ALDEBARAN : un système de vérification par réduction de processus communicants

Jean-Claude Fernandez

THESE

présentée par

Jean-Claude FERNANDEZ

pour obtenir le titre de **DOCTEUR**
de **L'UNIVERSITE JOSEPH FOURIER- GRENOBLE I**

(arrêté ministériel du 5 juillet 1984)
Spécialité : *INFORMATIQUE*

ALDEBARAN :
UN SYSTEME DE VERIFICATION PAR REDUCTION DE PROCESSUS COMMUNICANTS

Thèse soutenue le 27 mai 1988
Composition du jury :

Président : S. Krakowiak
Examinateurs : A. Arnold
G. Boudol
J. Sifakis
L. Trilling
J. Voiron

Thèse préparée au sein du laboratoire de Génie Informatique de Grenoble

UNIVERSITE Joseph FOURIER (GRENOBLE I)

Président de l'Université :
M. PAYAN Jean Jacques

Année Universitaire 1987 - 1988

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ère Classe

ARNAUD Paul	Chimie Organique
ARVIEU ROBERT	Physique Nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S
AURIAULT Jean-Louis	Mécanique
AYANT Yves	Physique Approfondie
BARBIER Marie-Jeanne	Electrochimie
BARJON Robert	Physique Nucléaire ISN
BARNOUD Fernand	Biochimie Macromoléculaire Végétale
BARRA Jean-René	Statistiques-Mathématiques Appliquées
BECKER Pierre	Physique
BEGUIN Claude	Chimie Organique
BELORISKY Elie	Physique
BENZAKEN Claude	Mathématiques Pures
BERARD Pierre	Mathématiques Pures
BERNARD Alain	Mathématiques Pures
BERTRANDIAS Françoise	Mathématiques Pures
BERTRANDIAS Jean-Paul	Mathématiques Pures
BILLET Jean	Géographie
BOELHER Jean-Paul	Mécanique
BONNIER Jane Marie	Chimie Générale
BOUCHEZ Robert	Physique Nucléaire ISN
BRAVARD Yves	Géographie
CARLIER Georges	Biologie Végétale
CAUQUIS Georges	Chimie Organique
CHARDON Michel	Géographie
CHIBON Pierre	Biologie Animale
COHEN ADDAD Jean-Pierre	Physique
COLIN DE VERDIERE Yves	Mathématiques Pures
CYROT Michel	Physique du Solide
DEBELMAS Jacques	Géologie Générale
DEGRANGE Charles	Zoologie
DEMAILLY Jean-Pierre	Mathématiques Pures
DENEUVILLE Alain	Physique
DEPORTES Charles	Chimie Minérale
DOLIQUE Jean-Michel	Physique des Plasmas
DOUCE Roland	Physiologie Végétale
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques Pures
GAGNAIRE Didier	Chimie Physique
GERMAIN Jean-Pierre	Mécanique,
GIRAUD Pierre	Géologie
HICTER Pierre	Chimie
IDELMAN Simon	Physiologie Animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques Pures
KAHANE André, détaché	Physique
KAHANE Josette	Physique
KRAKOWIAK Sacha	Mathématiques Appliquées

LAJZEROWICZ Jeanine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre-Jean	Mathématiques Appliquées
LEBRETON Alain	Mathématiques Appliquées
DE LEIRIS Joël	Biologie
LHOMME Jean	Chimie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences Nucléaires I.S.N.
LUNA Domingo	Mathématiques Pures
MACHE Régis	Physiologie Végétale
MASCLE Georges	Géologie
MAYNARD Roger	Physique du Solide
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (Biologie Végétale)
PAYAN Jean-Jacques	Mathématiques Pures
PEBAY-PEYROULA Jean-Claude	Physique
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
PIERRE Jean-Louis	Chimie Organique
RENARD Michel	Thermodynamique
RINAUDO Marguerite	Chimie CERMAV
ROSSI André	Biologie
SAXOD Raymond	Biologie Animale
SENGEL Philippe	Biologie Animale
SERGERAERT Francis	Mathématiques Pures
SOUCHIER Bernard	Biologie
SOUTIF Michel	Physique
STUTZ Pierre	Mécanique
TRILLING Laurent	Mathématiques Appliquées
VALENTIN Jacques	Physique Nucléaire I.S.N.
VAN CUTSEM Bernard	Mathématiques Appliquées
VIALON Pierre	Géologie

PROFESSEURS de 2ème Classe

ADIBA Michel	Mathématiques Pures
ANTOINE Pierre	Géologie
ARMAND Gilbert	Géographie
BARET Paul	Chimie
BLANCHI J.Pierre	STAPS
BLUM Jacques	Mathématiques Appliquées
BOITET Christian	Mathématiques Appliquées
BORNAREL Jean	Physique
BRUANDET J.François	Physique
BRUGAL Gérard	Biologie
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CERFF Rudiger	Biologie
CHIARAMELLA Yves	Mathématiques Appliquées
COURT Jean	Chimie
DUFRESNOY Alain	Mathématiques Pures
GASPARD François	Physique
GAUTRON René	Chimie
GENIES Eugène	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude	Sciences Nucléaires
GILLARD Roland	Mathématiques Pures
GIORNI Alain	Sciences Nucléaires
GONZALEZ SPRINBERG Gérardo	Mathématiques Pures
GUIGO Maryse	Géographie
GUMUCHAIN Hervé	Géographie
GUITTON Jacques	Chimie

HACQUES Gérard	Mathématiques Appliquées
HERBIN Jacky	Géographie
HERAULT Jeanny	Physique
JARDON Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKHOVE Claude	Géologie
LONGEQUEUE Nicole	Sciences Nucléaires I.S.N.
LUCAS Robert	Physique
MANDARON Paul	Biologie
MARTINEZ Francis	Mathématiques Appliquées
NEMOZ Alain	Thermodymamique CNRS - CRTBT
OUDET Bruno	Mathématiques Appliquées
PECHER Arnaud	Géologie
PELMONT Jean	Biochimie
PERRIN Claude	Sciences Nucléaires I.S.N.
PFISTER Jean-Claude	Physique du Solide
PIBOULE Michel	Géologie
RAYNAUD Hervé	Mathématiques Appliquées
RICHARD Jean-Marc	Physique
RIEDTMANN Christine	Mathématiques Pures
ROBERT Gilles	Mathématiques Pures
ROBERT Jean-Bernard	Chimie Physique
SARROT-REYNAULD Jean	Géologie
SAYETAT Françoise	Physique
SERVE Denis	Chimie
STOECKEL Frédéric	Physique
SCHOLL Pierre-Claude	Mathématiques Appliquées
SUBRA Robert	Chimie
VALLADE Marcel	Physique
VIDAL Michel	Chimie Organique
VIVIAN Robert	Géographie
VOTTERO Philippe	Chimie

MEMBRES DU CORPS ENSEIGNANT DE L' IUT 1

PROFESSEURS de 1ère Classe

BUISSON Roger	Physique IUT 1
DODU Jacques	Mécanique Appliquée IUT 1
NEGRE Robert	Génie Civil IUT 1
NOUGARET Marcel	Automatique IUT 1
PERARD Jacques	EEA. IUT 1

PROFESSEURS de 2ème classe

BOUTHINON Michel	EEA. IUT 1
CHAMBON René	Génie Mécanique IUT 1
CHEHIKIAN Alain	EEA. IUT 1
CHENAVAS Jean	Physique IUT 1
CHOUTEAU Gérard	Physique IUT 1
CONTE René	Physique IUT 1
GOSSE Jean-Pierre	EEA.IUT 1
GROS Yves	Physique IUT 1
KUHN Gérard, (Détaché)	Physique IUT 1
MAZUER Jean	Physique IUT 1
MICHOULIER Jean	Physique IUT 1
MONLLOR Christian	EEA.IUT 1
PEFFEN René	Métallurgie IUT 1
PERRAUD Robert	Chimie IUT 1
PIERRE Gérard	Chimie IUT 1
TERRIEZ Jean-Michel	Génie Mécanique IUT 1
TOUZAIN Philippe	Chimie IUT 1
VINCENDON Marc	Chimie IUT 1

PROFESSEURS DE PHARMACIE

AGNIUS-DELORD Claudine	Physique	Faculté La Tronche
ALARY Josette	Chimie Analytique	Faculté La Tronche
BERIEL Hélène	Physiologie et Pharmacologie	Faculté La Tronche
CUSSAC Max	Chimie Therapeutique	Faculté La Tronche
DEMENGE Pierre	Pharmacodynamie	Faculté La Tronche
FAVIER Alain	Biochimie	C.H.R.G.
JEANNIN Charles	Pharmacie Galénique	Faculté Meylan
LATURAZE Jean	Biochimie	Faculté La Tronche
LUU DUC Cuong	Chimie Générale	Faculté La Tronche
MARIOTTE Anne-Marie	Pharmacognosie	Faculté La Tronche
MARZIN Daniel	Toxicologie	Faculté Meylan
RENAUDET Jacqueline	Bactériologie	Faculté La Tronche
ROCHAT Jacques	Hygiéne et Hydrologie	Faculté La Tronche
SEIGLE-MURANDI Françoise	Botanique et Cryptogamie	Faculté Meylan
VERAIN Alice	Pharmacie Galénique	Faculté Meylan

MEMBRES DU CORPS ENSEIGNANT DE MEDECINE

PROFESSEURS CLASSE EXEPTIONNELLE ET 1ère CLASSE

AMBLARD Pierre	Dermatologie	C.H.R.G.
AMBROISE-THOMAS Pierre	Parasitologie	C.H.R.G.
BEAUDOING André	Pédiatrie-Puericulture	C.H.R.G.
BEZEZ Henri	Orthopédie-Traumatologie	Hopital SUD
BONNET Jean-Louis	Ophtalmologie	C.H.R.G.
BOUCHET Yves	Anatomie	Faculté La Merci
	Chirurgie Générale et Digestive	C.H.R.G.
BUTEL Jean	Orthopédie-Traumatologie	C.H.R.G.
CHAMBAZ Edmond	Biochimie	C.H.R.G.
CHAMPETIER Jean	Anatomie-Topographique et Appliquée	C.H.R.G.
CHARACHON Robert	O.R.L.	C.H.R.G.
COLOMB Maurice	Immunologie	Hopital sud
COUDERC Pierre	Anatomie-Pathologique	C.H.R.G.
DELORMAS Pierre	Pneumophtisiologie	C.H.R.G.
DENIS Bernard	Cardiologie	C.H.R.G.
GAVEND Michel	Pharmacologie	Faculté La Merci
HOLLARD Daniel	Hématologie	C.H.R.G.
LATREILLE René	Chirurgie Thoracique et Cardiovasculaire	C.H.R.G.
LE NOC Pierre	Bactériologie-Virologie	C.H.R.G.
MALINAS Yves	Gynécologie et Obstétrique	C.H.R.G.
MALLION Jean-Michel	Médecine du Travail	C.H.R.G.
MICOUD Max	Clinique Médicale et Maladies Infectieuses	C.H.R.G.
MOURIQUAND Claude	Histologie	Faculté La Merci
PARAMELLE Bernard	Pneumologie	C.H.R.G.
PERRET Jean	Neurologie	C.H.R.G.
RACHAIL Michel	Hépato-Gastro-Entérologie	C.H.R.G.
DE ROUGEMONT Jacques	Neurochirurgie	C.H.R.G.
SARRAZIN Roger	Clinique Chirurgicale	C.H.R.G.
STIEGLITZ Paul	Anestésiologie	C.H.R.G.
TANCHE Maurice	Physiologie	Faculté La Merci
VIGNAIS Pierre	Biochimie	Faculté La Merci

PROFESSEURS 2ème CLASSE

BACHELOT Yvan	Endocrinologie	C.H.R.G.
BARGE Michel	Neurochirurgie	C.H.R.G.
BENABID Alim Louis	Biophysique	Faculté La Merci
BENSA Jean-Claude	Immunologie	Hopital Sud
BERNARD Pierre	Gynécologie-Obstétrique	C.H.R.G.
BESSARD Germain	Pharmacologie	ABIDJAN
BOLLA Michel	Radiothérapie	C.H.R.G.
BOST Michel	Pédiatrie	C.H.R.G.
BOUCHARLAT Jacques	Psychiatrie Adultes	Hopital Sud
BRAMBILLA Christian	Pneumologie	C.H.R.G.
CHIROSSEL Jean-Paul	Anatomie-Neurochirurgie	C.H.R.G.
COMET Michel	Biophysique	Faculté La Merci
CONTAMIN Charles	Chirurgie Thoracique et Cardiovasculaire	C.H.R.G.
CORDONNIER Daniel	Néphrologie	C.H.R.G.
COULOMB Max	Radiologie	C.H.R.G.
CROUZET Guy	Radiologie	C.H.R.G.
DEBRU Jean-Luc	Médecine Interne et Toxicologie	C.H.R.G.
DEMONGEOT Jacques	Biostatistiques et Informatique Médicale	Faculté La Merci
DUPRE Alain	Chirurgie Générale	C.H.R.G.
DYON Jean-François	Chirurgie Infantile	C.H.R.G.
ETERRADOSSI Jacqueline	Physiologie	Faculté La Merci
FAURE Claude	Anatomie et Organogénèse	C.H.R.G.
FAURE Gilbert	Urologie	C.H.R.G.
FOURNET Jacques	Hépato-Gastro-Entérologie	C.H.R.G.
FRANCO Alain	Médecine Interne	C.H.R.G.
GIRARDET Pierre	Anesthésiologie	C.H.R.G.
GUIDICELLI Henri	Chirurgie Générale et Vasculaire	C.H.R.G.
GUIGNIER Michel	Thérapeutique et Réanimation Médicale	C.H.R.G.
HADJIAN Arthur	Biochimie	Faculté La Merci
HALIMI Serge	Endocrinologie et Maladies Métaboliques	C.H.R.G.
HOSTEIN Jean	Hépato-Gastro-Entérologie	C.H.R.G.
HUGONOT Robert	Médecine Interne	C.H.R.G.
JALBERT Pierre	Histologie-Cytogénétique	C.H.R.G.
JUNIEN-LAVILLAUROY Claude	O.R.L.	C.H.R.G.
KOLODIE Lucien	Hématologie Biologique	C.H.R.G.
LETOUBLON Christian	Chirurgie Générale	C.H.R.G.
MACHECOURT Jacques	Cardiologie et Maladies Vasculaires	C.H.R.G.
MAGNIN Robert	Hygiène	C.H.R.G.
MASSOT Christian	Médecine Interne	C.H.R.G.
MOUILLON Michel	Ophtalmologie	C.H.R.G.
PELLAT Jacques	Neurologie	C.H.R.G.
PHELIP Xavier	Rhumatologie	C.H.R.G.
RACINET Claude	Gynécologie-Obstétrique	Hopital Sud
RAMBAUD Pierre	Pédiatrie	C.H.R.G.
RAPHAEL Bernard	Stomatologie	C.H.R.G.
SCHAERER René	Cancérologie	C.H.R.G.
SEIGNEURIN Jean-Marie	Bactériologie-Virologie	Faculté La Merci
SELE Bernard	Cytogénétique	Faculté La Merci
SOTTO Jean-Jacques	Hématologie	C.H.R.G.
STOEBNER Pierre	Anatomie Pathologique	C.H.R.G.
VROUSOS Constantin	Radiothérapie	C.H.R.G.

Je tiens à remercier,

Monsieur Sacha Krakowiak, Professeur à l'Université Joseph Fourier (Grenoble 1), de m'avoir fait l'honneur de présider le jury de cette thèse,

Monsieur André Arnold, Professeur à l'Université de Bordeaux, pour avoir accepté de participer au jury,

Monsieur Gérard Boudol, Directeur de Recherches à l'INRIA, qui a accepté de juger ce travail. L'accueil qu'il m'a réservé constitue un encouragement à poursuivre ce travail,

Monsieur Joseph Sifakis, Directeur de Recherches au CNRS, qui m'a accueilli au sein du groupe Spécification et Analyse des Systèmes où j'ai bénéficié d'un excellent environnement de travail,

Monsieur Laurent Trilling, Professeur à l'Université Joseph Fourier, qui a patiemment relu le manuscrit et qui a bien voulu juger ce travail,

Monsieur Jacques Voiron, Maître de Conférences à l'Université Joseph Fourier, qui est le Directeur de cette thèse. Ses nombreux conseils et suggestions ont contribué à améliorer ce travail.

Je tiens également à remercier,

Pierre Berlioux, Fabienne Lagnier, Suzanne Graf, et Florence Maraninchi, qui ont relu et commenté certaines parties de la thèse,

les utilisateurs d'Aldébaran, notamment les membres du projet Lustre et H. Garavel qui ont contribué à l'amélioration du logiciel,

les membres du service reprographie pour le soin et l'efficacité qu'ils ont apporté au tirage de cette thèse.

Je remercie enfin Nicole, Juliette, Maud et Rémy qui ont supporté, avec patience, un compagnon et un père souvent distrait et parfois irascible.

ALDEBARAN : UN SYSTEME DE VERIFICATION PAR REDUCTION DE PROCESSUS COMMUNICANTS

RESUME

ALDEBARAN est un système de vérification de processus communicants, représentés par des systèmes de transitions étiquetées. Il permet de réduire et de comparer des systèmes de transitions étiquetées en tenant compte d'une relation d'équivalence. Les relations d'équivalence que nous avons considérées sont la congruence forte, l'équivalence et la congruence observationnelles, et la congruence par modèle d'acceptation. Nous présentons dans ce travail les bases théoriques d'ALDEBARAN, des algorithmes efficaces pour la comparaison et les réductions de systèmes de transitions étiquetées, une réalisation en langage C et deux exemples d'utilisation d'ALDEBARAN par d'autres logiciels.

MOTS CLES :
Parallélisme, Vérification de processus communicants, Systèmes de transitions étiquetées, relation d'équivalence, Efficacité.

ALDEBARAN : A SYSTEM OF VERIFICATION OF COMMUNICATING PROCESSES BY USING REDUCTION

ABSTRACT

ALDEBARAN is a system for verifying communicating processes, represented by transition systems whose arcs are labelled by action names. It allows the reduction and the comparison of labelled transition systems with respect to some equivalence relation. The following equivalences have been dealt with : strong congruence, both observational equivalence and congruence, and the congruence based on acceptance models. The first two chapters of this thesis present the theorical basis of ALDEBARAN and the efficient algorithms which have been designed for comparing and reducing labelled transition systems. An implementation of these algorithms in C and various examples of uses of ALDEBARAN follow.

INTRODUCTION

La complexité des systèmes répartis a rendu nécessaire l'utilisation de logiciels d'aide à la conception et à la vérification de ces systèmes. Par système réparti, nous entendons ensemble de composants communiquant par l'intermédiaire de ports de communication. Un composant est soit un système réparti, soit un processus séquentiel. Vérifier un système c'est montrer qu'une implantation *IMP* de ce système est conforme à ses spécifications *SPEC* .

D'un point de vue formel, on distingue généralement deux approches pour effectuer cette vérification. La première consiste à montrer que l'implantation *IMP* est un modèle de la spécification *SPEC*, qui est un ensemble de formules d'une logique. Chaque formule exprime une propriété du programme. La logique peut être soit le calcul des prédicats, soit une logique temporelle linéaire [Pnu77], soit une logique temporelle arborescente [GS86b]. Dans la deuxième approche la spécification et l'implantation sont des termes d'un calcul. Un calcul est la donnée d'un langage de termes et d'une relation d'équivalence ; *IMP* et *SPEC* sont des termes du calcul. La vérification consiste à comparer les termes *IMP* et *SPEC* modulo cette relation d'équivalence. Nous nous intéressons dans ce travail à cette deuxième approche.

La vérification peut être déductive, c'est l'approche *système à règles d'inférence* ou sémantique. Dans ce cas, la relation d'équivalence est interprétée sémantiquement, c'est-à-dire qu'elle est définie sur un domaine sémantique. Nous avons étudié, dans ce travail, la vérification sémantique pour des raisons d'efficacité.

Nous nous intéressons à la sémantique opérationnelle d'un calcul : à chaque terme est associé un système de transitions étiquetées. Un système de transitions étiquetées est défini par un ensemble d'états, une relation de transitions étiquetées et un état initial. Une étiquette désigne un port de communication par lequel un composant du système peut communiquer avec son environnement ou bien une action interne à un processus.

Pour interpréter une relation d'équivalence définie sur un calcul, il faut

pouvoir comparer deux systèmes de transitions étiquetées. Cette comparaison peut être effectuée par le calcul de la plus grande bisimulation entre deux systèmes de transitions étiquetées. Intuitivement, deux systèmes de transitions étiquetées "se bisimulent" si, à partir des états initiaux respectifs des deux systèmes, ils évoluent vers des états équivalents via la relation de transitions étiquetées. La plus grande bisimulation est une relation d'équivalence sur la classe des systèmes de transitions étiquetées. On peut efficacement (cf. chapitre II) associer à tout système de transitions étiquetées, un système de transitions étiquetées équivalent, modulo la plus grande bisimulation, ayant un nombre minimal d'états.

Différentes algèbres de processus communicants sont définies ACP [BK85], CCS [Mil80], Meije [Bou85a], SCCS [Mil83], TCSP [HBR]. Pour chacune de ces algèbres, une sémantique opérationnelle a été définie.

Différentes relations d'équivalence peuvent être définies sur ces algèbres :

- la congruence forte [Mil80],
- l'équivalence de test [EB87],
- l'équivalence observationnelle [Mil80],
- l'équivalence par modèle d'acceptation, [GS86a],
- l'équivalence par modèle de refus [HBR] et
- l'équivalence de trace.

Ces relations, définies sur les termes d'une algèbre, peuvent être définies sur les systèmes de transitions étiquetées. Parmi ces relations, nous distinguons la congruence forte, dont l'interprétation sur les systèmes de transitions étiquetées est la plus grande bisimulation. Dans ce travail, nous nous intéressons aux relations d'équivalence qui ont les deux caractéristiques suivantes : étant donné une relation d'équivalence,

- pour chaque classe d'équivalence, il existe un représentant canonique que nous appelons, par abus de langage, forme normale. Pour décider si deux systèmes sont équivalents il suffit de montrer que les formes normales associées sont égales au nom des états près.
- elle est moins fine que la plus grande bisimulation. Ceci permet de choisir une forme normale ayant un nombre minimal d'états. Cette forme normale est obtenue en construisant le système quotient après le calcul de la plus grande bisimulation.

Plusieurs outils de vérification ont été développés pour manipuler les systèmes de transitions étiquetées associés aux termes d'un calcul [LMV87], [Sor87], [Hil87]. Ces logiciels permettent la réduction d'un système de transitions étiquetées modulo une relation d'équivalence ou la comparaison de deux systèmes de transitions étiquetées modulo une relation d'équivalence. Ces logiciels ont montré l'intérêt de l'approche adoptée dans ce travail pour vérifier les systèmes répartis.

Nous proposons, dans cette thèse, la conception et la réalisation d'un outil pour la vérification des systèmes de transitions étiquetées, ALDEBARAN, répondant aux contraintes suivantes :

- *efficacité* : nous mettons en œuvre des algorithmes performants pour effectuer les différentes transformations sur les systèmes de transitions étiquetées : le calcul d'une forme normale est de l'ordre de la minute pour un système de transitions étiquetées de quelques milliers d'états. Cette implantation est réalisée en langage C pour permettre de choisir au mieux les structures de données adaptées à ces algorithmes et de contourner les problèmes de temps d'exécution et de place mémoire liés à des environnements de programmation comme Prolog ou Lisp.

- *paramétrisation* : l'opérateur de composition parallèle que nous proposons est paramétré par une relation de communication. Cette relation définit une opération de synchronisation sur l'ensemble des étiquettes.

- *modularité* : le logiciel ALDEBARAN est conçu pour permettre de prendre en compte de nouvelles relations d'équivalence, en définissant pour chaque relation un nouveau module contenant les transformations réalisant le calcul de la forme normale.

- *facilité d'intégration* : nous avons expérimenté l'utilisation d'ALDEBARAN par un autre logiciel de deux manières différentes. Nous avons intégré une partie d'ALDEBARAN au système Vénus écrit en Prolog. Un interface a été écrit pour minimiser les automates produits par le compilateur Lustre ou par Cæsar. Lustre, [CPH87], est un langage "flot de données" de programmation synchrone. Cæsar [Gar87] produit des systèmes de transitions étiquetées à partir de programmes écrits en LOTOS.

Dans le premier chapitre de cette thèse, nous définissons la syntaxe et la sémantique opérationnelle d'un langage de *processus réguliers*. La sémantique opérationnelle d'un processus régulier est un système de transitions étiquetées ayant un nombre fini d'états et de transitions.

Nous rappelons ensuite la définition des relations d'équivalence auxquelles nous nous sommes intéressés. Ce sont la congruence forte, l'équivalence et la congruence observationnelles définies sur l'algèbre CCS et la congruence par modèle d'acceptation définie sur une algèbre de processus [GS86a]. Ces relations d'équivalence sont définies sur le langage des processus réguliers. Milner [Mil85] a donné une axiomatisation *consistante* et *complète* de la congruence forte, de l'équivalence et de la congruence observationnelles. S. Graf et J. Sifakis [GS86a] ont procédé de même pour la congruence par modèle d'acceptation. Nous rappelons, pour chacune de ces relations, sa définition sur le langage des processus réguliers, et la transformation qui calcule, pour chaque système de transitions étiquetées, sa forme normale.

Nous définissons sur les systèmes de transitions étiquetées un opérateur *d'abstraction* [BK85] et les opérateurs de *composition parallèle* et de *restriction*. L'opérateur d'abstraction, défini sur l'ensemble des étiquettes, permet de ne pas distinguer des étiquettes considérées comme internes. Cela se traduit par un renommage de ces étiquettes en une étiquette unique notée τ.
L'opérateur de composition parallèle est paramétré par une relation de communication. Nous précisons les propriétés de cette relation pour exprimer les opérateurs de composition *synchrone* et *asynchrone*, le *rendez-vous à deux* ou le *rendez-vous multiple*.
L'opérateur de restriction est défini dans CCS pour permettre de supprimer certains comportements après composition.
La congruence forte et la congruence observationnelle sont des congruences pour les opérateurs de restriction, d'abstraction et de composition. La congruence par modèle d'acceptation n'est pas une congruence pour l'opérateur de composition parallèle.

Le deuxième chapitre est une étude des algorithmes mis en œuvre dans ALDEBARAN. Nous nous sommes particulièrement intéressés à un algorithme de partitionnement d'un ensemble d'états [PT86], permettant une mise en œuvre

du calcul de la plus grande bisimulation en un temps de l'ordre de $m \log n$, où m est le nombre de transitions et n le nombre d'états. Le calcul de la plus grande bisimulation est essentiel dans toutes les réductions effectuées par rapport à une relation d'équivalence : la comparaison des systèmes de transitions étiquetées, modulo une relation d'équivalence, est réalisée en "bisimulant" les formes normales de ces systèmes de transitions étiquetées. Les formes normales sont elles-même obtenues en minimisant le nombre des états par "autobisimulation". Nous présentons aussi la mise en œuvre du calcul des formes normales pour les différentes relations d'équivalence définies au chapitre I.

Un autre aspect important est la réduction de systèmes de transitions étiquetées composés. Lorsque plusieurs systèmes sont composés, nous proposons une méthode permettant de réduire progressivement le système résultat : étant donné un ensemble de systèmes à composer décrit par un terme de l'algèbre, et une relation de congruence, nous réarrangeons l'ordre des compositions lorsque c'est possible.

Le troisième chapitre est consacré à la description de l'outil ALDEBARAN, qui permet :

- la description de systèmes de transitions étiquetées et leur sauvegarde ;
- la réduction d'un système de transitions étiquetées modulo une relation d'équivalence, c'est-à-dire le calcul d'un forme normale ;
- la comparaison de deux systèmes de transitions étiquetées modulo une relation d'équivalence ;
- la composition de systèmes en tenant compte d'une relation de congruence ;
- la gestion d'environnements de travail.

Nous présentons à titre d'exemple l'utilisation de l'outil ALDEBARAN par le compilateur LUSTRE et par CÆSAR. Cette présentation est chaque fois suivie d'exemples.

Nous terminons ce chapitre en donnant quelques caractéristiques du logiciel réalisé.

CHAPITRE I

SYSTEMES DE TRANSITIONS ETIQUETEES

Dans ce chapitre, nous nous intéressons aux relations d'équivalence pouvant être définies sur les systèmes de transitions étiquetées et à la manière dont on peut composer ces systèmes.

Pour chaque relation d'équivalence étudiée dans ce chapitre, nous présentons sa définition syntaxique (i.e. sur un langage de processus réguliers) et sémantique (i.e. sur les systèmes de transitions étiquetées).

Un système de transitions étiquetées peut être défini de deux manières :
- par la description de sa relation de transition, et son état initial ; dans ces conditions, l'ensemble des états est constitué par tous les états atteignables à partir de l'état intial par la relation de transition.
- par composition de systèmes de transitions déjà construits.

Dans la premiere partie de ce chapitre, nous rappelons la syntaxe et la sémantique opérationnelle d'un langage de processus réguliers. Les opérateurs de ce langage sont **nil** (inaction), + (choix non-déterministe), préfixe par une action et **rec** (définition récursive). Les relations d'équivalence que nous avons étudiées sont définies sur ce langage.

La comparaison de deux systèmes de transitions étiquetées par une bisimulation a été développée par Park [Par81]. Elle est définie entre systèmes de transtions étiquetées sur le produit cartésien des ensembles d'états. L'ensemble des bisimulations entre systèmes de transitions étiquetées, ordonné par inclusion, possède un plus grand élément [Mil83] : la plus grande bisimulation est obtenue par un calcul de plus grand point fixe d'un opérateur monotone.
Lorsque les deux systèmes sont identiques, le calcul de la plus grande relation de bisimulation produit la partition sur les états, la moins fine compatible avec la relation de transitions étiquetées. Nous pouvons construire, à partir du système de transitions étiquetées et des classes d'équivalence, un système de transitions étiquetées quotient.

Le calcul de la plus grande relation de bisimulation peut être utilisé pour décider si deux systèmes de transitions étiquetées sont égaux aux noms des états près. Si deux systèmes ne se bisimulent pas par la plus grande bisimulation, ils ne se bisimulent par aucune autre.

Chaque relation d'équivalence étudiée dans ce travail est moins fine que la plus grande bisimulation et est caractérisée par la propriété suivante : pour chaque classe d'équivalence, il existe un représentant canonique appelé forme normale. Pour une relation d'équivalence, deux systèmes de transitions étiquetées sont équivalents si les formes normales associées sont identiques au nom des états près. Puisque ces relations d'équivalence contiennent la plus grande bisimulation, on peut définir une forme normale ayant un nombre minimal d'états. Pour comparer deux systèmes de transitions étiquetées modulo une relation d'équivalence nous appliquons l'algorithme suivant :
- calcul de la forme normale, modulo cette relation d'équivalence, de chaque système de transitions étiquetées,
- comparaison de ces formes normales. Ces formes normales sont identiques au renommage des états près.
De manière générale, pour toute relation d'équivalence moins fine que la congruence forte et pour laquelle il existe une forme normale nous pouvons réduire un système de transitions étiquetées modulo cette relation ou comparer deux systèmes de transitions étiquetées en appliquant l'algorithme précédent. Bergstra, Klop & Olderog, [BKO] ont proposé une congruence "par ensemble de refus", moins fine que la congruence forte, pour laquelle ils ont prouvé l'existence d'une forme normale.

Les relations d'équivalence étudiées dans ce chapitre sont :
- la *congruence forte*
- l'*équivalence observationnelle*,
- la *congruence observationnelle*,
- la *congruence par modèle d'acceptation* , telle qu'elle est définie par S.Graf et J. Sifakis.

La distinction entre action interne et port de communication est prise en compte par l'équivalence observationnelle, la congruence observationnelle et la congruence par modèle d'acceptation. Les actions internes ne sont pas

distinguées vis à vis de ces relations : elles sont étiquetées par une étiquette particulière τ.

Nous définissons un *opérateur d'abstraction* [BK85] sur les systèmes de transitions étiquetées. Il est paramétré par un ensemble d'étiquettes E et définit une partition sur l'ensemble de toutes les étiquettes. Son effet sur les systèmes de transitions étiquetées est le renommage des étiquettes de l'ensemble E par l'étiquette τ. La partition de l'ensemble de toutes les étiquettes, définie par cet opérateur, est un cas particulier des *critères d'abstraction* définis par G. Boudol [Bou85b] : un critère d'abstraction sur un ensemble d'étiquettes A est une partition partielle de A^*.

Dans la seconde partie de ce chapitre nous définissons un opérateur de composition parallèle et un opérateur de restriction. Ce qui différencie les opérateurs de composition parallèle des algèbres comme ACP, CCS, Meije, SCCS ou TCSP, ce sont les actions qui restent visibles après communication et le mode de composition *synchrone* ou *asynchrone*. De manière analogue à Winskel, [Win83], nous paramétrons un produit de systèmes de transitions étiquetées par une *algèbre de synchronisation*. Une algèbre de synchronisation est obtenue en structurant l'ensemble des étiquettes à l'aide d'un produit de synchronisation. Ce produit définit un mode de communication, *synchrone* ou *asynchrone*, ainsi que l'étiquette résultant de la communication. Il permet d'exprimer le *rendez-vous à deux* ou *multiple*. L'opérateur de restriction a été introduit dans CCS pour supprimer certains comportements après communication.

1. LANGAGE DES PROCESSUS REGULIERS

Dans ce paragraphe, nous définissons la syntaxe et la sémantique opérationnelle du langage des processus réguliers, qui nous permet de construire les systèmes de transitions étiquetées. Nous définissons la notion de bisimulation sur les systèmes de transitions étiquetées, qui est la base des relations d'équivalence définies dans ce chapitre. Nous introduisons l'opérateur d'abstraction.

1.1. Syntaxe et sémantique opérationnelle du langage des processus réguliers

A Notations et syntaxe

Soient,

P, un ensemble d'états, dont les éléments sont notés p, q, ...

Act, un ensemble d'étiquettes, dont les éléments sont notés a, b, c, ...

$\tau \notin Act$, une étiquette particulière représentant une action interne ou non observable,

$Act_\tau = Act \cup \{\tau\}$,

V, un ensemble de variables, dont les éléments sont notés X, Y, Z, ...

un ensemble d'opérateurs **nil**, +, a, **rec** X. où,

nil est un constante,

+ est un opérateur binaire,

$a \in Act_\tau$ est un opérateur unaire,

rec X, $X \in V$, est l'opérateur de définition récursive.

La syntaxe des processus réguliers est définie de la manière suivante :

$$t ::= \mathbf{nil} \mid X \mid a\,t \mid t + t \mid \mathbf{rec}\,X\,.\,t$$

Le langage des processus réguliers, noté **Trans**, est caractérisée par les contraintes syntaxiques suivantes :

un terme t appartient à **Trans** si et seulement si

(i) t est *fermé* (i.e. t ne contient pas de variable libre),

(ii) si t est de la forme **rec** X.t', t est *bien gardé* (i.e. toute occurrence de X dans t' apparaît dans un sous-terme de la forme a t, pour $a \in Act_\tau$).

B Systèmes de transitions étiquetées

Définition 1.1.1

Un *système de transitions étiquetées* est défini par un quadruplet $S = (Q, A, T, q_0)$ où :

- Q, sous-ensemble de P, désigne l'ensemble des états atteignables à partir de q_0.

- A est un sous-ensemble de Act_τ,
- T est la relation de transitions étiquetées par des éléments de A, sous-ensemble de $Q \times A \times Q$,
- q_0 est l'état initial du système.

Définition 1.1.2

Soit $S = (Q, A, T, q_0)$ un système de transitions étiquetées ; l'ensemble des états atteignables à partir de l'état initial peut être défini par induction de la manière suivante : c'est le plus petit sous-ensemble de P satisfaisant les conditions (i) et (ii) ci-dessous,

(i) q_0 est atteignable,

(ii) si q est atteignable et $(q, a, q') \in T$, alors q' est atteignable

Remarque

La notation **Trans** pour le langage des processus réguliers est justifiée par le fait qu'on peut associer à chaque processus régulier un système de transitions étiquetées ayant un nombre fini d'états. Réciproquement, tout système de transitions étiquetées ayant un nombre fini d'états peut être décrit par un terme de **Trans**.

Notations

Il est souvent commode de considérer l'image T [q] d'un état q par la relation de transition T, l'ensemble T_a [q] des successeurs de l'état q par la relation de transition étiquetée par a. Ces notations sont étendues à la relation inverse, notée T^{-1}. Formellement, nous avons :

$T [q] = \{(a, q') \mid (q, a, q') \in T\}$
$T^{-1} [q'] = \{(q, a) \mid (q, a, q') \in T\}$
$T_a [q] = \{ q' \mid (q, a, q') \in T\}$
$T_a^{-1} [q'] = \{ q \mid (q, a, q') \in T\}$

Ces notations sont étendues aux ensembles d'états; par exemple, si B désigne un ensemble d'états (i.e. B sous-ensemble de P),

$T [B] = \cup \{ T [p] \mid p \in B\}$.

Définition 1.1.3

Deux systèmes de transitions étiquetées $S_i = (Q_i, A_i, T_i, q_i)$, i = 1, 2 sont *égaux* si et seulement si

- il existe une bijection f de Q_1 dans Q_2 ,
- $A_1 = A_2$,
- $T_2 = \{(f(p_1), a, f(p'_1) \mid (p_1, a, p'_1) \in T_1\}$
- $f(q_1) = f(q_2)$.

C Règles de sémantique opérationnelle

La sémantique opérationnelle de **Trans** est la plus petite relation, notée -->, incluse dans **Trans** $\times$ Act_τ $\times$ **Trans** et satisfaisant les règles suivantes. Nous notons :

t - a -> t' pour (t, a, t') $\in$ -->

Pour définir la sémantique des opérateurs de **Trans**, nous utilisons les notations proposées par Plotkin [Plo81] :
$\forall$ a $\in$ Act_τ $\forall$ t, t' $\in$ **Trans**,

1) Préfixe

$$a\ t\ \text{-a->}\ t'$$

2) somme

$$\frac{t_1\ \text{-a->}\ t'}{t_1 + t_2\ \text{-a->}\ t'} \qquad \frac{t_2\ \text{-a->}\ t'}{t_1 + t_2\ \text{-a->}\ t'}$$

3) Définition récursive

$$\frac{t\ [\text{rec X.t} / X]\ \text{-a->}\ t'}{\text{rec X. t}\ \text{-a->}\ t'}$$

Exemple

Considérons le terme défini par l'expression suivante :

t = **rec** X. (a X) + b **rec** X. (b X)

Nous construisons le système de transitions étiquetées associé. Pour cela, on associe à chaque sous-terme bien gardé et fermé, un entier qui le désigne :

à t est associé l'entier 0,

à **rec** X . a X est associé l'entier 1

à b **rec** X . b X est associé l'entier 2

à **rec** X. b X est associé l'entier 3

Nous représentons l'ensemble de tous les états par l'ensemble des entiers naturels. Nous obtenons successivement, en appliquant la définition opérationnelle des opérateurs,

le système de transitions étiquetées S_1 associé au terme **rec** X.aX,

$S_1 = (\{1\}, \{a\}, \{(1, a, 1)\}, 1)$

le système de transitions étiquetées S_2 associé au terme b **rec** X.bX,

$S_2 = (\{2,3\}, \{b\}, \{(2, b, 3),(3, b, 3)\}, 2)$

le système de transitions étiquetées S associé au terme t,

$S = (\{0, 1, 2, 3\}, \{a, b\},$

$\{(0,a,1), (0, b, 3), (1, a, 1),(3, b, 3)\}, 0)$

La figure 1 est la représentation graphique du système de transitions étiquetées obtenu par application des règles de la sémantique opérationnelle au terme t.

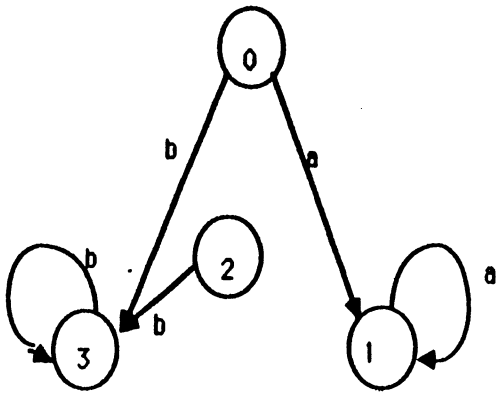


figure 1

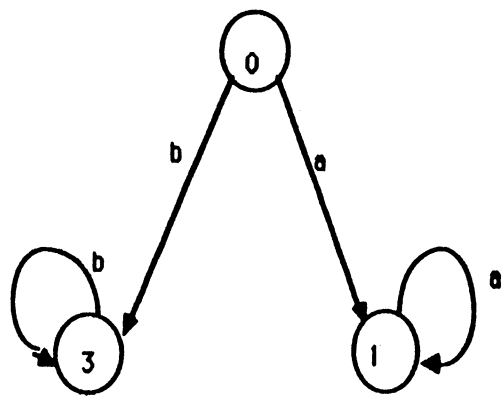


figure 2

Sur cet exemple, nous constatons que l'état 2 n'est pas atteignable à partir de l'état initial. Nous obtenons le système de transitions étiquetées (figure 2) après suppression de l'état 2.

Considérons le terme t' = **rec** X . (a X) + a **rec** X . (a X). Nous obtenons de la même manière le système de transitions étiquetées S' (figure 3).

Considérons le système de transitions étiquetées suivant (figure 4):

S" = ({0}, {a}, {(0, a, 0)}, 0).

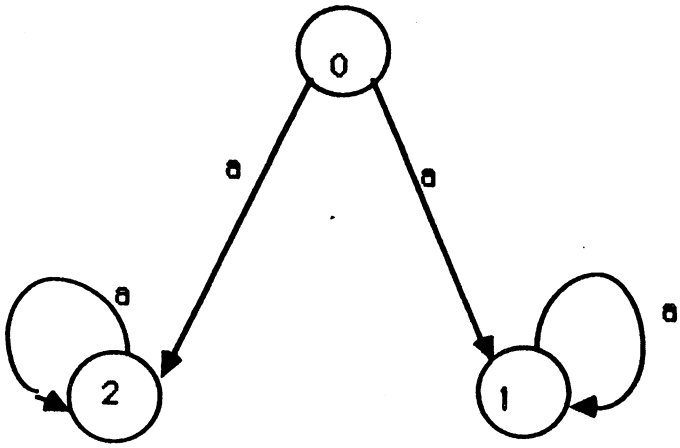


figure 3

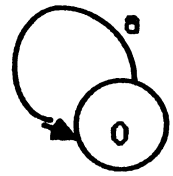


figure 4

Nous verrons au paragraphe 1.2., qu'il existe entre S" et S' une relation de bisimulation, et que S" est la forme normale de S', modulo la congruence forte.

1.2. Relations de bisimulation

Pour comparer deux systèmes de transitions étiquetées, nous définissons la notion de bismulation entre deux systèmes de transitions étiquetées. Deux états p et q se bisimulent si et seulement si, tout état p', successeur de p par une transition étiquetée par *a*, se bisimule avec un état q', successeur de q par une transition étiquetée par *a*, et inversement.

Définition

Soient deux systèmes de transitions étiquetées $S_i = (Q_i, A_i, T_i, q_i)$, $i = 1, 2$ et soit une relation binaire $\rho \subseteq Q_1 \times Q_2$.

ρ est une *bisimulation* si et seulement si

(i) $(q_1, q_2) \in \rho$,

(ii) $\forall (p_1, p_2) \in \rho$ et $\forall a \in Act_\tau \Rightarrow$

$(\forall p_1'.\ p_1' \in T_a[p_1] \Rightarrow \exists p_2'.\ p_2' \in T_a[p_2]$ et $(p_1', p_2') \in \rho)$

$(\forall p_2'.\ p_2' \in T_a[p_2] \Rightarrow \exists p_1'.\ p_1' \in T_a[p_1]$ et $(p_1', p_2') \in \rho)$

Remarque

On peut définir une fonctionnelle ψ sur les relations binaires sur l'ensemble P de tous les états :

$\Psi(\rho) = \{ (p_1, p_2) \mid \forall a \in Act_\tau,$

$(\forall p_1'.\ p_1' \in T_a[p_1] \Rightarrow \exists p_2'.\ p_2' \in T_a[p_2]$ et $(p_1', p_2') \in \rho)$

$(\forall p_2'.\ p_2' \in T_a[p_2] \Rightarrow \exists p_1'.\ p_1' \in T_a[p_1]$ et $(p_1', p_2') \in \rho)\}$

Soient deux systèmes de transitions étiquetées $S_i = (Q_i, A_i, T_i, q_i)$, $i = 1, 2$ et soit $\rho \subseteq Q_1 \times Q_2$. ρ est une bisimulation si et seulement si

(i) $(q_1, q_2) \in \rho$

(ii) $\rho \subseteq \Psi(\rho)$.

Exemple

1) Reprenons l'exemple figures 3 et 4 :

$S' = (\{0, 1, 2\}, \{a\}, \{(0,a,1),\ (0, a, 2),(1, a, 1), (2, a, 2)\}, 0)$

$S'' = (\{0\}, \{a\}, \{(0, a, 0)\}, 0)$.

La seule relation de bisimulation existant entre {0,1,2} et {0} est :

$\rho = \{(2, 0), (1, 0), (0, 0)\}$

2) Considérons l'exemple de la figure 3. Remarquons que la relation ρ_1 suivante vérifie la condition (ii) de la définition de la bisimulation. Pourtant ce n'est pas une bisimulation entre S' et lui-mêmecar le couple (0, 0) n'appartient pas à la relation.

$\rho_1 = \{(1, 2), (2, 1)\}$

Par contre, les relations suivantes sont des bisimulations.

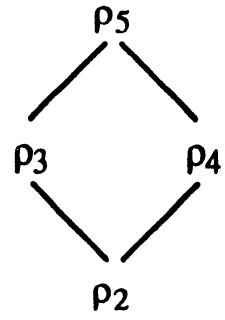
$\rho_2 = \{(0, 0), (1, 2), (2, 1)\}$,

$\rho_3 = \{(0, 0), (1, 1), (1, 2), (2, 1)\}$,

$\rho_4 = \{(0, 0), (2, 2), (1, 2), (2, 1)\}$,

$\rho_5 = \{(0, 0), (1, 1), (2, 2), (1, 2), (2, 1)\}$.

