VENUS : un outil d'aide à la vérification des systèmes communicants

Amelia Soriano Montes

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR DE 3éme CYCLE
<<Informatique>>

par

Amelia SORIANO MONTES

VENUS: UN OUTIL D'AIDE A LA VERIFICATION DES SYSTEMES COMMUNICANTS

Thèse soutenue le 9 janvier 1987 devant la Commission d'Examen.

Composition du jury:

J. MOSSIERE	*Président*
J. AZEMA	
Ph. JORRAND	
M. RAYNAL	*Examinateurs*
J. SIFAKIS	
J. VOIRON	

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH
Vice-Présidents : B. BAUDELET
H. CHERADAME
R. CARRE
J.M. PIERRARD

Année universitaire 1984-1985

Professeurs des Universités

E.N.S.E.E.G.

BESSON	Jean	LOUCHET	François
BONNETAIN	Lucien	PARIAUD	Jean-Charles
BONNIER	Etienne	RAMEAU	Jean-Jacques
DURAND	Françis	SOHM	Jean-Claude
GUYOT	Pierre	SOUQUET	Jean-Louis

E.N.S.E.R.G.

BARIBAUD	Michel	GENTY	Pierre
BLIMAN	Samuel	GUERIN	Bernard
BUYLE BODIN	Maurice	POUPOT	Christian
CHENEVIER	Pierre	SERMET	Pierre
COHEN	Joseph	ZADWORNY	François
COUMES	André		

E.N.S.I.E.G.

BARRAUD	Alain	JOUBERT	Jean-Claude
BAUDELET	Bernard	JOURDAIN	Geneviève
BLOCH	Daniel	LACOUME	Jean-Louis
BRISSONNEAU	Pierre	LONGEQUEUE	Jean-Pierre
CAVAIGNAC	Jean-François	MASSELOT	Christian
CHARTIER	Germain	MORET	Roger
CHERUY	Arlette	PAUTHENET	René
DURAND	Jean-Louis	PERRET	René
FELICI	Noël	PERRET	Robert
FOULARD	Claude	POLOUJADOFF	Michel
GAUBERT	Claude	SABONNADIERE	Jean-Claude
IVANES	Marcel	SCHLENKER	Claire
JALINIER	Jean-Michel	SCHLENKER	Michel
JAUSSAUD	Pierre		

E.N.S.H.G.

BOIS	Philippe	LESPINARD	Georges
BOUVARD	Maurice	MOREAU	René
LESIEUR	Marcel	PIAU	Jean-Michel

E.N.S.I.M.A.G.

ANCEAU	François	MOSSIERE	Jacques
FONLUPT	Jean	ROBERT	François
LATOMBE	Jean-Claude	SAUCIER	Gabrielle
MAZARE	Guy	VEILLON	Gérard

U.E.R.M.C.P.P.

CHERADAME	Hervé	RENAUD	Maurice
CHIAVERINA	Jean	ROBERT	André
GANDINI	Alessandro	SILVY	Jacques

Professeurs Associés

BLACKWELDER	Ronald	ENSHG
HAYASHI	Hirashi	ENSIEG
PURDY	Gary	ENSEEG

Professeurs à l'Université des Sciences Sociales (Grenoble II)

BOLLIET	Louis
CHATELIN	Françoise

Chercheurs du C.N.R.S.

Directeurs de recherche :

CARRE	René
FRUCHARD	Robert
JORRAND	Philippe
VACHAUD	Georges

Maître de recherche :

ALLIBERT	Michel	JOUD	Jean-Charles
ANSARA	Ibrahim	KAMARINOS	Georges
ARMAND	Michel	KLEITZ	Michel
BINDER	Gilbert	LANDAU	Ioan-Dore
BORNARD	Guy	LASJAUNIAS	Jean-Claude
DAVID	René	MERMET	Jean
DESPORTES	Jacques	MUNIER	Jacques
DRIOLE	Jean	PIAU	Monique
GIGNOUX	Damien	PORTESEIL	Jean-Louis
GIVORD	Dominique	THOLENCE	Jean-Louis
GUELIN	Pierre	VERDILLON	André
HOPFINGER	Emile	SUERY	Michel

Personnalités habilitées à diriger des travaux de recherche
(Décision du conseil scientifique)

E.N.S.E.E.G.

ALLIBERT	Colette	HAMMOU	Abdelkader
BERNARD	Claude	MALMEJAC	Yves (CENG)
BONNET	Roland	MARTIN GARIN	Régina
CAILLET	Marcel	NGUYEN TRUONG	Bernadette
CHATILLON	Catherine	RAVAINE	Denis
CHATILLON	Christian	SAINFORT	(CENG)
COULON	Michel	SARRAZIN	Pierre
DIARD	Jean-Paul	SIMON	Jean-Paul
EUSTATHOPOULOS	Nicolas	TOUZAIN	Philippe
FOSTER	Panayotis	URBAIN	Georges(ODEILLO)
GALERIE	Alain		

E.N.S.E.R.G.

BARIBAUD	Michel	DOLMAZON	Jean-Marc
BOREL	Joseph	HERAULT	Jeanny
CHOVET	Alain	MONLLOR	Christian
CHEHIKIAN	Alain		

E.N.S.I.E.G.

