy. *" Algebraic Theory of Processes "*. MIT Press, 1988.

[HI 90] M. Hennessy and A. Ingolfsdottir . *"A Theory of Communicating Processes with Value- Passing"* . In Proc. ICALP's90, Warwick, Lecture Notes in Computer Sciences(443) pp 205 -219, July 1990.

[HM 85] M. Hennessy and R. Milner. *"Algebraic laws for non deterministic and concurrency"*. Journal of the ACM, 32(1), pp 137- 161, january 1985.

[Hoa 78] C. A. R. Hoare. " Events in computation ", PhD thesis, Universite of Edinburgh, 1980.

[Hoa 85] C.A.R. Hoare. "*Communicating Sequential Processes*". Prentice Hall Int., 1985.

[Iba 90] M. B. Ibanez. " *Inférence parallèle et Processus Communicants pour les clauses de Horn. Extension au premier ordre à la méthode de connexion*". Thèse de Doctorat, I. N. P. de Grenoble, March 1990.

[Jor 86] Ph . Jorrand. *" Term Rewriting As a Basis for the Design of a Functional and Parallel Programming Language. A Case Study : The Language FP2"* . In Fundamentals of Artificial Intelligence, LNCS 232, pp 221 - 276. Springer - Verlag, 1988.

[Jor 91] Ph. Jorrand. *"Communication is Computation. A Non Von Neumann Abstract Machine Model"*. Unpublished Notes, 1992.

[Kel 76] R . M . Keller. *" Formal Verification of Parallel Programs "* . Communications of the ACM, 19(7), pp 371 - 384, July 1976.

[Lar 90a] K . G. Larsen. " *Compositional Theories Based over an Operational Semantics of Contexts* " LNCS 430.

[Lar 90b] K . G Larsen, " *Proof Systems for Satisfiability in Hennessy-Milner Logic With Recursion*" . Theoretical Computer Sciences (72, 1990), pp 265-288, Northolland, 1990.

[Mar 89] Z. Markov. " *A Framework for Network Modelling in Prolog*". In Proc. International Joint Conference in Artificial Intelligence, Detroit, pp. 78 - 83, 1989.

[May 87] D. May. " *Communicating Sequential Processes, Transputers and Occam* ". LNCS 272, Springer Verlag, 1987.

[MD 90] Z. Markov and Ch. Dichev." *The Net - Clause Language* " A tool for Data -Driven Inference. In J. van Eijck, editor, Logics and AI, Proc. European Workshop JELIA'90, Amsterdam, Lecture Notes in Artificial Intelligence, No. 478, Springer- Verlag, 1990.

[Mil 80] R . Milner. " *A Calculus of Communicating Systems* ". LNCS 92. Springer Verlag,1980.

[Mil 83] R . Milner. " *Calculi for Synchrony and Asynchrony* ". Theoretical Computer Science, 23(3) , pp 267 - 310, 1983.

[Mil 88] R. Milner. " *Operational and Algebraic Semantics of Concurrent Processes* ". Research Report ECS - LFCS - 88 - 46, Lab. for Foundations of Computer Science, Edinburgh, February 1988. To appear in the Handbook of Theroretical Computer Science.

[Mil 89] R. Milner. " *Communication and Concurrency* ". Prentice Hall Int, 1989.

[Mil 92] R. Milner. *"Functions As Processes"*, Research report INRIA, No 1154, February, 1990.

[Mis 91] J. Misra." *Loosely Coopled Processes (Preliminary Version)* ". Parle 91, LNCS A COMPLETER.

[Pal 90] C. Palamidessi. *"Algebraic Properties of Idempotent Substitutions"*. In Proc . ICALP'90 Warwick, LNCS 443 , pp 386 - 389 , July 1990.

[Par 81] D. Park. *" Concurrency and Automata On Infinite Sequences "*. In Proc. 5 th GI Conf . On Théor. Comp. Sci., LNCS 104, pp 167 - 183, Springer Verlag, March 1981.

[Plé 86] U. Pletat. *"Algebraic specifications of Abstract Data Types and CCS : An Operational Junction"*. In Proc of Sixth IFIP Workshop on Protocol Specification, Testing and Verification, Montreal, June, 1986.

[Plo 81] G. D. Plotkin, *"A structural approach to operational semantics"*. Report DAIMI - FN - 19, Computer Science Dept, Arhus University, Denmark, 1981.

[Pnu 85] A. Pnueli. *"Linear and Branching Structures in the Semantics and Logics of rective systems"*. In Proc. 12th ICALP, Nafplion, LNCS 194, pp 15 - 32. Springer - Verlag, July, 1985.

[Pom 86] L. Pomello.*" Some equivalence notions for concurrent systems"*. In Advances in Petri Nets . G. Rozemberg, editor., LNCS 222, Springer Verlag, pp 15 - 32, 1985.

[Rei 85] W. Reisig. *"Petri Nets"* . EATCS Monographs on the Theo Comp Science, Springer Verlag, 1985.

[Sar 90] V. A. Saraswat. *"Concurrent Constraint Programming"*. In Proc. Principle of Programming Languages (POPL 90), pp 232-245, January, 1990.