Ces relations de bisimulation sont ordonnées par l'inclusion ensembliste (figure suivante). Remarquons que ρ_2 n'est ni réflexive ni transitive.



Proposition 1.2.2

Soient deux systèmes de transitions étiquetées, $S_i = (Q_i, A_i, T_i, q_i)$, $i = 1, 2$.
L'opérateur Ψ est monotone croissant sur l'ensemble ordonné $(2^{(P \times P)}, \subseteq)$.
Il existe un plus grand point fixe de Ψ égal à :
$\cup \{\rho \mid \rho \subseteq \Psi(\rho)\}$

Preuve :

Notons $\rho_f = \cup \{\rho \mid \rho \subseteq \Psi(\rho)\}$.

Montrons que

(i) ρ_f est un point fixe (i.e. $\rho_f = \psi(\rho_f)$)

(ii) ρ_f est le plus grand point fixe ($\rho_g = \psi(\rho_g) \Rightarrow \rho_g \subseteq \rho_f$)

(i) $\rho_f \subseteq \psi(\rho_f)$

En effet, pout tout $\rho' \in \{\rho \mid \rho \subseteq \Psi(\rho)\}$, $\rho' \subseteq \cup \{\rho \mid \rho \subseteq \Psi(\rho)\} = \rho_f$.

D'autre part, $\psi(\rho') \subseteq \psi(\rho_f)$, puisque ψ est monotone croissant. On en déduit $\rho' \subseteq \psi(\rho_f)$. Cette dernière propriété étant vraie pout tout $\rho' \subseteq \psi(\rho')$, on déduit $\cup \{\rho \mid \rho \subseteq \Psi(\rho)\} \subseteq \psi(\rho_f)$, c'est-à-dire $\rho_f \subseteq \psi(\rho_f)$.

Inversement, $\psi(\rho_f) \subseteq \rho_f$. En effet, $\psi(\rho_f) \subseteq \psi(\psi(\rho_f))$, donc $\psi(\rho_f) \in \{\rho \mid \rho \subseteq \Psi(\rho)\}$ et $\psi(\rho_f) \subseteq \cup \{\rho \mid \rho \subseteq \Psi(\rho)\} = \rho_f$.

(ii) si $\rho_g = \psi(\rho_g)$ alors $\rho_g \in \{\rho \mid \rho \subseteq \Psi(\rho)\}$ et $\rho_g \subseteq \rho_f$.

□

Proposition 1.2.3

Si $Q_1 = Q_2$, et si ρ est réflexive (resp. symétrique, transitive) alors $\Psi(\rho)$ est réflexive (resp. symétrique, transitive).

Preuve :

soit ρ une relation binaire, sous-ensemble de $Q \times Q$;
si ρ est une relation réflexive ou symétrique, il en est de même pour $\psi(\rho)$
si ρ est transitive, soit $(p, q), (q, r) \in \psi(\rho)$; montrons que $(p, r) \in \psi(\rho)$.
-si $(p, q) \in \psi(\rho)$ alors $\forall a \in Act_\tau$,

$(\forall p'.\ p' \in T_a[p] \Rightarrow \exists q'.\ q' \in T_a[q]$ et $(p', q') \in \rho)$
$(\forall q'.\ q' \in T_a[q] \Rightarrow \exists p'.\ p' \in T_a[p]$ et $(p', q') \in \rho)$;

-si $(q, r) \in \psi(\rho)$ alors $\forall a \in Act_\tau$,

$(\forall q'.\ q' \in T_a[q] \Rightarrow \exists r'.\ r' \in T_a[r]$ et $(q', r') \in \rho)$
$(\forall r'.\ r' \in T_a[r] \Rightarrow \exists q'.\ q' \in T_a[q]$ et $(q', r') \in \rho)$;

on en déduit, par transitivité de ρ :
$\forall a \in Act_\tau$,

$(\forall p'.\ p' \in T_a[p] \Rightarrow \exists r'.\ r' \in T_a[r]$ et $(p', r') \in \rho)$
$(\forall r'.\ r' \in T_a[r] \Rightarrow \exists p'.\ p' \in T_a[p]$ et $(p', r') \in \rho)$;

Donc, $(p, r) \in \psi(\rho)$.

□

Soient deux systèmes de transitions étiquetées, $S_i = (Q_i, A_i, T_i, q_i)$, $i = 1, 2$. Nous nous intéressons au plus grand point fixe de l'opérateur Ψ, i.e. la plus grande bisimulation.
Lorsque $S_1 = S_2$, Ψ est définie sur le même ensemble d'états Q. Dans ce cas, Ψ détermine la plus grande relation d'équivalence sur Q, compatible avec la relation de transitions étiquetées considérée. Cela permet de construire un système de transitions étiquetées quotient ayant un nombre minimal d'états.

Définition d'une relation d'équivalence sur les systèmes de transitions étiquetées.

Deux systèmes de transitions étiquetées sont équivalents si et seulement si il existe une bisimulation entre ces deux systèmes.

Cette définition est correcte, la réflexivité et la symétrie sont facile à vérifier. La transitivité se déduit du fait que la composition de deux bisimulations est une bisimulation. Si deux systèmes sont équivalents, ils se bisimulent par la plus grande bisimulation. Nous notons $S_1 \sim S_2$ s'il existe une (plus grande)

bisimulation entre S_1 et S_2.

Définition du système de transitions étiquetées quotient

Soit $S = (Q, A, T, q_0)$ un système de transitions étiquetées, ρ une relation de bisimulation qui est une relation d'équivalence sur Q ; nous pouvons construire le système quotient de transitions étiquetées, noté S / ρ, de la manière suivante :

$S / \rho = (Q / \rho, A, T / \rho, [q_0])$ où :

- Q / ρ dénote l'ensemble des classes d'équivalence de la relation ρ :

$$Q / \rho = \{ B \mid B \subseteq Q \forall p, q \in B , (p, q) \in \rho \}$$

- $(B, a, B') \in T / \rho$ si et seulement si $T_a^{-1}[B'] \cap B \neq \emptyset$.

La construction la relation de transitions étiquetées quotient est possible du fait que la relation ρ est *compatible* avec la relation de transitions (cf. chap. II) :

$\forall a \in Act_\tau, \forall B, B' \in Q / \rho \Rightarrow (B' \subseteq T_a^{-1}[B]$ ou $B' \cap T_a^{-1}[B] = \emptyset)$.

1.3. Relations d'équivalence sur Trans

L'objet des paragraphes qui suivent est la définition, pour chaque relation d'équivalence, d'une transformation calculant la *forme normale* d'un système de transitions étiquetées.

R. Milner [Mil85] a défini un système formel pour la congruence forte, l'équivalence observationnelle et la congruence observationnelle. Le langage de ces systèmes formels est **Trans**, l'axiomatisation est donnée par un ensemble d'équations sur les termes de **Trans**. Pour chacune de ces relations il a montré la *consistance* et la *complétude* de l'axiomatisation pour une sémantique opérationnelle.

A. Soriano [Sor86] a défini, pour chacune des relations précédentes, un ensemble de transformations sur les systèmes de transitions étiquetées dont la composition calcule, pour chaque système de transitions étiquetées, sa forme normale.

S. Graf et J. Sifakis [GS86a] ont défini, pour la congruence par modèle

d'acceptation, un système formel dont le langage est **Trans**. Ils ont montré la consistance et la complétude de l'axiomatisation pour une sémantique opérationnelle. Ils ont défini une forme normale sur les systèmes de transitions étiquetées.

Chacune de ces relations d'équivalence définies sur **Trans**, peut être considérée comme une relation d'équivalence sur la classe des systèmes de transitions étiquetées. Cela provient du fait qu'il existe une bijection entre les systèmes de transitions étiquetées et les termes fermés et réguliers.

La forme normale d'un système de transitions étiquetées $S = (Q, A, T, q_0)$ modulo la congruence forte est obtenue en calculant la plus grande bisimulation sur Q : c'est le système quotient de transitions étiquetées associé à S. Du fait que la plus grande bisimulation sur Q est la relation d'équivalence sur Q, la moins fine compatible avec la relation de transitions étiquetées, le système quotient a le plus petit nombre d'états (i.e. de classes) possible.

L'équivalence observationnelle, la congruence observationnelle et la congruence par modèle d'acceptation sont moins fines que la congruence forte. Cela permet de trouver, pour chaque système de transitions étiquetées, une forme normale, par rapport à l'une de ces relations, ayant le moins d'états possible : étant donné une forme normale pour une des relations d'équivalence précédentes, le système quotient pour la congruence forte de cette forme normale est encore une forme normale.

Enfin, deux systèmes de transitions étiquetées sont équivalents modulo une de ces relations si et seulement si ils ont le même représentant canonique (i.e. la même forme normale) aux noms des états près.

La structure des paragraphes est la suivante : pour chaque relation d'équivalence, nous rappelons son axiomatisation sur **Trans** et la définition de la forme normale pour un système de transitions étiquetées.

1.4. Congruence forte

Définition axiomatique de la congruence forte

Les axiomes et les règles d'inférence sont ceux de la table 1 [Mil85].

TABLE 1:

Axiomes :

(+) :(A1) : $(t_1 + t_2) + t_3 = t_1 + (t_2 + t_3)$
(A2) : $t_1 + t_2 = t_2 + t_1$
(A3) : $t + t = t$
(A4) : $t + nil = t$

(μ) : (μ_1) : **rec** X. t = t [**rec** X.t / X]

Règle d'inférence :

$$\frac{t = t' [t / X]}{t = \textbf{rec } X.\ t'}$$

Forme normale sur les systèmes de transitions étiquetées pour la congruence forte

Milner [Mil85] a prouvé que le système de transitions étiquetées obtenu par application de l'algorithme suivant est une forme normale pour la congruence forte :

1) calcul de la plus grande bisimulation entre S et lui-même ; on obtient ainsi une partition de l'ensemble des états Q ;

2) construction du système de transitions étiquetées quotient S / ~.

Nous notons *ncf* (S) la forme normale d'un système de transitions étiquetées S pour la congruence forte.

Exemple

Les systèmes de transitions sont décrits par les figures 5, 6 et 7.

Le calcul de la plus grande bisimulation appliqué à S_1 (figure 5) produit les classes
{0, 1, 2}, {4, 5} et {6}

Le système quotient est S_2 (figure 6).

Le calcul de la plus grande bisimulation appliqué à S_3 (figure 7) produit les classes
{0, 1}, {2}, et {3}

Le système quotient est S_2

S_1 et S_3 sont fortement congrus ; S_2 est la forme normale de S_1 et de S_3.

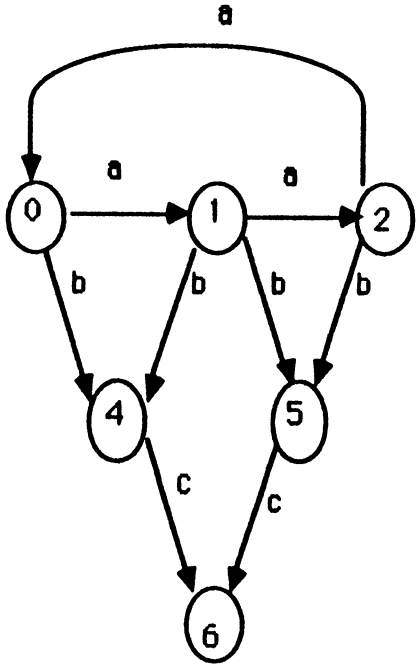


figure 5

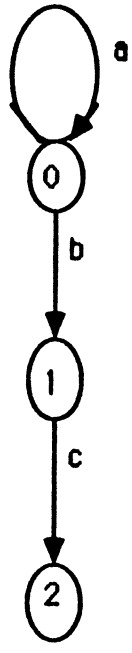


figure 6

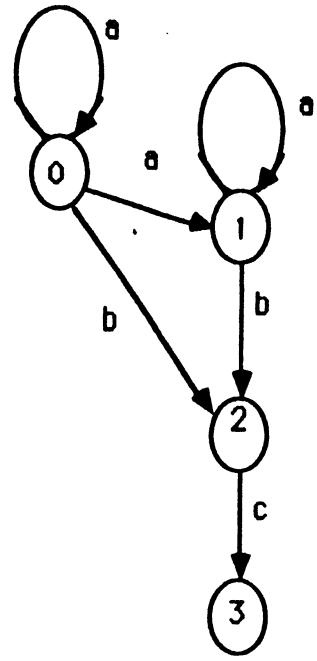


figure 7

1.5. Congruence observationnelle

Définition axiomatique de la congruence observationnelle

Les axiomes et les règles d'inférence sont ceux de la table 1 et de la table 2 [Mil85].

TABLE 2:

Axiomes :

(τ) :$(\tau 1)$: $\tau t + t = \tau t$

$(\tau 2)$: $a(\tau t_1 + t_2) = a t_1 + a(\tau t_1 + t_2)$

$(\tau 3)$: $a \tau t = a t$

(rec τ 1) : rec X. $(\tau X + t) =$ rec X. τt

(rec τ 2) : rec X. $(\tau (X + t) + t') =$ rec X. $(\tau X + t + t')$

Définitions des transformations sur les systèmes de transitions étiquetées

Les transformations qui suivent interviennent dans le calcul de la forme normale d'un système de transitions étiquetées pour la congruence observationnelle. Elles ont été définies par A. Soriano [Sor86].

a) Transformation *τ-chaîne* (élimination des chaînes d'actions τ)

La validité de la transformation τ-chaîne suivante peut être déduite de l'axiome $(\tau 3)$; elle distingue l'état initial des autres états.

$$(\tau\text{-chaîne}) \quad \frac{S = (Q, A, T, q_0),\ (q, \tau, q') \in T,\ q \neq q_0,\ |T[q]| = 1}{S' = (Q - \{q\}, A, T', q_0)}$$

où $T' = T - \{(q, \tau, q')\} - (T^{-1}[q] \times \{q\}) \cup (T^{-1}[q] \times \{q'\})$

$$(\tau\text{-chaîne}) \quad \frac{S = (Q, A, T, q_0),\ T[q_0] = \{(\tau, q)\}}{S' = (Q, A, T', q_0)}$$

où $T' = T \cup (T^{-1}[q_0] \times \{q\}) - T^{-1}[q_0] \times \{q_0\}$

Exemple

Considérons le système de transitions étiquetées S_1 (figure 8).

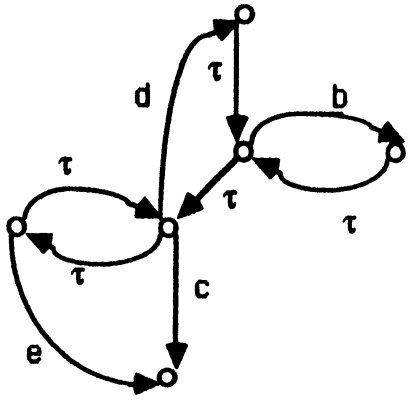


figure 8

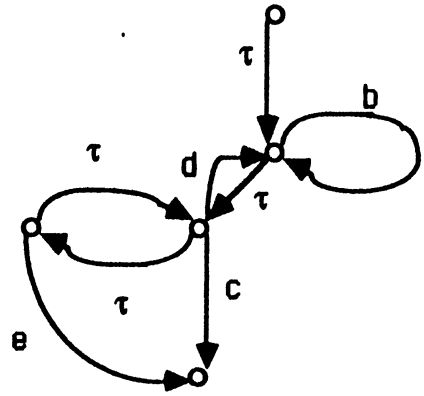


figure 9

S_2 (figure 9) est le transformé de S_1 par l'application τ-chaîne.

b) Transformation *τ-circuit* :

La validité de la transformation suivante (élimination des circuits d'actions τ) peut être déduite des axiomes (**rec** τ 1) (**rec** τ 2).

$$(\tau\text{-circuit}) \quad \frac{S = (Q, A, T, q_0),\ (q_i, \tau, q_{i+1}) \in T,\ i = 1,\ldots n,\ q_{n+1} = q_1}{S' = (Q - \{q_2, \ldots, q_n\}, A, T', q_0)}$$

où T' est définie de la manière suivante :

si $q_0 \notin \{q_1, \ldots, q_n\}$

$$\begin{aligned} T' = T &- \{\{q_i\} \times T[q_i] \mid i = 2, \ldots, n\} \\ &- \{T^{-1}[q_i] \times \{q_i\} \mid i = 2, \ldots, n\} \\ &\cup \{\{q_1\} \times T[q_i] \mid i = 1, \ldots, n\} \\ &\cup \{T^{-1}[q_i] \times \{q_1\} \mid i = 1, \ldots, n\} \end{aligned}$$

si $q_0 \in \{q_1, \ldots, q_n\}$, on considère q_0 à la place de q_1.

Exemple

Considérons le système de transitions étiquetées S_3, figure 10.

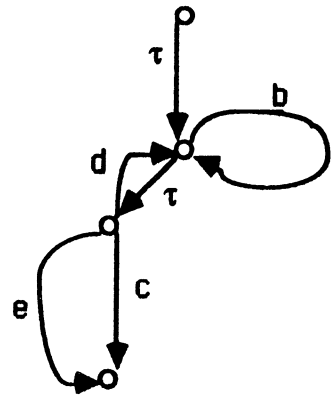


figure 10

S_3 est le transformé de S_2 par application de la transformation τ-circuit.

c) Transformation *τ-saturation* :

La validité de la transformation suivante (τ-saturation) peut être déduite des axiomes (τ1) et (τ 2) .

$$(\tau\text{-saturation}) \quad \frac{S = (Q, A, T, q_0),\ (q_i, \tau, q_{i+1}) \in T}{S' = (Q, A, T', q_0)}$$

où :

$$T' = T \cup \{\{q_i\} \times T[q_{i+1}]\} \cup \{T^{-1}[q_i] \times \{q_{i+1}\}\}$$

Exemple

Considérons le système de transitions étiquetées S_4 figure11. La notation

$$p \xrightarrow{\{a_i\}_{i \in I}} q$$

représente l'ensemble des transitions

$$\{p \xrightarrow{a_i} q\}_{i \in I}$$

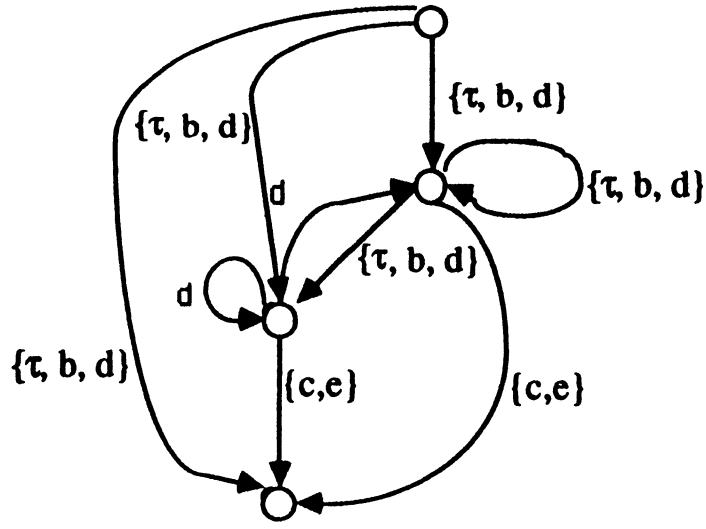


figure 11

S_4 est le transformé de S_3 par application de la transformation τ-saturation.

Forme normale sur les systèmes de transitions étiquetées pour la congruence observationnelle

A. Soriano [Sor86] a prouvé que le système de transitions étiquetées obtenu par application de l'algorithme suivant est une forme normale pour la congruence observationnelle :

1) application exhaustive des transformations (τ-chaine), (τ-circuit) et (τ-saturation) et fermeture réflexive de la relation de transitions étiquetées par τ.

2) calcul de la plus grande bisimulation ; on obtient ainsi une partition de l'ensemble des états.

3) construction du système de transitions étiquetées quotient et minimisation du nombre des transitions par application des transformations suivantes :

$$\frac{S = (Q, A, T, q_0),\ (q, \tau, q') \in T,\ (q, a, q'') \in T,\ (q', a, q'') \in T}{S' = (Q, A, T - \{(q, a, q'')\}, q_0)}$$

$$\frac{S = (Q, A, T, q_0), (q, a, q') \in T, (q', \tau, q'') \in T, (q, a, q'') \in T}{S' = (Q, A, T - \{(q, a, q'')\}, q_0)}$$

Commentaires : nous appelons *obs_réduction* la transformation composée de ces deux dernières transformations.

Nous notons *nco* (S) la forme normale d'un système de transitions étiquetées S. Nous notons $S_1 \sim_{co} S_2$ si les systèmes de transitions étiquetées S_1 et S_2 sont observationnellement congrus.

Exemple

La forme normale de S_4 est S_3.

1.6. Equivalence observationnelle

Définition axiomatique de l'équivalence observationnelle

Les axiomes et les règles d'inférence sont ceux de la table 1, de la table 2 et l'axiome suivant [Mil85] :

($\tau 4$) : $\tau t = t$

L'équivalence observationnelle n'est pas une congruence pour l'opérateur +. Pour s'en convaincre, il suffit de considérer l'exemple suivant :

Les termes τ a **nil** et a **nil**

sont observationnellement équivalents (cf. définition ci-dessous) mais

τ a **nil** + b **nil** et a **nil** + b **nil**

ne le sont pas.

Forme normale sur les systèmes de transitions étiquetées pour l'équivalence observationnelle

A. Soriano [Sor86] a prouvé que le système de transitions étiquetées obtenu par application de l'algorithme suivant est une forme normale pour l'équivalence observationnelle.

1) application exhaustive des transformations (τ-chaine), (τ-circuit) et (τ-saturation) et fermeture réflexive de la relation de transitions étiquetées par τ, où l'opération (τ-chaîne) est modifiée en supprimant la condition $q \neq q_0$.
2) application de la bisimulation en considérant pour tous les états la relation T_τ comme réflexive ; on obtient ainsi une partition de l'ensemble des états ;
3) construction du système de transitions étiquetées quotient et minimisation du nombre des transitions par application de la transformation *obs_réduction.*

Nous notons *neo* (S) la forme normale d'un système de transitions étiquetées S. Nous notons $S_1 \approx_{eo} S_2$ si les systèmes de transitions étiquetées S_1 et S_2 sont observationnellement équivalents.

1.7. Congruence par modèle d'acceptation

J.A. Bergstra, J.W. Klop et E.R. Olderog [BKO] ont défini une congruence par modèle d'acceptation sur l'algèbre ACP. Nous avons choisi la congruence par acceptation telle qu'elle a été définie par S. Graf et J. Sifakis, car elle est définie sur le même langage des processus réguliers **Trans.**

Définition axiomatique de la congruence par modèle d'acceptation

Les axiomes et les règles d'inférence sont ceux de la table 1 et de la table 3.

TABLE 3:

$(a\tau)$: $(a\tau\ 1) : a\,\tau\,t = a\ t$

$(a\tau\ 2) : a\,(\tau\,t + \Sigma\, a_i\, t_i) = a\,t + a\,(\Sigma\, a_i\, t_i)$

$(a\tau\ 3) : \tau\,(b\,t + t') + b\,t'' = \tau\,(b\,t + b\,t'' + t') + b\,t + b\,t''$

$(a\tau\ 4) : a\,(b\,t + t') + a\,(b\,t_1 + t_1') = a\,(b\,t + bt_1 + t') + a\,(b\,t + bt_1 + t_1')$

$(\mathbf{rec}\ \tau) : (\mathbf{rec}\ \tau\ 1) : \mathbf{rec}\ X.\ (\tau\,X + t) = \mathbf{rec}\ X.\ (\tau\,t + t)$

1.7.2. Définition des transformations sur les systèmes de transitions étiquetées

Les transformations qui suivent interviennent dans le calcul de la forme normale d'un système de transitions étiquetées pour la congruence par modèle d'acceptation.

a) Transformation *τ-chaîne*

La transformation (τ-chaîne) est identique à celle définie pour la congruence observationnelle.

b) Transformation *τ-init*

La validité de la transformation suivante (τ-init) peut être déduite de l'axiome (rec τ) ; elle ne concerne que l'état initial.

$$(\tau\text{-init})\ \frac{S = (Q, A, T, q_0),\ (q_0, \tau, q) \in T,\ q' \notin Q}{S' = (Q \cup \{q'\}, A, T', q')}$$

où : $T' = T \cup \{q'\} \times T\,[q_0] - \{(q_0, \tau, q_1) \mid q_1 \in T_\tau\,[q_0]\}$

Exemple

Le système de transitions étiquetées figure 13 est le transformé du système de transitions étiquetées figure 12, par la transformation τ-init

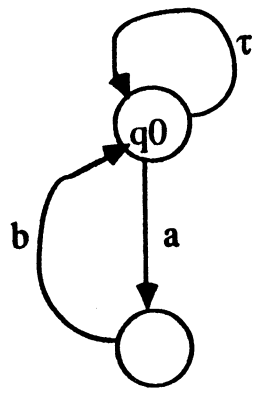


figure 12

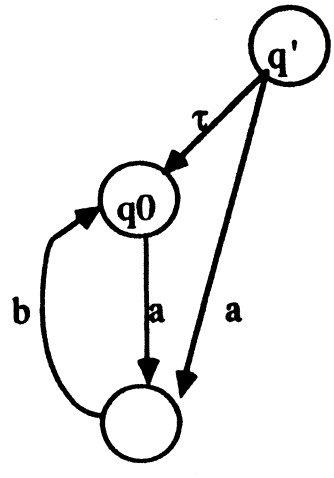


figure 13

c) Transformation *τ-élimination*

La validité de la transformation suivante (τ-élimination) peut être déduite de l'axiome (aτ2).

$$(\tau\text{-élimination}) \quad \frac{S = (Q, A, T, q_0), (q, \tau, q') \in T, q \neq q_0}{S' = (Q, A, T', q_0)}$$

où : $T' = T - \{(q, \tau, q')\} \cup T^{-1}[q] \times \{q'\}$

Exemple

Considérons le système de transitions étiquetées S_1 (figure 14).

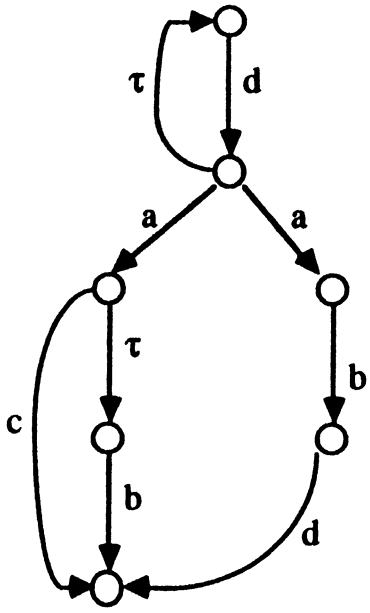


figure 14

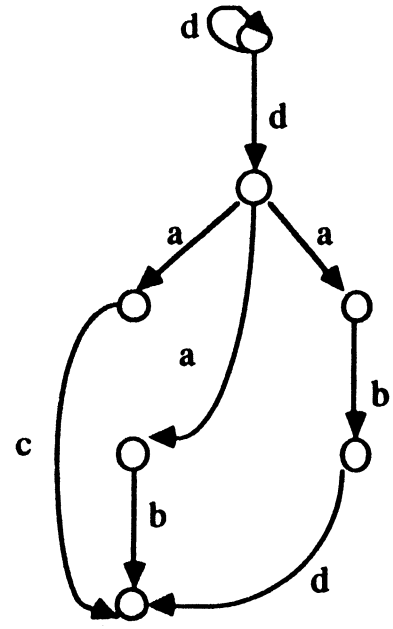


figure 15

Le système de transition étiquetées S_2, figure 15, est le transformé de S_1 par la transformation τ-élimination.

c) Transformation *τ-saturation-init*

La validité de la transformation suivante (τ -saturation init) peut être déduite de l'axiome (a τ 3).

$$\frac{S = (Q, A, T, q_0),\ (q_0, \tau, q) \in T,\ (q, b, q') \in T,\ (q_0, b, q'') \in T}{S' = (Q, A, T \cup \{(q_0, b, q'), (q, b, q'')\}, q_0)} \quad (a\tau 3)$$

d) Transformation *saturation*

La validité de la transformation suivante (saturation) peut être déduite de l'axiome (a τ4).

$$\frac{S = (Q, A, T, q_0),\ T_a^{-1}[q_1] \cap T_a^{-1}[q_2] \neq \emptyset,\ q_1' \in T_b[q_1],\ q_2' \in T_b[q_2]}{S' = (Q, A, T \cup \{(q_1, b, q_2'), (q_2, b, q_1')\}} \quad (\text{saturation})$$

Exemple

Le système de transitions étiquetées S_3, figure 16, est le transformé du système de transitions étiquetées S_2 par la transformation saturation.

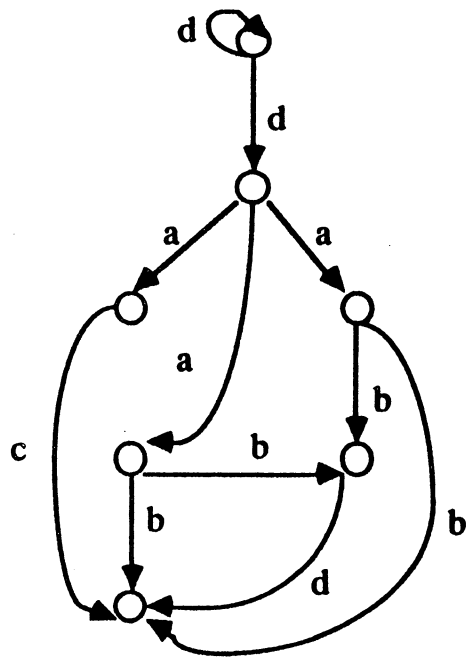


figure 16

Forme normale sur les systèmes de transitions étiquetées pour la congruence par modèle d'acceptation

S. Graf et J. Sifakis [GS86a] ont prouvé que le système de transitions étiquetées obtenu par application de l'algorithme suivant est une forme normale pour la congruence par modèle d'acceptation :

1) application des transformations (τ-init), (τ-chaîne), (τ-circuit), (τ-élimination), (τ-init-saturation) et (saturation) sur le système de transitions étiquetées.

2) Application de la bisimulation ; on obtient ainsi une partition de l'ensemble des états.

3) construction du système de transitions étiquetées quotient.

Nous notons *nac* (S) la forme normale d'un système de transitions étiquetées S. Nous notons $S_1 \sim_{ac} S_2$ si les systèmes de transitions étiquetées S_1 et S_2 sont congrus pour la congruence par modèle d'acceptation.

Exemple

Le système de transitions étiquetées S_4, figure 17, est la forme normale du système de transitions étiquetées S_1, figure 14.

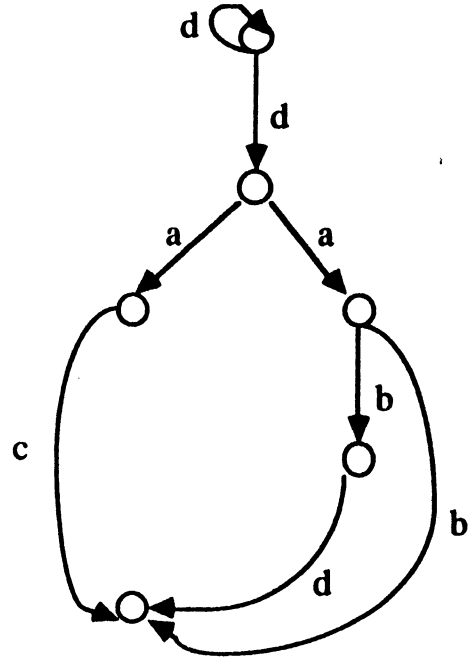


figure 17

Remarquons que le système de transitions étiquetées S_1, figure 14, est sous forme normale modulo la congruence ou l'équivalence observationnelles.

1.8. Opérateur d'abstraction

Dans ce paragraphe, nous étendons **Trans** en définissant un opérateur d'abstraction. Cette définition est inspirée de celle de l'algèbre ACP.

Définition

Nous définissons un opérateur *d'abstraction*, noté *abs*,

abs : **Trans** $\times 2^{Act} \rightarrow$ **Trans**:

Soient t, t' deux termes de **Trans** et I un sous-ensemble de *Act*

$$\frac{t \text{ -a-> } t' \text{ , } a \in I}{abs\,(I, t) \text{ -}\tau\text{-> } abs\,(I, t')} \qquad \frac{t \text{ -a-> } t', a \notin I}{abs\,(I, t) \text{ - a -> } abs\,(I, t')}$$

Cette définition engendre un opérateur *abs* sur les systèmes de transitions étiquetées.

$$\frac{S = (Q, A, T, q_0)}{abs\,(I, S) = (Q, A', T', q_0)}$$

où :

$A' = A \cup \{\tau\} - I$,

$T' = \{(q, \tau, q') \mid (q, a, q') \in T \text{ et } a \in I\} \cup$
$\{(q, a, q') \mid (q, a, q') \in T \text{ et } a \notin I\}$

La congruence forte, l'équivalence et la congruence observationnelles, et la congruence par modèle d'acceptation sont préservées par l'opérateur *abs*.

Exemple

Soit I = {a, b}, un ensemble "d'abstraction". Nous avons vu, dans l'exemple précédent, que $S_1 \sim_{ac} S_4$. Nous montrons, dans cet exemple, que
$abs\,(I, S_1) \sim_{ac} abs\,(I, S_4)$. Soient,

$S_5 = abs\,(I, S_1)$,

$S_6 = abs\,(I, S_4)$,

$S_7 = nac\ (S_5)$,

$S_8 = nac\ (S_6)$.

Les systèmes de transitions étiquetées S_5 et S_6 sont respectivement représentés, figure 18 et 19.

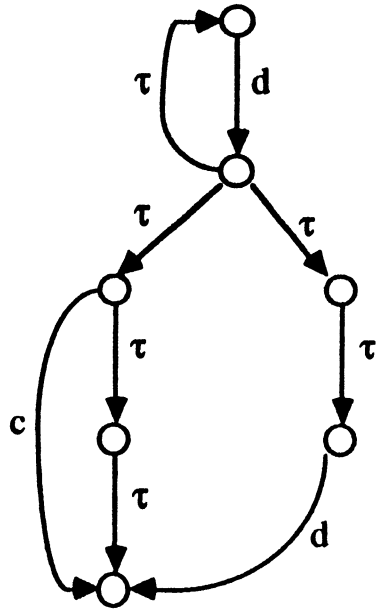


figure 18

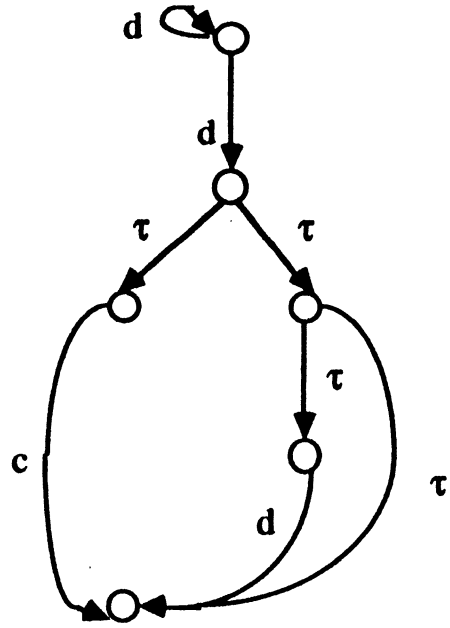


figure 19

Les systèmes de transitions étiquetées S_7 et S_8 sont identiques au système de transitions étiquetées, figure 20.

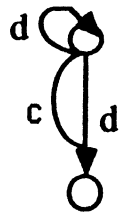


figure 20

2. Opérations de communication

Dans ce paragraphe, nous nous intéressons à la composition des systèmes de transitions étiquetées. Deux aspects de cette composition nous semblent importants :
- tout d'abord, l'opération de composition doit être générale pour permettre l'expression de différents modes de composition telle que la composition *synchrone* ou *asynchrone*, le *rendez-vous à deux* ou le *rendez-vous multiple* ;
- ensuite, il s'agit de déterminer si les relations d'équivalence définies au paragraphe précédent sont des congruences pour l'opérateur de composition. Lorsqu'une relation d'équivalence est une congruence pour l'opérateur de composition, on peut remplacer dans une composition un système de transitions étiquetées par sa forme normale (qui contient moins d'états). Cela permet de réduire en taille le calcul du système de transitions étiquetées produit.

Nous définissons deux opérateurs sur la classe des systèmes de transitions étiquetées : un *opérateur de composition* et un *opérateur de restriction.*

Nous adoptons une démarche analogue à celle de Winskel [Win83], pour définir un opérateur de composition. L'opérateur de composition est paramétré par une relation de communication. Cette relation est définie par un opérateur binaire sur l'ensemble des étiquettes, appelé par Winskel [Win83], *produit de synchronisation.* En modifiant les propriétés algébriques de cet opérateur, on peut décrire la composition synchrone, asynchrone, l'opération de communication par *rendez-vous à deux* ou par *rendez-vous multiple.*

L'opérateur de restriction [Mil80], noté \, est défini pour indiquer à l'environnement les portes de communication qui ne peuvent pas communiquer.

Nous montrons que la congruence forte et la congruence observationnelle définies au paragraphe 1, sont *compatibles* avec les opérateurs de restriction et de composition : ce sont des congruences sur la classe des systèmes de transitions étiquetées munie des opérateurs de restriction et de composition. Ceci permet d'optimiser la génération d'un système de transitions étiquetées composé en tenant compte de la congruence forte ou de la congruence observationnelle.

Lorsque l'on compose plusieurs systèmes de transitions étiquetées à l'aide des opérateurs de composition et de restriction, l'ordre dans lequel s'effectue cette composition n'est pas indifférent. Le nombre de transitions "temporaires" peut être plus ou moins important suivant qu'un opérateur de restriction s'applique ou non. Nous proposons, au chapitre II, deux stratégies pour minimiser le nombre de transitions temporaires.

2.1. Définition d'un opérateur de composition sur les systèmes de transitions étiquetées

Avant de définir formellement la composition de deux systèmes de transitions étiquetées, nous illustrons sur un exemple les résultats possibles de cette composition. Considérons les systèmes de transitions étiquetées S_1 (figure 1) et S_2 (figure 2).

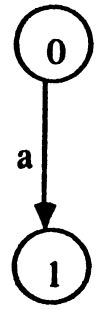


figure 1

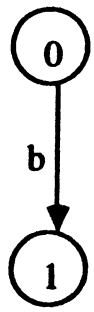


figure 2

Le résultat de la composition peut être le système de transitions étiquetées de la figure 3, si nous choisissons un opérateur de composition asynchrone ; le système de transitions étiquetées de la figure 4, si nous considérons l'opérateur d'entrelacement ; le système de transitions étiquetées de la figure 5, si nous considérons un opérateur synchrone. Pour exprimer ces différentes compositions avec un seul opérateur, nous définissons le *produit* de systèmes de transitions étiquetées. Intuitivement, l'ensemble des états est le produit cartésien des états et les transitions sont étiquetées par des couples d'étiquettes. Le produit des systèmes de transitions étiquetées de la figure 1 et 2 est représenté par la figure 6. Nous définissons un *produit de synchronisation*, noté •, qui permet d'associer une étiquette à un couple d'étiquettes. A l'aide du produit de systèmes

de transitions étiquetées et du produit de synchronisation, nous pouvons définir la composition parallèle de systèmes de transitions étiquetées.

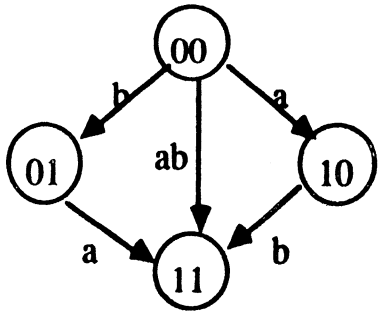


figure 3

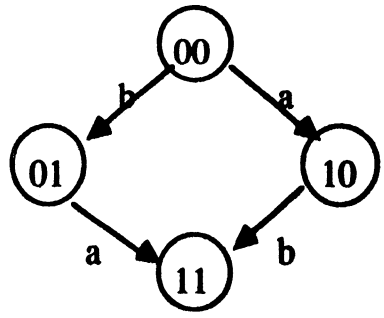


figure 4

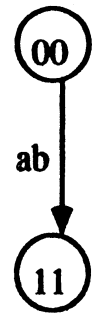


figure 5

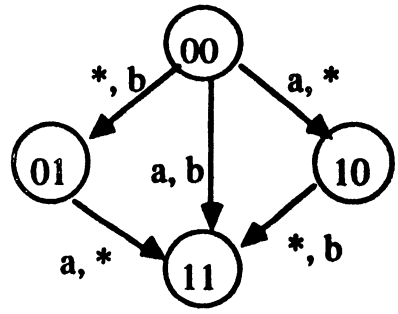


figure 6

Nous avons intoduit le symbole * qui permet, à l'aide du produit de synchronisation • de définir un opérateur de composition synchrone (figure 5) si a • * = 0 (i.e. 0 étiquette une transition vide) ou asynchrone (figures 3, 4) si a • * = a.

Dans cet exemple le produit de synchronisation • est commutatif,

La valeur du produit a • b peut être soit 0 (figure 4) soit une étiquette de Act_τ (figure 3).

Notations

Nous étendons l'ensemble Act_τ, en $Act' = Act_\tau \cup \{0, *\}$.

Définition du produit de deux systèmes de transitions étiquetées

Soient deux systèmes de transitions $S_i = (Q_i, A_i, T_i, q_i)$, $i = 1, 2$.

1) Nous définissons un produit entre sous-ensembles de *Act'*, noté $\times_*$, de la manière suivante : si A et B sont des sous-ensembles de Act_τ,

$A \times_* B = (A \times B) \cup (A \times \{*\}) \cup (\{*\} \times B)$.

2) Notons T_i^* l'ensemble $T \cup \{(q, *, q) \mid q \in Q_i\}$; nous définissons le *produit* $S_1 \times S_2$ des systèmes de transitions étiquetées S_1, et S_2 de la manière suivante,

$S_1 \times S_2 = (Q, A, T, q_0)$ où ,

$A = A_1 \times_* A_2$,

T et Q sont définis par la règle suivante :

$$\frac{(p_1, p_2) \in Q,\ (p_1, a, p'_1) \in T_1^*,\ (p_2, b, p'_2) \in T_2^*}{T \leftarrow T \cup \{(p_1, p_2), (a, b), (p'_1, p'_2)\},\ Q \leftarrow Q \cup \{(p'_1, p'_2)\}}$$

Initialement $T = \emptyset$ et $Q = \{q_0\}$ où

$q_0 = (q_1, q_2)$

Le produit de synchronisation défini sur *Act'*, structure *Act'* en une *algèbre de synchronisation.*

Définition de l'algèbre de synchronisation

Une *algèbre de synchronisation* [Win83] $L = (Act', \bullet)$ vérifie les conditions suivantes :

(i) $(Act', \bullet)$ est un monoïde (i.e. $\bullet$ est associatif),

(ii) 0 est absorbant (i.e. $\forall\ a \in Act'$, $a \bullet 0 = 0$).

(iii) $* \bullet * = *$ et $a \bullet b = *$ si et seulement si $a = b = *$

Dans la suite de ce travail, nous nous intéressons aux algèbres de synchronisation respectant les contraintes suivantes :

(1) le produit de synchronisation $\bullet$ est commutatif,

(2) $a \bullet \tau = 0$, pour $a \in Act_\tau$

(3) $* \bullet \tau = \tau$

(4) $* \bullet a = a$ ou $* \bullet a = 0$, pour $a \in Act$.

La condition (2) exprime qu'un événement interne ne peut être synchronisé avec un événement externe ; la condition (3) exprime que les événements internes apparaissent de manière asynchrone lors de la composition parallèle ; la condition (4) définit un produit de synchronisation asynchrone ou synchrone.

Définition de l'opérateur de composition

Soient,

- une algèbre de synchronisation L,
- deux systèmes de transitions étiquetées $S_i = (Q_i, A_i, T_i, q_i)$,
- une bijection, $< , > : P \times P \rightarrow P$.

On définit la composition $S = (Q, A, T, q_0) = (S_1 \|_L S_2)$ de S_1 et S_2 à partir de $S_1 \times S_2$ en remplaçant l'étiquette (a, b) par l' étiquette $a \bullet b$:

Si $S_1 \times S_2 = (Q, A_1 \times_* A_2, T, q_0)$,

$S_1 \|_L S_2 = (Q', A, T', q_0)$ où :

$Q' = \{<p, q> \mid (p, q) \in Q\}$

$A = \{a \bullet b \mid (a, b) \in A_1 \times_* A_2 \text{ et } a \bullet b \neq 0\}$

$T' = \{(<p_1, p_2>, a \bullet b, <q_1, q_2>) \mid a \bullet b \in A, \text{ et } ((p_1, p_2), (a, b), (q_1, q_2)) \in T\}$

Propriétés de l'opérateur de composition

Soit L une algèbre de synchronisation vérifiant les contraintes (1)-(4) ci-dessus. $\|_L$ est associatif et commutatif.

L'associativité et la commutativité de $\|_L$ se déduisent de l'associativité et de la commutativité du produit de synchronisation.

Exemples de quelques algèbres de synchronisation

1) Une algèbre L_1 de synchronisation pour l'opérateur | de CCS sans passage de valeurs.

Cette algèbre est caractérisée par les propriétés suivantes :

- pour chaque étiquette, a, il existe une étiquette a', appelée *complémentaire*,

telle que $a \bullet a' = \tau$;

- $a \bullet b = 0$ si et seulement si $b \neq a'$ (les seules synchronisations possibles sont les communications) ;

- $a \bullet * = a$ (cette propriété caractérise un opérateur asynchrone)· La table suivante définit l'opérateur $\bullet$ pour L_1.

$\bullet$	0	τ	a	b	*
0	0	0	0	0	0
τ	0	0	0	0	τ
a'	0	0	τ	0	a'
b'	0	0	0	τ	b'
*	0	τ	a	b	*

En composant les systèmes de transitions étiquetées S_1 et S_2 des figures 1 et 2, nous obtenons le système de transitions étiquetées de la figure 3 (resp. 4) si $b = a'$ (resp. $b \neq a'$).

2) Une algèbre L_2 de synchronisation pour l'opérateur || de SCCS [Mil 83] avec abstraction.