BORNARD	Guy	LEJEUNE	Gérard
DESCHIZEAUX	Pierre	MAZUER	Jean
GLANGEAUD	François	PERARD	Jacques
KOFMAN	Walter	REINISCH	Raymond

E.N.S.H.G.

ALEMANY	Antoine	OBLED	Charles
BOIS	Daniel	ROWE	Alain
DARVE	Félix	VAUCLIN	Michel
MICHEL	Jean-Marie	WACK	Bernard

E.N.S.I.M.A.G.

BERT	Didier	DELLA DORA	Jean
CALMET	Jacques	FONLUPT	Jean
COURTIN	Jacques	SIFAKIS	Joseph
COURTOIS	Bernard		

U.E.R.M.C.P.P.

CHARUEL	Robert

C.E.N.G.

CADET	Jean	NIFENECKER	Hervé
COEURE	Philippe (LETI)	PERROUD	Paul
DELHAYE	Jean-Marc (STT)	PEUZIN	Jean-Claude(LETI)
DUPUY	Michel (LETI)	TAIEB	Maurice
JOUVE	Hubert (LETI)	VINCENDON	Marc
NICOLAU	Yvan (LETI)		

Laboratoires extérieurs

C.N.E.T.

DEMOULIN	Eric	MERCKEL	Gérard
DEVINE	R.A.B.	PAULEAU	Yves
GERBER	Roland		

I.N.S.A. Lyon

GAUBERT	C.

****

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur	: M. M. MERMET
Directeur des Etudes et de la formation	: M. J. LEVASSEUR
Directeur des Recherches	: M. J. LEVY
Secrétaire Général	: Mle M. CLERGUE

Professeurs de la 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Metallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique-Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

****

Je tiens à remercier

Monsieur Jacques Mossière, Professeur à l'Institut National Polytechnique de Grenoble (INPG) qui m'a fait l'honneur de présider le jury de cette thèse.

Monsieur Joseph Sifakis, Directeur de Recherche au CNRS, pour m'avoir accueilli au sein du Groupe Spécification et Analyse des Systèmes et pour avoir jugé mon travail en me faisant des critiques et en me donnant toujours des conseils très utiles.

Monsieur Pierre Azema, Directeur de Recherche au CNRS, Monsieur Philippe Jorrand, Directeur de Recherche au CNRS et Monsieur Michel Raynal, Professeur à l'Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), pour avoir accepté de participer au jury de cette thèse.

Monsieur Jacques Voiron, Maître de Conférences à l'Université Scientifique et Medicale de Grenoble (USMG), pour le temps qu'il a consacré à l'encadrement de cette thèse et particulièrement pour les nombreuses discussions, toujours constructives, que nous avons eues, me permettant de profiter de son expérience pour la réalisation de ce travail.

Susanne Graf pour le soutien professionnel et amical qu'elle m'a donné et pour avoir bien voulu effectuer le dur travail de correction pendant la rédaction de cette thèse.

Concepción Montes et Andres Villanueva pour leurs conseils et encouragements qui ont été sans aucun doute un facteur très important pour la réalisation de ce travail.

Je remercie également tous ceux qui par leurs suggestions ont contribué à l'amélioration de cette thèse, et particulièrement Hubert Garavel.

Il me faut aussi remercier le Centre Régional des Oeuvres Universitaires et Scolaires (CROUS) et le Consejo de Desarrollo Científico y Humanístico le la Universidad Central de Venezuela (CDCH) pour leur soutien financier sans lequel ce travail n'aurait pas pu être accompli.

VENUS: UN OUTIL D'AIDE A LA VERIFICATION DES SYSTEMES COMMUNICANTS

RESUME

VENUS est un outil d'aide à la conception et à la vérification de systèmes communicants. Il est basé sur le calcul CCS de Milner et permet de normaliser et de vérifier des descriptions de comportements réguliers en tenant compte de la congruence forte et de l'équivalence et de la congruence observationnelle définies dans CCS. Nous présentons dans ce travail les bases théoriques de VENUS, une réalisation en PROLOG et quelques exemples de vérification de protocoles de communication.

MOTS CLES:

Parallelisme, Vérification de Systemes Communicants, CCS, Protocole de Communication, PROLOG.

VENUS: AN ENVIRONNEMENT TO SPECIFY AND VERIFY COMMUNICATING SYSTEMS.

ABSTRACT

An environnement to specify and verify communicating systems is presented. This environnement (called VENUS) is based on the Milner's CCS calculus. VENUS allows the normalisation and verification of communicating systems having a regular behaviour. It is realized using CCS strong congruence and the observation equivalence and congruence. This work describes both the theoretical bases of VENUS and its implementation in PROLOG. Some protocol verification exemples are also presented.

VENUS: UN OUTIL D'AIDE A LA VERIFICATION DES SYSTEMES COMMUNICANTS

TABLE DE MATIERES

INTRODUCTION.