[Sch 90] P. Schnoebelen. *" Sémantique du parallélisme et logique temporelle. Application au langage FP2"*. Thèse de Doctorat , I. N. P. de Grnoble, juin 1990.

[SJ 89] Ph. Schnoebelen and Ph. Jorrand. *" Principles of FP2. Term Algebras for Specifications of Parallel Machines "*. In J. W. de. Bakker, editor, Languages for Parallel Architectures design, Semantics, Implementation Models, chapter 5, pp 22" - 273. Wiley, 1989.

[Sti 85 a] C. Stirling. *" A complete modal proof system for a subset of SCCS"*. LNCS 185, pp 253-266, Berlin, March 1985.

[Sti 85 b] C. Stirling. *"A complete modal proof system for a subset of CCS"*, LNCS 194, pp 475-486, Nafplio, Greece, July 1985.

[Sti 87] C. Stirling. *"Modal logics for Communicating Systems"*. Theoretical Computer Science 49 , pp 311-347, North Holland, Amsterdam 1987.

[vGG 89] R. J. van Glabeek and U. Goltg. *"Equivalence Notions for Concurrent systems and Refinement of Actions"*. In Proc. Math. Found. Computer Science, Porabka - Kozubnik, LNCS 379, pp 237 - 248. Springer - Verlag, 1989.

[Win 80] G. Winskel.. *"Events in Computation "* . Ph D thesis, University of Edinburgh, 1980.

[Win 85 a] G. Winskel.. *"On the Composition and Decompositionof Assertions"* LNCS 197, pp ---- , Springer, Berlin 1987.

[Win 85 b] G. Winskel. *" A complete proof system for SCCS with modal assertions"*. LNCS 206,pp 392- 410, Springer, Berlin, 1985.

[Win 87] G. Winskel. *" Event Structures"*. In Advances in Petri Nets, LNCS 255, pp 325 - 397, Springer Verlag, 1985.

[Wol 88] P. Wolper. *" On the relation of programs and computations to Models of Temporal Logic"*. LNCS 357, 1988.

[Wol 89] T. Wolff. *" Implementation of transition semantics for parallel programs with shared variables"*. STACS 89, 6th Annual Symposiumon Theoretical Aspects of Computer Science. LNCS 349, pp 541 - 543, Paderborn, FRG, February, 1989.

[ZRv 85] J. Zwiers, W. P de Roever and P. Emde Boas. *" Compositionality and Concurrent Network : Soundness and Completness of a Proof System "*. LNCS 194, pp 509 - 520, Springer Verlag, July 1985.

Résumé

Dans cette thèse nous proposons une algèbre de processus HAL (The Herbrand Agent Language) servant d'outil de spécification de machines à inférence parallèles. Les applications particulières de cet outil sont la déduction automatique, la réécriture parallèle, la surréduction parallèle... etc. La spécification de machines parallèles consiste à compiler un programme écrit en logique de clauses de Horn, en logique équationnelle ...etc. pour obtenir un réseau de processus parallèles. L'intérêt d'un tel travail est purement théorique. Il ne s'agit pas d'exécuter des programmes parallèles, mais de décrire de manière abstraite des objets parallèles (réseaux de processus) . Nous proposons donc deux sémantiques pour HAL : la première considère la communication close et le parallélisme par envoi de messages, la deuxième considère la communication non close et le parallélisme à la fois par envoi de messages et partage de variables. Pour mieux comprendre certains opérateurs spécifiques à cette algèbre, nous considérons la logique HML (Logique de Hennessy et Milner) comme logique pour décrire des propriétés sur les processus et proposons un système de preuve HML compositionnel correct et complet pour un sous ensemble de HAL.

Mots Clés : Sémantique du Parallélisme, Algèbre de Processus, Sémantique close, Sémantique non close, Modèle avec Partage de Variables, Logique Modale, Système de preuve.

Abstract

In this thesis, we mainly propose a new process algebra HAL (The Herbrand Agent Language " aimed as abstract and concise descriptions of a large class of parallel machines. Particular applications of this algebra are automated deduction, parallel rewritting, parallel narrowing ... etc. The goal of a such goal is theoretical. In effect, we are not interested in exexuting parallel programs but in describing in an abstract way parallel objects (Network processes). We propose in this thesis two semantics for HAL. The first one considers that processus communicate by message passing and the second one considers that processes communicate by both message passing and shared variables. For a more understanding of HAL operators, we consider HML (The Hennessy and Milner Logic) as a language for the specification of properties over processes and we give a correct and complete modal proof system for HAL.

Key Words : Semantics of Concurrency, Process Algebra, Ground Semantics, Non Ground Semantics, Model with Shared Variables, Modal Logic, Proof System.

INPG département des études doctorales

AUTORISATION de SOUTENANCE

VU les dispositions de l'Arrêté du 23 novembre 1988 relatif aux Etudes doctorales

VU les rapports de présentation de

. Monsieur le Professeur ADAMO Jean-Marc
. Monsieur le Professeur BERTHOMIEUX Bernard

Madame Zineb TALANTIKITE épouse HABBAS

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, spécialité " Informatique "

Fait à Grenoble, le 7 septembre 1992

Pour le Président de l'INPG
et par délégation
le Directeur de l'Ecole Doctorale
J.L. LACOUME

INPG