Cette algèbre est caractérisée par les propriétés suivantes :

- nous distinguons un élément de Act', noté 1, élément neutre pour le produit de synchronisation $\bullet$, tel que le triplet (Act', $\bullet$, 1) a une structure de semi-groupe abélien (i.e. 1 est élément neutre) ;

- $a \bullet * = 0$ (cette propriété caractérise un opérateur synchrone).

En composant les systèmes de transitions étiquetées S_1 et S_2 des figures 1 et 2, nous obtenons le système de transitions étiquetées de la figure 5.
Dans la table suivante, qui définit l'opérateur $\bullet$, la valeur du produit $a \bullet b$ n'est pas précisée, lorsque ces étiquettes appartiennent à Act.

•	0	1	a	b	*
0	0	0	0	0	0
1	0	1	a	b	0
a	0	a	a• a	a• b	0
b	0	b	a• b	b • b	0
*	0	0	0	0	*

3) Une algèbre L_3 de synchronisation exprimant le *rendez-vous multiple.*

Dans cette algèbre, seules les transitions étiquetées par τ peuvent apparaître de manière asynchrone. Les étiquettes de même nom sont synchronisées. A la différence de l'opérateur | de CCS, plusieurs composants d'un même système peuvent être synchronisés . Le produit est synchrone. Cette algèbre est donc caractérisée par :

- $a \bullet * = 0$, pour $a \neq \tau$,
- $a \bullet a = a$, pour $a \neq \tau$,
- $a \bullet b = 0$, pour $a \neq b$.

•	0	τ	a	b	*
0	0	0	0	0	0
τ	0	0	0	0	τ
a	0	0	a	0	0
b	0	0	0	b	0
*	0	τ	0	0	*

En composant les systèmes de transitions étiquetées S_1 et S_2 des figures 1 et 2, nous obtenons le système de transitions étiquetées de la figure 5 si a = b et le système de transitions étiquetées vide (un état, une relation de transitions étiquetées vide) sinon.

4) Une algèbre L_4 de synchronisation pour l'opérateur *d'entrelacement*

Opération *d'entrelacement* : aucune étiquette n'est synchronisée avec une autre lors de la composition :

- a • b = 0 pour a et b différents de *;
- a • * = a.

•	0	τ	a	b	*
0	0	0	0	0	0
τ	0	0	0	0	τ
a	0	0	0	0	a
b	0	0	0	0	b
*	0	τ	a	b	*

En composant les systèmes de transitions étiquetées des figures 1 et 2, nous obtenons le système de transitions étiquetées de la figure 4.

Nous définissons un opérateur de restriction qui, étant donné un système de transitions étiquetées et un ensemble d'étiquettes, supprime les transitions étiquetées par un élément de cet ensemble.

Définition de l'opérateur de restriction

Nous définissons un opérateur de restriction \, qui, à un système de transitions étiquetées S = (Q, A, T, q_0) et à un ensemble B, sous-ensemble de *Act*, associe un système de transitions étiquetées,
S' = (Q', A', T', q_0) où :

- $A' = A - B$,
- T' est définie par la règle suivante :

$$\frac{(q, a, q') \in T,\ a \notin B}{(q, a, q') \in T'}$$

- Q' est l'ensemble des états atteignables à partir de l'état initial.

Une propriété intéressante est la distributivité, sous certaines conditions, de l'opérateur de restriction par rapport à la composition.

Notations

Soit L une algèbre de synchronisation.

(i) Nous étendons le produit de synchronisation aux ensembles :

Soient A et B, deux sous-ensembles de *Act*, nous notons

$A \bullet B = \{a \bullet b \mid a \in A \text{ et } b \in B \text{ et } a \bullet b \neq 0 \text{ et } a \bullet b \neq \tau\}$

(ii) Soit a un élément de *Act*. Nous définissons *Com* (a) de la manière suivante :

Com $(a) = \{b \mid b \in Act \text{ et } a \bullet b \neq 0\}$

Les propriétés suivantes se montrent facilement.

Propriétés

Soient $S_i = (Q_i, A_i, T_i, q_i)$ $i = 1, 2$ deux systèmes de transitions étiquetées et soit L une algèbre de synchronisation.

(i) $(S_1 \setminus I_1) \setminus I_2 = S_1 \setminus I_1 \cup I_2$

(ii) $(S_1 \setminus I_1) = S_1$ si $I_1 \cap A_1 = \emptyset$.

(iii) $(S_1 \parallel_L S_2) \setminus \{a\} = (S_1 \setminus \{a\}) \parallel_L S_2$ si

$(a \notin A_1 \bullet A_2$ et $a \notin A_2$ et $(a \in A_1 \Rightarrow Com\ (a) \cap A_2 = \emptyset))$ ou

$(a \in A_1 \bullet A_2$ et $((a = b \bullet c) \Rightarrow a = b = c))$

(iv) $(S_1 \parallel_L S_2) \setminus \{a\} = (S_1 \setminus \{a\}) \parallel_L (S_2 \setminus \{a\})$ si

$(a \notin A_1 \bullet A_2$ et $(a \in A_1 \Rightarrow Com\ (a) \cap A_2 = \emptyset)$

et $(a \in A_2 \Rightarrow Com\ (a) \cap A_1 = \emptyset))$ ou

$(a \in A_1 \bullet A_2$ et $((a = b \bullet c) \Rightarrow a = b = c))$

2.2. Congruences

Nous allons déterminer, pour chaque relation d'équivalence définie au paragraphe 1, celles qui sont des congruences sur la classe des systèmes de transitions étiquetées munies des opérateurs de composition et de restriction. Par abus de langage, nous notons $\mathbf{Trans}_L$ la classe des systèmes de transitions, munie des opérateurs $\|_L$ et $\setminus$, pour une algèbre de synchronisation donnée L. Pour qu'une relation d'équivalence $\sim_{eq}$ sur les systèmes de transitions étiquetées soit une congruence sur $\mathbf{Trans}_L$, il faut et il suffit que les deux propriétés suivantes soient vérifiées :

(i) $\forall S \in \mathbf{Trans}_L\ S_1 \sim_{eq} S_2 \Rightarrow (S_1 \|_L S) \sim_{eq} (S_2 \|_L S)$

(ii) $\forall I \subseteq Act\ S_1 \sim_{eq} S_2 \Rightarrow (S_1 \setminus I) \sim_{eq} (S_2 \setminus I)$

Les relations d'équivalence $\sim$ et $\sim_{CO}$ sont des congruences. Pour ces deux relations, nous montrons uniquement la propriété (i). La propriété (ii) est facile à montrer. Nous montrons sur un contre-exemple que la relation d'équivalence par modèle d'acceptation $\sim_{ac}$ n'est pas une congruence.

Proposition

La relation $\sim$ est une congruence sur $\mathbf{Trans}_L$.

Principe de démonstration

Soient trois systèmes de transitions étiquetées $S_i = (Q_i, A_i, T_i, q_i)$ pour i = 1, 2 et $S = (Q, A, T, q_0)$.

Notons ρ_{12} la plus grande bisimulation entre S_1 et S_2 . Il suffit de montrer qu'il existe une bisimulation entre $(S_1 \|_L S)$ et $(S_2 \|_L S)$. La relation

$\rho = \{(\langle p_1, p\rangle, \langle p_2, p\rangle) \mid (p_1, p_2) \in \rho_{12} \text{ et } p \in Q\}$,

est une bisimulation entre $S_1 \|_L S$ et $S_2 \|_L S$. En effet,

(i) $\langle q_1, q_0\rangle, \langle q_2, q_0\rangle) \in \rho$,

(ii) si $\langle p_1, p\rangle, \langle p_2, p\rangle \in \rho$ et $a \in Act$ et $q \in T_a\ [\langle p_1, p\rangle]$ alors trois cas sont possibles :

<u>cas 1</u> : q est de la forme $\langle r_1, p\rangle$ et $a = a \bullet *$.

$r_1 \in T_a\ [p_1]$ et $(p_1, p_2) \in \rho_{12} \Rightarrow \exists r_2\ r_2 \in T_a\ [p_2]$ et $(r_1, r_2) \in \rho_{12}$.

Par conséquent, $\langle r_1, p\rangle, \langle r_2, p\rangle \in \rho$.

<u>cas 2</u> : q est de la forme $\langle p_1, r\rangle$ et $a = * \bullet a$.

Nous avons, par définition de ρ, $\langle p_1, r\rangle, \langle p_2, r\rangle \in \rho$.

<u>cas 3</u> : q est de la forme $\langle r_1, r\rangle$ et $a = b \bullet c$.

$r_1 \in T_b\,[p_1]$ et $(p_1, p_2) \in \rho_{12} \Rightarrow \exists r_2\ r_2 \in T_b\,[p_2]$ et $(r_1, r_2) \in \rho_{12}$.
On déduit, de $r_2 \in T_b\,[p_2]$ et $r \in T_c\,[p]$, $\langle r_2, r\rangle \in T_a\,[\langle p_2, p\rangle]$. De plus, $\langle r_1, r\rangle, \langle r_2, r\rangle \in \rho$.

On procède de même pour la condition symétrique :
$\langle p_1, p\rangle, \langle p_2, p\rangle \in \rho$ et $a \in Act$ et $q \in T_a\,[\langle p_2, p\rangle]$.
□

Proposition

La congruence observationnelle $\sim_{co}$ est une congruence sur $\mathbf{Trans}_L$.

Principe de démonstration :

Soient trois systèmes de transitions étiquetées $S_i = (Q_i, A_i, T_i, q_i)$ pour $i = 1, 2$ et $S = (Q, A, T, q_0)$.

1) Nous utilisons une autre caractérisation de la congruence observationnelle [Mil85] :

- Soit $S = (Q, A, T, q_0)$ un système de transitions étiquetées. Nous définissons l'ensemble $T_a^+\,[p]$, pour $a \in Act_\tau$, de la manière suivante :

$T_a^+\,[p] = \{q \mid \exists q_1, \ldots q_n, r_1, \ldots, r_m$ avec $p = q_1$ et $q = r_m$ vérifiant :
$q_{i+1} \in T_\tau\,[q_i]$ pour $i < n$, $r_1 \in T_a\,[q_n]$ et $r_{j+1} \in T_\tau\,[q_j]$ pour $j < m\}$
$T_\varepsilon\,[p] = T_\tau^+\,[p] \cup \{p\}$.

- Soient deux systèmes de transitions étiquetées $S_i = (Q_i, A_i, T_i, q_i)$ pour $i = 1, 2$.
$S_1 \sim_{co} S_2$ si et seulement si il existe une relation ρ_{co} vérifiant :

(i) $(q_1, q_2) \in \rho_{co}$
(ii) $\forall (p_1, p_2) \in \rho_{co}$ et $\forall a \in Act \Rightarrow$
$\quad (\forall r_1 \in T_a\,[p_1] \Rightarrow \exists\, r_2 \in T_a^+\,[p_2]$ et $(r_1, r_2) \in \rho_{co})$
(iii) $\forall (p_1, p_2) \in \rho_{co}$ et $\forall a \in Act \Rightarrow$
$\quad (\forall r_2 \in T_a\,[p_2] \Rightarrow \exists\, r_1 \in T_a^+\,[p_1]$ et $(r_1, r_2) \in \rho_{co})$
(iv) $\forall (p_1, p_2) \in \rho_{co} \Rightarrow (\forall r_1 \in T_\tau\,[p_1] \Rightarrow \exists\, r_2 \in T_\varepsilon\,[p_2]$ et $(r_1, r_2) \in \rho_{co})$
(v) $\forall (p_1, p_2) \in \rho_{co} \Rightarrow (\forall r_2 \in T_\tau\,[p_2] \Rightarrow \exists\, r_1 \in T_\varepsilon\,[p_1]$ et $(r_1, r_2) \in \rho_{co})$
(vi) $(q_1, q_2) \in \rho_{co} \Rightarrow (\forall r_1 \in T_\tau\,[q_1] \Rightarrow \exists\, r_2 \in T_\tau^+\,[q_2]$ et $(r_1, r_2) \in \rho_{co})$
(vii) $(q_1, q_2) \in \rho_{co} \Rightarrow (\forall r_2 \in T_\tau\,[q_2] \Rightarrow \exists\, r_1 \in T_\tau^+\,[q_1]$ et $(r_1, r_2) \in \rho_{co})$

Nous appelons ρ_{co} "τ^*- bisimulation".

2) Nous procédons de la même manière que pour la démonstration de la congruence forte en remplaçant bisimulation par "τ^*- bisimulation".
□

La congruence par acceptation définie au premier paragraphe, n'est pas une congruence sur $\mathbf{Trans}_L$. Pour s'en convaincre, il suffit de considérer le contre-exemple suivant :

Soit L une algèbre de synchronisation asynchrone (i.e. a • * = a). Considérons les systèmes de transitions étiquetées S_1 (figure 7), S_2 (figure 8) et S (figure 9). $S_1 \sim_{ac} S_2$ mais $S_1 \|_L S$ n'est pas congru à $S_2 \|_L S$. En effet, la forme normale de $S_1 \|_L S$ (figure 10) et la forme normale de $S_2 \|_L S$ (figure 11) sont distinctes.

Nous pouvons trouver un contre-exemple pour une algèbre de synchronisation synchrone.

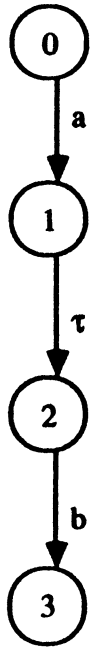


figure 7

figure 8

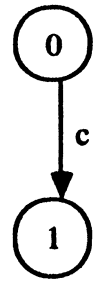


figure 9

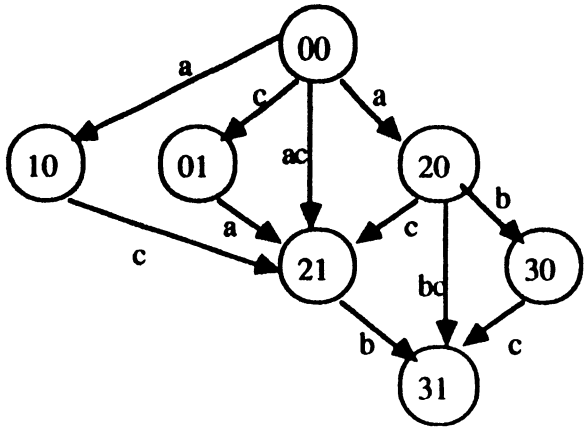


figure 10

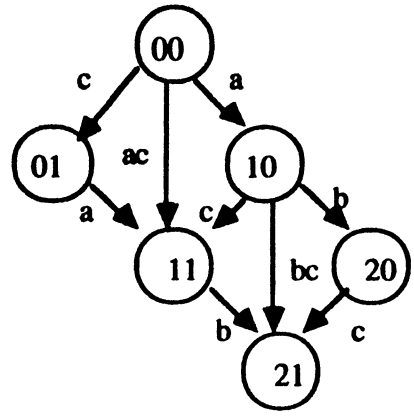


figure 11

3 Conclusion

Nous nous sommes intéressés dans ce chapitre à des relations plus grossières que la plus grande bisimulation et telles que, pour chaque classe d'équivalence, il existe un représentant canonique. Du fait que ces relations d'équivalence contiennent la plus grande bisimulation, le représentant canonique, ou forme normale, peut être choisi de telle sorte que le nombre de ses états soit minimal.

Nous avons défini un opérateur d'abstraction sur les systèmes de transitions étiquetées. Cet opérateur d'abstraction, combiné avec la congruence observationnelle, l'équivalence observationnelle ou la congruence par modèle d'acceptation, permet de comparer deux systèmes de transitions étiquetées "modulo un ensemble d'actions".

L'opérateur de composition parallèle sur les systèmes de transitions étiquetées est paramétré par une algèbre de synchronisation. Il permet de décrire la composition synchrone ou asynchrone, l'opération de communication par "rendez-vous" à deux ou multiple.

La mise en œuvre efficace d'un calcul de forme normale repose sur la mise en œuvre efficace du calcul de la plus grande bisimulation. Nous présentons, au chapitre II, un algorithme dont le coût en temps est quasi-linéaire par rapport à la taille de la relation de transitions étiquetées.

Les relations de congruence forte et observationnelle sont compatibles avec les opérateurs de composition et de restriction. Nous pouvons définir des stratégies de composition, en utilisant cette propriété de manière à réduire les résultats intermédiaires. Il s'agit de minimiser le nombre d'états et le nombre de transitions temporaires engendrés par la composition. Nous pouvons, de plus, utiliser certaines propriétés algébriques des opérateurs de composition et de restriction pour réduire encore le nombre d'états et le nombre de transitions temporaires. Nous développons ce point au chapitre II.

CHAPITRE II

TRANSFORMATIONS SUR LES SYSTEMES DE TRANSITIONS ETIQUETEES

Dans ce chapitre, nous décrivons la mise en œuvre des transformations définies au chapitre I. Ce sont, d'une part les transformations sur les systèmes de transitions étiquetées permettant le calcul d'une forme normale modulo une relation d'équivalence, d'autre part les transformations utilisées pour réduire le nombre d'états et le nombre de transitions temporaires enegendrés par la composition de systèmes de transitions étiquetées.

Nous nous intéressons dans ce chapitre à une mise en œuvre efficace des transformations sur les systèmes de transitions étiquetées. Nous distinguons deux sortes de transformations sur les systèmes de transitions étiquetées intervenant dans le calcul des formes normales :

- une *transformation globale*, c'est le calcul de la plus grande bisimulation,
- les *transformations locales*, qui interviennent dans l'algorithme de calcul d'une forme normale modulo une relation d'équivalence. Nous rappelons quelles sont les relations d'équivalence considérées : ce sont la congruence forte, la congruence et l'équivalence observationnelles, et la congruence par modèle d'acceptation.

L'algorithme choisi pour le calcul de la plus grande bisimulation est inspiré de celui décrit par [PT86] pour résoudre le problème de la partition la moins fine compatible avec une relation binaire. C'est à l'heure actuelle l'algorithme le plus performant du point de vue du temps d'exécution pour calculer la plus grande bisimulation : la complexité de cet algorithme est de l'ordre de $m \log n$ où m (resp. n) désigne le cardinal de la relation de transition (resp. de l'ensemble des états).

Les algorithmes décrivant une transformation locale sont des algorithmes sur les graphes et sont construits selon le principe suivant :

- parcours de la relation de transition,
- pour chaque transition vérifiant une certaine propriété, transformation

de la relation de transition. Par exemple, pour la mise en œuvre de la transformation *τ-chaîne*, la propriété qui permet la transformation de la relation de transitions est "l'état courant a un et un seul successeur par une transition étiquetée par τ".

Ce chapitre est divisé en trois parties :

Tout d'abord, nous présentons le problème de la partition la moins fine compatible avec une relation binaire, et la mise en œuvre de sa solution par Paige et Tarjan [PT86]. Nous étendons cet algorithme aux systèmes de transitions étiquetées, ce qui permet de calculer le système de transitions étiquetées quotient modulo la plus grande bisimulation. Le calcul de la bisimulation entre deux systèmes de transitions étiquetées est une conséquence de cette solution.

La deuxième partie est consacrée à la présentation des algorithmes décrivant les transformations locales sur les systèmes de transitions étiquetées.

Nous présentons ensuite la mise en œuvre de la génération d'un système de transitions étiquetées composé.

1. Transformation globale

Avant d'énoncer le problème de la partition la moins fine compatible avec une relation de transition, nous rappelons les définitions et notations utiles dans la suite de ce paragraphe.

Rappel et notations

Un *système de transitions* (Q, R) est la donnée d'un ensemble fini d'états Q et d'une relation de transition sur les états $R \subseteq Q \times Q$. A la différence d'un système de transitions étiquetées, il n'a pas d'état initial et les transitions ne sont pas étiquetées. Un système de transitions étiquetées (Q, A, T, q_0) est un système de transitions (Q, R) où :

$$R = \bigcup_{a \in A} \{(q, q') \mid (q, a, q') \in T\}$$

Nous adoptons les mêmes notations qu'au chapitre I :

$R[p] = \{q \mid (p, q) \in R\}$;

$R^{-1}[q] = \{p \mid (p, q) \in R\}$;

Dans toute la suite, Q désigne un ensemble d'états ayant n éléments ; pour plus de commodité, ces éléments sont identifiés aux n premiers entiers.

Une *partition* ρ de Q est une famille $\{E_i \mid i \in I\}$ de sous-ensembles de Q vérifiant $\forall i, j \in I . i \neq j \Rightarrow E_i \cap E_j = \emptyset$ et $Q = \cup \{E_i \mid i \in I\}$; toute partition d'un ensemble Q définit une *relation d'équivalence* sur Q et inversement ; dans toute la suite, nous employons indifféremment les termes relation d'équivalence ou partition ; un élément d'une partition est appelé *classe d'équivalence.*

Soient ρ et ρ' deux partitions de Q, ρ' est un *raffinement* de ρ, et on note $\rho' \subseteq \rho$ si et seulement si : $\forall X \in \rho' \exists Y \in \rho . X \subseteq Y$

Etant donné un système de transition (Q, R) et une relation d'équivalence $\rho = \{B_j \mid j \in J\}$, ρ est *compatible* avec la relation R ssi

$\forall i, j \in J \forall x, y \in B_i . R[x] \cap B_j \neq \emptyset$ si et seulement si $R[y] \cap B_j \neq \emptyset$

1.1. Présentation du problème

L'énoncé du problème est le suivant :

" Etant donné un système de transition (Q, R) et une relation d'équivalence ρ sur Q, trouver la relation d'équivalence ρ' la moins fine qui soit un raffinement de ρ *compatible* avec R".

Exemple 2.1:

On considère le système de transition (Q, R) où, $Q = \{1, 2, 3, 4, 5, 6\}$ et la relation R est représentée graphiquement par la figure 1.

$R[1] = \{2, 4\}$,
$R[2] = \{3, 4, 5\}$,
$R[3] = \{5, 1\}$,
$R[4] = \{5, 6\}$,
$R[5] = \{4, 6\}$,
$R[6] = \emptyset$.

Considérons les relations d'équivalence suivantes :

$\rho_1 = \{\{1, 2, 3\}, \{4, 5\}, \{6\}\}$,
$\rho_2 = \{\{1, 2, 3\}, \{4, 5, 6\}\}$;

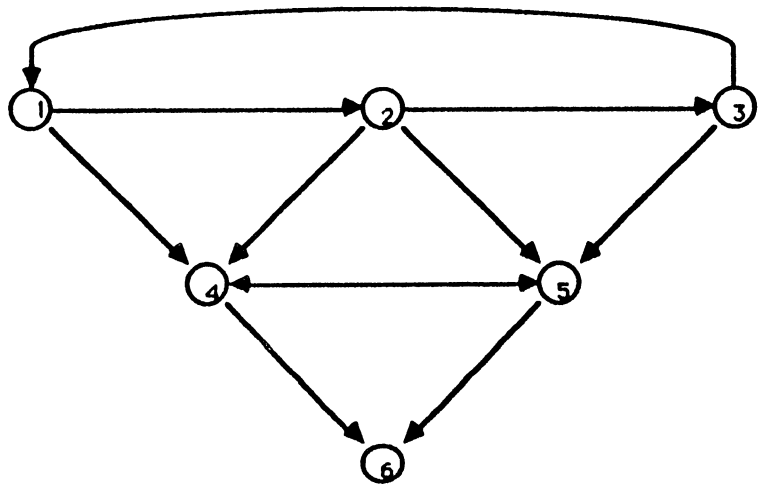


Figure 1

ρ_1 est *compatible* avec R, en effet,
pour chaque élément p de la classe {1, 2, 3},
R [p] $\cap$ {1, 2, 3} $\neq \emptyset$,
R [p] $\cap$ {4, 5} $\neq \emptyset$ et
R [p] $\cap$ {6} $= \emptyset$;
pour chaque élément p de la classe {4, 5},
R [p] $\cap$ {1, 2, 3} $= \emptyset$,
R [p] $\cap$ {4, 5} $\neq \emptyset$ et
R [p] $\cap$ {6} $\neq \emptyset$;
pour chaque élément p de la classe {6},
R [p] $\cap$ {1, 2, 3} $= \emptyset$,
R [p] $\cap$ {4, 5} $= \emptyset$ et
R [p] $\cap$ {6} $= \emptyset$.
ρ_2 *n'est pas compatible* avec R, en effet,
R [4] $\cap$ {4, 5, 6} $\neq \emptyset$ et
R [6] $\cap$ {4, 5, 6} $= \emptyset$;
ρ_1 est un *raffinement* de ρ_2, les classes {4, 5} et {6} de ρ_1 sont incluses dans la classe {4, 5, 6} de ρ_2.

Proposition 2.1.1. *Caractérisation de la compatibilité* :
Etant donné un système de transition (Q, R) et une relation d'équivalence ρ,
ρ est compatible avec R si et seulement si

$$\forall B, B' \in \rho, B' \subseteq R^{-1}[B] \text{ ou } B' \cap R^{-1}[B] = \emptyset.$$

Preuve :

La propriété

$\forall B, B' \in \rho, B' \subseteq R^{-1}[B]$ ou $B' \cap R^{-1}[B] = \emptyset$

est logiquement équivalente à la propriété

$\forall B, B' \in \rho, B' \cap R^{-1}[B] \neq \emptyset \Rightarrow B' \subseteq R^{-1}[B]$.

$\Rightarrow$: Soit ρ une relation d'équivalence compatible avec R, nous avons $\forall B, B' \in \rho$,

$B' \cap R^{-1}[B] \neq \emptyset \Rightarrow \exists p \in B', R[p] \cap B \neq \emptyset$

$\Rightarrow \forall q \in B', R[q] \cap B \neq \emptyset$

$\Rightarrow \forall q \in B', q \in R^{-1}[B] \neq \emptyset$

$\Rightarrow B' \subseteq R^{-1}[B]$

$\Leftarrow$: Réciproquement, si ρ est une relation d'équivalence vérifiant la propriété de la proposition , soient $B, B' \in \rho$, $p, q \in B'$

$R[p] \cap B \neq \emptyset \Rightarrow B' \cap R^{-1}[B] \neq \emptyset$

$\Rightarrow B' \subseteq R^{-1}[B]$

$\Rightarrow R[q] \cap B \neq \emptyset$

□

Soit (Q, R) un système de transitions, nous établissons maintenant le lien entre relation de bisimulation sur (Q, R) et relation d'équivalence sur Q compatible avec R. Pour cela, nous rappelons la définition de la bisimulation sur les systèmes de transitions étiquetées, pour l'adapter aux systèmes de transitions.

Soit $S = (Q, A, T, q_0)$ un système de transitions étiquetées et $\rho \subseteq Q \times Q$,

ρ est une bisimulation si et seulement si

(i) $(q_0, q_0) \in \rho$,

(ii) $\forall (p_1, p_2) \in \rho, \forall a \in Act_\tau, \Rightarrow$

$(\forall p_1' . p_1' \in T_a[p_1] \Rightarrow \exists p_2' . p_2' \in T_a[p_2]$ et $(p_1', p_2') \in \rho)$

$(\forall p_2' . p_2' \in T_a[p_2] \Rightarrow \exists p_1' . p_1' \in T_a[p_1]$ et $(p_1', p_2') \in \rho)$.

Un système de transitions n'a pas d'état initial. Nous définissons la notion de bisimulation sur les systèmes de transitions en supprimant la condition (i).

ρ est une bisimulation si et seulement si

$\forall (p_1, p_2) \in \rho \Rightarrow$

$(\forall p_1' . p_1' \in R[p_1] \Rightarrow \exists p_2' . p_2' \in R[p_2]$ et $(p_1', p_2') \in \rho)$

$(\forall p_2' . p_2' \in R[p_2] \Rightarrow \exists p_1' . p_1' \in R[p_1]$ et $(p_1', p_2') \in \rho)$.

La proposition suivante énonce que, étant donné un système de transitions (Q, R), une relation d'équivalence ρ est une bisimulation si et seulement si elle est compatible avec R.

Proposition 2.1.2

Soient (Q, R) un système de transition et ρ une relation d'équivalence définie sur Q,

ρ est une bisimulation si et seulement si ρ est compatible avec R.

La preuve se fait en comparant les définitions de la compatibilité et de la bisimulation.

□

La plus grande bisimulation sur Q est une relation d'équivalence, donc elle est plus petite que la relation d'équivalence la moins fine compatible avec R, raffinement de la partition universelle d'après la proposition précédente. Réciproquement, la relation d'équivalence la moins fine compatible avec R est une bisimulation, donc elle est contenue dans la plus grande bisimulation. Il revient au même de calculer la plus grande bisimulation sur Q ou la relation d'équivalence la moins fine compatible avec R, raffinement de la partition universelle.

Exemple 2.2

Considérons le système de transitions de la figure 1. La plus grande bisimulation est $\rho_1 = \{\{1, 2, 3\}, \{4, 5\}, \{6\}\}$; cette relation d'équivalence est la plus grande compatible avec R : elle est incluse dans $\rho_2 = \{\{1, 2, 3\}, \{4, 5, 6\}\}$ qui n'est pas compatible avec R et elle est incluse dans la relation universelle qui n'est pas compatible avec R.

1.2 Existence et unicité de la solution

Etant donné un système de transitions (Q, R) et une relation d'équivalence initiale ρ_0, l'existence et l'unicité de la solution proviennent du fait que l'ensemble des relations d'équivalence sur Q, muni de la relation d'ordre $\subseteq$, forme un treillis complet. Nous allons calculer le plus grand point fixe d'un opérateur Φ défini sur l'ensemble des relations d'équivalence sur Q, qui, étant donné une relation d'équivalence ρ, décompose chaque classe B' de ρ en utilisant une classe B. Cet opérateur Φ est défini à partir d'un opérateur Φ'.

Définition 2.1.2 *de l'opérateur Φ'*

Soient ρ une relation d'équivalence sur Q et B un sous-ensemble de Q.

$\Phi'(\rho, B) = \{ B' \cap R^{-1}[B] \mid B' \in \rho \} \cup \{B' - R^{-1}[B] \mid B' \in \rho\}$.

L'opérateur Φ' vérifie les propriétés suivantes :

Propriété *de l'opérateur Φ'*

Soient B, B_1 et B_2 des sous-ensembles de Q et soient ρ, ρ_1 et ρ_2 trois partitions de Q.

(i) $\Phi'(\rho, B)$ est une partition.

(ii) *monotonicité* : $\rho_1 \subseteq \rho_2 \Rightarrow \Phi'(\rho_1, B) \subseteq \Phi'(\rho_2, B)$,

(iii) $\Phi'(\rho, B) \subseteq \rho$,

(iv) $\Phi'(\Phi'(\rho, B_1), B_2) = \Phi'(\Phi'(\rho, B_2), B_1)$,

(v) $\Phi'(\Phi'(\rho, B_1), B_2) \subseteq \Phi'(\rho, B_1 \cup B_2)$,

(vi) $\forall B' \in \Phi'(\rho, B), B' \cap R^{-1}[B] = \emptyset$ ou $B' \subseteq R^{-1}[B]$,

(vii) Si $B = B_1 \cup B_2$ alors $\Phi'(\Phi'(\Phi'(\rho, B), B_1), B_2) = \Phi'(\Phi'(\rho, B_1), B_2)$.

Nous déduisons de la propriété (iv) que, quel que soit l'ordre dans lequel s'effectue la décomposition de la partition, le résultat est le même. Nous définissons l'opérateur Φ en tenant compte de cette propriété.

Définition 2.1.3 *de l'opérateur Φ*

Soit $\rho = \{B_i \mid 1 \leq i \leq n\}$ une partition de Q.

$\Phi(\rho) = \Phi'(\Phi'(\ldots \Phi'(\Phi'(\rho, B_1), B_2), \ldots), B_n)$.

Propriété *de l'opérateur* Φ

Soit B un sous-ensemble de Q et soient ρ, ρ_1 et ρ_2 trois partitions de Q. Les propriétés suivantes se déduisent des propriétés de l'opérateur Φ'.

(i) $\Phi(\rho)$ est une partition.

(ii) *monotonicité* : $\rho_1 \subseteq \rho_2 \Rightarrow \Phi(\rho_1) \subseteq \Phi(\rho_2)$,

(iii) $\Phi(\rho) \subseteq \rho$,

(iv) $\forall B' \in \Phi(\rho) \forall B \in \rho, B' \cap R^{-1}[B] \neq \emptyset$ ou $B' \subseteq R^{-1}[B]$.

Soient (Q, R) un système de transitions et ρ une relation d'équivalence sur Q. ρ est compatible avec R si et seulement si ρ est un point fixe de Φ. L'opérateur Φ est monotone sur l'ensemble des relations d'équivalence sur Q ; la relation d'équivalence la moins fine compatible avec ρ_0 est obtenue en calculant la limite de la suite :

$$\rho_{r+1} = \Phi(\rho_r).$$

Soient (Q, R) un système de transitions et ρ une relation d'équivalence sur Q ; nous présentons deux algorithmes pour calculer le plus grand point fixe de l'opérateur Φ.

La première solution montre le processus de raffinement de la relation d'équivalence ρ pour la rendre compatible avec la relation de transition R : étant donné une classe d'équivalence B, on produit un raffinement de la partition courante en décomposant chaque élément de la partition suivant l'inverse de la classe B par la relation R. La complexité de cet algorithme est la suivante : la taille mémoire est proportionnelle à *m*, le cardinal de la relation, et le temps d'exécution est proportionnel à *m n*, *n* étant le cardinal de l'ensemble des états.

Le deuxième algorithme que nous présentons est une extension du premier ; la taille mémoire est proportionnelle à *m* et le temps d'exécution est proportionnel à *m* log *n*.

1.3 Solution 1 : un algorithme en $O(m\,n)$

Nous définissons un opérateur Φ'', pour calculer la suite (ρ_r) en un temps proportionnel à *m n*. La propriété (vii) fait apparaître que si B_i décompose B_j en B_{j1} et B_{j2} au pas r, alors il n'est pas nécessaire de décomposer les éléments de ρ_r

suivant B_j : en effet, les éléments de ρ_{r+1} sont décomposés suivant B_{j1} et suivant B_{j2}.

Nous considérons deux sortes de classes : celles qui sont décomposées et celles qui servent à décomposer. Nous appelons *partitionneur* une classe qui sert à décomposer et nous notons W l'ensemble des partitionneurs. La décomposition d'une classe B' suivant un partitionneur B est *effective* si et seulement si

$$B' \not\subseteq R^{-1}[B] \text{ et } B' \cap R^{-1}[B] \neq \emptyset.$$

Nous notons *eff* (B, B') cette propriété. Dans la suite, si la décomposition de B' suivant B est effective, nous notons :

$$X_1 = B' \cap R^{-1}[B] \text{ et}$$
$$X_2 = B' - X_1.$$

L'opérateur Φ'' est défini de la manière suivante :

$(\rho', W') = \Phi''(\rho, W, B)$ où :

$\rho' = \Phi'(\rho, B)$ et

$W' = \{ B' \cap R^{-1}[B] \mid B' \in W\} \cup \{B' - R^{-1}[B] \mid B' \in W\} \cup$
$\{B' \cap R^{-1}[B] \mid B' \in \rho \text{ et } B' \notin W \text{ et } \mathit{eff}(B, B')\} \cup$
$\{B' - R^{-1}[B] \mid B' \in \rho \text{ et } B' \notin W \text{ et } \mathit{eff}(B, B')\} - \{B\}.$

Nous définissons une suite (ρ_r, W_r) :

$W_0 = \rho_0$,

...

$(\rho_{r+1}, W_{r+1}) = \Phi''(\rho_r, W_r, B)$, pour $B \in W_r$.

Le principe de l'algorithme de calcul de Φ'' est le suivant : calcul de la limite de la suite (ρ_r, W_r) par itérations. La condition d'arrêt est $W_r = \emptyset$, elle exprime que ρ_r vérifie la propriété de compatibilité avec R (proposition 2.1.1). Soit B le partitionneur ayant servi au raffinement de ρ_r. Remarquons que :

- la propriété suivante est vérifiée :

$$\forall B' \in \rho_{r+1}, B' \subseteq R^{-1}[B] \text{ ou } B' \cap R^{-1}[B] = \emptyset ;$$

de plus, tout raffinement ultérieur ρ_k, $k > r$, de la partition ρ_r, vérifie encore cette propriété. Cette dernière propriété permet de supprimer de W_r le partitionneur B ayant servi au raffinement de ρ_r : $B \notin W_{r+1}$.

- Soit $B' \in \rho_r$; nous pouvons distinguer trois cas :

1) si $B' \in W_r$ et si B' a été décomposée par B en

$$X_1 = B' \cap R^{-1}[B] \text{ et}$$
$$X_2 = B' - X_1,$$

alors $X_1, X_2 \in \rho_{r+1}$, et $X_1, X_2 \in W_{r+1}$;

2) si $B' \notin W_r$ et si B' a été décomposée par B en

$X_1 = B' \cap R^{-1}[B]$ et

$X_2 = B' - X_1$,

alors $X_1, X_2 \in \rho_{r+1}$, et $X_1, X_2 \in W_{r+1}$;

3) si B' n'a pas été décomposée par B, alors $B' \in \rho_{r+1}$; de plus, si $B' \in W_r$ alors $B' \in W_{r+1}$.

1.3.1 Algorithme 1

Soient, (Q, R) un système de transitions,

$\rho = \{B_i \mid B_i \subseteq Q, i \in [0, \ldots, k-1]\}$, la relation d'équivalence initiale sur Q,

$n = |Q|$,

$m = |R|$,

W l'ensemble des partitionneurs,

interpred l'ensemble des couples (B', X_1) pour lesquels la décomposition suivante est *effective* :

$X_1 = B' \cap R^{-1}[B]$

$X_2 = B' - X_1$.

1. $W = \rho$;
2. **tant-que** $W \neq \emptyset$
3. choisir (arbitrairement) et supprimer B dans W ;
4. calculer l'ensemble interpred =
   $\{(B', X_1) \mid \mathit{eff}(B, B')\ B' \in \rho$ et $X_1 = B' \cap R^{-1}[B]\}$
5. **pour tout** $(B', X_1) \in$ interpred
6. $X_2 \leftarrow B' - X_1$;
7. remplacer B' par X_1 et X_2 dans ρ ;
8. **si** $B' \in W$
9. **alors** remplacer B' par X_1 et X_2 dans W ;
10. **sinon** ajouter X_1 et X_2 dans W ;

fsi

fin_pour

fin_tant-que

fin

Exemple 2.3

Nous calculons par cet algorithme la plus grande bisimulation pour le système de transitions défini à la figure 2. Nous donnons les pas de calcul de l'exécution de l'algorithme 1 .

Tout d'abord, rappelons la définition de la relation inverse :

$R^{-1}[1] = \{3\}$,
$R^{-1}[2] = \{1\}$,
$R^{-1}[3] = \{1, 2\}$,
$R^{-1}[4] = \{1, 2\}$,
$R^{-1}[5] = \{2, 3, 4\}$,
$R^{-1}[6] = \{4, 5\}$.

Nous désignons respectivement par ρ_r et W_r la partition courante et l'ensemble des partitionneurs au pas r.

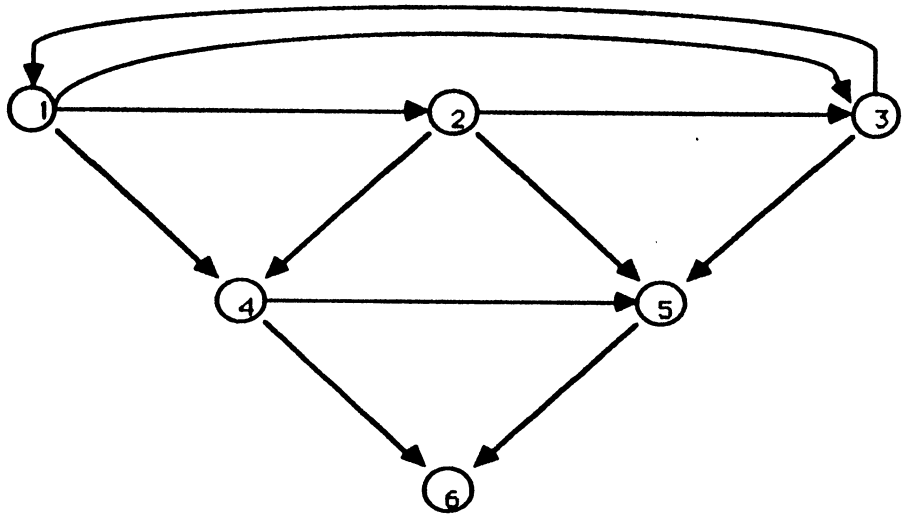


figure 2

Initialement

$B_0 = \{1, 2, 3, 4, 5, 6\}$
$\rho_0 = \{B_0\}$
$W_0 = \{B_0\}$

pas 1 :

-la décomposition est faite suivant B_0

$R^{-1}[B_0] = \{1, 2, 3, 4, 5\}$, B_0 est décomposée en B_1 et B_2

- $B_1 = \{1, 2, 3, 4, 5\}$
- $B_2 = \{6\}$

$\rho_1 = \{B_1, B_2\}$

$W_1 = \{B_1, B_2\}$

pas 2 :

-la décomposition est faite suivant B_1 ; ρ reste inchangée :

$R^{-1}[B_1] = \{1, 2, 3, 4, 5\}$

- $B_1 = \{1, 2, 3, 4, 5\}$
- $B_2 = \{6\}$

$\rho_2 = \{B_1, B_2\}$

$W_2 = \{B_2\}$

remarquons que la partition courante n'a pas été décomposée par B_1 et que B_1 est supprimée de W_2.

pas 3 :

-la décomposition est faite suivant B_2

$R^{-1}[B_2] = \{4, 5\}$, B_1 est décomposée en B_3 et B_4

- $B_3 = \{4, 5\}$
- $B_2 = \{6\}$
- $B_4 = \{1, 2, 3\}$

$\rho_3 = \{B_2, B_3, B_4\}$

$W_3 = \{B_3, B_4\}$

pas 4 :

-la décomposition est faite suivant B_3

$R^{-1}[B_3] = \{1, 2, 3, 4\}$, B_3 est décomposée en B_5 et B_6

- $B_2 = \{6\}$
- $B_4 = \{1, 2, 3\}$
- $B_5 = \{4\}$
- $B_6 = \{5\}$

$\rho_4 = \{B_2, B_4, B_5, B_6\}$

$W_4 = \{B_4, B_5, B_6\}$

pas 5 :

-la décomposition est faite suivant B_4 ; ρ reste inchangée :

$R^{-1}[B_4] = \{1, 2, 3\}$

$\rho_5 = \{B_2, B_4, B_5, B_6\}$

$W_5 = \{B_5, B_6\}$

pas 6 :

- la décomposition est faite suivant B_5

$R^{-1}[B_5] = \{1, 2\}$, B_4 est décomposée en B_7 et B_8

- $B_2 = \{6\}$
- $B_5 = \{4\}$
- $B_6 = \{5\}$
- $B_7 = \{1, 2\}$
- $B_8 = \{3\}$

$\rho_6 = \{B_2, B_5, B_6, B_7, B_8\}$

$W_6 = \{B_6, B_7, B_8\}$

pas 7 :

- la décomposition est faite suivant B_6

$R^{-1}[B_6] = \{2, 3, 4\}$, B_7 est décomposée en B_9 et B_{10}

- $B_2 = \{6\}$
- $B_5 = \{4\}$
- $B_6 = \{5\}$
- $B_8 = \{3\}$
- $B_9 = \{1\}$
- $B_{10} = \{2\}$

$\rho_6 = \{B_2, B_5, B_6, B_8, B_9, B_{10}\}$

$W_6 = \{B_8, B_9, B_{10}\}$ (B_7 est remplacée dans W par B_9 et B_{10}).

Aux pas 8, 9 et 10, la décomposition est faite respectivement suivant B_8, B_9 et B_{10} ; nous constatons que :

- ρ_6, ρ_7, ρ_8, ρ_9 et ρ_{10} sont identiques ;
- $W_7 = \{B_9, B_{10}\}$, $W_8 = \{B_{10}\}$ et $W_9 = \emptyset$.

Nous obtenons le même résultat quel que soit l'ordre dans lequel sont choisis les éléments de W.

Nous présentons la preuve de l'algorithme, une réalisation de l'algorithme et enfin l'évaluation de cette réalisation.

1.3.2 Correction

L'itération se termine : la condition de terminaison $W = \emptyset$ est vérifiée après un nombre fini d'exécutions du corps de l'itération. En effet, à chaque pas,

l'élément partitionneur est supprimé de W (ligne 3) et les classes rajoutées (ligne 10 ou 11), proviennent de la décomposition stricte d'une classe en deux sous-classes. Une classe a au plus *n* éléments, elle est au plus décomposée *n* fois.

Nous définissons la notion de *compatibilité* d'une relation d'équivalence avec la relation de transitions pour un sous-ensemble d'états X de Q :

Etant donné un système de transitions (Q, R), une relation d'équivalence ρ est *compatible* avec R *pour* X si et seulement si :

$\forall B \in \rho, B \subseteq R^{-1}(X)$ ou $B \cap R^{-1}(X) = \emptyset$

L'invariant de l'itération est la condition suivante (cf. proposition 2.1.5) : "$W \subseteq \rho$ et ρ est compatible pour chaque élément de ρ n'appartenant pas à W ".

La correction partielle de l'algorithme revient à montrer la propriété suivante : si l'algorithme se termine, la partition obtenue ρ_r est compatible avec la relation de transition pour chacun des éléments de ρ_r.
Cette correction partielle est établie par la proposition 2.1.6.

Proposition 2.1.4.

Soient (Q, R) un système de transitions et ρ une relation d'équivalence sur Q. Désignons par ρ_r la relation obtenue après r pas d'itération. Si ρ_r est compatible avec R pour X, alors ρ_{r+1} est compatible avec R pour X.

preuve :

La preuve découle de la décomposition de B' en deux sous-classes X_1 et X_2 aux lignes 4 et 6 de l'algorithme, en remarquant que toute classe de ρ_{r+1} est incluse dans une classe de ρ_r.
□

Proposition 2.1.5

Soit ρ_r et W_r désignant respectivement la partition obtenue et l'ensemble des partitionneurs obtenu après r pas d'itération.

$B \in \rho_r$ et $B \notin W_r \Rightarrow \forall B' \in \rho_r \,.\, B' \subseteq R^{-1}[B]$ ou $B' \cap R^{-1}[B] = \emptyset$

Preuve :

La preuve se fait par induction sur r, le nombre de pas de l'itération :
- soit r = 0, la propriété est vérifiée, tous les éléments de ρ_0 étant dans W_0,
- soit r > 0, remarquons que la propriété suivante est vérifiée :

$B \in \rho_r$ et $B \notin W_r \Rightarrow B \in \rho_{r-1}$.

En effet, supposons qu'il existe B vérifiant : $B \in \rho_r$ et $B \notin W_r$ et $B \notin \rho_{r-1}$; dans ce cas, B est issue de la décomposition d'une classe de ρ_{r-1} et $B \notin \rho_r$, ce qui contredit l'hypothèse $B \in \rho_r$.

Soit $B \in \rho_r$ et $B \notin W_r$ après le r-ième pas, nous avons deux cas possibles :

cas 1 $B \in W_{r-1}$ avant le r-ième pas, alors B est le partitionneur au r-ième pas, et

$\forall B' \in \rho_r . B' \subseteq R^{-1}[B]$ ou $B' \cap R^{-1}[B] = \emptyset$;

cas 2 $B \notin W_{r-1}$ avant le r-ième pas, alors par hypothèse d'induction,

$\forall B' \in \rho_{r-1} . B' \subseteq R^{-1}[B]$ ou $B' \cap R^{-1}[B] = \emptyset$;

de plus, chaque classe B" de ρ_r est incluse dans une classe B' de ρ_{r-1} ;

d'où :

$B' \subseteq R^{-1}[B] \Rightarrow B'' \subseteq R^{-1}[B]$ et

$B' \cap R^{-1}[B] = \emptyset \Rightarrow B'' \cap R^{-1}[B] = \emptyset$; on déduit :

$\forall B'' \in \rho_r . B'' \subseteq R^{-1}[B]$ ou $B'' \cap R^{-1}[B] = \emptyset$.

□

Proposition 2.1.6

$W_r = \emptyset \Rightarrow \forall B, B' \in \rho_r \; B' \subseteq R^{-1}[B]$ ou $B' \cap R^{-1}[B] = \emptyset$.

Preuve : La preuve est une conséquence de la proposition **2.1.5**

□

1.3.3 Réalisation de l'algorihme.

Nous présentons dans ce paragraphe les structures de données choisies pour respecter les contraintes que nous nous sommes fixées :

- complexité en temps de l'algorithme : $m\,n$,
- complexité en espace mémoire : m.

Ces contraintes nous ont conduits à choisir des structures de données de telle sorte que l'exécution d'un pas de l'itération soit proportionnel à $|R^{-1}[B]|$ où B est le partitionneur choisi (ligne 2).

1) Un état est représenté par un entier.
2) Chaque classe B_i est représentée par une *liste d'entiers*. Nous considérons les deux primitives suivantes de manipulation de liste : *élément* et *suivant*, qui étant donné une liste l, retournent respectivement le premier élément et le reste de la liste l. Par convention, Ø désigne la liste vide. Pour déterminer si la décomposition d'une classe est effective (ligne 4) en un temps constant, à chaque classe est associé son cardinal. Nous utilisons un tableau *bloc* [0 .. n - 1] pour représenter les classes : le i-ième élément désigne un couple (c', B') où c' est le cardinal de la classe B'. Ce tableau est initialisé avec la partition initiale.
3) Nous utilisons un tableau *prédécesseur* [0 .. n - 1] dont le p-ième élément désigne le prédécesseur de l'état p dans la classe de p. Ceci permet le transfert de chaque élément de B' vers X_2 (ligne 6), en un temps constant. Ce tableau est mis à jour pendant le calcul de *interpred*.
4) A chaque état nous associons la classe qui le contient : c'est l'indice dans le tableau *bloc* de la liste contenant cet état, ce qui permet de calculer *interpred* à la ligne 4 en un temps proportionnel à $|R^{-1}[B]|$; nous considérons un tableau *inclasse* [0 .. n - 1], dont le p-ième élément désigne l'indice de la classe contenant p.
5) Pour calculer *interpred* (ligne 4) en un temps proportionnel à $|R^{-1}[B]|$, nous utilisons les structures suivantes :

- *premier*, l'indice de la première classe élément de *interpred*,
- $interpred_0$ [0 .. n - 1], un tableau dont le j-ième élément est soit une valeur appartenant à l'intervalle [0, n - 1], soit une valeur n'appartenant pas à cet intervalle et notée par convention *null*. Si la valeur appartient à cet intervalle alors elle désigne un élément de *interpred*. Initialement,

premier = *null* et $interpred_0$ [j] = *null* pour tout j.

- $interpred_1$ [0 .. n - 1], un tableau dont le j-ième élément désigne un couple (c', X') où c' = |X'|. Initialement, $interpred_1$ [j] = (0, Ø) pour tout j. Soit B le partitionneur de la partition courante. A la fin du calcul de *interpred*, $interpred_1$ [j] = (c', X') où $X' = B' \cap R^{-1}[B]$.

Pour chaque état de R^{-1} [B], le temps nécessaire à la mise à jour de *interpred* est constant (cf. procédure maj_interpred, ci-dessous).
6) La substitution de la ligne 7 est réalisée comme suit :

- création d'une nouvelle classe d'indice *nb_classe* contenant $(|X_1|, X_1)$,
- *bloc* [j] = $(|X_2|, X_2)$ par construction.

L'entier *nb_classe* désigne le cardinal de la partition courante.

7) L'ensemble des partitionneurs *W* est représenté par une liste d'entiers. Un entier de cette liste désigne un indice du tableau *bloc*.

8) Pour effectuer le test de la ligne 8, nous considérons un tableau *in_W* [0 .. n - 1], dont le j-ième élément vaut 1 si j est dans *W*, 0 sinon. La substitution (ligne 9) consiste alors à insérer l'indice *nb_classe* dans *W*.

9) La relation de transition est représentée par son inverse : c'est un tableau R^{-1} [0 .. n - 1], dont le p-ième élément désigne R^{-1} [p].

Nous proposons une réalisation de l'algorithme 1. Nous conservons sa numérotation pour montrer comment ont été réalisées ses différentes actions.

Soient nb_classe i, j et premier des entiers,
bloc [0 .. n - 1] un tableau de couples (c', B') où c' est un entier, B une liste d'entiers et c' = |B'|,
W l'ensemble des partitionneurs,
$interpred_0$ [0 .. n - 1] un tableau d'entiers,
$interpred_1$ [0 .. n - 1] un tableau de couples (x , X) où x est un entier, X une liste d'entiers et x = |X|,
prédécesseur [0 .. n - 1] un tableau d'entiers,
inclasse [0 .. n - 1] un tableau d'entiers,
in_W [0 .. n - 1] un tableau d'entiers.

1. initialisations ;
2. **tant-que** W ≠ Ø
3. i ← élément (W) ; W ← suivant (W) ; (c, B) ← bloc [i] ;
4. calcul_interpred (B) ;
5. j ← $interpred_0$ [premier] ; $interpred_0$ [premier] ← *null* ;
   tant-que j ≠ premier
   décomposer_majw (j) ;
   j ← $interpred_0$ [j] ; $interpred_0$ [j] ← *null* ;
   fin_tant-que
   décomposer_majw (premier)
   fin_tant-que

fin

```
procédure calcul_interpred (B)
Soit,
pre la liste d'entiers désignant l'ensemble des prédécesseurs du partitionneur.
    tant-que B ≠ Ø
        p ← élément (B) ; B ← suivant (B) ; pre ← R-1 [p] ;
        tant-que pre ≠ Ø
            q ← élément (pre) ; pre ← suivant (pre) ;
            maj_interpred (q) ;
        fin_tant-que
    fin_tant-que
fin_procédure

procédure maj_interpred (q)
Soient,
    x, j, c des entiers,
    X une liste d'entiers.
    j ← inclasse [q] ;
    si interpred0 [j] = null
    alors si premier = null
          alors
                premier ← j ; interpred0 [j] ← j
            sinon
                interpred0 [j] ← interpred0 [premier] ;
                interpred0 [premier] ← j ;
            fsi
            X ← insérer (q, Ø) ; interpred1 [j] ← (1, X) ;
    sinon
            (c, X) ← interpred1 [j] ;
            X ← insérer (q, X) ; interpred1 [j] ← (c+1, X) ;
    fsi
fin_procédure
```

```
fonction insérer (q, X)
Soit e une liste d'entiers.
    e ← suivant (prédécesseur[q]) ;
    suivant (prédécesseur [q]) ← suivant (e) ;
    prédécesseur [élément (suivant (e))] ← prédécesseur [q] ;
    si X = Ø
    alors suivant (e) ← e ; prédécesseur [q] ← e ;
    sinon suivant (e) ← suivant (X) ;
        prédécesseur [élément (suivant (X))] ←e ;
        suivant (X) ← e ; prédécesseur [q] ← X
    fsi
    retourner (e)
fin_fonction

procédure décomposer_majw (j)
Soient, x, c' des entiers,
    X, B' une liste d'entiers.
    si c = x        -- la décomposition n'est pas effective
    alors bloc [j] ← interpred1 [j] ; interpred1 [j] ← (0, Ø)
    sinon (x, X) ← interpred1 [j] ;
6.      (c', B') ← bloc [j] ;
7.      nb_classse ← nb_classe + 1 ;
        bloc [nb_classe] ← interpred1 [j] ; interpred1 [j] ← (0, Ø) ;
        bloc [j] ← (c' - x, B') ; maj_classe (nb_classe) ;
8.      si in_W [j] = 1
9.      alors insérer_W (nb_classe) ;
10.     sinon insérer_W (nb_classe) ; insérer_W (j) ;
    fsi
fin_procédure

procédure insérer_W (j)
Soit l une liste d'entiers.
    l ← allouer_liste (1) ;      -- alloue un élément de type liste d'entiers
    élément (l) ← j ; suivant (l) ← W ; W ← l ;
fin_procédure
```

procédure maj_classe (nb_classe)
Soient B', B" deux listes d'entiers,
x' un entier.
(x', B') ← bloc [nb_classe] ; B" ← B' ;
inclasse [élément (B')] ← nb_classe ;
B' ← suivant (B') ;
tant-que B' ≠ B"
inclasse [élément (B')] ← nb_classe ;
B' ← suivant (B') ;
fin_tant-que
fin_procédure

1.3.4 Evaluation de l'algorithme

Evalution de la taille mémoire nécessaire pour exécuter l'algorithme 1.

La taille mémoire nécessaire à l'exécution de cet algorithme est proportionnelle à m. En effet, outre la représentation de la relation, nous utilisons les tableaux de dimension n : *classe*, *inclasse*, *in_W*, *bloc*, $interpred_0$ et $interpred1$. La somme des tailles des classes de la partition courante est égale à n. La liste W a au plus n éléments.

Evalution du temps nécessaire pour exécuter l'algorithme 1

Avant d'évaluer le temps nécessaire à l'exécution de l'algorithme nous montrons les deux propositions suivantes :

Proposition 2.1.7
Le corps de l'itération est exécuté en un temps proportionnel à $|R^{-1}[B]|$, où B est le partitionneur servant à décomposer la partition.

Principe de démonstration :
Nous évaluons le temps nécessaire à l'exécution de la procédure *calcul_interpred* et à l'exécution de l'itération (ligne 5).

1) *Temps d'exécution de la procédure calcul_interpred*

Le temps nécessaire à l'exécution de la procédure maj_interpred est constant quelle que soit la valeur du paramètre q. Ceci provient du fait que le temps nécessaire à l'exécution de la procédure insérer est constant : les instructions de cette procédure sont des instructions d'affectation (6 au plus) et une instruction de comparaison. L'évaluation du coût de la procédure calcul_interpred consiste à dénombrer les appels de la procédure maj_interpred et les affectations pour chaque élément de l'ensemble $R^{-1}[B]$.

2) *Temps d'exécution de l'itération (ligne 5)*

Soient B_j la suite de bloc [j], X_j la suite de $interpred_1$ [j] et interpred l'ensemble des indices j vérifiant $interpred_0$ [j] $\neq$ *null*. Le temps nécessaire à l'exécution de la procédure décomposer_majw est proportionnel au temps nécessaire à l'exécution de la procédure maj_classe. Celui-ci dépend de j, puisque l'exécution de la procédure maj_classe consiste, pour chaque élément q de X_j, à mettre à jour *inclasse* [q]. Le temps nécessaire à l'exécution de la procédure décomposer_majw est proportionnel à $|X_j|$. Le temps nécessaire à l'exécution de l'itération (ligne 5) est proportionnel à $\Sigma_{j \in interpred} [X_j|$. La famille d'ensemble $(X_j)_{j \in interpred}$ est une partition de l'ensemble $R^{-1}[B]$. On en déduit que $\Sigma_{j \in interpred} [X_j| = |R^{-1}[B]|$.

□

Proposition 2.1.8

Le temps total d'exécution de l'itération est proportionnel à

$\Sigma_{q \in Q} | R^{-1}[q]| * nb\_w\ (q)$

où : nb_w (q) est le nombre de fois où l'état q appartient à une classe insérée dans W.

Preuve :

- la partie initialisation est exécutée en un temps proportionnel à $n \leq m$.

- la partie **tant-que** (ligne 2) est exécutée en un temps proportionnel à $|R^{-1}[B]|$ (proposition **2.1.7**), où B est le partitionneur. On en déduit que toutes les opérations sont effectuées relativement aux éléments de $R^{-1}[q]$, chaque fois qu'un état q est élément de B.

□

Proposition 2.1.9

nb_w (p) ≤ n

Preuve

Soit p un état de Q. Initialement, p appartient à un élément de W dont la taille est inférieure ou égale à n. Lorsque cet élément de W est sélectionné pour raffiner la partition courante, il est extrait de W (ligne 3). Si la classe contenant p est insérée dans W (ligne 9 ou 10), sa taille est strictement inférieure à la taille de la classe qui contenait précédemment p. Il y a au plus n classes contenant p et insérées dans W.

□

Des propositions **2.1.7.** et **2.1.8.**, et du fait que $\Sigma_{q\in Q} | R^{-1}[q]| = m$, on déduit que le temps total d'exécution de l'algorithme 1 est proportionnel à *m n.*

1.4 Solution 2 : un algorithme en *O* (*m* log *n*)

L'objet de ce paragraphe est de modifier l'opérateur Φ" et l'algorithme précédent de telle sorte que le temps d'exécution soit de l'ordre de *m* log *n*. Pour cela, il suffit de majorer la fonction nb_w (p) définie à la proposition 2.1.8 par log *n*, pour tout état p. L'idée est de procéder de manière analogue au cas où R est une fonction [AHU74] : après décomposition d'une classe n'appartenant pas à W en deux sous-classes, il suffit de rajouter dans W (ligne 10, algorithme 1) la plus petite des deux sous-classes. Malheureusement, lorsque R est une relation quelconque, on ne peut pas procéder de la sorte, comme le montre l'exemple 2.4 ci-dessous. Nous montrons qu'il n'est pas possible de remplacer dans l'algorithme 1, la ligne 10 par :
"**sinon** soit X la plus petite des classes X_1 et X_2, ajouter X dans W".

Exemple 2.4

Reprenons le système de transitions de l'exemple 2.3, en appliquant l'algorithme 1 modifié : lorsqu'une classe n'appartenant pas à W est décomposée, on ajoute la plus petite des deux sous-classes dans W. Nous obtenons :

<u>pas 1 :</u>

-la décomposition est faite suivant B_0

B_0 est décomposée en B_1 et B_2

- $B_1 = \{1, 2, 3, 4, 5\}$
- $B_2 = \{6\}$

$\rho_1 = \{B_1, B_2\}$

$W_1 = \{B_2\}$

pas 2 :

-la décomposition est faite suivant B_2

B_1 est décomposée en B_3 et B_4

- $B_3 = \{4, 5\}$
- $B_4 = \{1,2,3\}$

$\rho_2 = \{B_2, B_3, B_4\}$

$W_2 = \{B_3\}$

pas 3 :

-la décomposition est faite suivant B_3

B_3 est décomposée en B_5 et B_6

- $B_5 = \{4\}$
- $B_6 = \{5\}$

$\rho_3 = \{B_2, B_4, B_5, B_6\}$

$W_3 = \{B_5\}$ ou $W_3 = \{B_6\}$

Nous considérons en parallèle les deux "valeurs" de W_3.

pas 4 : ($W_3 = \{B_5\}$)

-la décomposition est faite suivant B_5

B_4 est décomposée en B_7 et B_8

- $B_7 = \{1, 2\}$
- $B_8 = \{3\}$

$\rho_4 = \{B_2, B_5, B_6, B_7, B_8\}$

$W_4 = \{B_8\}$

pas 5 :

-la décomposition est faite suivant B_8

ρ_4 est inchangée

$\rho_5 = \{B_2, B_5, B_6, B_7, B_8\}$

$W_5 = \emptyset$

pas 4 : ($W_3 = \{B_6\}$)

-la décomposition est faite suivant B_6

B_4 est décomposée en B_7 et B_8

- $B_7 = \{2, 3\}$
- $B_8 = \{1\}$

$\rho_4 = \{B_2, B_5, B_6, B_7, B_8\}$

$W_4 = \{B_8\}$

pas 5 :

-la décomposition est faite suivant B_8

B_7 est décomposée en B_9 et B_{10}

- $B_9 = \{2\}$
- $B_{10} = \{3\}$

$\rho_5 = \{B_2, B_5, B_6, B_8, B_9, B_{10}\}$

$W_5 = \{B_9\}$

pas 6:

-la décomposition est faite suivant B_9

ρ_5 est inchangée
$\rho_6 = \{B_2, B_5, B_6, B_8, B_9, B_{10}\}$
$W_6 = \emptyset$

Nous remarquons sur cet exemple que, suivant l'élément ajouté dans W_3, le résultat de l'algorithme n'est pas le même. De plus, en insérant B_5 dans W_3, la partition obtenue n'est pas compatible avec R : ($B_7 = \{1, 2\}$)
$R[1] \cap B_7 = \{2\}$ et $R[2] \cap B_7 = \emptyset$.

Paige et Tarjan [PT86] ont considéré deux sortes d'éléments dans W : les partitionneurs représentant une classe, appelés *blocs simples*, et les partitionneurs représentant une union de classes, appelés *blocs arbres*. Lorsqu'une classe B n'appartenant pas à W est décomposée en B_1 et B_2, on rajoute dans W l'arbre binaire ayant B_1 et B_2 pour fils. Lorsqu'un arbre est sélectionné dans W pour raffiner la partition courante, la décomposition de chaque classe est réalisée par rapport au plus petit des fils du partitionneur.

Nous utilisons les notations et les définitions du paragraphe précédent :
- (Q, R) désigne un système de transitions,
- $\rho = \{B_j \mid B_j \subseteq Q, j \in [0, \ldots, k-1]\}$ désigne une relation d'équivalence sur Q,
- nous considérons la suite ρ_r, W_r définie au paragraphe précédent,

$(\rho_0, W_0) = (\rho, \rho)$

...

$(\rho_{r+1}, W_{r+1}) = \Phi''(\rho_r, W_r, B_i)$

B_i désigne le partitionneur au pas r : $(\rho_{r+1}, W_{r+1}) = \Phi''(\rho_r, W_r, B_i)$
ρ_r désigne la partition courante au pas r,
W_r désigne l'ensemble courant des partitionneurs au pas r,
$n = |Q|$,
$m = |R|$.

Proposition 2.1.10

Soit $B_i \in \rho_r$ le partitionneur au pas r ; si la classe B_i est elle-même décomposée en B_{i1} et B_{i2}, toutes les autres classes B_j de la partition ρ_{r+1} sont décomposables en :

$X_1 = B_j \cap (R^{-1}[B_{i1}] - R^{-1}[B_{i2}])$
$X_2 = B_j \cap (R^{-1}[B_{i2}] - R^{-1}[B_{i1}])$
$X_3 = B_j \cap R^{-1}[B_{i1}] \cap R^{-1}[B_{i2}]$

Preuve :

Ceci provient du fait qu'un partitionneur ayant servi au raffinement de la partition vérifie la propriété suivante :

$\forall B_j \in \rho_{r+1}\ B_j \subseteq R^{-1}[B_i]$ ou $B_j \cap R^{-1}[B_i] = \emptyset$;

la décomposition d'un élément B_j de ρ_{r+1} en X_1, X_2 et X_3 est effective uniquement dans le cas où $B_j \subseteq R^{-1}[B_i]$.

□

Remarquons que, lorsque R est une fonction, $X_3 = \emptyset$ et la décomposition d'une classe B_j peut être réalisée uniquement en fonction de la plus petite des deux sous-classes B_{i1} ou B_{i2} [AHU74]. En effet,

$X_1 = B_j \cap R^{-1}[B_{i1}] = B_j - R^{-1}[B_{i2}]$

$X_2 = B_j - R^{-1}[B_{i1}] = B_j \cap R^{-1}[B_{i2}]$

Dans ce cas, la suite (B_i) des partitionneurs contenant un état p, utilisés pour raffiner la partition, vérifie la propriété suivante :

$$|B_{i+1}| \leq \frac{|B_i|}{2}$$

Ce qui permet de justifier, dans le cas d'une fonction, une complexité de l'ordre de $m \log n$.

L'idée de Paige et Tarjan consiste à considérer deux sortes de partitionneurs appelés *blocs simples* (i.e. classes d'équivalence) et *blocs arbres* (i.e. union de classes d'équivalence).

Pour représenter la décomposition de la classe B' en X_1 et X_2, à l'aide d'un arbre t_i (figure 3), on associe à la racine une fonction qui, pour chaque état p, retourne le nombre de ses successeurs dans B'.

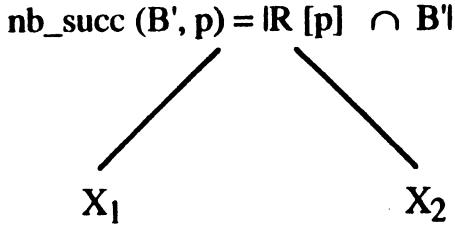


figure 3

Si la feuille X_1 (resp. X_2) est décomposée, elle est remplacée dans l'arbre par un

nœud n_1 (resp. n_2) représentant cette décomposition (figure 3-bis).

$$nb\_succ\ (B', p) = |R\ [p] \cap B'|,$$

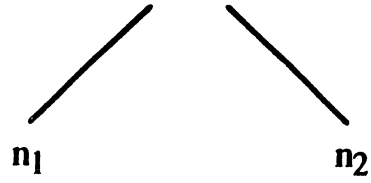


figure 3-bis

L'information associée à un nœud qui n'est pas la racine est le cardinal de l'union des ses feuilles.

La fonction nb_succ est étendue aux nœuds de la manière suivante :

si $B_{i1}, \ldots, B_{ik}$ désignent les feuilles d'un nœud n_i, alors

$nb\_succ\ (n_i, p) = nb\_succ\ (B_{i1}, p) + \ldots + nb\_succ\ (B_{ik}, p)$.

Lorsque un bloc simple B_i est sélectionné pour raffiner la partition, la fonction $\lambda p.nb\_succ\ (B_i, p)$ peut être calculée en un temps proportionnel à $|R^{-1}\ [B_i]|$ (cf. 1.4.6 évaluation de l'algorithme).

Lorsque un bloc arbre t_i est sélectionné pour raffiner la partition, les fonctions $\lambda p.nb\_succ\ (n_{i1}, p)$ et $\lambda p.nb\_succ\ (n_{i2}, p)$ peuvent être calculées en un temps proportionnel au plus petit des deux sous-arbres n_{i1} ou n_{i2}. Un arbre t est plus petit qu'un arbre t' si le cardinal de l'union des feuilles de t est inférieur au cardinal de l'union des feuilles de t'. Par conséquent, la décomposition d'une classe B_j de ρ_{r+1} telle qu'elle est définie dans la proposition **2.1.10**, peut être réalisée en un temps proportionnel au plus petit des deux sous-arbres en appliquant les règles suivantes :

$$\frac{nb\_succ\ (n_{i1}, p) > 0,\ nb\_succ\ (n_{i2}, p) > 0}{X_3 = X_3 \cup \{p\}}$$

$$\frac{nb\_succ\ (n_{i1}, p) > 0,\ nb\_succ\ (n_{i2}, p) = 0}{X_1 = X_1 \cup \{p\}}$$

$$\frac{nb\_succ\ (n_{i1}, p) = 0,\ nb\_succ\ (n_{i2}, p) > 0}{X_2 = X_2 \cup \{p\}}$$

Nous décrivons, l'opérateur Φ'' en considérant deux cas :

cas 1 : le partitionneur est un bloc simple B ; $(\rho', W') = \Phi''(\rho, W, B)$ où :

$\rho' = \Phi'(\rho, B)$ et

W' est construit de la manière suivante : soit $B' \in \rho$.

1) Si $B' \in W$ et B' n'est pas décomposée, alors $B' \in W'$.

2) Si $B' \in W$ et B' est décomposée en X_1 et X_2, alors $X_1, X_2 \in W'$.

3) Si $B' \notin W$ et B' est décomposée et B' n'est pas une feuille, alors l'arbre représentant cette décomposition (figure 3) appartient à W'.

4) Si $n' \in W$, alors $n'' \in W'$ où n'' est obtenu à partir de n' en remplaçant les feuilles de n', pour lesquelles la décomposition suivant B est effective, par les nœuds représentant cette décomposition (figure 3-bis).

cas 2 : le partitionneur est un bloc arbre n ; $(\rho', W') = \Phi''(\rho, W, n)$ où :

ρ' est construite de la manière suivante :

1) si $B' \in \rho$ et B' n'est pas décomposée, alors $B' \in \rho'$.

2) Si $B' \in \rho$ et B' est décomposée en X_1, X_2 et X_3 (proposition 2.1.10), alors X_1, X_2 et $X_3 \in \rho'$.

W' est construit de la manière suivante :

1) Soient n_1 et n_2 les fils gauche et droit de n. Si n_i n'est pas une feuille, alors $n_i \in W'$, pour i = 1, 2.

2) Si $B' \in W$ et B' n'est pas décomposée, alors $B' \in W'$.

3) Si $B' \in W$ et B' est décomposée en X_1, X_2 et X_3 (proposition 2.1.10), alors X_1, X_2 et $X_3 \in W'$.

4) Si $B' \notin W$ et B' est décomposée en X_1, X_2 et X_3 (proposition 2.1.10) et B' n'est pas une feuille, alors l'arbre représentant cette décomposition appartient à W' (figure 3-ter ou figure 3 si un $X_i = \emptyset$).

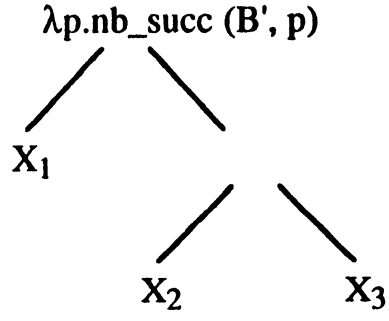


figure 3-ter

5) si $n' \in W$, alors $n'' \in W'$ où n'' est obtenu à partir de n' en remplaçant les feuilles de n', décomposées en X_1, X_2 et X_3 (proposition 2.1.10), par

les nœuds représentant cette décomposition (figure 3-ter, sans l'information associée à la racine, ou figure 3-bis, si un $X_i = \emptyset$).

Nous présentons l'algorithme qui calcule la suite (ρ_r, W_r) pour les systèmes de transitions, son extension aux systèmes de transitions étiquetées, la correction de l'algorithme étendu, sa réalisation et enfin son évaluation.
La fonction *nb_succ* est représentée par son graphe et nous considérons, pour un partitionneur B, l'ensemble *incluspred* des classes B' telles que $B' \subseteq R^{-1}[B]$.

1.4.1 Algorithme 2

Soient, (Q, R) un système de transitions,
$\rho = \{B_i \mid B_i \subseteq Q, i \in [0, \ldots, k-1]\}$, la relation d'équivalence initiale sur Q,
$n = |Q|$,
$m = |R|$,
W l'ensemble des partitionneurs,
x un élément de W,
interpred l'ensemble des couples (B', X_1) pour lesquels la décomposition suivante est *effective* :
$X_1 = B' \cap R^{-1}[B]$,
$X_2 = B' - X_1$,
nb_succ l'ensemble des triplets (p, B, $|B \cap R[p]|$), où B désigne soit une classe, soit une union de classes,
incluspred l'ensemble des classes $B_j \subseteq R^{-1}[B_i]$,
pour un *bloc arbre* t_i fils_gauche(t_i) (resp. fils_droit (t_i)) l'accès au fils gauche (resp. au fils droit),
(nous prenons la convention suivante : $|$ fils_gauche (t_i) $| \leq |$ fils_droit (t_i) $|$).
$W = \rho$
tant-que $W \neq \emptyset$
choisir et supprimer x de W
si x est un *bloc simple* B_i
alors raffiner_simple (B_i)
sinon raffiner_arbre (t_i) -- x est un *bloc arbre* t_i
fin_tant-que
fin

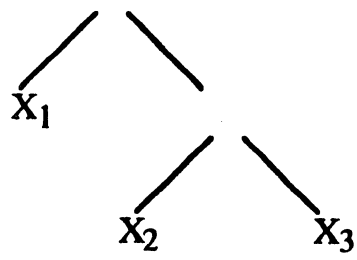
```
procédure raffiner_simple (Bi)
    pour tout p ∈ Bi
        pour tout q∈ R -1 [Bi]
            maj_interpred (q) ; maj_nb_succ ( Bi, q) ;
        fin_pour
    fin_pour
    pour tout (B', X1) ∈ interpred
        maj_bloc (B', X1)
    fin_pour
fin_procédure

procédure maj_interpred (q)
    soit B' la classe contenant q supprimer q de B' ;
    si il existe (B', X1) ∈ interpred
    alors insérer q dans X1
    sinon ajouter (B', {q}) dans interpred
    fsi
fin_procédure

procédure maj_nb_succ (Bi, q)
    si il existe (p, Bi, n) ∈ nb_succ
    alors remplacer n par n+1
    sinon ajouter (p, Bi, 1) dans nb_succ
    fsi
fin_procédure

procédure maj_bloc (B', X1)
    si | X1| ≠ |B'|
    alors X2 = B' - X1 ;
        remplacer B' par X1 et X2 dans ρ ;
        si B' ∈ W
        alors remplacer B' par X1, X2 dans W ;
        sinon si B' ∉ W et B' n'est pas une feuille
        alors créer un arbre t' ayant pour fils_gauche X1 et fils_droit X2 ;
                -- on suppose | X1 | ≤ |X2 |, sinon on intervertit les rôles
```

-- de X_1 et X_2
ajouter t' dans W ;
sinon remplacer la feuille B' par un nœud ayant pour fils_gauche X_1 et fils_droit X_2 ;
-- on suppose $|X_1| \leq |X_2|$, sinon on intervertit les rôles
-- de X_1 et X_2
fsi
fsi
fin_procédure

procédure raffiner_arbre (t_i)
calcul_nb_succ (t_i, fils_gauche (t_i)) -- on en déduit , nb_succ (fils_droit (t_i))
-- incluspred est mis à jour
si fils_gauche (t_i) n'est pas une feuille, **alors** l'insérer dans W ; **fsi**
si fils_droit (t_i) n'est pas une feuille, **alors** l'insérer dans W ; **fsi**

pour tout $B_j \in$ incluspred
décomposer_majw (B_j, fils_gauche (t_i), fils_droit (t_i))
fin_pour
fin_procédure

procédure décomposer_majw (B_j, n_{i1}, n_{i2})
calculer X_1, X_2, X_3 par le procédé de la proposition 2.1.10 ;
remplacer B_j par X_1, X_2 et X_3 dans ρ ;
mettre à jour W de façon analogue à la procédure raffinement_simple, en remarquant que si $X_1 \neq \emptyset$, $X_2 \neq \emptyset$ et $X_3 \neq \emptyset$, les arbres ou les nœuds sont de la forme :



fin_procédure

1.4.2. Extension aux systèmes de transitions étiquetées

Soit $S = (Q, A, T, q_0)$ un système de transitions étiquetées. La relation de transition T est considérée ici comme une union de relations de transition indicées par les éléments de A. Le raffinement de la partition courante est réalisé suivant un partitionneur donné et suivant une étiquette. Nous modifions la définition **2.1.2** de l'opérateur Φ' en introduisant un opérateur $\Phi^{(3)}$.

Définition 2.1.4 *de l'opérateur* $\Phi^{(3)}$

Soient ρ une relation d'équivalence sur Q et B un sous-ensemble de Q.

$\Phi^{(3)}(\rho, B, a) = \{ B' \cap T_a^{-1}[B] \mid B' \in \rho \} \cup \{B' - T_a^{-1}[B] \mid B' \in \rho\}$.

L'opérateur $\Phi^{(3)}$ vérifie les propriétés suivantes :

Propriété *de l'opérateur* $\Phi^{(3)}$

Soient a, a_1 et a_2 des éléments de A, soit B un sous-ensemble de Q et soient ρ, ρ_1 et ρ_2 trois partitions de Q.

(i) $\Phi^{(3)}(\rho, B, a)$ est une partition.

(ii) *monotonicité* : $\rho_1 \subseteq \rho_2 \Rightarrow \Phi^{(3)}(\rho_1, B, a) \subseteq \Phi^{(3)}(\rho_2, B, a)$,

(iii) $\Phi^{(3)}(\rho, B, a) \subseteq \rho$,

(iv) $\Phi^{(3)}(\Phi^{(3)}(\rho, B, a_1), B, a_2) = \Phi^{(3)}(\Phi^{(3)}(\rho, B, a_2), B, a_1)$,

(v) $\forall B' \in \Phi^{(3)}(\rho, B, a)$, $B' \cap T_a^{-1}[B] = \emptyset$ ou $B' \subseteq T_a^{-1}[B]$.

L'opérateur Φ' devient, si $A = \{a_1, \ldots a_n\}$:

$\Phi'(\rho, B) = \Phi^{(3)}(\Phi^{(3)}(\ldots \Phi^{(3)}(\Phi^{(3)}(\rho, B, a_1), B, a_2), \ldots), B, a_n)$.

Les propriétés (i)-(v) et la propriété (vii) de la définition **2.1.2** sont conservées. La propriété (vi) devient :

(vi) $\forall a \in A \forall B' \in \Phi'(\rho, B)$, $B' \cap T_a^{-1}[B] = \emptyset$ ou $B' \subseteq T_a^{-1}[B]$.

Les propriétés (i)-(iii) de la définition **2.1.3** sont conservées. La propriété (iv) devient :

(iv) $\forall a \in A \forall B' \in \Phi(\rho) \forall B \in \rho$, $B' \cap T_a^{-1}[B] = \emptyset$ ou $B' \subseteq T_a^{-1}[B]$.

L'algorithme 2 est modifié de la manière suivante :

- (Q, A, T, q_0) désigne un système de transitions étiquetées,

- a désigne un élément de A,
- $T_a^{-1}[q]$ désigne l'ensemble $\{(p, a) \mid (p, a, q) \in T\}$.
- incluspred désigne les couples (B_j, a) tels que $B_j \subseteq T_a^{-1}[B_i]$
- nb_succ désigne les triplets $(a, p, B_i, |B_i \cap T_a[p]|)$.

Les procédures raffiner_simple, raffiner_arbre, maj_nb_succ et décomposer_majw sont modifiées de la manière suivante :

```
procédure raffiner_simple (Bi)
    Ci ← Bi                    -- sauvegarde de Bi
    pour tout a ∈ A
        pour tout p ∈ Ci
            pour q∈ Ta -1 [Ci]
                maj_interpred (q) ;
                maj_nb_succ (Bi, a, q) ;
            fin_pour
        fin_pour
        pour (B', X1) ∈ interpred
            maj_bloc (B', X1)
        fin_pour
    fin_pour
fin_procédure

procédure raffiner_arbre (ti)
    calcul_nb_succ (ti, fils_gauche (ti))  -- on en déduit nb_succ (fils_droit (i))
                                           -- mise à jour de incluspred
    si fils_gauche (ti) n'est pas une feuille alors l'insérer dans W fsi
    si fils_droit (ti) n'est pas une feuille alors l'insérer dans W fsi
    pour tout a ∈ A
        pour (Bj, a) ∈ incluspred
            décomposer_majw (a, Bj, fils_gauche (ti), fils_droit (ti))
        fin_pour
    fin_pour
fin
```

procédure maj_nb_succ (B_i, a, q)
    si il existe (a, p, B_i, n) $\in$ nb_succ
    alors remplacer n par n+1
    sinon ajouter (a,p, B_i, 1) dans nb_succ
    fsi
fin_procédure

procédure décomposer_majw (a, B_j, n_{i1}, n_{i2})
    calculer X_1, X_2, X_3 par le procédé de la proposition 2.1.10 en remplaçant R par T_a :
        $X_1 = B_j \cap (T_a^{-1}[n_{i1}] - T_a^{-1}[n_{i2}])$
        $X_2 = B_j \cap (T_a^{-1}[n_{i2}] - T_a^{-1}[n_{i1}])$
        $X_3 = B_j \cap T_a^{-1}[n_{i1}] \cap T_a^{-1}[n_{i2}]$;
    remplacer B_j par X_1, X_2 et X_3 dans ρ ;
    mettre à jour W de la même manière que dans l'algorithme 2
fin_procédure

1.4.3. Exemple

Considérons le système de transitions étiquetées décrit à la figure 4 suivante:

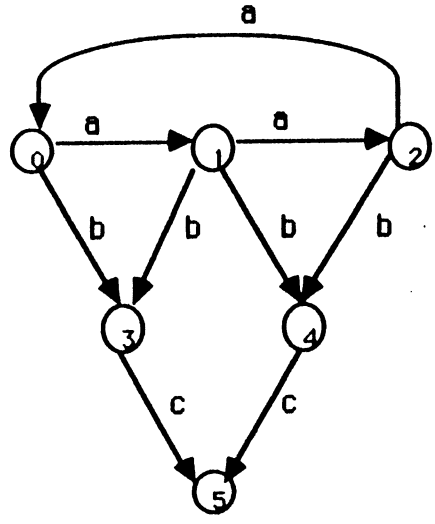


figure 4

Initialement, tous les états sont dans la même classe. Nous allons montrer que la partition la moins fine compatible avec cette relation de transitions étiquetées est {{0, 1, 2}, {3, 4}, {5}}. Nous montrons les pas de calcul du programme faisant apparaître le résultat de la procédure raffiner_simple ou raffiner_arbre. Les classes sont représentées par B_i au lieu de classe [i]. Soit t un *bloc_arbre* et *a* une étiquette. Nous représentons, à la racine de l'arbre, *nb_succ* par une liste de couples (p, k) où k désigne le nombre de successeurs de l'état p par T_a.

Initialement,

$B_0 = \{0,1,2,3,4,5\}$

$\rho_0 = \{B_0\}$

$W_0 = \{B_0\}$

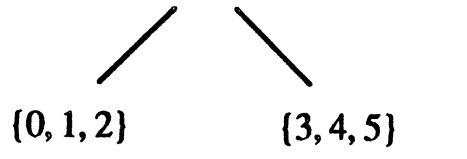
- raffinement_simple (B_0)
  - étiquette a :

    $B_1 = \{0, 1, 2\}$

    $B_2 = \{3, 4, 5\}$

    $\rho_1 = \{ B_1, B_2\}$

    $W_1 = \{t_0\}$ où



  - étiquette b :

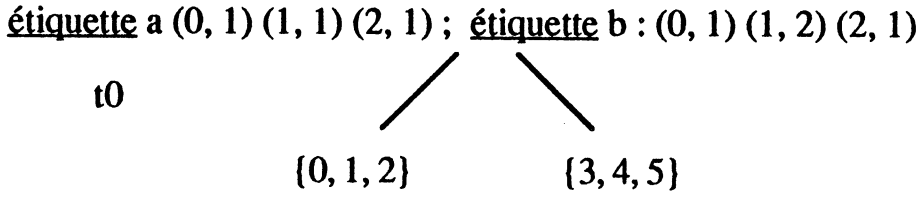
    $B_1 = \{0, 1, 2\}$

    $B_3 = \{3, 4\}$

    $B_4 = \{5\}$

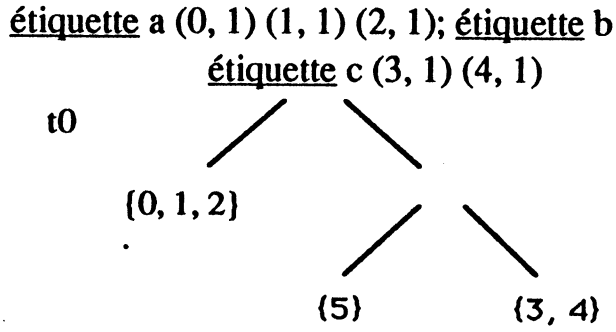
    $\rho_1 = \{ B_1, B_3, B_4\}$

    $W_1 = \{t_0\}$ où t_0 est modifié de la manière suivante :



- étiquette c :

W_1 et ρ_1 ne sont pas modifiés. t_0 est modifié de la manière suivante :



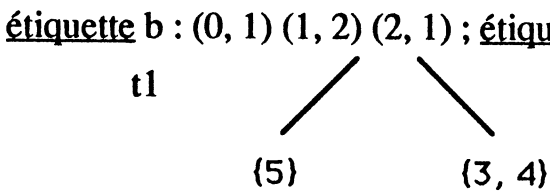
raffinement_arbre (t_0)

Le fils gauche de t_0 est B_1, le fils droit est t_1.

λp . nb_succ (B_1, p) est définie de la manière suivante :

étiquette a (0, 1) (1, 1) (2, 1); étiquette b Ø ; étiquette c Ø. On en déduit :

$W_2 = \{t_1\}$ où



la partition n'est pas modifiée ;

$\rho_2 = \{ B_1, B_3, B_4\}$

$W_2 = \{t_1\}$ où

raffinement_arbre (t_1)

Le fils gauche de t_1 est B_4, le fils droit est t_1.

λp . nb_succ (B_4, p) est définie de la manière suivante :

étiquette a Ø; étiquette b Ø ; étiquette c (3, 1) (4, 1). On en déduit :
λp . nb_succ (B_3, p)
étiquette a Ø; étiquette b (0, 1) (1, 2) (2, 1) ; étiquette c Ø.
$W_3 = \emptyset$
la partition n'est pas modifiée ; donc W_3 reste vide.

1.4.4. Correction

Soit S = (Q, A, T, q_0) un système de transitions étiquetées. Désignons par W_r l'ensemble des *blocs simples* ou *blocs arbres* après r exécutions de la procédure raffiner simple ou raffiner_arbre. Désignons par ρ_r la relation obtenue après r exécutions de la procédure raffiner_simple ou raffiner_arbre. Nous modifions la définition de la compatibilité d'une relation d'équivalence avec la relation de transition pour un sous-ensemble d'états X :

Une relation d'équivalence ρ sur Q est compatible avec R pour $X \subseteq Q$ si et seulement si :

$\forall a \in A \; \forall B \in \rho \Rightarrow B \subseteq T_a^{-1}[X]$ ou $B \cap T_a^{-1}[X] = \emptyset$

L'argument de terminaison est identique à l'argument de terminaison de l'algorithme 1.

L'invariant de l'itération de l'algorithme 2 est la condition suivante :
"ρ est compatible avec T pour les classes B vérifiant :

$B \in \rho$ et $B \notin W$ et B n'est pas une feuille".

La correction partielle :

$W_r = \emptyset \Rightarrow \forall a \in A \forall B' \in \rho_r (B' \subseteq T_a^{-1}[B]$ ou $B' \cap T_a^{-1}[B] = \emptyset)$,
est une conséquence de la proposition 2.1.14.

Proposition 2.1.11

Si ρ_r est compatible avec T pour X, alors ρ_{r+1} est compatible avec T pour X.

preuve :

La preuve découle de la décomposition d'une classe B' en deux sous-classes X_1 et X_2 (procédure maj_bloc ou procédure décomposer_majw) ou trois sous-classes X_1, X_2 et X_3 (procédure décomposer_majw) en remarquant que

toute classe de ρ_r est incluse dans une classe de ρ_{r+1}.
□

Proposition 2.1.12

Soit t_i le partitionneur au r-ième pas ; après l'exécution de la procédure décomposer_majw, la propriété suivante est vérifiée :
$(\forall B' \in \rho_{r+1}.\ B' \subseteq T_a^{-1}[\text{fils\_gauche}(t_i)]$ ou $B' \cap T_a^{-1}[\text{fils\_gauche}(t_i)] = \emptyset)$ et
$(\forall B' \in \rho_{r+1}.\ B' \subseteq T_a^{-1}[\text{fils\_droit}(t_i)]$ ou $B' \cap T_a^{-1}[\text{fils\_droit}(t_i)] = \emptyset)$

Preuve

Par construction, la fonction nb_succ permet de réaliser la décomposition précédente pour chaque classe.
□

Proposition 2.1.13

Soit t_i le partitionneur au r-ième pas ; après exécution de la procédure raffiner arbre, la propriété de la proposition précédente est vérifiée pour tout a de A :
$\forall a \in A$
$(\forall B' \in \rho_{r+1}.\ B' \subseteq T_a^{-1}[\text{fils\_gauche}(t_i)]$ ou $B' \cap T_a^{-1}[\text{fils\_gauche}(t_i)] = \emptyset)$ et
$(\forall B' \in \rho_{r+1}.\ B' \subseteq T_a^{-1}[\text{fils\_droit}(t_i)]$ ou $B' \cap T_a^{-1}[\text{fils\_droit}(t_i)] = \emptyset)$.
□

Proposition 2.1.14

$B \in \rho_r$ et $B \notin W_r$ et B n'est pas une feuille $\Rightarrow$
$\forall a \in A \forall B' \in \rho_r.\ B' \subseteq T_a^{-1}[B]$ ou $B' \cap T_a^{-1}[B] = \emptyset$.

Preuve :
La preuve se fait par induction sur r, le nombre de pas de l'itération.
- soit r = 0, la propriété est vérifiée, tous les éléments de ρ_0 étant dans W_0,
- soit r > 0, remarquons que,

$B \in \rho_r$ et $B \notin W_r$ et B n'est pas une feuille $\Rightarrow B \in \rho_{r-1}$;
Par l'absurde, supposons que $B \notin \rho_{r-1}$, alors B provient de la décomposition d'un élément B" de ρ_{r-1}. Trois cas sont possibles :

cas 1 $B'' \in W_{r-1}$,

$B \in W_r$ (cf procédure maj_bloc), ce qui contredit l'hypothèse $B \notin W_r$;

cas 2 : $B'' \notin W_r$ et B" est une feuille,

B est une feuille du nœud remplaçant la feuille B" (cf procédure maj_bloc) ce qui contredit l'hypothèse que B n'est pas une feuille ;

cas 3 : $B \notin W_r$ et B" n'est pas une feuille,

B est une feuille de l'arbre représentant la décomposition de B" (cf procédure maj_bloc) ce qui contredit l'hypothèse que B n'est pas une feuille.

Soit B tel que $B \in \rho_r$ et $B \notin W_r$ et B n'est pas une feuille ; montrons que la propriété suivante est vérifiée :

$B \in \rho_r$ et $B \notin W_r$ et B n'est pas une feuille $\Rightarrow$

$\forall a \forall B' \in \rho_r.\ B' \subseteq T_a^{-1}[B]$ ou $B' \cap T_a^{-1}[B] = \emptyset$;

trois cas sont possibles :

cas 1 : $B \in W_{r-1}$,

B est le partitionneur au r-ième pas, puisque $B \notin W_r$;

B n'est pas décomposé, puisque $B \in \rho_r$; la propriété est vérifiée ;

cas 2 : $B \notin W_{r-1}$ et B est une feuille avant le r-ième pas,

soit t_i le bloc arbre dont B est une feuille ; puisque B n'est pas une feuille après le r-ième pas, B est le fil_gauche ou le fils_droit de t_i ; la propriété est vérifiée, d'après les propositions 2.1.12 et 2.1.13;

cas 3 : $B \notin W_{r-1}$ et B n'est pas une feuille avant le r-ième pas,

par hypothèse d'induction,

$\forall a \in A \forall B' \in \rho_{r-1}.\ B' \subseteq T_a^{-1}[B]$ ou $B' \cap T_a^{-1}[B] = \emptyset$;

puisque toute classe de ρ_r est incluse dans une classe de ρ_{r-1}, la propriété précédente est vérifiée en remplaçant ρ_{r-1} par ρ_r.

□

1.4.5 Réalisation de l'algorithme

Nous supposons l'existence d'une constante *c* telle que, pour chaque état *p* et chaque étiquette *a*, $|T_a[p]| \leq c$. L'analyse des protocoles en **César** a montré qu'un état avait généralement au plus 5 successeurs par la relation de transitions étiquetées. La représentation de la relation de transitions (cf. §2.2) et celle de la

fonction *nb_succ* sont choisies en tenant compte de cette hypothèse. Nous présentons uniquement les structures de données choisies pour réaliser l'algorithme 2. L'algorithme lui-même n'est pas décrit.

1) Nous reprenons les structures de données de l'algorithme 1 en modifiant le tableau *bloc* de la manière suivante : soit *nb_classe* le nombre de classes de la partition courante,

classe [0 .. n - 1] est un tableau dont le j-ième élément désigne une classe de la partition courante,

bloc est un tableau dont le k-ième élément est un triplet (c, i_1, i_2) ; par convention,

si k ≤ *nb_classe*, *bloc* [k] désigne un bloc simple où

c désigne le cardinal de *classe* [i_1]

$i_2 = 0$, si $i_1 \in W$,

$i_2 = 1$, si $i_1 \notin W$,

$i_2 \geq$ *nb_classe*, *bloc* [i_2] est une feuille ;

si k > *nb_classe*, *bloc* [k] est un *bloc arbre* qui est soit une racine, soit un nœud intermédiaire ; une racine provient d'une décomposition d'une classe *classe* [i] en *classe* [i_1] et *classe* [i_2]. Dans ces conditions,

c désigne (λp.nb_succ (i, p)),

i_1 le fils gauche de k,

i_2 le fils droit de k.

Dans le cas où *bloc* [k] est un nœud intermédiaire, il représente une union de classes,

c désigne le nombre d'éléments de cette union,

i_1 le fils gauche de k,

i_2 le fils droit de k.

2) Une étiquette de l'ensemble A est représentée par un entier. *e* désigne le cardinal de l'ensemble A.

3) Nous représentons *incluspred* de manière analogue à *interpred* :

- *premier* [0 .. e - 1] est un tableau indicé par A ; le a-ième élément désigne le premier indice j du tableau *bloc* (et du tableau *classe*) inséré dans *incluspred* tel que, pour le partitionneur B, *classe* [j] $\subseteq R^{-1}$ [B]. Initialement,
premier [a] = *null.*

- *incluspred* [0 .. e - 1, 0 .. n - 1] est un tableau à 2 dimensions. La première

dimension est indicée par l'ensemble des étiquettes A et la deuxième est indicée par les classes. Initialement, *incluspred* [a, j] = *null*. Si le (a, j)-ième élément n'est pas *null*, alors il désigne un indice k des tableaux *bloc* et *incluspred*, suivant de j dans la liste *incluspred* et tel que, pour le partitionneur B, *classe* [k] $\subseteq R^{-1}$ [B].

4) *nb_succ* est un tableau à deux dimensions, la première étant indicée par les états, la seconde par les étiquettes. Le (q, a)-ième élément désigne la liste des couples (k_j, c_j) tel que $|T_a [q] \cap (\cup k_j)| = c_j$, où $\cup k_j$ représente l'union des feuilles de k_j. Cette liste a au plus *c* éléments.

1.4.6. Evaluation de l'algorithme 2

La taille mémoire nécessaire à l'exécution de l'algorithme 2 est proportionnelle à *m* : il suffit, pour s'en convaincre, de faire l'inventaire des différentes structures de données utilisées dans l'algorithme 2.

Evaluation du temps nécessaire pour exécuter l'algorithme 2

Le temps nécessaire à l'exécution de l'algorithme 2 est de l'ordre de *c m* log *n*. C'est une conséquence de la proposition 2.1.15.

Proposition 2.1.14.

raffiner_simple (i) s'exécute en un temps proportionnel au nombre d'éléments de **classe**[i].

raffiner_arbre (i) s'exécute en un temps proportionnel à la taille du plus petit de ses fils.

Principe de démonstration :

Une classe d'équivalence est une liste dont les fonctions d'accès sont *élément* et *suivant* ; l'insertion ou la suppression d'un élément consiste à modifier l'accès à son prédécesseur et l'accès à son successeur ; le temps d'exécution de chacune de ces opérations est constant.

1) Considérons l'appel de la procédure raffiner_simple (i). Cette procédure est composée de trois actions principales :

- la sauvegarde du partitionneur, $C_i \leftarrow$ *classe* [i],
- une itération pour mettre à jour les structures $interpred_0$, $interpred_1$ et

nb_succ,

- une itération pour mettre à jour les structures *bloc*, *inclasse*, *W* et *in_W*.

Nous montrons que le temps nécessaire à l'exécution de chacune de ces actions est de l'ordre de $\Sigma_{a \in A}|T_a^{-1}[C_i]|$.

(i) La sauvegarde de *classe* [i] est proportionnelle à |*classe* [i]| ; |*classe* [i]| est majoré par $\Sigma_{a \in A}|T_a^{-1}[C_i]| + 1$. En effet, seul l'état initial est susceptible de ne pas avoir de prédécesseur.

(ii) Le temps nécessaire à l'exécution de la procédure maj_interpred (q) est constant (cf. algorithme 1).

(iii) Le temps nécessaire à l'exécution de maj_nb_succ (i, a, q) est de l'ordre de *c*. En effet, le (q, a)-ième du tableau *nb_succ* désigne la liste des couples (k_j, c_j) tel que $|T_a[q] \cap (\cup k_j)| = c_j$. D'après l'hypothèse que nous avons faite, cette liste a au plus *c* éléments. Soit *suc* la liste *nb_succ* [q, a]. L'action principale de la procédure maj_nb_succ (i, a, q) est le parcours de la liste *suc* jusqu'à trouver un couple (i, c_i). Si ce existe alors il est remplacé par le couple (i, c_i+1) sinon le couple (i, 1) est inséré dans la liste *nb_succ* [q, a].

(iv) On déduit de (ii) et (iii) que le temps nécessaire à l'exécution de la première itération de la procédure raffiner_simple est de l'ordre de $\Sigma_{a \in A}|T_a^{-1}[C_i]|$.

(v) La procédure maj_bloc est réalisée de manière analogue à la procédure décomposer_majw de l'algorithme 1. Le temps nécessaire à l'exécution de la procédure maj_bloc est proportionnel au temps nécessaire à l'exécution de maj_classe. On en déduit que le temps nécessaire à l'exécution de la deuxième itération est de l'ordre de

$$\Sigma_{a \in A}|T_a^{-1}[\,C_i\,]| = \Sigma_{j \in \text{interpred}}|X_j|$$

où X_j = *classe* [j] $\cap\, T_a^{-1}[C_i]$ et $\bigcup_{j \in \text{interpred}} X_j = T_a^{-1}[\,C_i]$.

Finalement, le temps nécessaire à l'exécution de raffiner_simple est de l'ordre de $c * \Sigma_{a \in A}|T_a^{-1}[C_i\,]|$.

2) Considérons l'appel de la procédure raffiner_arbre. Il est tout aussi facile de vérifier que le temps nécessaire à l'exécution de la procédure raffiner_arbre (i) est de l'ordre de $c * \Sigma_{a \in A}|T_a^{-1}[C]|$, où C est l'union des classes "feuilles" du

plus petit des fils de i. Pour cela, il suffit de montrer que le temps nécessaire à l'exécution de la procédure calcul_nb_succ est de l'ordre de $c * \Sigma_{a \in A} |T_a^{-1}[C]|$. Par un argument analogue à (v), nous pouvons montrer que le coût de la réalisation de l'itération :

pour tout $a \in A$
  pour tout $(B_j, a) \in$ *incluspred*
    décomposer_majw (a, B_j, fils_gauche(t_i) , fils_droit (t_i))
  fin_pour
fin_pour

est proportionnel à $c * \Sigma_{a \in A} |T_a^{-1}[C]|$.

Soit (F_k) la suite des feuilles du plus petit des fils (i.e. le fils gauche) de l'arbre partitionneur. L'action principale de la procédure calcul_nb_succ est la mise à jour, pour chaque étiquette *a* de l'ensemble A et chaque état *q* d'un F_k, de la liste *nb_succ* [q, a]. Cette mise à jour est réalisée de la manière suivante :
soit fgi l'indice dans le tableau *bloc* de fils_gauche (i) et fdi l'indice dans le tableau *bloc* de fils_droit (i),
- le couple (i, k) est remplacé par le couple (i, k-1) dans la liste *nb_succ* [q, a],
- le couple (fgi, k') est remplacé par le couple (fgi, k'+1) dans la liste *nb_succ* [q, a].

Après l'exécution de la procédure calcul_nb_succ, les couples (i, k) de *nb_succ* [q, a] désignent le nombre de successeurs de l'état *p*, par une transition étiquetée par *a*, dans l'ensemble représentant l'union des feuilles du fils droit du partitionneur.
□

Proposition 2.1.15.

Le temps total d'exécution de l'algorithme 2 est proportionnel à

$\Sigma_{q \in Q, a \in A} |T_a^{-1}[q]| * nb\_w(q)$

où nb_w (q) est le nombre de fois qu'un état appartient à un bloc simple de W, ou à une feuille du fils gauche d'un élément de W.
□

On déduit que le temps total d'exécution de l'algorithme 2 est proportionnel à $c\ m \log n$, où c est le nombre maximum de successeurs que peut avoir un état par une étiquette donnée.

1.4.7 Critique de la mise en œuvre

Deux critiques principales de la mise en œuvre de l'algorithme précédent peuvent être faites :

- la mise à jour de nb_succ n'est pas effectuée en temps constant, et dépend du nombre de successeurs d'un état par une étiquette, ce qui rajoute le facteur c à l'évaluation de l'algorithme. Nous avons fait l'hypothèse qu'un état avait en moyenne 5 successeurs par une étiquette, ce qui nous a conduits à considérer la solution adoptée comme satisfaisante.

- la première action de la procédure raffiner_simple consiste à sauvegarder des éléments du partitionneur. Le coût de cette sauvegarde est bien entendu proportionnel au nombre d'éléments du partitionnneur. Nous avons adopté cette solution pour conserver une mise à jour simple des classes d'équivalence dans le cas où le partitionneur se "décompose" lui-même.

1.5 Applications

Nous décrivons deux applications de la mise en œuvre de l'algorithme 2. Nous l'utilisons pour calculer la plus grande bisimulation entre deux systèmes de transitions étiquetées et pour calculer le système quotient à partir des classes d'équivalence produites par l'algorithme 2. D'autres applications peuvent être envisagées : la partition initiale n'est pas nécessairement la partition universelle. Elle peut être définie par un ensemble de prédicats sur l'ensemble des états et on peut chercher à raffiner cette partition en une partition compatible avec la relation de transitions étiquetées.

1.5.1 Calcul de la plus grande bisimulation entre deux systèmes de transitions étiquetées

Pour décider si deux formes normales sont égales, on peut appliquer

l'algorithme du calcul de la plus grande bisimulation à la somme disjointe de deux systèmes de transitions étiquetées. Nous utilisons la somme disjointe de préférence au produit cartésien, dans la mesure où le cardinal de l'ensemble des états du système "somme" est égal à la somme des cardinaux des ensembles d'états des systèmes de transitions étiquetées composants. Construire la bismulation sur la somme disjointe impose un parcours linéaire de chaque relation de transitions étiquetées, ce qui coûte moins en temps que le parcours du produit cartésien des états.

Etant donné deux systèmes de transitions étiquetées $S_i = (Q_i, A_i, T_i, q_i)$, $i = 1, 2$ on peut toujours renommer les ensembles d'états, et par conséquent les relations de transitions étiquetées, de telle sorte que $Q_1 \cap Q_2 = \emptyset$. Par exemple, on peut renommer les états de Q_1 avec les entiers appartenant à l'intervalle $[0, |Q_1| - 1]$ et les états de Q_2 avec les entiers appartenant à l'intervalle $[|Q_1|, |Q_1| + |Q_2| - 1]$. Par convention, 0 est l'état initial de S_1 et $|Q_1|$ est l'état initial de S_2. Nous notons $\oplus$ l'opérateur "somme".

Exemple

Considérons les deux systèmes de transitions étiquetées S_1 et S_2 décrits par la figure suivante. L'état initial de S_1 est 0 et nous avons renommé l'état initial de S_2 en 3.

$S_1 \oplus S_2 = (\{0, 1, 2, 3\}, \{(0, a, 1), (1, a, 2), (2, a, 0), (3, a, 3)\}$

S_1 S_2

La relation universelle est compatible avec l'union des relations de transitions

étiquetées de S_1 et de S_2.
0 et 3 appartiennent à la même classe. Il existe une bisimulation entre S_1 et S_2.

1.5.2 Construction du système de transitions étiquetées quotient

Nous présentons l'algorithme de calcul du système de transitions étiquetées quotient à partir des classes d'équivalence produites par l'algorithme 2. Soit $S = (Q, A, T, q_0)$ un système de transitions étiquetées. Le système de transitions étiquetées quotient est $S' = (Q / \rho, A, T / \rho, C_0)$ où :
- Q / ρ désigne l'ensemble des classes d'équivalence, notées C_i,
- T / ρ est construite en appliquant la procédure maj_quotient suivante :

```
procédure maj_quotient
    pour tout Ci ∈ Q / ρ
        pour tout p ∈ Ci
            pour tout (a, q) ∈ Ta [p]
                insérer (Ci, a, Cq) dans T / ρ  -- Cq désigne la classe contenant q
            fin_pour
        fin_pour
    fin_pour
fin_procédure
```

Exemple

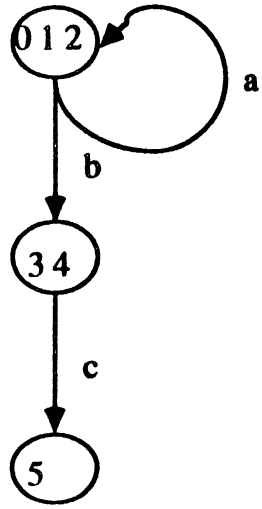


figure 5

Le système de transitions étiquetées quotient du système de transitions étiquetées de la figure 4 est représenté figure 5.

Soit c le majorant de $|T_a[p]|$ pour tout état p et toute étiquette a. Il est facile de voir que l'on peut réaliser l'algorithme précédent de telle sorte que le temps nécessaire à son exécution soit de l'ordre de $c\ m$.

1.6 Discussion

Soit $S = (Q, A, T, q_0)$ un système de transitions étiquetées. Nous avons mis en œuvre un algorithme qui "raffine" une partition de Q pour la rendre compatible avec la relation de transitions étiquetées T. Cet algorithme est quasi-linéaire par rapport à la taille de la relation de transitions étiquetées. Dans ce travail, nous considérons deux applications de cet algorithme : la comparaison de deux systèmes de transitions étiquetées, sous forme normale pour une relation d'équivalence donnée, et le calcul de la plus grande bisimulation sur un système de transitions étiquetées. Pour déterminer la complexité du calcul d'une forme normale par rapport à une relation d'équivalence, nous étudions la complexité des transformations de la relation de transitions étiquetées intervenant dans le calcul de cette forme normale.

2 Transformations locales

Dans ce paragraphe, nous étudions la complexité des transformations intervenant dans le calcul des formes normales décrites au chapitre I. Soit S un système de transitions étiquetées. Dans le paragraphe précédent, nous avons vu comment associer efficacement à S le système quotient de transitions étiquetées modulo la plus grande bisimulation (i.e. la congruence forte). Ce calcul intervient aussi lorsqu'on veut déterminer la forme normale de S modulo une des relations d'équivalence suivantes : l'équivalence observationnelle, la congruence observationnelle et la congruence par modèle d'acceptation. Nous présentons, dans cette partie, la réalisation des autres transformations intervenant dans le calcul de la forme normale modulo une de ces relations d'équivalence, en vue de leur évaluation. Ces transformations sont les transformations τ-chaîne, τ-boucle, τ-circuit, τ-saturation, τ-élimination, saturation, obs_réduction, τ-init et τ-saturation-init. Nous ne traitons pas ces deux dernières transformations : elles ne concernent que l'état initial du système de transitions étiquetées dont on calcule la forme normale. Le temps nécessaire à l'exécution de ces transformations est constant. Il ne dépend pas de la taille de la relation de transitions étiquetées.

Les algorithmes des transformations locales sont tous basés sur le même schéma : parcours de la relation de transitions étiquetées et traitement des états vérifiant une certaine propriété *P*. Les transformations qui nous intéressent peuvent être classées en deux catégories :

- les transformations réalisées par un parcours *en profondeur d'abord* de la relation de transitions étiquetées. Ce sont les transformations τ-chaîne, τ-boucle, τ-circuit, τ-saturation et τ-élimination.
- les transformations réalisées par un parcours *en largeur d'abord* de la relation de transitions étiquetées. Ce sont les transformations saturation et obs_réduction.

Les transformations de la relation de transitions étiquetées sont composées d'insertions et de suppressions de transitions dans la relation de transitions étiquetées. Le critère retenu pour la représentation de la relation de transitions étiquetées est la rapidité d'accès à l'information.

Dans toute la suite de ce paragraphe, nous considérons un système de transitions étiquetées S = (Q, A, T, q_0) où $n = |Q|$, $m = |T|$, $e = |A|$ et $c = \max \{|T_a [p]|, p \in Q, a \in A\}$.

2.1 Représentation de la relation de transitions étiquetées

Nous discutons dans ce paragraphe du choix des structures de données pour la représentation de S.

Un état (resp. une étiquette) est un élément de l'intervalle [0..n -1] (resp. de [0 .. e - 1]). Par convention l'état initial est 0.

La relation de transitions étiquetées est représentée par un tableau *T* indicé par l'ensemble des états ; le p-ième élément est une liste de couples (a, al) où *a* est une étiquette et *al* la liste des états successeurs de p par la relation de transition étiquetée par $a : al = T_a [p]$. Cette liste est ordonnée suivant l'ordre lexicographique. Nous prenons la convention que l'étiquette τ est représentée par 0. Si un état p vérifie $T_\tau [p] \neq \emptyset$ alors la liste $(a, T_a [p])$ a pour premier élément le couple $(\tau, T_\tau [p])$. Les deux primitives de manipulation de liste d'entiers sont *élément* et *suivant*. La liste vide est notée Ø.

Nous définissons deux fonctions insérer (q, l) et supprimer (q, l) pour insérer et supprimer l'élément q "en tête" de la liste l.

Les deux primitives de manipulation de la relation de transitions étiquetées (i.e. le tableau *T*) sont *insertion* et *suppression*. On peut remarquer que *insertion* (p, a, q, T) et *suppression* (p, a, q, T) ont une complexité en temps de l'ordre de *c*. Ces deux primitives sont réalisées par un parcours de la liste T [p] des couples (a', al') jusqu'à ce que a = a'. L'état q est alors inséré dans la liste al' ou supprimé de cette liste.

Avec la représentation choisie, les fonctions T_a et T_a^{-1} retournent un accès au premier élément de la liste *al*. L'avantage de cette représentation est la rapidité d'accès à l'information. L'inconvénient est l'emplacement mémoire trop important pour représenter la relation de transitions étiquetées.

2.2. Transformations de la relation de transitions étiquetées

Nous présentons l'algorithme effectuant un parcours *en profondeur d'abord* de la relation de transitions étiquetées (algorithme 3). A titre d'exemple, nous décrivons une réalisation des transformations τ-chaîne et τ-élimination. Nous présentons ensuite les deux transformations obs_réduction et saturation.

Outre la relation de transitions étiquetées, nous utilisons :
- un tableau *visit* [0 .. n - 1], indicé par l'ensemble des états ; le p-ième élément est 0 si l'état n'a pas été visité, 1 sinon.
- un tableau *bloc* [0 .. n - 1], indicé par l'ensemble des états : le p-ième élément désigne une liste d'états.
- un tableau *inbloc* [0 .. n - 1], indicé par l'ensemble des états ; le p-ième élément désigne un entier.

Ces trois structures sont globales aux transformations réalisées par un parcours en profondeur d'abord.

Algorithme 3

Soient,
T [0 .. n - 1] un tableau dont le p-ième élément est une liste de couples (a, al)
visit [0 .. n - 1] un tableau d'entiers,
bloc [0 .. n - 1] un tableau dont le p-ième élément désigne une liste d'entiers,
inbloc [0 .. n - 1] un tableau d'entiers,
p un entier.
pour tout p ∈ [0, n - 1]
bloc [p] ← Ø ; visit [p] ← 0 ;
fin_pour
pour tout p ∈ [0, n - 1]
si visit [p] = 0 **alors** traiter (p) **fsi**
fin_pour
maj_T ;
fin

Commentaires :
Les procédures traiter et maj_T diffèrent suivant que l'algoritme désigne la transformation τ-chaîne, τ-boucle, τ-circuit, τ-saturation ou τ-élimination.

Exemple : *Réalisation de la transformation τ-chaîne*

Soit S = (Q, A, T, q_0) un système de transitions étiquetées. La transformation τ-chaîne, appliquée à S, supprime de T les transitions (q, τ, q') telles que T [q] = {(t, q')}.
La procédure traiter met à jour le tableau *inbloc*. Pour cela, nous utilisons une liste d'entiers, *chaîne_courante*, qui contient tous les éléments de la chaîne courante. La description de la procédure traiter et maj_T est donnée ci-dessous :

```
procédure traiter (p)
Soient q , q' deux entiers désignant des états.
    visit [p] ← 1 ;
    chaîne_courante ← insérer ( p, chaîne_courante) ;
    si T [p] = {(τ, q)} alors suppression(p, τ, q, T) ;
        si  visit [q] = 0 alors traiter (q)
        sinon                       --q appartient à une autre chaîne
            tant-que chaîne_courante ≠ Ø
                q' ← élément (chaîne_courante) ; inbloc [q'] ← inbloc [q] ;
                chaîne_courante ← supprimer (q', chaîne_courante) ;
            fin_tant-que
        fsi
    sinon                           -- p est la fin d'une chaîne
        tant-que chaîne_courante ≠ Ø
            q' ← élément (chaîne_courante) ; inbloc [q'] ← p ;
            chaîne_courante ← supprimer (q', chaîne_courante) ;
        fin_tant-que
    fsi
fin_procédure
```

```
procédure maj_T
Soient p, q, q' des entiers,
    al une liste d'entiers.
    pour tout p ∈ [0, n - 1]
        pour tout a ∈ [0, e - 1]
            al ← T_a^-1 [p] ;
            tant-que al ≠ Ø
                q ← élément (al) ; q' ← inbloc [q] ;
                insertion (p, a, q', T) ; suppression (p, a, q, T) ;
            fin_tant-que
        fin_pour
    fin_pour
fin_procédure
```

Evaluation :

Remarquons que le temps nécessaire à l'exécution de la transformation τ-chaîne est proportionnel à c^2n. En effet, la procédure traiter est appelée n fois et le temps d'exécution de la procédure traiter est :
- constant pour un état n'appartenant pas à la chaîne courante,
- constant pour un état appartenant à la chaîne courante et ne désignant pas la fin de la chaîne courante,
- de l'ordre de la longueur de la chaîne courante pour un état désignant la fin de la chaîne courante.

De même, il est facile de voir que le temps d'exécution de la procédure maj_T est de l'ordre de $c^2 n$.

Exemple *transformation* τ-élimination

Soit S = (Q, A, T, q_0) un système de transitions étiquetées. La transformation τ-élimination, appliquée à S, supprime de T les transitions (q, τ, q') telles que $q \neq q_0$ et remplace les transitions (p, a, q) par les transitions (p, a, q').

Le tableau *bloc* est mis à jour par la procédure traiter. Le p-ième élément du tableau *bloc* désigne l'ensemble des états atteignables par T^* à partir de p. La

procédure maj_T met à jour le tableau T à l'aide du tableau *bloc*.

```
procédure traiter (p)            -- p ≠ 0
Soit q un entier,
    al une liste d'entiers.
    visit [p] ← 1 ; al ← T_τ [p] ;
    copier (bloc [p], T_τ [p]) ;  -- chaque élément de T_t [p] est inséré dans la liste
                                  -- ordonnée bloc [p]
    tant-que al ≠ Ø
        q ← élément (al) ; al ← suivant (al) ;
        si visit [q] = 0 alors traiter (q) fsi
        bloc [p] ← fusion (bloc [p], bloc [q]); -- fusion de deux listes ordonnées
    fin_tant-que
fin_procédure

procédure maj_T
Soient,p, q, q'des entiers,
    al, bq des listes d'entiers.
    pour tout p ∈ [1, n - 1]
      pour tout  a ∈ [1, e - 1]    -- a ≠ τ
         al ← T_a [p] ;
        tant-que al ≠ Ø
          q ← élément (al) ; al ← suivant (al) ;
          si T_τ [q] ≠ Ø et ∀b ≠ τ T_b [q] = Ø alors suppression (p, a, q, T) ; fsi
         bq ← bloc [q] ;
          tant-que bq ≠ Ø
             q' ← élément (bq) ; bq ← suivant (bq) ;
             si T_τ [q'] = Ø ou ∃b ≠ τ T_b [q] ≠ Ø alors insertion (p, a, q',T) ; fsi
          fin_tant-que
       fin_tant-que
     fin_pour
   fin_pour
```

pour tout p ∈ [1, n - 1]
    al ← T_τ [p] ;
    tant-que al ≠ Ø
        q ← élément (al) ; al ← suivant (al) ; suppression (p, τ, q, T) ;
    fin_tant-que
fin_pour
fin_procédure

Evaluation :

Soit n_τ le nombre d'états à partir desquels il existe un τ-transition : $n_\tau = |\{p \mid T_\tau [p] \neq \emptyset\}|$. Le temps nécessaire à l'exécution de la transformation τ-élimination est de l'ordre de n_τ^3 : c'est un algorithme de calcul de fermeture transitive.

La transformation τ-saturation est aussi un algorithme de calcul de fermeture transitive de la relation T_τ. Son temps d'exécution est de l'ordre de n_τ^3. La procédure traiter (p) met à jour le tableau *bloc* de façon analogue à la procédure τ-élimination et insère les transitions de l'ensemble {p} × T [q] dans *T*, pour un élément q de l'ensemble T_τ [p] (figure 5) :

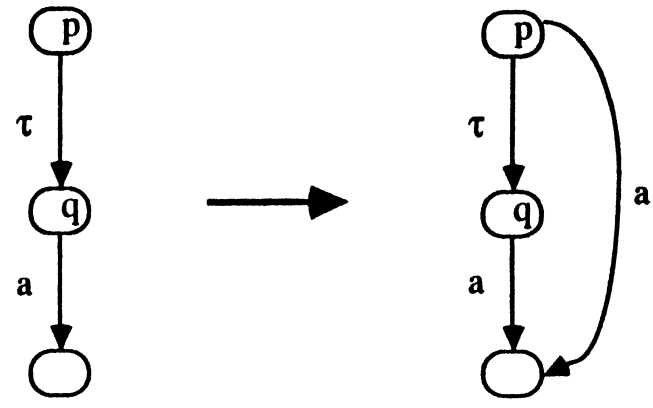


figure 5

La procédure maj_T inverse la relation de transitions étiquetées et pour chaque état q, et chaque état r de la liste *bloc* [q], insère les éléments de $T^{-1}[q] \times \{r\}$ dans *T* (figure 6).

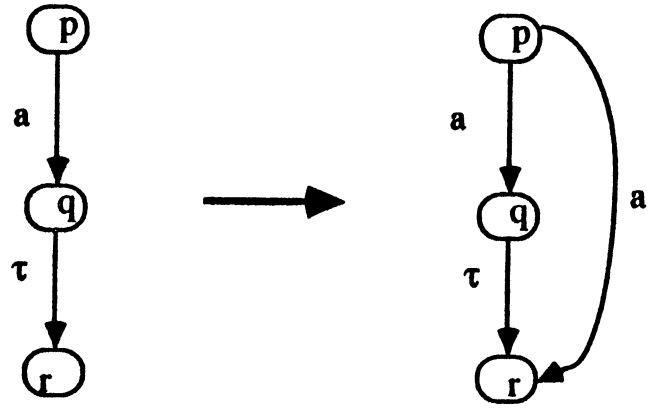


figure 6

L'inversion de la relation de transitions est justifiée par deux raisons. D'une part, la liste T_a^{-1} [q] est directement accessible. D'autre part, la construction du système quotient modulo la congruence et l'équivalence observationnelles, opération qui suit l'application de la τ-saturation, utilise la relation de transitions étiquetées inverse. Remarquons que l'inversion de la relation de transitions étiquetées est linéaire en *m*.

La transformation τ-circuit est analogue à la transformation τ-chaîne. La procédure traiter détermine les composantes fortement connexes de la relation T_τ et met à jour le tableau *inbloc*. Le temps nécessaire à l'exécution de la procédure traiter est de l'ordre de *m* [AHU74]. La procédure maj_T est analogue à la procédure maj_T de la transformation τ-chaîne. Finalement, le temps nécessaire à l'exécution de la transformation τ-circuit est majoré par c^2n.

La transformation τ-boucle est linéaire en *n* : elle consiste, pour chaque état p, à supprimer la transition (p, τ, p).

Transformation obs_réduction

La transformation obs_réduction et la procédure maj_T de la tranformation t-saturation sont "symétriques".

Après la construction du système quotient, la relation de transitions étiquetées est représentée par $T\rho^{-1}$. La transformation obs_réduction est composée des trois actions suivantes :

- pour chaque état r et chaque état q élément de $T\rho_\tau^{-1}$ [r], suppression de la transition (p, a, r) si (p, a, q) $\in$ Tρ (figure 7).
- inversion de la relation T_ρ^{-1}.
- pour chaque état p et chaque état q élément de $T\rho_\tau$ [p], suppression de la transition (p, a, r) si (q, a, r) $\in$ Tρ (figure 8).

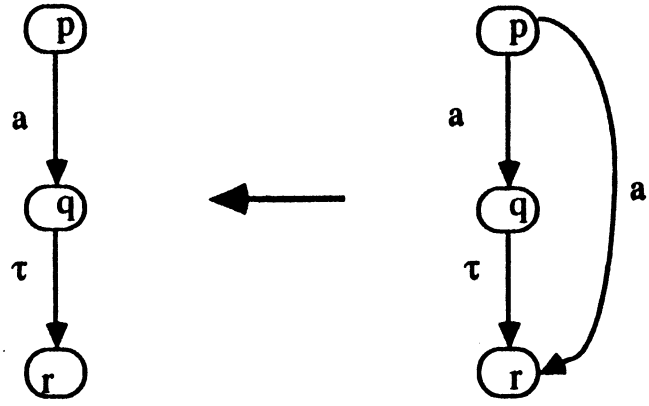


figure 7

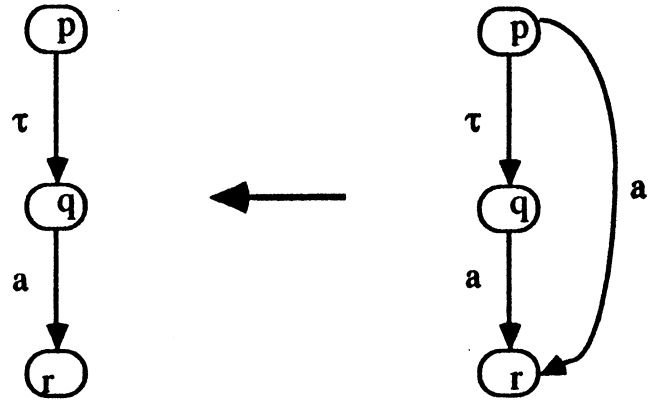


figure 8

Evaluation :

Le temps nécessaire à l'exécution de la transformation obs_réduction est proportionnel à $n_\rho\ m_\rho$, où m_ρ est le cardinal de T_ρ et n_ρ le cardinal de l'ensemble des états du quotient.

Transformation saturation

Cette transformation est décrite par l'algorithme 4 ci-dessous.

Algorithme 4

Soient p, q_1, q_2 des entiers,
al_1, al_2, bl des listes d'entiers.
pour tout p ∈ [0 ,n - 1]
pour tout a ∈ [1 .. e - 1]
si ($|T_a [p]| > 1$)
alors al ← T_a [p] ;
tant-que al ≠ Ø
q_1 ← élément (al) ; al_1 ← suivant (al) ;
tant-que al_1 ≠ Ø
q_2 ← élément (al_1) ; al_1 ← suivant (al_1) ;
pour tout b ∈ [1, e - 1]
bl ← $T_b [p] \cup T_b [q_2]$; $T_b [q_1]$ ← bl ; $T_b [q_2]$ ← bl ;
fin_pour
fin_tant_que
fin_tant-que
fsi
fin_pour
fin_pour
fin_procédure

Evaluation :

Le temps nécessaire à l'exécution de l'algorithme 4 est majoré par $n^3e^2\ m^2$. Ce majorant est atteint dans l'hypothèse la plus défavorable : soit *p* un état et soit *a* une étiquette, $T_a [p] \neq \emptyset$. La fusion des deux listes effectuée dans la boucle **pour** la plus interne est de l'ordre de m^2. Cette fusion est effectuée *e* fois pour chaque couple d'états (q_1, q_2). La transformation saturation est la plus coûteuse

des transformations intervenant dans le calcul des formes normales. Cependant, l'hypothèse la plus défavorable n'est pas vérifiée dans les exemples traités en pratique.

2.3 Discussion

Nous pouvons évaluer le temps nécessaire au calcul d'une forme normale pour la congruence forte, l'équivalence observationnelle, la congruence observationnelle ou la congruence par modèle d'acceptation. En pratique, ce temps est quasi-linéaire par rapport à la taille de la relation de transitions étiquetées "transformée". Certaines transformations sont basées sur le calcul de la fermeture transitive de la relation étiquetée par τ. Cette fermeture est calculée après la transformation τ-cycle qui réduit la relation étiquetée par τ. Nous avons observé dans les exemples traités que le coût du calcul de la forme normale est toujours de l'ordre de celui du calcul de la plus grande bisimulation. Nous pensons qu'il est intéressant de "normaliser" les systèmes de transitions pendant l'opération de composition. Cela permet de minimiser les transitions et les états *temporaires* (celles et ceux qui ne figurent pas dans le résultat final).

3 Réduction de la composition

Généralement, un système réparti est défini, de manière hiérarchique, à partir de processus séquentiels (i.e. de systèmes de transitions étiquetées) de l'opérateur d'abstraction, de l'opérateur de composition, et de l'opérateur de restriction. La vérification consiste à associer à ce système un système de transitions étiquetées sous forme normale pour une relation de congruence. L'objet de ce paragraphe est l'étude de stratégies pour optimiser la génération du système de transitions étiquetées composé. Il s'agit de minimiser le nombre d'états et de transitions "temporaires", générés pendant la composition puis supprimés.

Nous considérons à part l'opérateur d'abstraction qui est généralement utilisé avec la congruence observationnelle pour réduire un système de transitions étiquetées. Nous envisageons uniquement dans ce qui suit les opérateurs de composition et de restriction pour définir les stratégies.

Un système réparti est décrit par une *expression de composition*. La syntaxe abstraite définissant les expressions de composition est décrite de la manière suivante : soit t, S et I désignant respectivement une expression de composition, un système de transitions étiquetées et un sous-ensemble de *Act*,

$t ::= S \mid t \|_L t \mid t \backslash I$.

Les stratégies que nous proposons consistent à décorer l'arbre abstrait associé à l'expression de composition avec un système d'attributs et à générer le système de transitions étiquetées associé à la racine de l'arbre en fonction des attributs de chaque nœud. Une feuille est un système de transitions étiquetées ou un ensemble d'étiquettes ; un nœud qui n'est pas une feuille représente un opérateur. Nous présentons informellement ces stratégies sur un exemple, puis nous définissons deux stratégies particulières.

Exemple

Dans cet exemple, l'algèbre de synchronisation considérée est celle de CCS, et la relation de congruence, la congruence observationnelle ; on note $\|$ au lieu de $\|_{L\setminus}$ l'opérateur de composition correspondant. Par convention, on utilise le caractère " ' " pour désigner le complémentaire d'une étiquette :

$x \bullet x' = \tau$

Nous présentons une version simplifiée du problème d'ordonnancement

[Mil80], (page 33) :
Nous supposons que nous avons n processus communicants p_i pour i variant de 1 à n, chaque processus exécutant une certaine tâche de façon répétitive. Nous construisons un contrôleur, pour garantir que ces tâches s'exécutent de manière cyclique. Plus précisement, chaque p_i reçoit un signal a_i lui permettant d'exécuter sa tâche et émet un signal b_i après exécution de celle-ci. Les deux contraintes que doit vérifier le "scheduler" sont :

(i) Lorsque toutes les occurrences de b_i ne sont pas *visibles*, le système doit être observationnellement congru au système de transitions étiquetées figure 9.

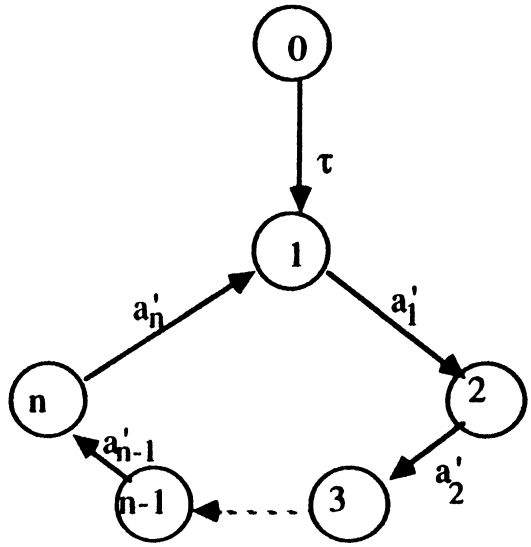


figure 9

(ii) pour chaque i, lorsque toutes les occurrences de $a_j b_j$ ($i \neq j$) sont supprimées, le système doit être observationnellement congru au système de transitions étiquetées figure 10.

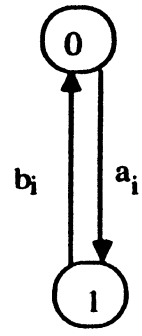


figure 10

Pour décrire le comportement du contrôleur, nous construisons un anneau composé de n "cyclers" élémentaires C_i (figure 11) et d'un "starter" S (figure

12). La représentation en CCS est la suivante :

$C_i = g_i a'_i (b'_i g'_{i+1} C_i + g'_{i+1} b'_i C_i)$, avec la convention $g_{n+1} = g_0$.

$S = g'_1$ **nil**

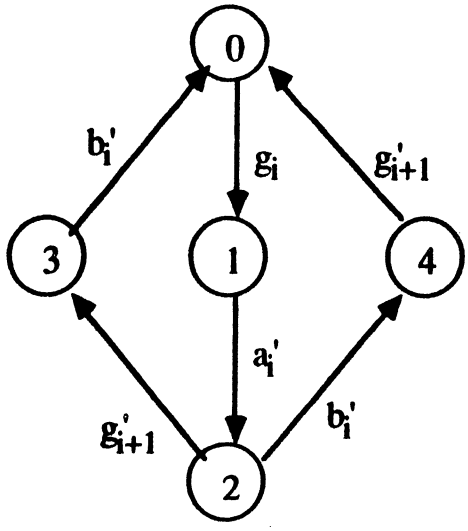


figure 11

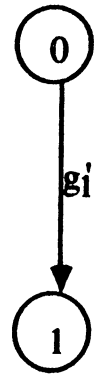


figure 12

Le comportement Sch du contrôleur peut être décrit de la manière suivante :

$$Sch = (S \parallel C_1 \parallel C_2 \parallel \ldots \parallel C_n) \setminus \{g'_1, \ldots, g'_n, g_1, \ldots, g_n\}$$

Pour montrer que la contrainte (i) est satisfaite, nous "fermons" les portes de communication b_i avec les processus R_i (figure 13), dont la représentation en CCS est $R_i = b_i R_i$.

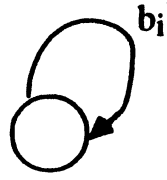


figure 13

Nous montrons que :

$$Sch' = Sch \parallel R_1 \parallel \ldots \parallel R_n) \setminus \{b'_1, \ldots, b'_n, b_1, \ldots, b_n\}$$

est observationnellement congru à $SchR = \tau\, a_1' \ldots a_n'\, SchR$ (figure 9).

Nous nous restreignons au cas ou n = 2.

L'expression de composition est :

$$Sch' = (S \parallel C_1 \parallel C_2) \setminus \{g_1, g_2, g'_1, g'_2\} \parallel R_1 \parallel R_2)) \setminus \{b_1, b_2, b'_1, b'_2\}.$$

Le système de transitions étiquetées associé à Sch' a 3 états et 3 transitions (figure 9)

Comment générer un système de transitions étiquetées à partir de l'arbre abstrait associé à Sch' ?

1) La méthode la plus simple est la suivante :

- pour un nœud *composition parallèle*, générer le système de transitions pour chacun des fils et effectuer la composition (cf. la définition de l'opérateur de composition).
- pour un nœud *restriction*, évaluer le fils "ensemble de restriction", générer un système de transitions étiquetées pour l'autre fils et effectuer la restriction (cf. la définition de l'opérateur de restriction).

Avec cette méthode, trop de transitions temporaires sont générées : au nœud représentant l'expression $(S \parallel C_1 \parallel C_2)$ est associé un système de transitions dont le nombre d'états (resp. de transitions) est de l'ordre de 50 (resp. 200).

2) Nous pouvons améliorer cette méthode en tenant compte du *contexte de réduction* et de la relation de congruence. Par exemple, lors de la génération du système de transitions étiquetées associé au nœud représentant l'expression $(S \parallel C_1 \parallel C_2)$, si l'on tient compte de l'ensemble de restriction $\{g_1, g_2, g'_1, g'_2\}$, le système de transitions associé à $(S \parallel C_1 \parallel C_2) \setminus \{g_1, g_2, g'_1, g'_2\}$ a 13 états et 19 transitions. Sa forme normale pour la congruence observationnelle a 9 états et 13 transitions.

3) Enfin, nous pouvons utiliser les propriétés algébriques des opérateurs $\parallel$ et $\setminus$ étudiées au premier chapitre. En effet,

$$\begin{aligned} Sch' \sim ((&R_1 \parallel C_1) \setminus \{b_1, b'_1\} \parallel \\ &(R_2 \parallel C_2) \setminus \{b_2, b'_2\} \parallel \\ &S) \setminus \{g_1, g_2, g'_1, g'_2\} \end{aligned}$$

Nous associons un système de transitions étiquetées St_1 (resp. St_2) à $(R_1 \parallel C_1) \setminus \{b_1, b'_1\}$ (resp. $(R_2 \parallel C_2) \setminus \{b_2, b'_2\}$) en tenant compte de la restriction pendant la composition. Nous obtenons un système de transitions étiquetées qui a 5 états et 6 transitions. Nous calculons sa forme normale notée St_{1co} (resp. St_{2co}), qui a 3 états et 3 transitions. Enfin, nous associons Sch' à $(S \parallel St_{1co} \parallel St_{2co}) \setminus \{g_1, g_2, g'_1, g'_2\}$ qui a 3 états et 3 transitions. Le résultat est sous forme normale pour la congruence observationnelle.

Il apparaît sur cet exemple que, dans le cas 1 et, dans une moindre mesure dans le cas 2, beaucoup de transitions sont engendrées puis supprimées. Nous

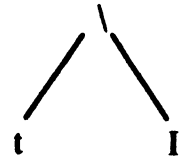
proposons deux stratégies permettant de prendre en compte le contexte et d'effectuer la composition telle qu'elle est décrite en 2 ou en 3.

Pour associer à une expression de communication un système de transitions étiquetées, nous procédons en plusieurs étapes :

1) *Définition de l'ensemble de communication.* Cet ensemble précise, pour les couples d'étiquettes dont le produit de synchronisation est non nul, la valeur de ce produit.

2) *Construction d'un arbre abstrait binaire associé au terme.* Un noeud de l'arbre est soit un opérateur de communication, soit une feuille désignant un système de transitions étiquetées, soit une feuille désignant un ensemble de restriction. A chaque noeud sont associés trois attributs précisant :

- l'ensemble des étiquettes effacées ; à la racine, l'ensemble qu'il représente est vide ; cet ensemble est enrichi sur les sous-termes t de



par I .

- les systèmes de transitions étiquetées "feuilles" ;

- l'ensemble des étiquettes visibles ; pour une feuille, il représente l'ensemble des étiquettes du système de transitions étiquetées associé à la feuille ; il est modifié par l'opérateur de restriction.

3) *Parcours de l'arbre abstrait pour mettre à jour l'attribut associé à chaque noeud* :

- l'attribut ensemble des étiquettes effacées est hérité,

- les attributs ensemble des étiquettes visibles et systèmes de transitions feuilles sont synthétisés. Par exemple, à l'expression de composition
$Sch' = (S \parallel C_1 \parallel C_2) \setminus \{g_1, g_2, g'_1, g'_2\} \parallel R_1 \parallel R_2)) \setminus \{ b_1, b_2, b'_1, b'_2\}$,
il correspond l'arbre de la figure 14.

4) *Parcours de l'arbre abstrait modifié afin de générer le graphe de communication* : les sommets de ce graphe sont les systèmes de transitions étiquetées et un arc relie deux sommets si et seulement si il existe une communication entre les deux systèmes de transitions étiquetées correspondants. Les arcs sont étiquetés par l'ensemble des étiquettes visibles après

communication. Le graphe de communication associé à Sch' est celui de la figure 15.

5) *Réduction du graphe de communication.* Cette réduction est faite en choisissant deux sommets du graphe de communication, en les composant et en réalisant la fusion, dans le graphe, des deux sommets venant d'être composés.

Les attributs d'un nœud t de la syntaxe abstraite sont notés de la manière suivante :

t. V désigne l'attribut "actions visibles",

t . E dénote l'attribut "actions effacées",

t . S désigne l'attribut "systèmes de transitions étiquetées".

La mise à jour des attributs est régie par les règles suivantes :

Règles de la grammaire abstraite	*Calcul des attributs*
$t \rightarrow S = (Q, A, T, q_0)$	$t.V \leftarrow A$ $t.S = \{S\}$
$t \rightarrow t_1 \setminus I$	$t.V \leftarrow t_1.V - I$ $t.S \leftarrow t_1.S$ $t.E \leftarrow (t.E) \cup I$ $t_1.E \leftarrow (t.E)$
$t \rightarrow t_1 \|\|_L t_2$	$t.V \leftarrow t_1.V \cup t_2.V \cup (t_1.V) \bullet (t_2.V)$ $t.S \leftarrow t_1.S \cup t_2.S$ $t_1.E \leftarrow t.E$ $t_2.E \leftarrow t.E$

Remarques

(i) Si le noeud est une feuille $S_i = (Q_i, A_i, T_i, q_i)$ alors :

- l'ensemble des étiquettes effacées est hérité du noeud père,
- l'ensemble des étiquettes visibles est A_i.
- l'ensemble des systèmes de transitions étiquetées est $\{S_i\}$.

(ii) Si le noeud est l'opérateur de composition parallèle alors :

- l'ensemble des étiquettes effacées est hérité du noeud père et transmis aux fils gauche et droit,
- l'ensemble des étiquettes visibles est l'union des ensembles des étiquettes visibles des fils gauche et droit et de l'ensemble des étiquettes résultant des communications,
- l'ensemble des systèmes de transitions étiquetées est l'union des ensembles des systèmes de transitions étiquetées des fils gauche et droit.

(iii) Si le noeud est l'opérateur de restriction, si le fils gauche désigne l'ensemble des étiquettes effacées alors :

- l'ensemble des étiquettes effacées est l'union de l'ensemble des étiquettes effacées hérité du noeud père, et de l'ensemble désigné par le fils gauche. Si le fils droit est encore un noeud de restriction, on réitère cette opération.
- l'ensemble des étiquettes visibles est la différence ensembliste entre l'ensemble des étiquettes visibles du fils droit et l'ensemble des étiquettes effacées.
- l'ensemble des systèmes de transitions étiquetées est celui du fils droit.

Exemple

Nous associons à Sch' l'arbre abstrait de la figure 14. Les attributs sont représentés dans un carré et désignent, dans l'ordre :

- l'attribut *actions effacées*,
- l'attribut *actions visibles*,
- l'attribut *systèmes de transitions étiquetées.*

Les étapes de construction et de réduction du graphe diffèrent suivant la stratégie choisie. Nous définissons la notion de graphe de communication puis les deux stratégies **comp1** et **comp2** permettant de construire et de réduire de graphe de communication.

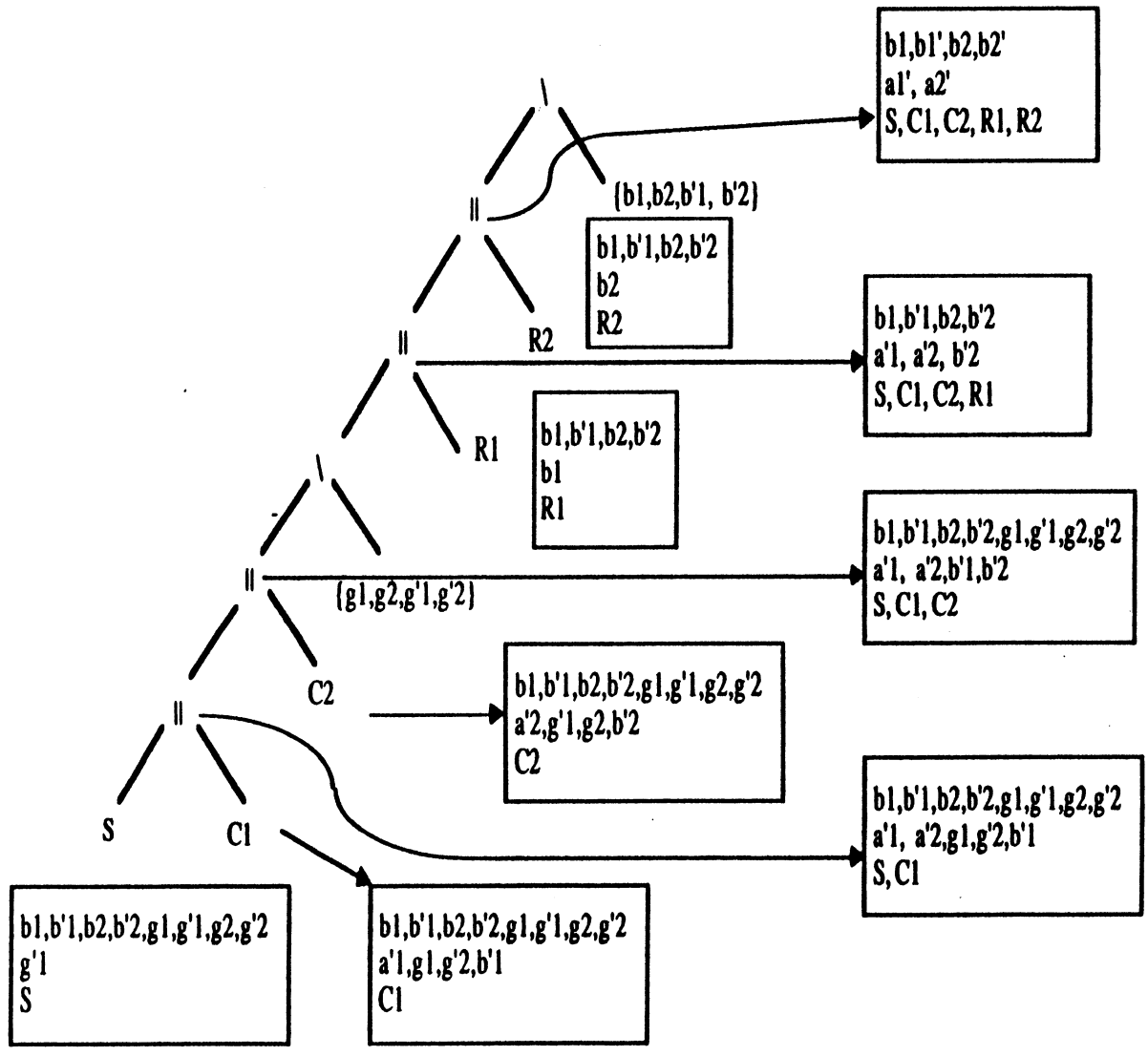


figure 14

3.1 Graphe de communication

Soit t une expression de composition dont les feuilles sont S_1 ,..., S_n, où $S_i = (Q_i, A_i, T_i, q_i)$. Soit E_i l'attribut *actions effacées* associé à S_i.
Dans cette partie, nous définissons le graphe de communication associé à t. Tout d'abord, nous définissons les informations nécessaires à la construction et à la mise à jour de ce graphe. Ces informations sont :

- la fonction *nb_aut* qui, étant donné un système de transitions étiquetées S_i et une étiquette *a*, détermine le nombre de systèmes de transitions étiquetées avec lesquels S_i communique par l'intermédiaire de *a* ;
- l'ensemble d'étiquettes visibles après communication entre deux systèmes

de transitions S_i et S_j. Cet ensemble est déterminé en fonction de nb_aut et des attributs actions effacées E_i et E_j ;

- la fonction booléenne *com_aut* qui détermine si deux systèmes de transitions communiquent.

Définitions

1) La fonction *nb_aut* est définie de la manière suivante :

$nb\_aut\,(a, S_i) = |\{b \mid \exists\, j \text{ et } j \neq i \;\; b \in A_j \text{ et } a \bullet b \neq 0\}|$

2) Les ensembles d'*actions visibles après communication* V_{ij} sont définis de la manière suivante :

$$\begin{aligned} V_{ij} = \;& \{a \mid a \in V_i \text{ et } a \notin E_i\} \cup \\ & \{a \mid a \in V_i \text{ et } a \in E_i \text{ et } nb\_aut\,(a, S_i) \neq 1\} \cup \\ & \{a \mid a \in V_i \text{ et } a \in E_i \text{ et } nb\_aut\,(a, S_i) = 1 \text{ et } a \bullet b \neq 0 \text{ et } b \notin A_j\} \cup \\ & \{a \mid a \in V_j \text{ et } a \notin E_j\} \cup \\ & \{a \mid a \in V_j \text{ et } a \in E_j \text{ et } nb\_aut\,(a, S_j) \neq 1\} \cup \\ & \{a \mid a \in V_j \text{ et } a \in E_j \text{ et } nb\_aut\,(a, S_j) = 1 \text{ et } a \bullet b \neq 0 \text{ et } b \notin A_i\} \cup \\ & \{a \bullet b \mid a \in V_i \text{ et } b \in V_j \text{ et } a \bullet b \neq 0 \text{ et } a \bullet b \notin E_i \cup E_j\}. \end{aligned}$$

3) La fonction *com_aut* est définie de la manière suivante :

$com\_aut\,(S_i, S_j) = \exists\; a \in A_i \text{ et } \exists\, b \in A_j \text{ et } a \bullet b \neq 0$

4) Le *graphe de communication* associé à t est un graphe non orienté $(F, \rightarrow)$:

- les sommets sont les couples (S_i, E_i) :

$F = \{(S_i, E_i) \mid S_i \text{ est une feuille de t et } E_i \text{ l'attribut "actions effacées" associé}\}$

- les arcs sont étiquetés par les étiquettes visibles après communication si et seulement si les sommets communiquent :

$(S_i, V_{ij}, S_j) \in \rightarrow$ si et seulement si $com\_aut\,(S_i, S_j)$ est vrai.

Exemple

Le graphe associé à Sch' est donné figure 15. Nous notons :

$E = \{b_1, b'_1, b_2, b'_2\}$,

$E' = E \cup \{g_1, g'_1, g_2, g'_2\}$

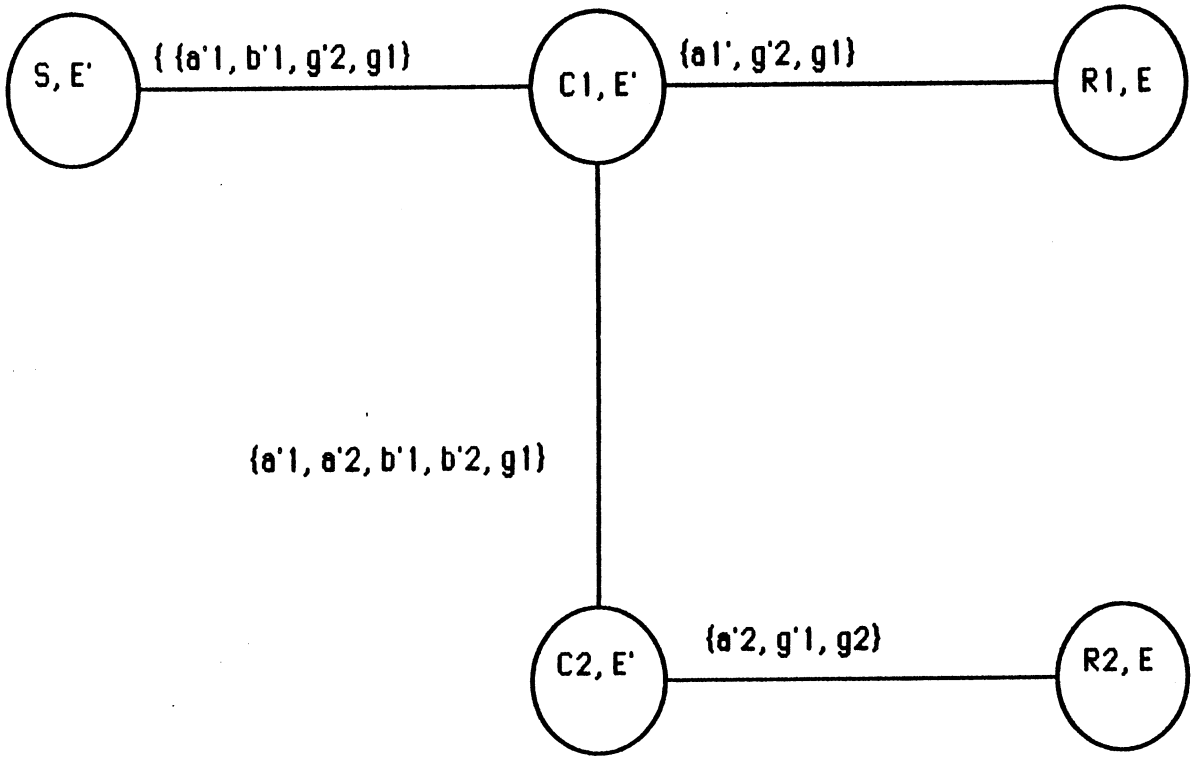


figure 15

3.2 Définition de la stratégie comp1.

Les phases de construction et de réduction du graphe de communication sont confondues et sont définies de la manière suivante :

si le noeud est un *noeud composition parallèle* **alors**

- construction et réduction du graphe de communication pour le fils droit et pour le fils gauche ; soient sg et sd les systèmes de transitions étiquetées associés respectivement au fils gauche et au fils droit.
- détermination des étiquettes visibles étiquetant l'arc de communication entre sg et sd.
- composition de sg et sd et calcul de la forme normale du résultat pour la congruence choisie.

sinonsi le nœud et un nœud *restriction* **alors**

construction et réduction du graphe de communication pour le fils droit et calcul de la forme normale du résultat pour la congruence choisie.

fsi

3.3 Définition de la stratégie comp2.

Soit G = (F, →) le graphe de communication associé à l'expression de composition t et $\sim_c$ une congruence. La stratégie **comp2** consiste à appliquer tant que c'est possible une *réduction du graphe de communication*. Une réduction du graphe de communication **red**, associe à un graphe G de n sommets un graphe G' de n-1 sommets et peut être définie par la procédure suivante :

red (G, $\sim_c$)
- choisir deux sommets (S_i, E_i) et (S_j, E_j) de G ;
- effectuer la composition $S_i \|_L S_j$; soit S la forme normale du résultat pour $\sim_c$;
- remplacer dans G les sommets (S_i, E_i) et (S_j, E_j) par le sommet (S, $E_i \cap E_j$);
- mettre à jour les *ensembles d'actions visibles après communication* (cf. définitions du paragraphe 3.1) ;
- retourner G'.

fin

La procédure précédente peut être mise en œuvre de plusieurs façons distinctes : il faut préciser comment choisir les deux sommets à composer. Ce choix est déterminé par les considérations suivantes :

(i) Puisqu'on s'intéresse aux systèmes qui communiquent, on choisit des systèmes appartenant à la même composante connexe. Dans l'exemple de la figure 15, le graphe de communication est réduit à une seule composante connexe.

(ii) Parmi les systèmes de transitions étiquetées appartenant à une même composante connexe, on détermine l'ensemble des systèmes de transitions étiquetées qui communiquent le moins (i.e. les sommets du graphe de degré minimum, le degré d'un sommet étant le nombre d'arcs ayant ce sommet pour extrémité). Dans l'exemple de la figure 15, ce sont les systèmes de transitions étiquetées S, R_1 et R_2.

(iii) On choisit alors deux systèmes de transitions, dont l'un appartient à l'ensemble précédent, et tels que l'ensemble des actions visibles après communication soit le plus petit possible.

Les considérations précédentes conduisent à réduire le graphe de communication en deux phases :

a) Chaque composante connexe G' = (F', →) de G est réduite à un élément en itérant la procédure **red** où l'action de choisir les deux éléments à composer est précisée.

b) Si le graphe G a plus d'une composante connexe, le graphe est réduit à un élément en itérant la procédure **red** où l'action de choisir les deux éléments à composer est aléatoire.

L'itération appliquée à une composante connexe G' (resp. au graphe G) est la construction de la suite :

$\mathbf{Red}^{i+1}(G', \sim_c) = \mathbf{Red}(\mathbf{Red}^{i}(G, \sim_c), \sim_c)$.

Lorsque $\mathbf{red}^{n}(G')$ (resp. $\mathbf{red}^{n}(G)$) est réduit à un seul élément, nous obtenons le système de transitions étiquetées associé à la composante connexe (resp. l'expression de composition t).

a) Réduction d'une composante connexe du graphe de communication

Le choix des deux éléments à composer, pour réduire une composante connexe G' = (F', →), est déterminé par les actions suivantes :

- détermination du degré minimum de G' :

  $degré\_min\ (G') = min\ \{degré\ (S) \mid S \in F'\}$

- détermination des sommets F'' de G' vérifiant :

  $F'' = \{S \in F' \mid degré(S) = degré\_min\ (G')\}$

- détermination des deux sommets à composer :

c'est un couple (S_i, S_j) vérifiant la propriété suivante,

$S_i \in F''$ et $|V_{ij}| = min\ \{|V_{kl}| \mid S_k \in F'', S_l \in F'\}$

et $com\_aut\ (i, j) =$ vrai

Dans l'exemple de la figure 15, réduit à une seule composante connexe,

$degré\_min\ (G) = 1$,

$F' = \{S, C_1, C_2\}$,

le couple choisi est soit (C_1, R_1), soit (C_2, R_2).

b) Réduction du graphe de communication

Si le graphe G a plus d'une composante connexe, chaque composante connexe contient un seul système de transitions étiquetées. La composition, par application de la procédure **red**, est un entrelacement, puisqu'aucun système de transitions étiquetées ne communique avec un autre. Il n'y a pas lieu de définir

un choix particulier pour deux systèmes de transitions étiquetées devant être composés.

c) Construction du graphe de communication

La construction du graphe de communication associé à l'expression de composition t est déterminée par la propriété suivante (chapitre I) :

Soient $S_i = (Q_i, A_i, T_i, q_i)$, i = 1, 2, deux systèmes de transitions étiquetées et L une algèbre de synchronisation.

(i) $(S_1 \|_L S_2) \setminus \{a\} = (S_1 \setminus \{a\}) \|_L S_2$ si et seulement si

$(a \notin A_1 \bullet A_2$ et $a \notin A_2$ et $(a \in A_1 \Rightarrow Com\ (a) \cap A_2 = \emptyset))$ ou

$(a \in A_1 \bullet A_2$ et $((a = b \bullet c) \Rightarrow a = b = c))$

Cette propriété se généralise aux expressions de composition de la manière suivante :

(ii) $(t_1 \|_L t_2) \setminus I = (t_1 \setminus I) \|_L t_2$ si et seulement si

$\forall a \in I$

$(a \notin ((t_1.V) \bullet (t_2.V))$ et $a \notin (t_2.V)$ et

$(a \in (t_2.V) \Rightarrow Com\ (a) \cap (t_2.V) = \emptyset))$ ou

$(a \in ((t_1.V) \bullet (t_2.V))$ et $((a = b \bullet c) \Rightarrow a = b = c))$

Nous transformons l'arbre en appliquant les deux transformations suivantes.

1) Soit n un nœud de la forme $n_1 \|_L n_2$. Si n_1 contient un sous-arbre de la forme $n_{11} \setminus I_1$ et si n_2 contient un sous-arbre de la forme $n_{22} \setminus I_2$, alors on remplace le nœud $t_{11} \setminus I_1$ par une feuille correspondant au système de transitions étiquetées associé à $t_{11} \setminus I_1$ par la stratégie **comp2**. On procède de la même manière avec le nœud $t_{22} \setminus I_2$. Remarquons que si l'arbre contient des sous-arbres de la forme $t \setminus I$, après l'application de cette transformation, alors ceux-ci sont totalement ordonnés vis-à-vis de la relation d'ordre *parent* ; n est *parent* de n' si n' est un fils de n ou si un fils de n est *parent* de n'.

2) La deuxième transformation consiste à déterminer les systèmes de transitions étiquetées feuilles appartenant au même graphe. Elle consiste à appliquer la propriété (ii) dans le sens droite gauche.

Etant donné deux nœuds *restriction* $n_1 \setminus I_1$ et $n_2 \setminus I_2$ tels que $n_1 \setminus I_1$ soit parent de $n_2 \setminus I_2$,

si $\forall S_1 \in (n_1 . S)$ et $S_1 \notin (n_2 . S) \Rightarrow (S_1 \|_L n_2) \setminus I_2 = S_1 \|_L (n_2 \setminus I_2)$,

alors les systèmes de transitions étiquetées de $(n_1 . S)$ et de $(n_2 . S)$ appartiennent au même graphe de communication, sinon on remplace le nœud $t_2 \setminus I_2$ par une

feuille correspondant au système de transitions étiquetées associé à $t_2 \setminus I_2$ par la stratégie **comp2.**

La génération du graphe est alors réalisée en parcourant les feuilles de l'arbre.

3.4. Exemple

Nous illustrons la construction et la réduction du graphe de communication associé à Sch' en utilisant la stratégie **comp1** et **comp2.**

a) stratégie comp1

- composition de S et C_1

les étiquettes visibles après composition sont :

$\{a'_1, g_1, g'_2, b'_1\}$

le système de transitions étiquetées composé, noté StS_1, a

10 états,

13 transitions,

la normalisation de StS_1 pour la congruence observationnelle laisse StS_1 inchangé.

- composition de StS_1 et C_2

les étiquettes visibles après composition sont :

$\{a'_1, a'_2, b'_1, b'_2\}$

le système de transitions étiquetées composé, noté St_{12}, a

13 états,

19 transitions,

le normalisé pour la congruence observationnelle de St_{12}, noté St_{12co}, a

9 états,

13 transitions.

- composition de St_{12co} et R_1

les étiquettes visibles après composition sont :

$\{a'_1, a'_2, b'_2\}$

le système de transitions étiquetées composé, noté StR_2, a

9 états,

13 transitions,

le normalisé pour la congruence observationnelle de St_{12}, noté StR_{2co}, a

5 états,

6 transitions.

- composition de StR_{2co} et R_2

les étiquettes visibles après composition sont :

$\{a'_1, a'_2\}$

le système de transitions étiquetées composé, noté StR, a

5 états,

6 transitions,

le normalisé pour la congruence observationnelle de StR, noté Sch', a

3 états,

3 transitions.

b) stratégie comp2

Le graphe de communication est celui donné par la figure 16.

- Les sommets de plus petit degré sont S, R_1 et R_2,
- nous pouvons composer soit C_1 et R_1, soit C_2 et R_2 , nous choisissons de composer C_1 et R_1 :

le système de transitions étiquetées composé, noté St_{11}, a

5 états,

6 transitions,

le normalisé pour la congruence observationnelle de St_{11}, noté St_{11co}, a

3 états,

3 transitions,

- mise à jour du graphe de communication : nous obtenons le graphe de communication de la figure 16.

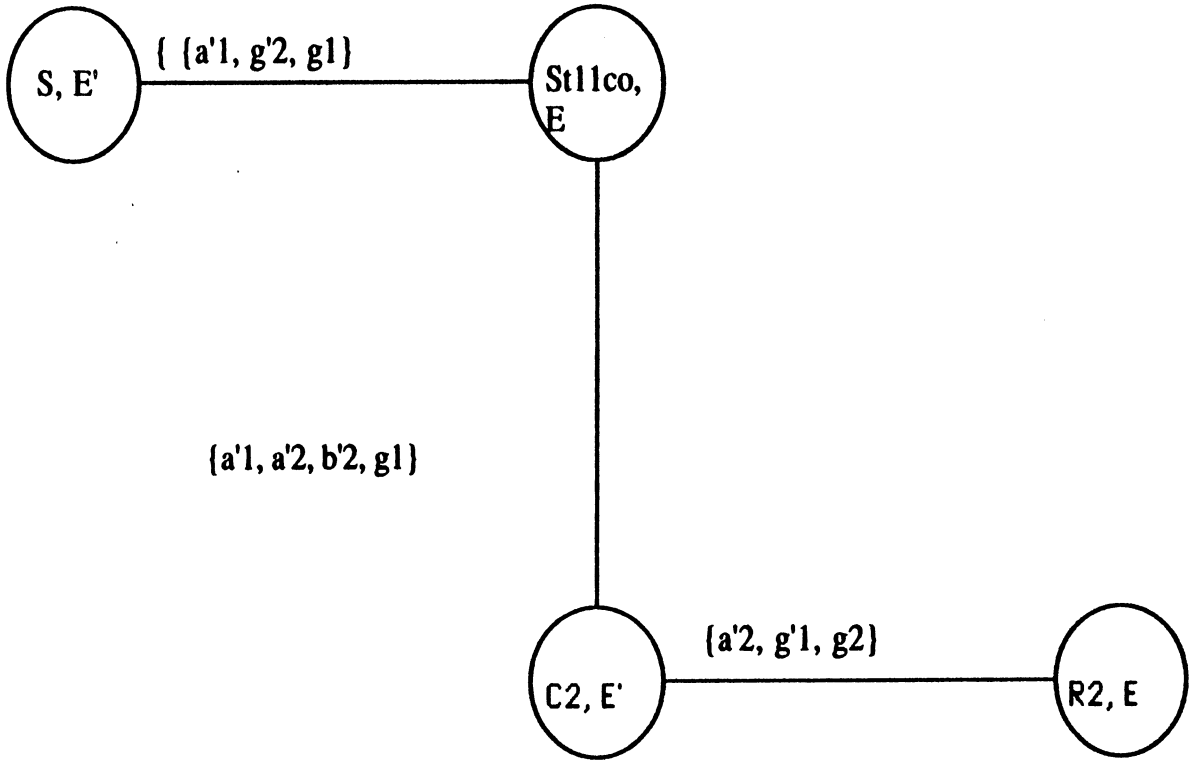


figure 16

- Les sommets de plus petit degré sont S et R_2

composition de C_2 et R_2

le système de transitions étiquetées composé, noté St_{22}, a

5 états,

6 transitions,

le système de transitions étiquetées normalisé pour la congruence observationnelle de St_{22}, noté St_{22co}, a

3 états,

3 transitions,

mise à jour du graphe de communication : nous obtenons le graphe de communication de la figure 17.

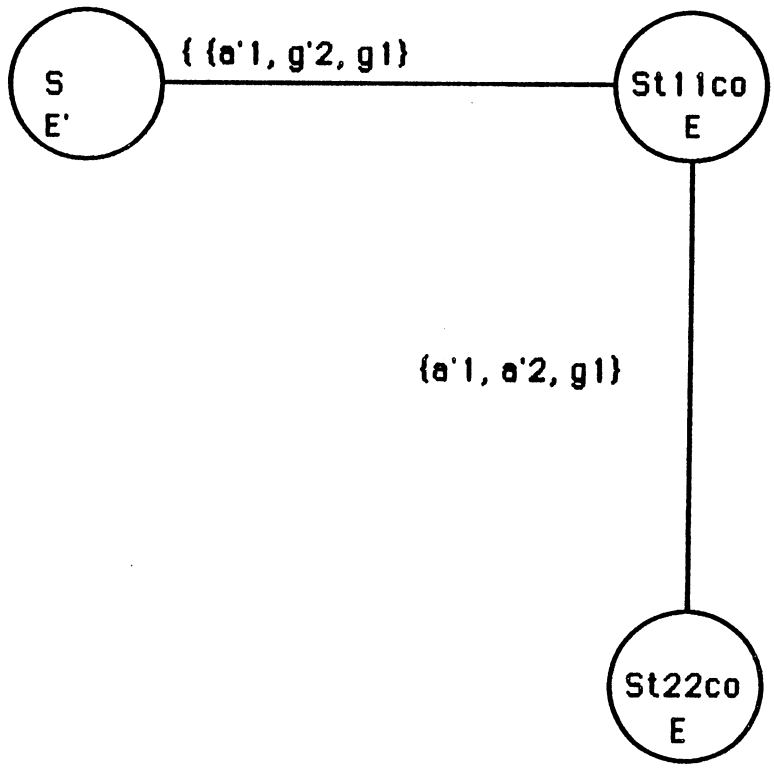


figure 17

- Les sommets de plus petit degré sont S et St_{22co}
nous pouvons composer soit S et St_{11co}, soit St_{22co} et St_{11co}
composition de S et St_{11co}
le système de transitions étiquetées composé, noté StS_1, a
6 états,
7 transitions,
le normalisé pour la congruence observationnelle de StS_1 noté StS_{1co} a
6 états,
7 transitions.

Enfin, le normalisé pour la congruence observationnelle de la composition de St_{1co} et St_{22co}, notée Sch', a trois états et trois transitions (figure 18) :

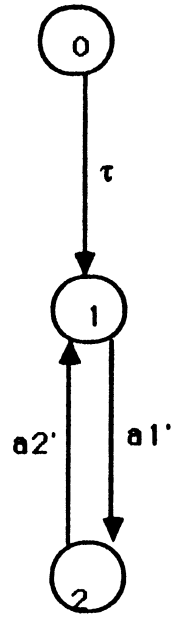


figure 18

3. 5 Réalisation

Nous étudions dans ce paragraphe la mise en œuvre des stratégies **compl** et **comp2**. Nous ne présentons pas l'évaluation de nos algorithmes. Ceux-ci manipulent essentiellement les noms des systèmes de transitions que l'on veut composer et les étiquettes. Le temps nécessaire à l'exécution des procédures définies pour les stratégies **compl** et **comp2** est négligeable par rapport aux temps nécessaires à l'exécution des transformations étudiées dans les paragraphes précédents.

La table du produit de synchronisation est représentée par une matrice triangulaire (par commutativité de l'opérateur •) *act_com* [a, b] ; le (a, b)-ième élément désigne l'étiquette *c* si a • b = c.

Soient n systèmes de transitions étiquetées, numérotés de 0 à n - 1 et soient i, j ∈ [0 .. n - 1] deux systèmes de transitions étiquetées. Les structures de données suivantes sont utilisées pour construire et réduire le graphe de communication.

Le graphe de communication est représenté par deux structures :

- une matrice triangulaire (par commutativité de l'opérateur •),*visible* [i, j] ; le (i, j)-ième élément désigne un couple (n, l) où l est une liste d'étiquettes visibles après communication de i et j et n le cardinal de cette liste ;

- un tableau *gra_com*, dont le i-ième élément désigne un couple (degré, l), où degré est un entier désignant le degré du sommet i et l la liste de systèmes de

transitions étiquetées communiquant avec i.
Un tableau *classe* représente les composantes connexes du graphe de communication. Il est construit à partir du tableau *gra_com.*
Un tableau *inclasse* dont le i-ième élément désigne la classe de i.
Un tableau *ste_com* dont le i-ème élément désigne le noeud dans lequel i apparaît.

La génération du système de transitions étiquetées associé à un noeud n est réalisée pour la stratégie **comp2** en appliquant successivement les trois procédures suivantes :

parcours1 (noeud, Ø) ;
parcours2 (noeud) ;
générer (i, noeud) ; i est un entier désignant le système de transitions à créer.

La rôle de la procédure parcours1 est la mise à jour des attributs.
La procédure parcours2 transforme l'arbre avant la génération du graphe de communication. C'est cette procédure qui détermine si toutes les feuilles de l'arbre appartiennent au même graphe de communication. Soient n et n' deux nœuds tels que n soit *parent* de n'. Si un système de transitions étiquetées, attribut *système de transitions étiquetées* du nœud n, ne peut être dans le même graphe que les feuilles du nœud n', alors un système de transitions étiquetées, associé au nœud n' est généré.

La génération du système de transitions étiquetées associé à un noeud n est réalisée pour la stratégie **comp1** en appliquant successivement les deux procédures suivantes :

parcours1 (noeud, Ø) ;
générer (i, noeud) ; i est un entier désignant le système de transitions à créer.

a) *Parcours de l'arbre pour mettre à jour les attributs*

Nous employons les notations suivantes :
visibles(n) désigne les étiquettes visibles du noeud n,
effacées(n) désigne les étiquettes visibles du noeud n,
systèmes_de_transitions_étiquetées(n) désigne la liste des systèmes de transitions étiquetées feuilles, descendants du noeud n.

```
procédure parcours1 (n : nœud, R : ensemble d'étiquettes)
    selon la nature du nœud n ∈ {\, ||, Idf_s}
        \:  A ← R ∪ fils_gauche(res (n)) ; parcours1(n, A);
            effacées(n) ← A ;
            visibles (n) ← visibles(fils_droit(n)) ;
            systèmes_de_transitions_étiquetées (n) ←
                systèmes_de_transitions_étiquetées (fils_droit(n)) ;
        || : parcours1 (fils_gauche (n), R) ; parcours1 (fils_droit (n), R) ;
            effacées (n) ← R ;
            visibles (n) ← visibles(fils_gauche(n)) ∪ visibles(fils_droit(n)) ∪
            {c | c = a • b et a • b ≠ 0 et a • b ≠ τ et
                a ∈ visibles(fils_gauche(n)) et b ∈ visibles(fils_droit(n))};
            systèmes_de_transitions_étiquetées (n) ←
                systèmes_de_transitions_étiquetées (fils_gauche(n)) ∪
                systèmes_de_transitions_étiquetées (fils_droit(n)) ;

        Idf_s : effacées (n) ← R ;
            systèmes_de_transitions_étiquetées (n) ← {Idf_s}
fin_procédure
```

b) *Transformation de l'arbre avant la génération du graphe de communication*

- Un parcours de l'arbre met à jour une liste de nœuds *parent*, dont le dernier élément est la racine. Si n_1 est le *parent* de n_2 alors n_1 est le suivant de n_2 dans cette liste. Les primitives de manipulation de listes de nœuds sont *élément* et *suivant*. La liste vide est notée *null*.

Nous notons *Prop* (A_1, A_2, I) le prédicat :

$\forall a \in I$ $(a \notin A_1 \bullet A_2$ et $a \notin A_2$ et $(a \in A_1 \Rightarrow Com(a) \cap A_2 = \emptyset))$ ou
$(a \in A_1 \bullet A_2$ et $((a = b \bullet c) \Rightarrow a = b = c))$

La procédure suivante met à jour *ste_com* et construit et détermine les systèmes de transitions étiquetées appartenant au graphe de comunication.

---

parcours2 (n : nœud)
Soient,
- $nœud_1$, $nœud_2$ deux nœuds,
- S_2 un entier désignant un système de transitions étiquetées.

tant-que parent ≠ null
$nœud_2$ ← élément (parent) ;
$nœud_1$ ← suivant (parent) ;
pour tout $S_2 \in$ (fils_droit ($nœud_1$)) . S
si $S_2 \notin$ (fils_droit ($nœud_2$)) . S
alors
si *Prop* (fils_gauche ($nœud_2$).V, S_2.V, fils_droit ($nœud_2$).E) = faux
alors mk_ste ($nœud_2$) ; **retour** ;
fsi
fsi
finpour
fin_tant-que
fin_procédure

Commentaires : La fonction mk_ste crée un nouveau nom, appelle générer et met à jour le noeud. L'instruction **retour** permet de "sortir" de la boucle **pour**.

c) *Génération et réduction pour la stratégie **comp2***

Nous donnons uniquement la description de la procédure telle qu'elle est réalisée.

---

générer2 (i : entier, n : nœud)

Description :
Construction du graphe :
- maj de visible en tenant compte de la définition 2.3.3 du chapitre I,
- maj _de gra_com , pour deux systèmes qui communiquent.

Réduction du graphe :
- détermination des composantes connexes du graphe ;
- réduction de chacune de ces composantes connexes en composant les

systèmes deux à deux suivant le critère défini au chapitre I : composition d'un couple de deux systèmes de transitions étiquetées dont un des éléments est de degré minimum et dont le cardinal des étiquettes visibles est minimum.

- chacune des composantes connexes ne comporte plus qu'un seul élément. On compose, s'il y a lieu, l'ensemble de ces éléments représentant une composante connexe.

---

d) *Génération et réduction pour la stratégie* ***compl***

```
générer1 (i, n)
    selon la nature du nœud ∈ {\, ||, Idf_s}
        || :  générer1 (i1, fils_gauche(nœud)) ;
              générer1 (i2, fil_droit(noeud)) ;
              maj_visible (i1, i2) ;
              composer (i, i1, i2)
        \:    générer1 (i, fils_droit(n)) ;
        Idf_s : i ← Idf_s ;
    fin
fin_procédure
```

---

e) *Composition de deux systèmes de transitions étiquetées*

```
composition (i1, i2)
    pour tout (p, a, q) ∈ T1
        pour tout (p', b, q') ∈ T2
            selon (a • *, b • * )
                a • * = 0 et b • * = 0:
                    si a • b ≠ 0 et a • b ∈ visible [i1, i2]
```

alors insérer $p * |Q_2| + p', a \bullet b, q' * |Q_2| + p'$) dans T_i
fsi
$a \bullet * = a, b \bullet * = b$:
si $a = \tau$ ou $a \in$ visible $[i_1, i_2]$
alors insérer $(p * |Q_2| + p', a, q * |Q_2| + p')$ dans T_i ;
fsi
si $b = \tau$ ou $b \in$ visible $[i_1, i_2]$
alors insérer $(p * |Q_2| + p', b, p * |Q_2| + q')$ dans T_i
fsi
si $a \bullet b \neq 0$ et $a \bullet b \in$ visible $[i_1, i_2]$
alors insérer $p * |Q_2| + p', a \bullet b, q * |Q_2| + q')$ dans T
fsi
fin
fin
fin
fin

3.6 Discussion

Pour déterminer la forme normale d'un système de transitions étiquetées composé par rapport à une relation d'équivalence, nous proposons des stratégies permettant de minimiser le nombre de transitions *temporaires* et le nombre d'états *temporaires* engendrés pendant la composition. Ce sont les états et les transitions engendrés puis supprimés. Ces stratégies sont fondées sur le calcul des formes normales des résultats *intermédiaires* et sur une "réorganisation" de l'expression définissant le système de transitions étiquetées composé. Ces stratégies sont justifiées par le fait que nous pouvons calculer efficacement la forme normale d'un système de transitions étiquetées, pour une relation d'équivalence donnée.

4. Conclusion

Nous avons présenté dans ce chapitre la réalisation des transformations sur les systèmes de transitions étiquetées présentées au chapitre I.

La mise en œuvre de la bisimulation telle qu'elle est faite dans ce travail, permet son application sur des systèmes de transitions importants avec des performances correctes. Ce calcul est de l'ordre de la minute pour un système de transitions étiquetées de 2 000 états et 10 000 transitions, et de l'ordre de l'heure pour 20 000 états et 100 000 transitions.

En pratique, nous avons remarqué que le temps de calcul d'une forme normale pour une relation d'équivalence est de l'ordre de celui du calcul de la plus grande bisimulation. En effet, les transformations coûteuses intervenant dans le calcul de la forme normale, comme le calcul de la fermeture transitive de la relation de transition étiquetée par τ, sont locales.

Cela nous a conduits, lors du calcul de la forme normale d'un système de transitions étiquetées composé, par rapport à une congruence, à proposer deux stratégies pour composer et réduire des systèmes de transitions étiquetées. Elles consistent à réduire les résultats intermédiaires en calculant leur forme normale. De plus, en utilisant les propriétés algébriques de l'opérateur de composition et de l'opérateur de restriction, nous pouvons encore minimiser la taille des résultats intermédiaires.

CHAPITRE III

LE SYSTEME ALDEBARAN

Plusieurs logiciels de spécification et d'analyse des systèmes ont été réalisés, comme par exemple les systèmes AUTO [LMV87], [Ver87], CESAR [Qu82],[Sc83], ECRINS [SM87], VENUS [Sor87].

Le système AUTO permet de construire des termes d'un calcul de processus et de calculer les automates associés à ces termes. Il permet de tester l'équivalence forte ou observationnelle de deux automates et de calculer le quotient d'un automate par un *critère d'observation régulier.*

En CESAR, la spécification d'un programme est exprimée par un ensemble de formules d'une logique, et le programme vérifie cette spécification s'il satisfait chacune de ses formules. La compilation du programme produit un graphe d'états. Le programme satisfait une formule si son graphe d'états est un modèle pour cette formule.

Le système ECRINS est conçu pour la manipulation de termes dans des calculs de processus communicants. La comparaison de termes est faite par une notion d'équivalence observationnelle. L'utilisateur a la possibilité de décrire le calcul de son choix en précisant pour chaque opérateur de l'algèbre sa sémantique opérationnelle.

Le système VENUS permet la description et la transformation de termes du calcul CCS, et leur comparaison modulo certaines relations d'équivalence. Ces relations d'équivalence sont la congruence forte, la congruence observationnelle et l'équivalence observationnelle. La comparaison des termes finis est réalisée par réécriture, la comparaison des termes réguliers est réalisée sur les systèmes de transitions étiquetées associés aux termes.

Nous proposons le système ALDEBARAN qui est conçu pour comparer et réduire des systèmes de transitions étiquetées de plusieurs milliers d'états. Le système ALDEBARAN permet de comparer deux systèmes de transitions étiquetées modulo une relation d'équivalence. Les relations définies sont la congruence forte, la congruence observationnelle, l'équivalence observationnelle et la congruence par modèle d'acceptation. Il permet de composer des systèmes de transitions étiquetées déjà définis en appliquant des

stratégies de réduction pendant la composition. Le système est conçu de telle sorte que d'autres relations d'équivalence puissent être définies.

Ce chapitre est organisé en quatre parties.

Nous présentons les objectifs et les contraintes qui nous ont conduits à la réalisation du système ALDEBARAN.

Dans une deuxième partie, nous présentons une description générale du système ALDEBARAN.

Nous décrivons certains aspects de la réalisation d'ALDEBARAN.

Nous donnons des exemples d'utilisation du système ALDEBARAN. Nous décrivons les interfaces avec les compilateurs LUSTRE et CÆSAR.

1 Objectifs et contraintes

Le système ALDEBARAN a été conçu pour répondre à plusieurs objectifs et contraintes :

a)*Description algébrique des systèmes répartis*

Un système réparti est un terme d'une algèbre décrite par un ensemble d'opérateurs (sa *signature*), et une sémantique opérationnelle. A chaque terme de l'algèbre est associé un système de transitions étiquetées par l'intermédiaire de la sémantique opérationnelle. Nous pouvons appréhender de deux manières différentes la description algébrique des systèmes répartis :

- Un méta-langage permet de définir une algèbre, en spécifiant la sémantique opérationnelle de chaque opérateur. Le système produit un analyseur syntaxique pour les termes de l'algèbre. C'est l'approche adoptée en ECRINS.

- Un système réparti a une structure hiérarchique. Chaque sous-système est soit un système réparti soit un processus séquentiel. Tout processus séquentiel peut être décrit par un terme de **Trans**. Nous avons vu au premier chapitre comment interpréter différents opérateurs de composition ou de communication. Un opérateur de composition parallèle, paramétré par une algèbre de synchronisation, est défini sur les systèmes de transitions étiquetées. Cette algèbre de synchronisation décrit les communications entre les systèmes.

Nous avons préféré la deuxième approche. Les opérateurs de base du calcul sont ceux de **Trans**. Ils permettent la description de tout système de transitions

étiquetées ayant un nombre fini d'états et de transitions. Nous ajoutons un opérateur de composition parallèle, paramétré par une algèbre de synchronisation L, et un opérateur de restriction pour obtenir $\mathbf{Trans_L}$.

b) *Définition de relations d'équivalence*

Les relations d'équivalence sont définies sur les termes d'une algèbre pour pouvoir transformer ou comparer deux termes. Nous nous intéressons aux relations d'équivalence plus grossières que la congruence forte et pour lesquelles il existe un algorithme calculant la forme normale d'un système de transitions étiquetées. Dans ces conditions, décider si deux termes sont équivalents, c'est comparer leurs formes normales par la plus grande bisimulation (i.e. la congruence forte). Les relations d'équivalence prédéfinies en ALDEBARAN sont la congruence forte, la congruence observationnelle, l'équivalence observationnelle et la congruence par modèle d'acceptation. De manière générale, toute relation d'équivalence plus grossière que la congruence forte et pour laquelle il existe un algorithme calculant la forme normale d'un système de transitions étiquetées, peut être ajoutée à ALDEBARAN. Un aspect qui nous semble important est de permettre l'extension du système ALDEBARAN par l'ajout de nouvelles relations d'équivalence. La définition d'une nouvelle relation d'équivalence peut ainsi être faite en créant un module contenant les transformations qui lui sont associées. Nous avons préféré cette façon de procéder à la génération automatique de transformations à partir de la définition équationnelle d'une relation d'équivalence.

c) *Facilité d'intégration*

Tout logiciel manipulant des systèmes de transitions étiquetées peut utiliser ALDEBARAN pour réduire et composer les systèmes de transitions étiquetées.

d) *Efficacité*

Dans le processus de comparaison de deux systèmes de transitions étiquetées, le calcul de la forme normale est une opération qui peut être coûteuse en temps. Nous avons étudié des algorithmes, de telle sorte que le coût du calcul d'une forme normale soit de l'ordre de la minute pour un système de transitions étiquetées de quelques milliers d'états.

2 Description générale du système

ALDEBARAN peut être utilisé de manière interactive ou de manière "silencieuse". Il possède les caractéristiques suivantes :

a)*Modularité des descriptions*

Un système de transitions étiquetées peut être obtenu soit à partir d'un terme de **Trans**, soit par composition de systèmes de transitions étiquetées existants. Dans ce dernier cas, une algèbre de synchronisation doit être définie. La composition de plusieurs systèmes de transitions étiquetées est faite en appliquant la stratégie **comp1** ou la stratégie **comp2**.

b) *Choix d'une relation d'équivalence*

Ce choix intervient lorsque l'utilisateur veut comparer deux systèmes de transitions étiquetées, composer plusieurs systèmes de transitions étiquetées ou, plus simplement, calculer la forme normale d'un système de transitions étiquetées modulo une relation d'équivalence.

c) *abstraction*

Un opérateur d'abstraction, combiné avec la congruence observationnelle, l'équivalence observationnelle ou la congruence par modèle d'acceptation, permet d'enrichir la notion d'équivalence entre systèmes.

d) *Conservation et désignation de l'information*

Des primitives de gestion d'objets sont disponibles pour créer, conserver, modifier ou supprimer des objets.

e) Facilité d'intégration

Le système ALDEBARAN peut être utilisé de manière "silencieuse" par d'autres logiciels manipulant les automates ou les systèmes de transitions étqiuetées. L'étiquette τ est désignée en ALDEBARAN par l'identificateur "i". Cela nous permet de vérifier des systèmes de transitions étiquetées issus de LOTOS. L'étiquette τ peut être redéfinie.

3 Réalisation

Dans ce paragraphe, nous présentons certains aspects de la réalisation du système ALDEBARAN. Nous avons utilisé le langage C et le générateur d'analyseur syntaxique YACC pour réaliser une maquette d'ALDEBARAN. La version actuelle de ce prototype est développée à la fois sur VAX 790 et sur une station de travail SUN 3. Le système d'exploitation est UNIX. Ces choix assurent la portabilité d'ALDEBARAN sur toute machine UNIX.

Architecture du système ALDEBARAN

Nous réalisons les fonctionnalités du système ALDEBARAN par huit modules :

- Module *Moniteur* : interprétant les commandes de l'utilisateur.
- Module *Gestion des environnements* : permettant la définition et la gestion des objets.
- Module *Système de transitions étiquetées :* ce module contient les fonctions de mise à jour des structures utilisées pour représenter ou transformer un système de transitions étiquetées.
- Module *Composition* : permettant à l'utilisateur de définir un produit de synchronisation puis de composer les systèmes de transitions étiquetées.
- Module *Partition* : contient les procédures pour calculer la partition la moins fine compatible avec une relation de transitions étiquetées. Ce module est utilisé par les procédures du module *Relations d'équivalence.*
- Module *Relations d'équivalence* : ce sont les transformations et les procédures de décision relatives à une relation d'équivalence qui constituent ce module.
- Module *Entrées / Sorties* : assure les communications entre le système ALDEBARAN et le système hôte.
- Module *Gestion mémoire* : gère la zone mémoire allouée aux programmes ALDEBARAN.

Les relations entre les modules sont représentées figure 1.

Notations

Dans ce chapitre, nous utilisons les notations suivantes pour décrire les aspects syntaxiques :

<xxx> symbole non terminal ;
xxx mot clé ;
[xxx] occurrence éventuelle de xxx ;
{xxx} occurrence éventuelle de xxx un nombre quelconque de fois ;
[xxx]+ occurrence de xxx un nombre strictement positif de fois.
Le caractère "|" désigne l'alternative et les symboles terminaux sont écrits en gras.

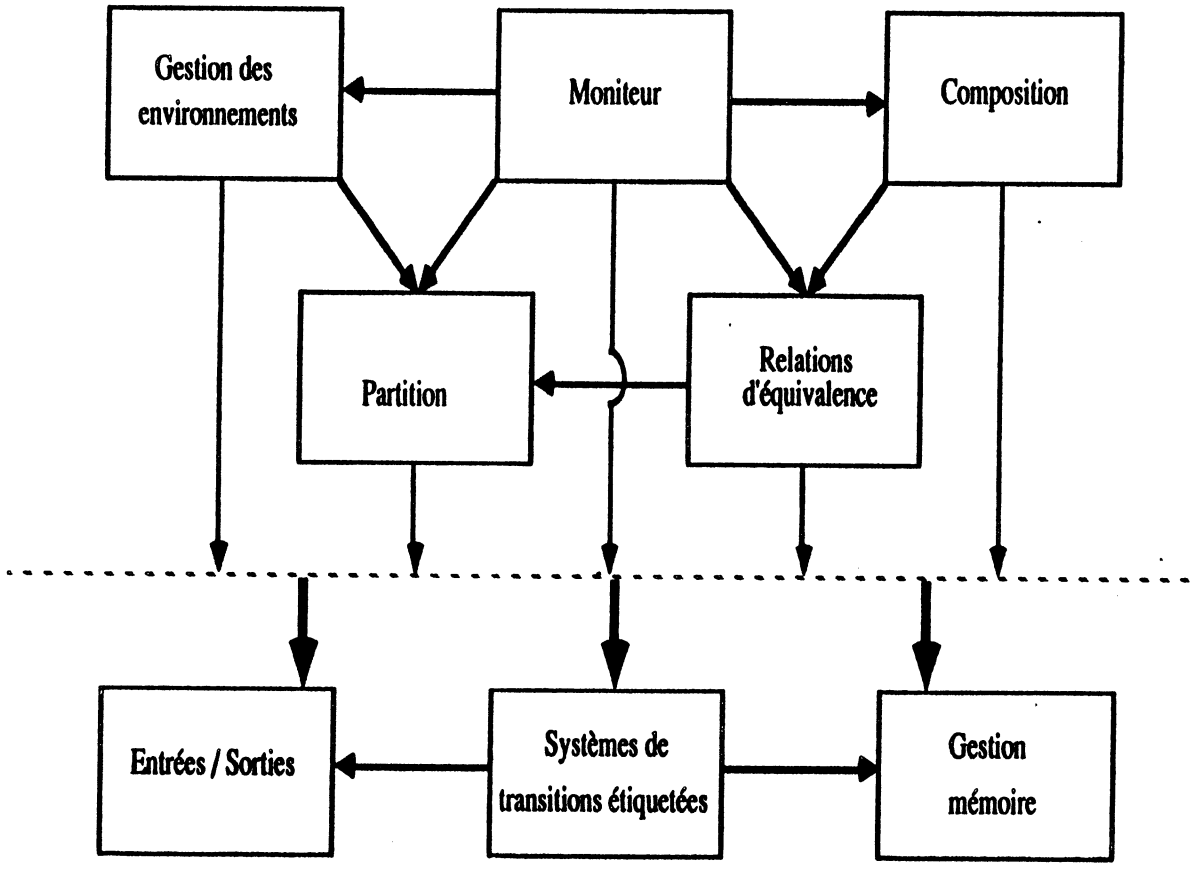


figure 1

Commentaires

Les modules *Entrées / Sorties*, *Systèmes de transitions étiquetées* et *Gestion mémoire* sont utilisés par tous les autres modules.

Nous présentons certains aspects des différents modules. Une description complète des fonctionnalités des modules est donnée dans le manuel utilisateur d'ALDEBARAN [Fer88] et résumée en annexe III.

Module *Moniteur*

C'est l'interprète de commandes. Il initialise le système ALDEBARAN et analyse la ligne de commande. Pour permettre différentes utilisations du système ALDEBARAN, celle-ci peut être constituée d'options et de paramètres. Deux options sont associées à chaque relation d'équivalence : une option de réduction "-r" (calcul de forme normale) et une option de comparaison "-c"(procédure de décision). Un paramètre est un fichier qui contient une description, dans le format ALDEBARAN, d'un système de transitions étiquetées. Lorsque l'option "-r" (resp. "-c") est présente, un paramètre (resp. deux paramètres) doivent être présents ; dans ce cas, la réduction (resp. la comparaison) est effectuée et l'on sort d'ALDEBARAN. Si aucune option n'est présente, les commandes d'ALDEBARAN sont exécutées de manière cyclique. Le cycle d'interprétation est le suivant : acquisition, analyse et exécution de la commande utilisateur. Il est initialisé par l'activation du module principal d'ALDEBARAN. Une erreur de commande replace le système en attente de commandes et une commande de terminaison interrompt le cycle.
La syntaxe de la ligne de commande est décrite de la manière suivante :

<ligne de commande> ::= **ALDEBARAN** <option-params>
<option-params> ::= **-r***rel* <fichier>**.aut** |
-c*rel* <fichier1>**.aut** <fichier2>**.aut** |
{<fichier>**.aut**}

La chaîne de caractères *rel* désigne une relation d'équivalence. Lorque la chaîne de caractères *rel* est égale à *cf* (resp. *co*, *ac*) elle désigne la relation de congruence forte (resp. observationnelle, par modèle d'acceptation).

Module *Gestion des environnements*

Un environnement est la donnée d'un ensemble de systèmes de transitions étiquetées. Il peut être défini de deux façons :

- soit par la commande **gen_env** (), qui récupère les paramètres de la ligne de commande ALDEBARAN ;

- soit par la commande **cre_env** (<editeur>), dont le rôle est la définition à l'aide d'un éditeur de texte, <editeur>, d'un ensemble de termes de **Trans**,

l'analyse de ces termes par l'analyseur syntaxique du système, et la génération d'un système de transitions étiquetées pour chaque terme d'une description de comportement.

a) Le langage de description des termes

Nous présentons dans ce paragraphe la syntaxe concrète permettant de définir des termes de **Trans**. Dans ce langage, deux types sont proposés :
- le type *étiquette*, qui est un sous-ensemble des chaînes de caractères constituées de chiffres, de lettres et des symboles '_', '.', '?', '-', '!'.

Un élément du type étiquette commence par une lettre.

L'étiquette τ est dénotée par l'identificateur "i". Une commande particulière permet de redéfinir l'étiquette τ :

def_tau (<idf_eti>)

Cette commande intervient avant la création d'un environnement. Dans une définition ultérieure d'un environnement, l'étiquette τ est identifiée par la chaîne de caractères *idf_eti*.

Une étiquette dont le premier caractère est "n" désigne une étiquette complémentaire lorsque l'algèbre considérée est CCS.
- le type *comportement*, qui est décrit par **Trans**, à ceci près que l'on remplace l'opérateur de point fixe **rec** X.t par une équation "X =t".

Le langage de description des termes est défini par la grammaire ci-dessous, dont l'axiome est le nom terminal <env>.

<env> ::= **definitions**
<idf> = part_d_def> {; <idf> = part_d_def>}
fin_definitions

<liste_idf> ::= <idf> {; <liste_idf>}
<part_d_def> ::= <part_d_def> + <part_pref> |
<part_pref>
<part_pref> ::= idf <part_d_def> |
nil |
(<part_d_def>)

Les opérations définies sur les environnements permettent de conserver, de visualiser et de détruire les environnements ou les systèmes de transitions étiquetées.

b) Analyse syntaxique et vérification du langage de description des termes

Un analyseur syntaxique est produit par YACC. Cet analyseur admet comme entrée une description de la syntaxe concrète du langage de description des termes, ainsi qu'un ensemble de programmes. Il vérifie qu' il n'y a pas de définition récursive non gardée dans les équations de la partie **définition**. Il produit en sortie :

- un descripteur d'environnement,
- un ensemble de systèmes de transitions étiquetées.

Le descripteur d'environnement contient les informations suivantes :

- le nom de l'environnement,
- une table des identificateurs contenant, pour chaque identificateur, un attribut.
- les listes des noms d'étiquette et des noms de comportements définis dans cet environnement.

c) Descripteur d'environnement

Le descripteur d'environnement contient les informations nécessaires à la conservation et à la modification des systèmes de transitions étiquetées. Il contient, notamment, une table des caractères, une table des identificateurs, une table contenant la description de chaque système de transitions étiquetées, une table contenant, pour chaque système de transitions étiquetées, son descripteur.

La table des identificateurs, **IDF**, est construite pendant l'analyse syntaxique d'un environnement (commande **cre_env**) ou pendant la génération de l'environnement (commande **gen_env**). Une fonction de "hashage" associe à chaque chaîne de caractères représentant un identificateur de comportement ou d'étiquette, un accès dans la table **IDF**. Le i-ième élément de **IDF** désigne un couple (nom, type, attribut).

Le nom est un accès à une table de chaînes de caractères, où sont rangées toutes les chaînes de caractères,
le type est étiquette ou comportement,
l'attribut est fonction du type de l'identificateur :

i) le type est *étiquette*, l'attribut est un couple (lc, code) où lc (resp. code) désigne la liste des comportements dans lesquels elle intervient (resp. le code de l'étiquette) : par convention, l'étiquette "i" est codée par 0.

ii) le type est *comportement*, l'attribut est un quintuplet (la, c, ty', trans,a) où :
- *la* est la liste des étiquettes intervenant dans la définition du comportement,
- *c* désigne le code dans la table **IDF** , dans laquelle elle est définie,
- *ty'* indique si le terme qui lui est associé est fermé ou ouvert,
- *trans* est un triplet (0, {(t, j)}) (resp. (1, t, j)) où j est un accès dans **IDF**, désignant le transformé de i par t, (resp. i est le transformé de j par t),
- *a* est un accès au système de transitions étiquetées.

La table **sys_trans** des systèmes de transitions étiquetées associe à chaque code de système de transitions étiquetées, la description de sa relation suivant son mode (i.e. description *directe* ou description *inverse*).

La table **desc_tab** des descripteurs associe à chaque entrée dans la table **sys_trans** un descripteur ; c'est un sextuplet
(état_initial, card_trans, card_état, card_act, t_present, mode) où,
- *état_initial* désigne l'état initial du système,
- *card_trans* désigne le cardinal de la relation de transitions,
- *card_état* désigne le cardinal de l'ensemble des états,
- *card_act* désigne le cardinal de l'ensemble des étiquettes,
- *t_présent* est vrai si i étiquette une transition, sinon est faux,
- *mode* indique si la relation de transitions étiquetées est représentée sous forme directe ou sous forme inverse.

Le descripteur d'environnement contient aussi deux tables **des_act** et **des_nom** associant à chaque code un indice dans la table de chaînes de caractères suivant que le type désigne une étiquette ou un système de transitions étiquetées.

d) Représentation de l'environnement

Il est représenté par un catalogue UNIX, constitué du fichier source, du fichier descripteur et d'un catalogue objet. Celui-ci contient, pour chaque système de transitions étiquetées associées à un terme, un fichier contenant le descripteur du système de transitions étiquetées et une liste de triplets représentant la relation de transitions étiquetées. Pour chaque triplet (i, j, k) où i et k sont des entiers positifs et j un entier, i et k désignent respectivement un état de départ et un état d'arrivée et j un code d'étiquette.

e) Le format ALDEBARAN

Le format de description des systèmes de transitions étiquetées comprend un descripteur de système ainsi que la liste exhaustive des transitions. Une transition est décrite par un triplet (q_1, a, q_2) où q_2 désigne le successeur de l'état q_1 par une transition étiquetée par a. Le descripteur du système de transitions étiquetées est un triplet (q_0, card_trans, card_état) où q_0 désigne l'état initial du système, *card_trans* désigne le nombre de transitions du système et *card_état* désigne le nombre d'états du système. Le format ALDEBARAN est décrit de la manière suivante :

<format_ALDEBARAN> ::= <descripteur> [<triplet>] +
<descripteur> ::= <état_initial>, <card_trans>, <card_état>
<triplet> ::= <état>, <idf>, <état>

Module *Systèmes de transitions étiquetées*

Dans ce module sont regroupées les fonctions d'accès aux systèmes de transitions étiquetées et de modification des systèmes de transitions étiquetées. Il est utilisé par tous les autres modules à l'exception des modules *Entrées / Sorties* et *Gestion mémoire.* Certaines fonctionnalités de ce module sont accessibles à l'utilisateur. Ce sont celles permettant d'associer un terme à un système de transitions étiquetées, de renommer certaines étiquettes d'un système de transitions étiquetées ou de faire *abstraction* de certaines étiquettes. La syntaxe est décrite dans le manuel utilisateur d'ALDEBARAN.

Module *Composition*

C'est le module qui génère un système de transitions étiquetées pour un comportement décrit par une *expression de composition*. Le langage de description des expressions de composition est défini par la grammaire ci-dessous, dont l'axiome est exp_comp. Les opérations d'abstraction, de composition et de restriction sont notées respectivement $, & et #.

<exp_comp> ::= <exp_comp> & <term_comp> |
<term_comp>
<term_comp> ::= <term_comp> # <liste_idf> |
<term_fact>
<term_comp> ::= <term_comp> $ <liste_idf> |
<term_fact>
<term_fact> ::= <idf> |
(<exp_comp>)

Une composition est possible uniquement si un produit de synchronisation a été défini par la commande suivante :

def_com (ty_com, oper_com)

Cette commande lit deux entiers *ty_com* et *oper_com*.
Les valeurs possibles de *ty_com* sont 0 ou 1. Si *ty_com* = 0 alors le produit de synchronisation est asynchrone (i.e. a • * = a). Si *ty_com* = 1 alors le produit de synchronisation est synchrone (i.e. a • * = 0).
Les valeurs possibles de *oper_com* sont 0, 1, 2, 3.
Si *oper_com* = 0, alors le produit de synchronisation est a • na = τ et a • b = 0, pour a ≠ b et a ≠ nb et b ≠ na.
Si *oper_com* = 1, alors le produit de synchronisation est a • a = a et a • b = 0, pour a ≠ b.
Si *oper_com* = 2, alors le produit de synchronisation est a • a = τ et a • b = 0, pour a ≠ b.
Si *oper_com* = 3, alors un argument supplémentaire *liste_triplet* est lu. *liste_triplet* est une liste de triplets $[(a_1, b_1, c_i), \ldots, (a_n, b_n, c_n)]$, où a_i, b_i, c_i pour $1 \leq i \leq n$, sont des étiquettes. Le produit de synchronisation est défini de la manière suivante : $a_i \bullet b_i = c_i$ pour $1 \leq i \leq n$.

On associe un système de transitions étiquetées à une expression de composition par la commande **comp** :

comp (sem, stratégie, idf, exp_comp)

Cette commande lit deux entiers *sem* et *stratégie*, un identificateur *idf* et analyse l'expression de composition *exp_comp*.
Les valeurs possibles de *sem* sont 0, 1 ou 2.
Si *sem* = 0 alors aucune forme normale n'est calculée pendant la composition.
Si *sem* = 1 alors les formes normales sont calculées pour la congruence forte.
Si *sem* = 2 alors les formes normales sont calculées pour la congruence observationnelle.
Les valeurs possibles de *stratégie* sont 0, 1 ou 2.
Si *stratégie* = 0 alors le système de transitions dénoté par *exp_comp* est calculé et associé à *idf*.
Si *stratégie* = 1 alors le système de transitions dénoté par *exp_comp* est calculé en appliquant la stratégie **comp1**, et associé à *idf*.
Si *stratégie* = 2 alors le système de transitions dénoté par *exp_comp* est calculé en appliquant la stratégie **comp2**, et associé à *idf*.

Remarque

Pour les deux commandes précédentes, l'expression de composition se trouve sur le flux d'entrée. Celui-ci peut être modifié (voir fonctionnalité du module *Entrées / Sorties*), ce qui permet de définir la composition soit de manière interactive soit par l'intermédiaire d'un fichier.

Module *Relations d'équivalence*

Ce module contient les transformations et les procédures de décision relatives aux relations d'équivalence. Les procédures de décision et les transformations prédéfinies sont celles relatives à la congruence forte, à l'équivalence observationnelle, à la congruence observationnelle et à la congruence par modèle d'acceptation. Ces fonctions opèrent sur les systèmes de transitions étiquetées associés aux termes. Ce module est organisé en sous-modules : à chaque relation d'équivalence correspond un sous-module.

Les fonctionnalités sont les suivantes :

A chaque relation prédéfinie sont associées deux commandes. La première permet de calculer la forme normale d'un système de transitions étiquetées modulo cette relation. La deuxième permet de décider si deux systèmes sont équivalents modulo cette relation.
Pour les autres relations d'équivalence à définir, deux commandes sont prévues, paramétrées par ces nouvelles relations : le calcul de forme normale et la procédure de décision.

Module *Partition*

L'algorithme de calcul de la partition la moins fine compatible avec la relation de transitions étiquetées est dans un module à part. Cet algorithme est en effet utilisé dans certains cas indépendamment du module sémantique (cf. interface avec le compilateur LUSTRE). Il est utilisé par le module *Relation d'équivalence*. Il contient aussi l'algorithme de construction du quotient.

Module *Entrées / Sorties*

Ce module contient les fonctions utilisées par les autres modules pour réaliser les communications entre ALDEBARAN et le système hôte. Il contient aussi des commandes permettant de rediriger le flux d'entrée ou de sortie :

- **open_in_stream** (<idf>)	ouverture d'un flux en entrée
- **open_out_stream** (<idf>)	ouverture d'un flux en sortie
- **close_stream** (<idf>)	fermeture d'un flux
- **status_in_stream**	liste des flux ouverts en entrée
- **status_out_stream**	liste des flux ouverts en sortie
- **close_all**	fermeture de tous les flux

Module *Gestion mémoire*

Ce module gère l'espace mémoire utilisé par le système ALDEBARAN.

4 Exemples d'utilisation

Nous considérons l'exemple du "scheduler", pour comparer les stratégies **comp1** et **comp2.**

Nous illustrons l'utilisation du système ALDEBARAN par d'autres systèmes existants. Nous avons envisagé la communication avec d'autres systèmes existants de deux manières différentes :

a) *L'intégration partielle du système ALDEBARAN dans un autre système*

Une tentative dans ce sens a été réalisée avec le système VENUS [Sor87]. Un prototype de VENUS a été programmé en C-Prolog (version1.5) sous UNIX 4.2, supporté par un VAX 790. Nous avons transformé le source de VENUS en Prolog de Delphia [Del87] qui permet une édition de liens avec des modules écrits en langage C. Nous avons écrit un interface qui, à partir de clauses Prolog définissant un système de transitions étiquetées, construit un système de transitions étiquetées dans la forme interne d'ALDEBARAN et réciproquement. Les modules contenant les relations d'équivalence ont pu être ajoutés à VENUS.

b) *La communication par l'intermédiaire de fichiers textes*

Pour ce faire, nous avons défini un format de représentation des systèmes de transitions étiquetées, appelé format ALDEBARAN. Des interfaces ont été réalisés avec LUSTRE et CÆSAR. Un interface est actuellement développé entre ALDEBARAN et XESAR.

4.1. Le "Scheduler"

Nous illustrons, par cet exemple, une utilisation interactive d'ALDEBARAN, et une comparaison des stratégies **comp1** et **comp2.** Nous proposons une amélioration de la stratégie **comp2.**

Reprenons l'exemple du "Scheduler" développé au chapitre II. Nous voulons montrer :

$Sch' \sim_{co} (((S \,\&\, C_1 \,\&\, C_2) \,\#\, \{g_1, ng_1, g_2, ng_2\}) \,\&\, R_1 \,\&\, R_2) \,\#\, \{b_1, nb_1, b_2, nb_2\}$.

Les étiquettes ng_1, ng_2, nb_1 et nb_2 remplacent respectivement les étiquettes g'_1, g'_2, b'_1 et b'_2 dans l'expression de composition. Chacun des systèmes de transitions étiquetées est dans un fichier suffixé par ".aut", au format

ALDEBARAN.
Après l'exécution de la commande UNIX,

Aldebaran S.aut C_1.aut C_2.aut R_1.aut R_2.aut Sch'.aut

l'utilisateur se trouve dans l'environnement ALDEBARAN. Le système signale qu'il est en attente de commande en affichant la chaîne de caractères "ALDEBARAN>". La session se déroule de la manière suivante. Les messages affichés par ALDEBARAN sont en italiques :

ALDEBARAN> **gen_env**

Génération de l'environnement qui est constitué des cinq systèmes de transitions étiquetées décrits dans les fichiers.

ALDEBARAN>**def_com**

choix du type de communication, 0 : asynchrone, 1 synchrone
0
choix de l'opérateur de communication :0, 1, 2 ou 3
0

Commentaire : nous avons choisi l'opérateur de composition de CCS.

ALDEBARAN>**comp**

Préciser la sémantique :
0, 1 (forte), 2 (observationnelle)
2
Préciser la stratégie :
0, 1 (en réordonnant), 2 (sans réordonner)
2
Sch $(((S \,\&\, C_1 \,\&\, C_2) \,\#\, \{g_1, ng_1, g_2, ng_2\}) \,\&\, R_1 \,\&\, R_2) \,\#\, \{b_1, nb_1, b_2, nb_2\}$

Commentaire : Après l'analyse de l'expression de composition, un système de transitions étiquetées est généré en appliquant la stratégie **comp2**. Il est associé, dans l'environnement, à l'identificateur Sch.

ALDEBARAN>**décide_eo** Sch Sch'

systèmes équivalents modulo l'équivalence observationnelle

ALDEBARAN>**quit**

L'utilisateur est dans l'environnement UNIX.

Comparaison des stratégies **comp1** et **comp2**

Nous comparons les stratégies **comp1** et **comp2** du point de vue des temps d'exécution. Nous obtenons les résultats suivants :

Nombre de cyclers	temps en seconde comp2	temps en seconde comp1
2	1	1.21
3	1.11	4.03
4	2.02	13
5	4	105.31
6	6.24	saturation espace memoire
10	97.47	

Nous constatons qu'à partir de six "cyclers", nous ne pouvons plus appliquer la stratégie **comp1**.

Amélioration de la stratégie **comp2**

Etant donné une expression de composition, e $A, l'opérateur d'abstraction $ est descendu le plus "profond" possible dans l'arbre associé à e $ A. Pour cela, nous appliquons les transformations suivantes :

Si $e = e_1 \,\&\, e_2$ et $(e \,.\, V) \cap A = \emptyset$ **alors** $(e_1 \,\&\, e_2)\ \$\ A \rightarrow (e_1 \,\#\, A) \,\&\, (e_2 \,\#\, A)$ **fsi**
si $e = e_1\ \$\ A'$ **alors** $(e_1\ \$\ A')\ \$\ A \rightarrow e_1\ \$\ (A \cup A')$ **fsi**
si $e = e_1 \,\#\, R$ et $A \cap R \cap (e_1 . V) = \emptyset$ **alors** $(e_1 \,\#\, R)\ \$\ A \rightarrow (e_1\ \$\ A) \,\#\, R$ **fsi**

4.2. Interface avec LUSTRE

Dans ce paragraphe, nous présentons l'interface entre le compilateur LUSTRE et ALDEBARAN, et un exemple illustrant son fonctionnement. Le langage LUSTRE est décrit en annexe I. Le compilateur LUSTRE associe un

automate à un programme. Le langage de description des automates est le *code portable*. Le code portable [CPS86] est la sortie commune aux compilateurs LUSTRE et ESTEREL. Il décrit les automates produits par l'un ou l'autre des compilateurs. Un automate est décrit par des couples (état, liste des actions réalisées dans cet état) ; les actions réalisées dans un état sont mémorisées sous forme d'arbre : chaque nœud correspond à un test et chaque feuille à un branchement vers un nouvel état. L'expérience montre qu'en utilisant ESTEREL l'automate produit est généralement proche de l'automate minimal (par rapport à la plus grande bisimulation), ce qui semble ne pas toujours être le cas en LUSTRE : ceci est dû au fait que LUSTRE est un langage déclaratif ; le style de programmation d'un langage déclaratif n'est pas contraint par des soucis d'efficacité. L'utilisation d'ALDEBARAN permet de réduire la taille du code produit par le compilateur LUSTRE. Pour ce faire, un interface a été développé entre ALDEBARAN et LUSTRE par A.C. Glory et C. Ratel [GR87] :

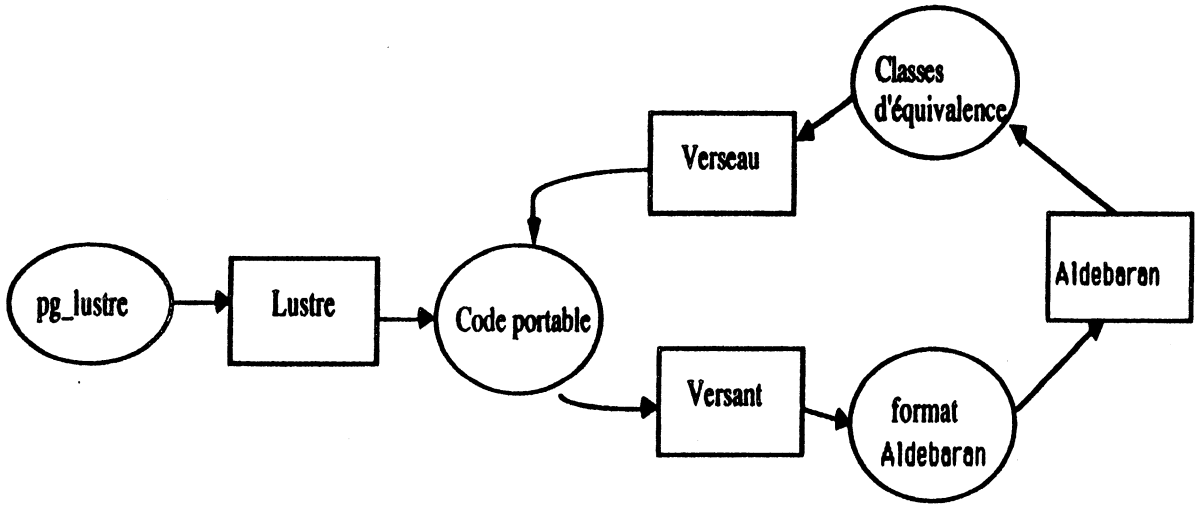


figure 2

VERSANT transforme une description d'automate faite à l'aide du code portable en une description d'automate au format ALDEBARAN. Cette transformation est réalisée en concaténant les éléments de la liste des actions (tests compris) conduisant d'un état à un autre.

VERSEAU met à jour le code portable à l'aide des classes d'équivalence.

Le cycle de la figure précédente peut être exécuté plusieurs fois (cf exemple). Une fois la minimisation effectuée par ALDEBARAN, il se peut qu'à partir d'un état, quel que soit le résultat du test, l'état d'arrivée soit le même. Dans ces conditions, le test n'est plus nécessaire. Le code portable est alors tranformé pour supprimer les tests inutiles et une minimisation par ALDEBARAN est de nouveau réalisée. Le cycle précédent est effectué tant que l'on peut minimiser le nombre des états. L'exemple suivant illustre le processus cyclique de minimisation.

Exemple

Soit le programme LUSTRE suivant :

```
node test (c : bool)
    returns (s, x : bool) ;
var état0, état2, état3, état4 : bool ;
let
    état0 = true -> false ;
    état2 = false -> pre état0 and c or pre état4 ;
    état3 = false -> pre état0 and not c ;
    état4 = false -> pre état3 or pre état2 ;
    s = (état0 or état4) and not état2 and not état3 ;
    x = true ;
tel
```

La représentation graphique du code portable est donnée par la figure ci-dessous.

Nous définissons les étiquettes suivantes :

a étiquette l'instruction {output x ; s := 1 ; output s} et

b étiquette l'instruction {output x ; s := 0 ; output s}

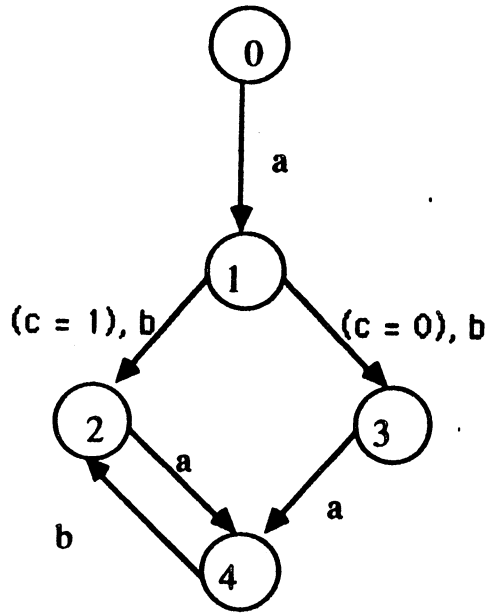


figure 3

Le calcul de la plus grande bisimulation sur cet automate produit les classes suivantes :

{0}, {1}, {2, 3}, {4}

Le test évaluant la valeur de c n'est plus nécessaire puisque quel que soit le résultat de son évaluation, la séquence d'actions qui suit conduit à des états équivalents. On assimile les actions (c = 0), b et (c = 1), b à l'action b :

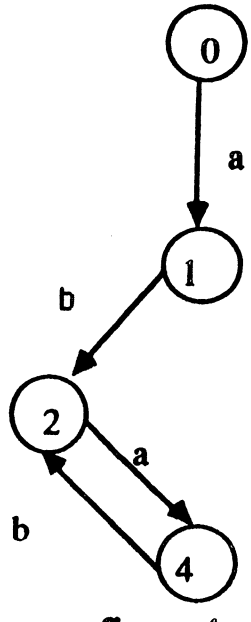


figure 4

Le calcul de la plus grande bisimulation sur cet automate produit les classes :

{0, 2}, {1, 4}.
L'automate réduit est :

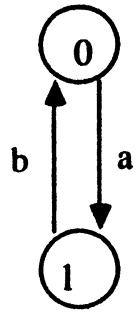


figure 5

Bien entendu, l'utilisation du système ALDEBARAN dans son entier n'est pas nécessaire. C'est pourquoi nous avons écrit un interface permettant d'utiliser uniquement les modules nécessaires pour la minimisation de l'automate :
le module *Partition*, le module *Système de transitions étiquetées*, le module *Entrées / Sorties* et le module *Gestion mémoire*.

4.3 Interface avec CÆSAR

Cæsar [Gar87] est un logiciel qui transforme une description LOTOS contenue dans un fichier de nom suffixé par ".lotos" en un réseau de Pétri interprété ou en graphes. Différents formats de graphes sont possibles, en particulier le format ALDEBARAN ".aut".

LOTOS [Lot87] (Language of Temporal Ordering Specification) est un langage de spécification conçu pour la définition formelle des protocoles et des services dans le cadre des réseaux. Actuellement, LOTOS est l'un des outils de description formelle employée par l'ISO.

LOTOS est une extension du langage CCS. Le concept de base est la définition des relations existant entre les événements dans le comportement observable du système. Un autre aspect de LOTOS est l'utilisation des types abstraits basée sur le langage ACT ONE, pour la représentation des structures de données. Nous n'avons pas étudié cet aspect.

L'intérêt d'un interface avec LOTOS est la possibilité de vérifier des protocoles de communication avec notre système. Jusqu'à présent,

ALDEBARAN a permis de comparer les automates produits par deux versions du compilateur modulo la congruence forte. Nous envisageons deux possibilités de coopération avec Cæsar. La première concerne la vérification de spécifications LOTOS à l'aide des relations d'équivalence définis en ALDEBARAN.
La deuxième porte sur la récupération des descriptions d'automates avant composition, pour les composer à l'aide des stratégies définies au chapitre II.

Plus d'une centaine d'automates produits par Cæsar ont été traités par ALDEBARAN. Le plus gros de ces automates a 28836 transitions et 5444 états. Le temps mis pour traiter cet exemple est de l'ordre de 245 secondes.

Exemple de vérification : le protocole du bit alterné

Nous présentons une vérification du protocole du bit alterné. La description en LOTOS a été faite par H. Garavel et est donnée en annexe II.

Le protocole du bit alterné est un protocole de transmission qui permet le transfert de données entre deux entités, pour lesquelles une connexion bi-directionnelle a été établie. Pour simplifier le problème, nous considérons une entité émettrice E et une entité réceptrice R.

a) Spécification

Dans cette modélisation, la valeur du message n'est pas significative. Le comportement global du système doit satisfaire les contraintes suivantes :

- un message émis par E doit être reçu par R ;
- l'ordre des messages est préservé.

On peut spécifier le comportement du système par le système de transitions étiquetées (figure 7) et le terme suivant :

Spec_bitalt = PUT GET Spec_bitalt.

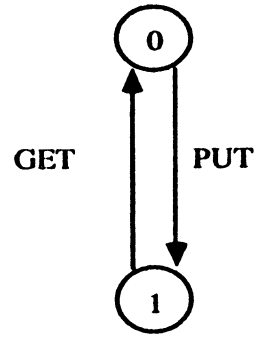


figure 6

Commentaires : L'étiquette *PUT* représente l'émission de E vers R. L'étiquette *GET* représente la réception par R d'un message en provenance de E.

b) Réalisation

Le protocole est constitué d'un émetteur, d'un récepteur et de deux médiums (figure 7).

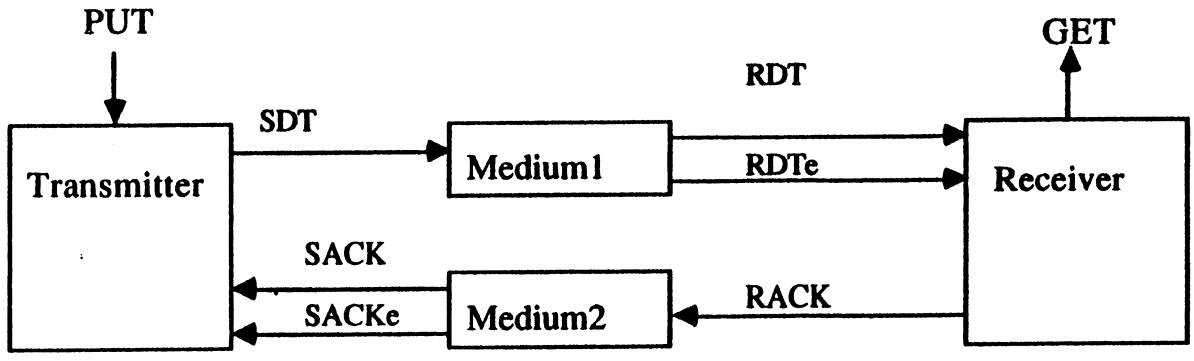


figure 7

L'émetteur (Transmitter) prend en charge les messages en provenance de E, les transmet à travers la ligne Medium1 au récepteur (Receiver) qui les transmet vers E. Ensuite, le récepteur transmet à travers la ligne Medium2 un acquittement. On considère ces lignes non fiables : il est possible qu'un message ou un acquittement soit perdu. Dans ce cas le médium peut signaler cette perte au destinataire.

Pour détecter les pertes non signalées, les messages et les acquittements contiennent un bit de contrôle. Le bit de contrôle de chaque acquittement est égal au bit de contrôle du message qu'il acquitte. Les bits de contrôle de deux

messages successivement émis ont des valeurs distinctes.

Si l'émetteur reçoit une indication de perte de message ou un message avec un bit de contrôle erroné, la ligne Medium1 réémet le dernier message envoyé.

Si le récepteur reçoit une indication de perte d'acquittement ou un acquittement avec un bit de contrôle erroné, la ligne Medium2 réémet le dernier acquittement envoyé.

c) Vérification

L'automate *Bitalt* produit par le compilateur LOTOS a 244 états et 710 transitions. Nous montrons, à l'aide d'ALDEBARAN, qu'il est observationnellement équivalent à *Spec_bitalt* :

La vérification a été conduite de la manière suivante : Nous essayons dans l'ordre suivant, la congruence forte, la congruence observationnelle et l'équivalence observationnelle.

Aldebaran Bitalt.aut Spec_bitalt.aut

ALEDBARAN>**gen_env**
ALEDBARAN>**time**

Commentaire : Indique le temps d'exécution de chaque commande qui suit la commande **time**. Ce temps d'exécution est décomposé en temps utilisateur et en temps système. Le temps utilisateur et le temps système sont indiqués en secondes. Le temps utilisateur est suffixé par "u", le temps système est suffixé par "s".

ALDEBARAN>**decide_f** Bitalt Spec_bitalt
Systèmes non équivalents modulo la congruence forte
2.18u 0.8s
ALDEBARAN>**decide_co** Bitalt Spec_bitalt
Systèmes non équivalents modulo la congruence observationnelle
3.3u 0.8s
ALDEBARAN>**decide_eo** Bitalt Spec_bitalt
Systèmes équivalents modulo l'équivalence observationnelle
3.0u 0.12s
ALDEBARAN>**nco** Bitalt
3.8u 0.11s
ALDEBARAN>**no_time**

Commentaire : cette commande annule l'effet de la commande **time.**

ALDEBARAN>**pr_ste** Bitaltnco

Commentaire :on obtient le système de transitions étiquetées (figure 8). L'étiquette *i* initiale correspond au fait que le message entre Medium1 et la ligne peut être perdu.

ALDEBARAN>**time**

ALDEBARAN>**nac1** Bitalt

3.18u 0.12s

ALDEBARAN>**pr_ste** Bitaltnac1

Commentaire : on obtient le système de transitions étiquetées (figure 9).

ALDEBARAN>**quit**

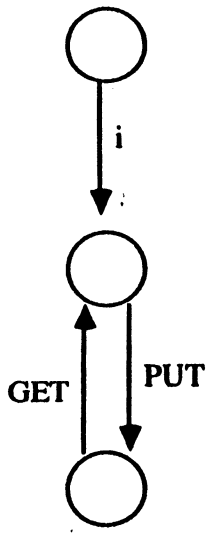


figure 8

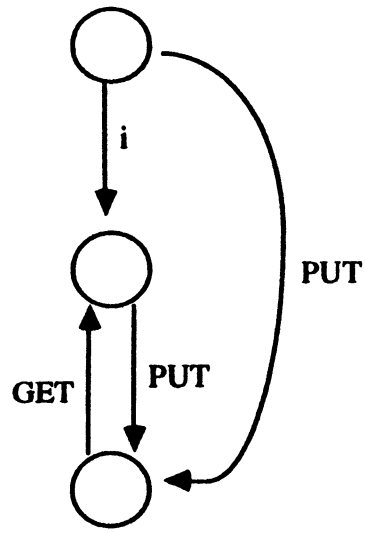


figure 9

4.4 Bilan et évaluation

Dans ce travail, nous avons cherché à réaliser de manière efficace les calculs de formes normales et les procédures de décision. Cela nous a permis de proposer des stratégies de réduction pour calculer la forme normale d'un système de transitions étiquetées composé. La contrainte d'efficacité nous a conduits, après expérimentation, à éliminer des environnements de programmation tels que ML ou Prolog. En effet, nous avons été limités à la fois en place mémoire (environ 2000 transitions en ML) et en temps d'exécution. De plus, la portabilité est incertaine avec ces environnements.

a) Expérimentation d'ALDEBARAN

ALDEBARAN a été expérimenté essentiellement de manière "silencieuse". Il est utilisé pour minimiser des automates (CÆSAR, LUSTRE) ou pour comparer des automates (CÆSAR).

Le plus gros exemple traité, issu de LOTOS, consistait à comparer deux systèmes de transitions étiquetées modulo la congruence forte.
Le premier système de transitions étiquetées a 28836 transitions et 5444 états ; le deuxième système de transitions étiquetées a 9728 transitions et 3584 états.
Le temps mis pour comparer ces deux systèmes fut 245 secondes.

Nous donnons ci-dessous deux tableaux. Le premier donne des ordres de grandeur du temps nécessaire à la construction d'un système de transitions étiquetées en mémoire suivie de l'exécution de l'algorithme de partition. Le deuxième est une évaluation du temps d'exécution de la commande **nco**. On applique successivement cette commande à la composition de 2, 3, 4 et 5 "cyclers".

états	transitions	temps (s)
200	1000	1. 5
1000	5000	18
10 000	50 000	1412
20 000	100 000	5512

nombre de cylers	états	transitions	temps (secondes)
2	25	47	0.36u 0s
3	125	320	27.50u 1.49s
4	625	2025	34.13u 0.49s
5	3125	12250	322.32u 1.13s

b) Interface avec Xésar

Actuellement un interface avec Xésar est en cours de réalisation. Nous avons atteint les limites de la maquette en traitant un exemple issu de Xésar : nous avons analysé le système de transitions étiquetées produit à partir de la description Estelle / R du protocole de Stenning [RRSV87]. Ce protocole est une extension du protocole du bit alterné : l'ordre des messages n'est pas nécessairement préservé. Ce système de transitions étiquetées, que nous notons *stenning*, a 13716 états et 28338 transitions. Nous avons tenté de réduire *stenning* modulo la congruence observationnelle. Après exécution de la transformation τ-saturation, nous avons obtenu un système de transitions étiquetées *st'* ayant 230 436 transitions. Le calcul de la plus grande bisimulation sur *st'* a provoqué une saturation de l'espace mémoire. La solution que nous avons retenue est la suivante :
Nous minimisons *stenning* modulo la congruence forte. Le résutlat obtenu est minimisé modulo la congruence observationnelle.
Nous nous intéressons aux actions *IN*, *OUT*, et *IN_TR_LOS*. Par analogie avec le protocole du bit alterné, les étiquettes *IN* et *OUT* correspondent aux étiquettes *PUT* et *GET*. Nous faisons abstraction des autres actions, à savoir *TR_LOS*, *RT_LOS* et *chi*. Nous minimisons le résultat de l'abstraction modulo la congruence par acceptation.

aldebarnan stenning.aut
ALDEBARAN>**gen_env**
ALDEBARAN>**time**
ALDEBARAN>**ncf** stenning
946.7u 14.2s
Commentaire : Le système de transitions étiquetées produit, noté *stenningnf* a 1424 états et 2906 transitions.
ALDEBARAN>**nco** stenningnfnco
1082u 16.2s
Commentaire : Le système de transitions étiquetées produit, noté *stenningnfnco* a 428 états et 878 transitions.
ALDEBARAN>**abst** sten_abs stenningnfnco $ [TR_LOS, RT_LOS, chi] ;;
11u 0.8s

ALDEBARAN>**nco** sten_abs

39.39u 0.56s

Commentaire : Le système de transitions étiquetées produit, noté *sten_absnco* a 167 états et 476 transitions.

ALDEBARAN>**nac1** sten_absnco

78.50u 1.17s

Commentaire: Le système de transitions étiquetées produit, noté *steabsnfnconac1* a 8 états et 25 transitions. Cela nous a permi de vérifier que le séquencement des entrées et celui des sorties sont corrects.

Remarque

Nous avons envisagé une extension du système ALDEBARAN pour mettre en œuvre de nouvelles relations d'équivalence définies par un ensemble d'axiomes (i.e. un ensemble d'équations). Pour cela, nous adoptons une approche système de réécriture. A chaque équation $t_1 = t_2$, nous associons deux tranformations sur **Trans**, *gauche* : $t_1 \rightarrow t_2$ et *droit* :$t_2 \rightarrow t_1$.
La transformation *gauche* associe à tout terme t, tel qu'il existe une *instanciation* σ vérifiant $\sigma(t_1) = t$, le terme $\sigma(t_2)$.
De même, la transformation *droit* associe à tout terme t, tel qu'il existe une *instanciation* σ vérifiant $\sigma(t_2) = t$, le terme $\sigma(t_1)$.

Nous avons conçu et réalisé en langage ML un module de manipulation de transformations sur les termes. Pour réaliser ce module, nous nous sommes inspirés d'un logiciel que nous avons écrit en ML, en collaboration avec A. Bouajjani. Le rôle de ce logiciel est le calcul d'une forme normale pour un terme d'une logique arborescente. Les fonctionnalités de ce module sont réalisées par les opérateurs suivants sur les fonctions, [ML85] :

then_fun,
or_else_fun,
fail_fun,
repeat_fun,
first_fun,
every_fun.

Pour contrôler l'exécution d'une combinaison de transformations, on peut forcer celles-ci à échouer à l'exécution. Lorsqu'une transformation ne

s'applique pas à un terme alors elle retourne la valeur *échec*, sinon elle retourne la valeur *succès*.
En utilisant une lambda-notation, on peut définir des opérateurs de la manière suivante :
f then_fun g = $\lambda x.\ g\ (f\ (x))$
f or_else_fun g = $\lambda x\ .f\ (x)$, si $f\ (x) \neq$ *échec*
= $\lambda x\ g\ (x)$ sinon
fail_fun f = *échec*
repeat_fun f = (*f then_fun* (*repeat_fun f*)) *or_else I*, où *I* désigne l'identité.
first_fun fl x applique à x la première fonction qui "n'échoue pas" de la liste de fonctions *fl*
every_fun fl x compose les fonctions de la liste *fl* pour former une fonction qui s'applique à x.

Nous n'avons pas, pour le moment, retenu cette solution dans la mesure où elle est inefficace. Pour définir une nouvelle relation d'équivalence, on modifie ALDEBARAN en rajoutant un nouveau module.

5 Prolongements

Nous pensons qu'une expérimentation sur de véritables protocoles de communication sera intéressante pour évaluer les stratégies de composition des systèmes de transitions étiquetées. D'autres critères peuvent être pris en compte, comme par exemple le nombre de transitions étiquetées par une action. D'autre part, il nous semble nécessaire qu'il y ait une plus grande interaction entre l'utilisateur et le processus de génération d'un système de transitions étiquetées à partir d'une expression de composition.

Une amélioration d'ALDEBARAN qui nous semble importante, est un diagnostic plus explicite lorsque deux systèmes ne sont pas équivalents, pour une relation d'équivalence donnée. Par exemple, un contre-exemple peut être fourni sous la forme d'un chemin partant de l'état initial et conduisant à des états qui ne sont pas équivalents.

D'autres relations d'équivalence seront étudiées (notamment la "testing equivalence") et intégrées dans le système ALDEBARAN.

CONCLUSION

L'objectif de ce travail a été la définition et la mise en œuvre de méthodes pour la description et la vérification des systèmes répartis. Partant d'un langage de termes et d'une sémantique opérationnelle, une relation d'équivalence peut être interprétée sur les systèmes de transitions étiquetées.

Les relations d'équivalence que nous avons étudiées sont la congruence forte, la congruence observationnelle, l'équivalence observationnelle et la congruence par modèle d'acceptation. La congruence forte est la plus petite de toutes ces relations d'équivalence. Chacune de ces relations est caractérisée par l'existence d'un représentant canonique pour chaque classe d'équivalence. Du fait que la congruence forte est la plus fine, ce représentant est minimal en nombre d'états.

La première partie de ce travail a consisté à définir, pour chacune des relations d'équivalence, un procédé de calcul de forme normale. Nous avons défini un opérateur d'abstraction sur les systèmes de transitions étiquetées. Cet opérateur, combiné avec les relations de congruence observationnelle, d'équivalence observationnelle et de congruence par modèle d'acceptation, permet de définir d'autres notions d'équivalence entre les systèmes de transitions étiquetées.
Nous avons proposé un opérateur de composition parallèle paramétré par une relation de communication. Cela permet d'exprimer différents modes de composition tels que la composition synchrone ou asynchrone, le rendez-vous à deux ou le rendez-vous multiple. L'opérateur de composition parallèle est paramétré par une relation de communication. Cette relation binaire est structurée par un produit de synchronisation qui décrit le résultat d'une synchronisation entre deux étiquettes. Nous avons vu que la congruence forte et la congruence observationnelle sont compatibles avec l'opérateur de composition.

Dans la deuxième partie de ce travail, nous avons présenté des algorithmes efficaces réalisant les transformations associées aux relations d'équivalence. Deux sortes de transformations on été distinguées : les transformations globales et locales.

La transformation globale que nous avons considérée est celle qui calcule la relation d'équivalence la moins fine compatible avec la relation de transitions étiquetées, à partir d'une relation d'équivalence donnée. Nous appliquons cet algorithme au calcul de la forme normale par rapport à la congruence forte. Cet algorithme est également applicable lorsque la relation d'équivalence initiale n'est pas la partition universelle, mais peut être, par exemple, définie par un ensemble de prédicats.

Les algorithmes associés aux transformations locales sont des algorithmes sur les graphes. La méthode qui s'applique à toutes les transformations locales est la suivante : parcours de la relation de transitions étiquetées, et pour chaque transition vérifiant une certaine propriété, l'action spécifique correspondant à la transformation est exécutée.

Nous avons étudié des stratégies pour minimiser le nombre de transitions et d'états temporaires pendant la composition de plusieurs processus.

Dans la troisième partie, nous avons présenté une implantation de ces algorithmes en langage C. Cela a conduit à la réalisation du système ALDEBARAN. Les raisons qui nous ont conduits à choisir le langage C plutôt que les environnements de programmation ML ou Prolog sont l'efficacité et la portabilité.

Diverses voies sont envisagées pour poursuivre ce travail. Nous pensons qu'il serait intéressant, lorsque deux systèmes ne sont pas équivalents modulo une relation d'équivalence, d'exhiber un contre-exemple. Ce peut être la construction de séquences d'exécution partant des états initiaux des deux systèmes et conduisant à des états qui ne sont pas équivalents. Une autre extension prévue est l'étude et la programmation d'autres relations d'équivalence contenant la congruence forte. Enfin l'utilisation du système ALDEBARAN devrait permettre de définir d'autres stratégies pour composer les systèmes de transitions étiquetées.

REFERENCES

[AHU74] Aho A., Hopcroft J., Ullman J., "Design and analysis of computer Algorithms, Addison Wesley, 1974.

[BCG87] G. Berry, Ph. Couronné, G. Gonthier, "Programmation synchrone des systèmes réactifs en langage ESTEREL", TSI, april 1987.

[BK85] J.A. Bergstra, J.W. Klop "Algebra of communicating system with abstraction", TCS 37, 1985, p77-121

[BKO] J.A. Bergstra, J.W. Klop, E.R. Olderog, "Readiness and failures in the algebra of communicating processes", Report CS-R8609, Centre for Mathematics and Computer Science, Amsterdam.

[Bou85a] G. Boudol, "Le calcul Meije", Parallélisme, Communication, Synchronisation, édité par J.P. Verjus et G. Roucairo, CNRS, 1985.

[Bou85b] G. Boudol, "Calcul de processus et vérification", rapport INRIA, RR424, 1985.

[CPH87] P. Caspi, D. Pilaud, N. Halbwachs, J.A. Plaice, "LUSTRE, a declarative language for programming synchronous systems", 14th ACM Symposium on Principles of Programming Languages, Munich, jan. 1987.

[CPS86] P. Couronné, J.A. Plaice, J.B. Saint, " Le format portable LUSTRE ESTEREL", non publié, 1986.

[Del87] Prolog de Delphia, Manuel de référence, 1987.

[EB87] E.H. Eertink, E. Brinksma, "The implementation of a testderivation algorithm", SEDOS /C2/N82, University of Twente, 6/10/87.

[Fer87] J. Cl. Fernandez, "ALDEBARAN 1.1., Manuel utilisateur", 1987.

[Gar87] H.Garavel, "CÆSAR 1.1., User Manual", à paraître 1987.

[GR87] A.C. Glory, C. Ratel, "Versant, Verseau, Lesar : Manuel utilisateur", nov. 1987, non publié.

[GS86a] Graf S., Sifakis J., "Readiness Semantics for Regular Processes with Silent Actions", rapport technique César n° 3, Laboratoire de Génie Informatique, Grenoble, nov 86.

[GS86b] Graf S., Sifakis J., "An expressive logic for a Process Algebra with Silent Actions", rapport technique César n° 4, Laboratoire de Génie Informatique, Grenoble, dec. 86.

[HBR] Hoare CAR, Brookes SD and Roscoe AW, "A theory of communicating Processes", JACM 31.

[Hil87] M. Hillerström, "Verification of CCS-processes", AALBORG Universitetscenter, 1987.

[KS83] Kanellakis P., and Smolka S., "CCS Expressions, Finite State Processes, and three problems of Equivalence". Proceeding ACM Symp. on principles of Distribued Computing, 1983.

[LMV87] Lecompte V., Madelaine E., Vergamini D., "AUTO, un système de vérification de processus parallèles et communicants", rapport technique INRIA 83, Mars 1987.

[Lot87]LOTOS , A Formal Description Technique based on the temporal Ordering of Observational Behaviour. International Organization for Standardization , Info. Proc. Sys - OSI-. Draft International Standard 8807-ANSI (USA), Jul 87.

[Mil80] Milner R "A calculus of communicating systems", LNCS 92, Springer Verlag 1980.

[Mil83] Milner R, "A calculi for synchrony and asynchrony", TCS 25, 1983, p267-310.

[Mil85] Milner R, "A complete axiomatisation for observational congruence of finite state behaviours", manuscript, Uni. of Edimburgh, 1985.

[ML85] "The ML Handbook", Version 6.1., INRIA, July 1985.

[Par81] Park D. "Concurrency and Automata on infinite sequences", LNCS 104, Theorical Computer Science, 5 th Gl. Conference, Karlsruhe, March 81.

[Plo81] Plotkin G.D. "A structural approach to operational semantics", Aarhus University , DAIMI FN-19, Sept. 81.

[Pnu77] Pnueli A., "The temporal logic of concurrent programs", 19th FOCS, 1977.

[PT86] Paige R., Tarjan R.E., "Efficient Algorithms based on Partition Reffinement", 1986.

[Qu82] Queille J.P., "Le système CESAR : Description, Spécification et Analyse des Applications Réparties", Thèse de Docteur-ingénieur, INPG, 1982.

[RRSV87] Richier J.L., Rodriguez C., Sifakis J., Voiron J., " Verification In Xesar of the sliding window protocol", 17th International Workshop on Protocol Specification, Testintg and Verification, 1987.

[San82] Sanderson, M.T. "Proof techniques for CCS", Ph. D. Thesis, CST-19-82, University of Edimburgh, Nov 1982.

[Sc83] Schwartz J.P., "QUASAR, une réalisation du système CESAR : Description, Spécification et Analyse des Applications Réparties", Thèse de Docteur-ingénieur, INPG, 1983.

[SM87] De Simone R., Madelaine E., "ECRINS, un laboratoire de preuves pour les calculs de processus communicants", Actes du Colloque C^3, 1987.

[Sor87] Soriano A, "Venus : un outil d'aide à la vérification des systèmes communicants", Thèse de troisième cycle, Université de Grenoble.

[Tar72] Tarjan R.E. "Depth first search and linear graph algorithms", SIAM J., Computing 1:2 146-160.

[Ver87] Vergamini D., "Vérification d'automates finis par Equivalences Observationnelles : Le système AUTO, Thèse Université de Nice, 1987.

[Win83] Winskel G, "Synchronisation tree", Proc. 10th Int. Colloq. Automata Lang & Programming Barcelonna (J. Diaz Ed.), 695-711, Lecture Notes in Computer Science, 154, Springer Verlag.

ANNEXE I

Présentation du langage LUSTRE

LUSTRE est un langage "data-flow" qui opère sur des suites infinies de valeurs : chaque constante ou variable est égale à la suite des valeurs qu'elle prend pendant l'exécution du programme. L'interprétation synchrone est la suivante : un programme a un comportement cyclique calculant au n-ième cycle la n-ième valeur de chaque variable. Un programme est un ensemble d'équations qui définissent les variables. L'équation "X = E", où X est une variable et E une expression représentant respectivement les suites $(x_1, ..., x_n, ...)$ et $(e_1, ..., e_n, ...)$, signifie $x_n = e_n$ pour tout entier n.

Nous présentons un sous-ensemble de LUSTRE permettant d'illustrer le processus de compilation du langage [CPH87].

Les opérateurs du langage LUSTRE sont les opérateurs arithmétiques, booléens et conditionnels habituels et les opérateurs temporels **pre** et ->.

L'opérateur **pre** retourne la valeur de la variable au cycle précédent : si E représente la suite $(e_1, ..., e_n, ...)$, **pre** (E) représente la suite $(nil, e_1, ..., e_n, ...)$ où nil est une valeur indéfinie correspondant à une variable non initialisée dans un langage impératif.

- L'opérateur -> permet la définition de valeurs initiales : si E (resp. F) représente la suite $(e_1, ..., e_n, ...)$ (resp. $(f_1, ..., f_n, ...)$), E -> F représente la suite $(e_1, f_2, ..., f_n, ...)$

Sémantique de LUSTRE

La sémantique du langage LUSTRE est définie par une fonction de l'ensemble des histoires dans lui-même. Une histoire η est une suite infinie de mémoires. Une mémoire σ est une fonction de l'ensemble des identificateurs

dans l'ensemble des valeurs. La sémantique est décrite par un ensemble de règles :

$\eta \models eqs$: η'étant donnée une histoire η, le programme constitué par l'ensemble d'équations eqs produit l'histoire η' ;

$eq \text{-}\sigma\rightarrow eq'$ si X est la partie gauche de l'équation eq, l'évaluation de X dans l'état σ produit l'équation eq' ;

$\sigma \models e\text{-}v\rightarrow e'$ L'évaluation de l'expression e à v dans l'état σ produit l'expression e'.

Nous donnons ci-dessous en exemple la sémantique des opérateurs **pre** et ->.

$$\frac{\sigma \models e\text{-}v\rightarrow e',\ \sigma \models f\text{-}w\rightarrow f'}{\sigma \models e\text{->}f\text{-}v\rightarrow f'}$$

Cette règle signifie que la valeur courante de l'expression e->f est la valeur courante de e, et que dans le futur, ce sera celle de f.

La valeur retournée par l'opérateur **pre** est considérée comme un paramètre :

$$\frac{\sigma \models e\text{-}w\rightarrow e'}{\sigma \models pre(e,\ v)\text{-}v\rightarrow pre(e',w)}$$

2.3. Contrôle dirigé par les données

La structure de contrôle du programme est une boucle infinie. A chaque cycle,

1) les valeurs courantes des variables sont calculées,
2) le programme est réécrit en un nouveau programme.

Le programme est un automate dont les états sont les différentes formes qu'il peut prendre lorsqu'il est réécrit. Le programme peut être transformé par l'introduction de variables auxiliaires, de telle sorte qu'une équation contienne au plus un opérateur temporel (i.e. **pre** ou ->). Après cette transformation, un état est caractérisé par un ensemble de *variables d'état* :

- une variable booléenne, appelée "init", indique si l'état est initial,
- à chaque équation contenant un opérateur **pre** est associée une variable dont la valeur est celle retournée par cet opérateur dans l'état courant.

L'automate a une infinité d'états ; il faut le transformer en un automate d'états finis qui est la structure de contrôle. Ceci est réalisé en considérant uniquement les variables booléennes. Les calculs des entiers étiquettent les transitions et les expressions booléennes dépendantes d'entiers sont traitées comme des entrées.

Exemple

Considérons le programme LUSTRE suivant qui a deux booléens b_0 et b_1 en entrée et qui calcule un entier n :

```
n =0 -> if left then pre(n) + 1
        else if right then pre(n) - 1
        else pre(n)

left = false -> b0 and not pre(b0) and b1

right = false -> b0 and not pre(b0) and not b1
```

La transformation de ce programme est obtenue en introduisant la variable auxiliaire pre_b_0, de sorte qu'une équation contient au plus un opérateur temporel, et en faisant du résultat de l'opérateur pre un paramètre, lorsque cet opérateur est appliqué à une expression booléenne.

```
n =0 -> if left then pre(n) + 1
        else if right then pre(n) - 1
        else pre(n)
```

left = false -> b_0 and not pre_b_0 and b_1

right = false -> b_0 and not pre_b_0 and not b_1

pre_b_0 = pre(b_0, nil)

Il y a deux variables d'état, init et la valeur courante de pre_b_0. L'état initial S_0 est caractérisé par {init = true, preb_b_0 = nil}. En appliquant les règles de la sémantique, nous avons dans l'état initial : n = 0, left = right = false et, dans l'état suivant, init = false, pre_b_0 = true ou false suivant l'entrée b_0.

Si nous définissons

S_{10}= {init = false, pre_b_0 = true}, S_{11} = {init = false, pre_b_0 = false}

un code possible pour S_0 est

S_0 : n :=0 ; put (n) ; get(b_0) ; if b_0 then goto S_{10} else goto S_{11} fi ;

En appliquant les règles de la sémantique à S_{10} nous obtenons un code possible pour S_{10} :

S_{10} : n:= 0 ; put(n) ; get(b_0) ; if b_0 then goto S_{10} else goto S_{11} fi ;

Dans l'état S_{11}, où pre_b_0 est vrai,

S_{11}: get(b_0)
if b_0 then get(b_1) ;
if b_1 then n := n + 1 else n := n - 1 fi ;
put (n) ; goto S_{10}
else put(n) ; goto S_{11} ;
fi ;

Tous les états accessibles peuvent être construits de cette manière.

Cette méthode a été introduite la première fois pour la compilation du langage ESTEREL [BCG87] et est utilisée par l'actuelle version du compilateur de LUSTRE.

Nous présentons un exemple d'utilisation du système ALDEBARAN par le compilateur LUSTRE.

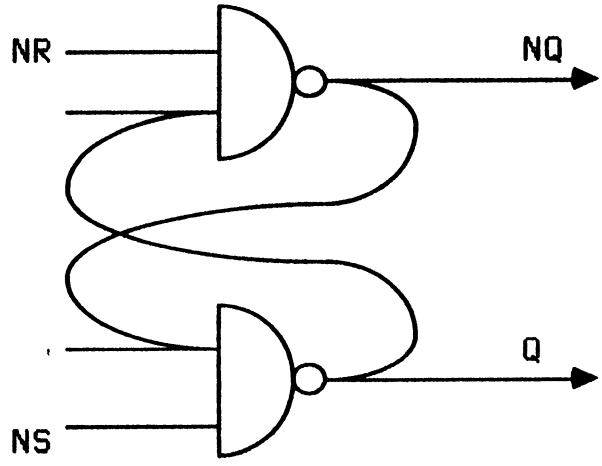
Exemple bascule RS

La bascule RS comporte deux entrées, NR et NS, et deux sorties Q et NQ. Le comportement attendu de cette bascule est le suivant :
Lorsque NR = 0 et NS = 1 alors Q = 0 et NQ = 1,
NR = 1 et NS = 0 alors Q = 1 et NQ = 0 ,
NR = 1 et NS = 1 alors Q(t) = q(t - 1) et NQ (t) = NQ (t - 1)

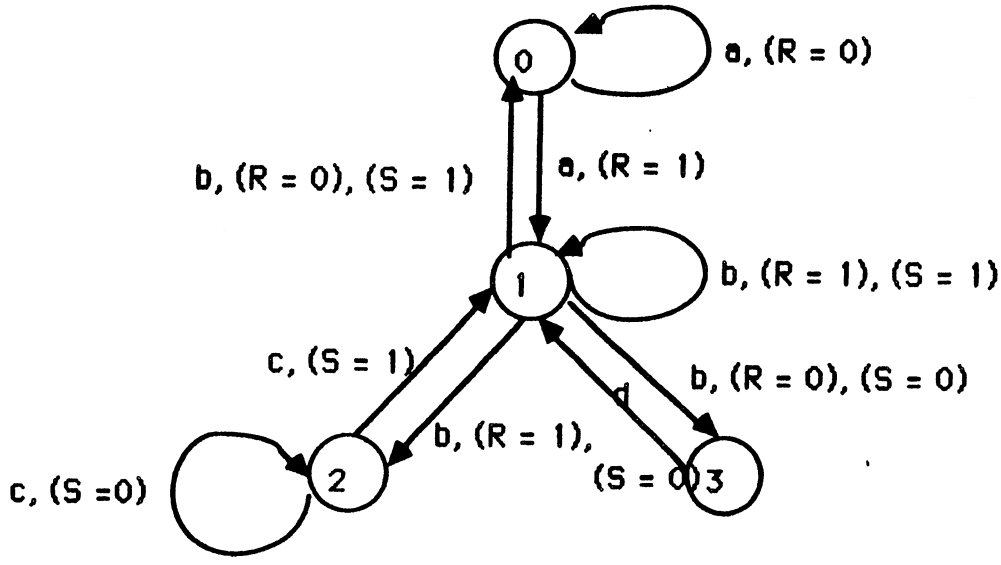
. Nous présentons une spécification de la bascule RS en LUSTRE :

```
node rs
(R, S : bool) returns (Q, NQ : bool) ;
let
    Q = true -> not pre (NQ) or pre (S) ;
    NQ = false -> not pre (Q) or pre (R)
tel
```

Une réalisation possible de cette bascule est proposée (catalogue Texas Instrument) dans la figure suivante :



Ce programme a deux entrées R et S et deux sorties Q et NQ, définies chacune par une équation. Le code portable qui lui est associé décrit un automate à 16 états. Une fois réduit, cet automate a quatre états. Nous donnons une description de l'automate réduit :
Pour simplifier la présentation, définissons les étiquettes suivantes :
a étiquette l'instruction {NQ := 0 ; output NQ ; Q := 1 ; output Q},
b étiquette l'instruction {NQ := 1 ; output NQ ; Q := 1 ; output Q},
c étiquette l'instruction {NQ := 0 ; output NQ ; Q := 0 ; output Q} et
d étiquette l'instruction {NQ := 1 ; output NQ ; Q := 0 ; output Q}



Les transitions sont étiquetées par les actions définies ci-dessus (i.e. a, b, c ou d) et par le résultat de l'évaluation des tests. Ce comportement est bien le comportement attendu du système, aux états transitoires près.

ANNEXE II
Description du Protocole du Bit Alerné en LOTOS

```
specification ALTERNATING_BIT_PROTOCOL [PUT, GET] : noexit behaviour
   hide SDT0, SDT1, RDT0, RDT1, RDTe, RACK0, RACK1, SACK0, SACK1, SACKe in
   (
      (
      TRANSMITTER [PUT, SDT0, SDT1, SACK0, SACK1, SACKe]
      |||
      RECEIVER [GET, RDT0, RDT1, RDTe, RACK0, RACK1]
      )
   |[SDT0, SDT1, RDT0, RDT1, RDTe, RACK0, RACK1, SACK0, SACK1, SACKe]|
      (
      MEDIUM1 [SDT0, SDT1, RDT0, RDT1, RDTe]
      |||
      MEDIUM2 [RACK0, RACK1, SACK0, SACK1, SACKe]
      )
   )

where

process MEDIUM1 [SDT0, SDT1, RDT0, RDT1, RDTe] : noexit :=
   SDT0;              (* reception d'un message *)
      (
      RDT0;           (* transmission correcte *)
         MEDIUM1 [SDT0, SDT1, RDT0, RDT1, RDTe]
      []
      RDTe;           (* perte avec indication *)
         MEDIUM1 [SDT0, SDT1, RDT0, RDT1, RDTe]
      []
      (hide LOSS in
      LOSS;           (* perte silencieuse *)
```

```
        MEDIUM1 [SDT0, SDT1, RDT0, RDT1, RDTe]
      ))
   []
   MEDIUM1 [SDT1, SDT0, RDT1, RDT0, RDTe]
endproc

process MEDIUM2 [RACK0, RACK1, SACK0, SACK1, SACKe] : noexit :=
   RACK0;          (* reception d'un acquittement *)
      (
      SACK0;       (* transmission correcte *)
         MEDIUM2 [RACK0, RACK1, SACK0, SACK1, SACKe]
      []
      SACKe;       (* perte avec indication *)
         MEDIUM2 [RACK0, RACK1, SACK0, SACK1, SACKe]
      []
      (hide LOSS in
      LOSS;        (* perte silencieuse *)
         MEDIUM2 [RACK0, RACK1, SACK0, SACK1, SACKe]
      ))
   []
   MEDIUM2 [RACK1, RACK0, SACK1, SACK0, SACKe]
endproc

process TRANSMITTER [PUT, SDT0, SDT1, SACK0, SACK1, SACKe] : noexit :=
   PUT;          (* aquisition d'un message *)
      TRANSMIT [PUT, SDT0, SDT1, SACK0, SACK1, SACKe]
where
   process TRANSMIT [PUT, SDT0, SDT1, SACK0, SACK1, SACKe] : noexit :=
      SDT0;              (* emission du message *)
         (
         SACK0;          (* bit de controle correct *)
            TRANSMITTER [PUT, SDT1, SDT0, SACK1, SACK0, SACKe]
         []
         SACK1;          (* bit de controle incorrect => reemission *)
            TRANSMIT [PUT, SDT0, SDT1, SACK0, SACK1, SACKe]
         []
         SACKe;          (* indication de perte => reemission *)
            TRANSMIT [PUT, SDT0, SDT1, SACK0, SACK1, SACKe]
         []
         (hide TIMEOUT in
         TIMEOUT;        (* timeout => reemission *)
            TRANSMIT [PUT, SDT0, SDT1, SACK0, SACK1, SACKe]
         ))
   endproc
```

```
endproc

process RECEIVER [GET, RDT0, RDT1, RDTe, RACK0, RACK1] : noexit :=
   RDT0;                    (* bit de controle corect *)
      GET;                  (* livraison du message M *)
         RACK0;             (* envoi d'un acquittement correct *)
            RECEIVER [GET, RDT1, RDT0, RDTe, RACK1, RACK0]
   []
   RDT1;                    (* bit de controle incorrect => *)
      RACK1;                (* envoi d'un acquittement incorrect *)
         RECEIVER [GET, RDT0, RDT1, RDTe, RACK0, RACK1]
   []
   RDTe;                    (* indication de perte => *)
      RACK1;                (* envoi d'un acquittement incorrect *)
         RECEIVER [GET, RDT0, RDT1, RDTe, RACK0, RACK1]
   []
   (hide TIMEOUT in
   TIMEOUT;                 (* timeout => *)
      RACK1;                (* envoi d'un acquittement incorrect *)
         RECEIVER [GET, RDT0, RDT1, RDTe, RACK0, RACK1])
endproc

endspec
```

ANNEXE III
Aldébaran
Manuel de l'utilisateur

Jean-Claude Fernandez
LGI
BP53X
38041 GRENOBLE Cedex
Tel : 76 51 49 15
e-mail fernand@imag.imag.fr

1 Environnement

Aldébaran est écrit en Langage C et utilise le générateur d'analyseur Yacc. Aldébaran est portable sur toute machine UNIX.

2 Appel

La syntaxe de la ligne de commande est décrite de la manière suivante :

aldebaran $file_1 \ldots file_n$

où : $file_i$ est un nom de fichier contenant la description d'un système de transitions étiquetées au format Aldébaran et suffixé par ''.aut''.

Le format Aldébaran comporte le descripteur du système de transitions étiquetées et une liste de triplet :

format Aldébaran := *descripteur* [*triplet*]$^+$
descripteur ::= *état_initial card_trans card_état*
triplet ::= *état idf état*

3 Liste des commandes

Commande "quit"

Syntaxe **quit**
Permet de sortir de Aldébaran.

Commande "gen_env"

Syntaxe **gen_env**
Génère un environnement constitué des systèmes de transitions étiquetées, contenus dans les fichiers paramètres de la ligne de commande. Pour chaque système de transitions étiquetées, contenu dans `file.aut`, sa description sous forme binaire est sauvegardée dans un fichier `file.sto`.

Commande "cre_env"

Syntaxe **cre_env (éditeur)**
Appel de l'éditeur `éditeur`, analyse syntaxique du fichier produit et création de l'environnement.
(**Remarque** : Commande non implantée).

Commande "pr_env"

Syntaxe **pr_env**
Visualisation du contenu de l'environnement.

Commande "sa_env"

Syntaxe **sa_env**
Sauvegarde de l'environnement courant. Un fichier nommé `environ.des` est créé.

Commande "ch_env"

Syntaxe **ch_env**
Chargement de l'environnement décrit dans le fichier `environ.des`.

Commande "pr_ste"

Syntaxe **pr_ste idf**
Visualisation du système de transitions étiquetées désigné par **idf**.

Commande "pr_desc"

Syntaxe **pr_desc idf**
Visualisation du descripteur du système de transitions étiquetées désigné par **idf**.

Commande "ls_ste"

Syntaxe **ls_ste**
Liste des systèmes de transitions étiquetées de l'environnement courant.

Commande "rm_ste"

Syntaxe **rm_ste idf**
Suppression du système de transitions étiquetées de l'environnement courant.

Commande "ecr_ste"

Syntaxe **ecr_ste idf**
Sauvegarde sous forme de fichier texte dans le fichier **idf.aut**.

Commande "sa_ste"

Syntaxe **sa_ste idf**
Sauvegarde sous forme de fichier binaire dans le fichier **idf.ste**.

Commande "ch_ste"

Syntaxe **ch_ste idf**
Chargement d'un système de transitions étiquetées existant dans un fichier **idf.ste**.

Commande "def_tau"

Syntaxe **def_tau idf**

Redéfinition de l'étiquette tau. Par défaut, elle est désignée par i. Cette commande ne peut intervenir qu'avant la création d'un environnement.

Commande "def_com"

Syntaxe **def_com ty_com oper_com**

Définition de la relation de communication. **ty_com** et **oper_com** sont deux entiers. Le premier entier est le *le type de communication* : 1 désigne le *type synchrone* et 0 le *type asynchrone*. Les valeurs possibles du deuxième entier sont 0, 1, 2 ou 3.

oper_com = 0 : Le produit de synchronisation est celui associé à l'algèbre CCS: a . na = tau et a . b = 0, pour a != b et a != nb et b != na.

oper_com = 1 : Le produit de synchronisation est a . a = a et a . b = 0 pour a != b.

oper_com = 2 : Le produit de synchronisation est a . a = tau et a . b = 0 , pour a != b.

oper_com = 3 : Un argument supplémentaire *listetriplets* est nécessaire. *listetriplets* est une liste de triplets dont la syntaxe est :
$[(a_1, b_1, c_1), \ldots, (a_n, b_n, c_n)]$
Le produit de synchronisation est défini de la manière suivante :
$a_i \cdot b_i = c_i$ pour i = 1, n.

Commande "comp"

Syntaxe **comp som strategie idf exp_comp ;;**

L'entier **som** désigne la relation d'équivalence choisie et l'entier **strategie** précise la stratégie pour composer les systèmes de transitions étiquetées. **idf** désigne le système de transitions étiquetées résultat de la composition. **exp_comp** est une expression de composition dont la syntaxe est décrite de la manière suivante :

exp_comp := *exp_comp* & *term_comp* |
term_comp
term_comp := *term_comp* # [*liste_idf*] |
term_comp $ [*liste_idf*] |
term_fact
term_fact := *idf* | *(exp_comp)*
liste_idf := *liste_idf* , *idf* |

idf.

Après l'analyse de l'expression *exp_comp*, un système de transitions étiquetées est généré en appliquant la stratégie `strategie` et en tenant compte de la relation d'équivalence `sem`.

Commande "abst"

Syntaxe *abst idf* `$ [` *liste_idf* `] ;;`

Renommage en *tau* de toutes les étiquettes du système de transitions étiquetées *idf*, appartenant à *liste_idf*.

Commande "open_in_stream"

Syntaxe `open_in_stream idf`

Ouverture du flux d'entrée désigné par *idf*.

Commande "close_all"

Syntaxe `close_all`

Fermeture de tous les flux.

Commande "system"

Syntaxe `system ''chaine'' ;`

Exécution de la commande `Unix chaine`.

Commande "help"

Syntaxe `help`

Liste des commandes disponibles.

Commande "decide_f"

Syntaxe `decide_f` idf_1 idf_2

Procédure de décision pour la congruence forte.

Commande "ncf"

Syntaxe `ncf idf`

Calcul de la forme normale pour la congruence forte. Le système de tran-

sitions étiquetées résultat est `idfnf`.

Commande "decide_co"

Syntaxe **decide_co** $\mathtt{idf}_1$ $\mathtt{idf}_2$
Procédure de décision pour la congruence observationnelle.

Commande "nco"

Syntaxe **nco idf**
Calcul de la forme normale pour la congruence observationnelle. Le système de transitions étiquetées résultat est `idfnco`.

Commande "decide_eo"

Syntaxe **decide_eo** $\mathtt{idf}_1$ $\mathtt{idf}_2$
Procédure de décision pour l'équivalence observationnelle.

Commande "neo"

Syntaxe **neo idf**
Calcul de la forme normale pour l'équivalence observationnelle. Le système de transitions étiquetées résultat est `idfneo`.

Commande "decide_acl"

Syntaxe **decide_acl** $\mathtt{idf}_1$ $\mathtt{idf}_2$
Procédure de décision pour la congruence par acceptation, notée acl. (**Remarque** : Commande non implantée).

Commande "nacl"

Syntaxe **nacl idf**
Calcul de la forme normale pour la congruence par acceptation, notée acl. Le système de transitions étiquetées résultat est `idfacl`.

Commande "decide_ac2"

Syntaxe **decide_ac2 idf$_1$ idf$_2$**

Procédure de décision pour la congruence par acceptation, notée ac2.
(**Remarque** : Commande non implantée).

Commande "nac2"

Syntaxe **nac2 idf**

Calcul de la forme normale pour la congruence par acceptation, notée ac2.
Le système de transitions étiquetées résultat est **idfac2**.
(**Remarque** : Commande non implantée).

Commande "cmp"

Syntaxe **cmp idf$_1$ idf$_2$**

Test de l'égalité des deux systèmes de transitions étiquetées désignés par **idf$_1$** et **idf$_2$**.

Commande "cl_cf"

Syntaxe **cl_cf idf**

Détermination des classes d'équivalence sur l'ensemble des états du système de transitions étiquetées désigné par **idf**, modulo la congruence forte.

Commande "time"

Syntaxe **time**

Visualisation du temps d'exécution d'une commande.

Commande "no_time"

Syntaxe **no_time**

Commande inverse de la précédente.

4 Utilisation "silencieuse" d'Aldébaran

4.1 Forme normale pour la congruence forte

min *file*

min est un fichier de commande "shell" contenant :

```
    FILE1='basename $1 .aut'.aut
cp $FILE1 _.aut
aldebaran _.aut > _.min << .
gen_env
ncf _
pr_ste _nf
quit
.
/bin/grep , _.min
rm _.aut _.ste _.min
```

4.2 Forme normale pour la congruence observationnelle

omin *file*

min est un fichier de commande "shell" contenant :

```
    FILE1='basename $1 .aut'.aut
cp $FILE1 _.aut
aldebaran _.aut > _.min << .
gen_env
nco _
pr_ste _nco
quit
.
/bin/grep , _.min
rm _.aut _.ste _.min
```

4.3 Décision pour la congruence forte

cmp_f *file*$_1$ *file*$_2$

min est un fichier de commande "shell" contenant :

```
    FILE1='basename $1 .aut'.aut
FILE2='basename $2 .aut'.aut
cp $FILE1 com1.aut
cp $FILE2 com2.aut
aldebaran com1.aut com2.aut > _.com << .
gen_env
decide_f com1 com2
quit
.
rm com1.ste com2.ste com1.aut com2.aut
if '/bin/grep -s non _.com'
then
    echo 0
    rm _.com
    exit 0
else
    echo 1
    rm _.com
    exit 1
fi
```

4.4 Décision pour la congruence observationnelle

cmp_co *file*$_1$ *file*$_2$
    min est un fichier de commande "shell" contenant :

```
    FILE1='basename $1 .aut'.aut
FILE2='basename $2 .aut'.aut
cp $FILE1 com1.aut
cp $FILE2 com2.aut
aldebaran com1.aut com2.aut > _.com << .
gen_env
decide_co com1 com2
quit
.
rm com1.ste com2.ste com1.aut com2.aut
if '/bin/grep -s non _.com'
then
    echo 0
    exit 0
else
    echo 1
```

```
    exit 1
fi
```

5 Exemple de session

```
1] aldebaran *.aut

ALDEBARAN>time
0.4u 0.15s

ALDEBARAN>gen_env
0.52u 0.36s

ALDEBARAN>def_com
 choix du type de communication, 0 : asynch., 1 synch
0
choix de l'operateur de communication : 0, 1, 2 ou 3
0
0.4u 0.15s

ALDEBARAN>system more exp_comp_abs ;
((start&s1&s2&s3&s4&s5)
        #[ng1,ng3,ng2,ng4,ng5,g5,g4,g3,g1,g2])
        $[nb1,nb2,nb3,nb4,nb5]
;;
0.52u 0.37s

ALDEBARAN>open_in_stream exp_comp_abs
0.4u 0.20s

ALDEBARAN>comp
preciser la semantique :
        0,1 (forte), 2 (observationnelle), 3 (acc)
2
preciser la strategie :
        0,1 (sans reordonner), 2(en reordonnant)
2
scheduler_abs
30.18u 1.3s

ALDEBARAN>pr_sto scheduler_abs
```

```
descripteur(scheduler_abs)
        card_trans : 6
        card_etat : 7
        card_act : 5
        t_present : 1
        mode : 0
                (0, i, 6)
                (2, na5, 6)
                (3, na2, 4)
                (4, na3, 5)
                (5, na4, 2)
                (6, na1, 3)
0.4u 0.29s

ALDEBARAN>system more exp_comp ;
(((start&s1&s2&s3&s4&s5) #[ng1,ng2,ng3, ng4, ng5,g1,g2,g3, g4,g5])
&r1&r2&r3&r4&r5) #[nb1,nb2,nb3, nb4,nb5,b1,b2,b3,b4,b5];;
30.18u 1.4s

ALDEBARAN>close_all
0.4u 0.29s

ALDEBARAN>open_in_stream exp_comp
30.18u 1.4s

ALDEBARAN>comp
preciser la semantique :
        0,1 (forte), 2 (observationnelle), 3 (acc)
2
preciser la strategie :
        0,1 (sans reordonner), 2(en reordonnant)
2
scheduler
4.38u 0.39s

ALDEBARAN>pr_ste scheduler
descripteur(scheduler)
        card_trans : 6
        card_etat : 7
        card_act : 6
        t_present : 1
        mode : 0
                (0, i, 6)
                (2, na5, 6)
```

```
                (3, na2, 4)
                (4, na3, 5)
                (5, na4, 2)
                (6, na1, 3)
30.18u 1.10s

ALDEBARAN>decide_f scheduler scheduler_abs
systemes equivalents modulo la congruence forte
4.59u 0.42s

ALDEBARAN>quit
35.5u 3.3s 3:34 18% 5+42k 114+26io 0pf+0w
```

ANNEXE IV
Description du Protocole de Stenning en Estelle /R

```
specification stenning;
const
    k = 6;        (* all numbers are modulo k *)
    k_minus_1 = 5;
    M = 2;        (* Capacity of the media *)
    TWS = 2;      (* Transmitter Window Size *)
    RWS = 2;      (* Receiver Window Size *)
    TOE = 5;      (* Transmit Time Out *)

channel data;
        message(val: integer);

module transmitter process(tm: data; mt: data);

body transmission for transmitter;
    var
(* the transmitter has got (from the user) the messages numbered
                lowest_unacked .. lowest_unacked+number_entered-1 *)
(* the transmitter has sent (on the line) the messages numbered
                lowest_unacked .. lowest_unacked+number_in_transit-1 *)
        lowest_unacked: 0 .. k_minus_1;
        number_in_transit: 0 .. k_minus_1;
        number_entered_message: 0 .. k_minus_1;
        in_transit: packed array[0..k_minus_1] of boolean;

(* test if i is in the interval base<= <base+width  (mod k) *)
    function in_window(i, base, width: integer): boolean;
        begin
            if (base+width) <= k then
```

```
            in_window := (base <= i) and (i < base+width)
        else
            in_window := (base <= i) or (i < base+width-k)
    end;

(* set to false in_transit[i] for low<=i<=high (mod k) *)
(*  the interval low..high (mod k) is not empty *)
    procedure cancel_timer(low, high: integer);
        var i: integer;
        begin
            if high < low then high := high + k;
            for i := low to high do in_transit[i mod k]:= false;
        end;

    initialize
        var i: 0..k_minus_1;
        begin
            number_in_transit := 0;
            number_entered_message := 0;
            lowest_unacked := 0;
            for i := 0 to k_minus_1 do in_transit[i] := false;
        end;

    trans (* transmission of a new message *)
        provided (number_in_transit < TWS) and
                 (number_in_transit >= number_entered_message)
        delay(1,*) begin
            {: mess_in } output tm.message(
                            (lowest_unacked+number_in_transit) mod k);
            in_transit[(lowest_unacked+number_in_transit) mod k] := true;
            number_in_transit := (number_in_transit+1)mod k;
            number_entered_message := (number_entered_message+1)mod k;
        end;

    trans (* retransmission of a message *)
        provided (number_in_transit < TWS) and
                 (number_in_transit < number_entered_message)
        delay(1,*) begin
            output tm.message((lowest_unacked+number_in_transit) mod k);
            in_transit[(lowest_unacked+number_in_transit) mod k] := true;
            number_in_transit := (number_in_transit+1) mod k;
        end;

    trans (* time out on a message *)
```

```
        any j: 0..k_minus_1 do provided in_transit[j] delay(TOE)
            begin
                cancel_timer(j, (lowest_unacked+number_in_transit-1+k) mod k);
                number_in_transit := (j - lowest_unacked +k) mod k;
            end;

    trans (* acknowledgement *)
        when mt.message(j) begin
            if in_window(j, lowest_unacked, number_in_transit) then begin
                cancel_timer(lowest_unacked, j);
                number_entered_message := (number_entered_message -
                    (j + 1 - lowest_unacked) + k) mod k;
                number_in_transit := (number_in_transit -
                    (j + 1 - lowest_unacked) + k) mod k;
                lowest_unacked := (j+1) mod k
        end end;
end;

module receiver process(mr: data; rm: data);

body reception for receiver;
(* test if i is in the interval base<= <base+width (mod k) *)
    function in_window(i, base, width: integer): boolean;
        begin
            if (base+width) <= k then
                in_window := (base <= i) and (i < base+width)
            else
                in_window := (base <= i) or (i < base+width-k)
        end;

    state: (wait, deliver);

    var
        next_required: 0.. k_minus_1; (* first message not received *)
        received: packed array[0..k_minus_1] of boolean;

    initialize to wait
        var i: integer;
        begin
            for i:=0 to k_minus_1 do received[i]:= false;
            next_required:= 0;
        end;

    trans
```

```
        from wait when mr.message(i)
            provided in_window(i, next_required, RWS)
                            and not received[i]
                to deliver
                    begin
                        received[i] := true;
                    end;
            provided otherwise  (* the message is out of sequence *)
                to wait
                    begin
                        output rm.message((next_required - 1 + k) mod k)
                    end;
    trans {: mess_out } (* deliver to the user *)
        from deliver provided received[next_required] to deliver
            begin
                received[next_required] := false;
                next_required := (next_required+1) mod k;
            end;
    trans       (* end of deliver, send an acknowledge *)
        from deliver provided not received[next_required] to wait
            begin
                output rm.message((next_required - 1 + k) mod k)
            end;
end;

module medium process(min, mout: data; param loss: boolean; dup: boolean);
body line for medium;
    state: (wait, reset);
    var mess: packed array[1..M] of -1..k_minus_1;
    procedure shift;
        var i: integer;
        begin
            for i:= 2 to M do mess[i-1]:= mess[i]
        end;
    initialize to wait
        var i: integer;
        begin
            for i:=1 to M do mess[i]:= -1;
        end;

    trans (* receive a message *)
        from wait to wait when min.message(i) begin shift; mess[M]:= i end;
    trans {: loss} (* receive a message and lose it *)
        provided loss from wait to wait when min.message(i)
```

```
                                                    begin shift; mess[M]:= -1 end;
    trans (* try to send any message *)
        from wait to reset any j: 1..M do provided mess[j]<>-1 delay(1)
           begin output mout.message(mess[j]); mess[j]:= -1 end;
    trans {: dup} (* try to send any message, with duplication *)
        provided dup from wait to reset any j: 1..M do
           provided mess[j]<>-1 delay(1)
           begin output mout.message(mess[j]) end;
    trans
        from reset to wait begin end;
end;

var t: transmitter;
    r: receiver;
    t_to_r, r_to_t: medium;
initialize begin
    init t with transmission;
    init r with reception;
    init t_to_r with line(true, false);
    init r_to_t with line(true, false);
    connect t.tm to t_to_r.min;
    connect t.mt to r_to_t.mout;
    connect r.rm to r_to_t.min;
    connect r.mr to t_to_r.mout;
end;
end.
```

TABLE DES MATIERES

AUTORISATION DE SOUTENANCE

DOCTORAT 3ème CYCLE, DOCTORAT-INGENIEUR, DOCTORAT USTMG

Vu les dispositions de l'Arrêté du 16 avril 1974,

Vu les dispositions de l'Arrêté du 5 juillet 1984,

Vu les rapports de M. Gérard Boudol, Directeur de Recherche

M. Laurent Trilling, Professeur .....

M. FERNANDEZ Jean-Claude ..... est autorisé

à présenter une thèse en vue de l'obtention du DOCTORAT de .....

GRENOBLE 1 Mention Informatique .....

Grenoble, le .....

Le Président de l'Université Scientifique
Technologique et Médicale

J.J PAYAN