Actuellement avec le développement des systèmes distribués et la conception des langages de programmation capables d'exprimer le parallèlisme, il est possible de construire des systèmes de plus en plus puissants et rapides. En contrepartie, ces systèmes sont de plus en plus complexes; c'est pour cette raison que de nos jours, dans la conception des systèmes communicants, il est indispensable d'avoir une étape de vérification, permettant de garantir la réalisation fiable des systèmes.

La vérification des systèmes communicants est aujourd'hui un domaine actif de recherche. C'est le problème de la vérification automatique des systèmes communicants réguliers qui a motivé la réalisation de cette thèse.

Vérifier la description d'un système consiste à la comparer à une spécification. D'un point de vue théorique, il est possible de distinguer entre deux traitements différents. Un traitement basé sur la logique [AS 70] [Pn 77] [Ab 70] où les propriétés à vérifier sont représentées par un ensemble d'assertions sur le comportement. Ce traitement permet de vérifier des propriétés générales des systèmes comme par exemple l'absence de blocage. L'autre traitement est basé sur des théories algèbriques [Mi 80] [Mi 83] [BK 85] [AB 84] [BH 84] où les descriptions des systèmes et leurs spécifications sont écrites comme termes d'une algèbre et les preuves sont faites à l'aide d'une notion d'équivalence définie entre les termes de l'algèbre; cette approche permet de vérifier des propriétés spécifiques des systèmes, comme par exemple la spécification du service d'un protocole de communication. C'est cette deuxième forme de spécification que nous considérons par la suite.

Dans la littérature, il y a plusieurs exemples de vérification de systèmes communicants fondés sur une algèbre [Mi 80] [BK 84] [Ko 85]. Mais le problème principal de ce type de preuves est lié à la manipulation de termes, et plus les systèmes sont complexes, plus la quantité de calcul nécessaire pour la vérification augmente; d'où la nécessité de l'automatiser.

Dernièrement il y a eu un effort pour produire des outils informatiques d'aide à la vérification des systèmes communicants.

Cette thèse est orientée dans cette voie et l'objectif principal est de concevoir un outil d'aide à la conception et à la vérification des systèmes communicants. Les caractéristiques principales de cet outil doivent être la modularité, la flexibilité et la facilité d'utilisation. Nous proposons dans cette thèse l'outil VENUS basé sur le calcul CCS (Calculus of Communicating Systems) [Mi 80][HM 85][Mi 85].

Dans la première partie du travail nous décrivons l'algèbre de CCS et la définition et l'axiomatisation de la congruence observationnelle. Nous étudions les problèmes posés par cette axiomatisation pour la construction d'un ensemble simple de règles de réecriture et nous proposons des solutions permettant de définir une procédure de construction de "formes normales" et une procédure de décision pour des équivalences et congruences entre termes.

La deuxième partie du travail est consacrée à la description de l'outil VENUS. Cet outil permet principalement:

- de concevoir des systèmes par la définition de différentes composantes du système et la définition de l'architecture par composition de composantes,
- de spécifier et de vérifier la description de chaque module ainsi que le comportement global du système, en utilisant ou non des critères d'observation,
- de simplifier la définition donnée d'un comportement,
- de réaliser des transformations (normalisations, réductions, etc.) sur les comportements,
- de comparer deux comportements,
- de créer des environnements de travail avec des comportements définis ou construits par transformation d'autres comportements,

- de gérer un ensemble d'environnements.

Nous donnons une présentation générale de l'outil, du langage de définition des comportements, des possibilités offertes pour la gestion des environnements de travail et pour réaliser des vérifications et des transformations de comportements définis dans l'environnement.

Nous décrivons une réalisation dans le langage Prolog d'un prototype modulaire de VENUS en donnant son architecture générale et la description de chaque module.

PARTIE I: Un calcul pour la description et pour la vérification des systèmes communicants.

CCS [Mi 80] est un calcul pour la description, la spécification et la vérification de systèmes communicants asynchrones. Ce calcul est fondé sur deux idées principales: l'observation et la communication synchronisée.

Informellement, dans CCS, on peut décrire le comportement observable d'un système (*notion d'observation*). C'est à partir de cette notion d'observation que les critères de comparaison de systèmes (équivalence et congruence observationnelle) ont été définis.

La description d'un système peut être faite aussi par composition parallèle (opération notée par ||) de processus qui communiquent entre eux. La communication de processus est considerée comme une action indivisible du système (*idée de communication synchronisée*), cette action est appellée communication interne et notée "τ".

D'une manière plus formelle, les vérifications des systèmes consistent à transformer et comparer un terme décrivant un système à sa spécification représentée par un autre terme.

Un certain nombre de travaux présentant une axiomatisation complète de l'équivalence et de la congruence observationnelle ont été publiés récemment [HM 85] [BK 85] [Mi 85].

Il est important de disposer de procédures de verification automatisables. Nous consacrons cette partie de la thèse à la définition des procédures de décision pour la congruence forte et pour l'équivalence et la congruence observationnelle.

Nous définissons aussi une procédure de construction de formes normales. Les formes normales construites correspondent aux termes bien gardés "minimaux" de leurs classes d'équivalence.

Pour les termes réguliers, ces procédures sont basées sur des *transformations* ad-hoc définies sur des *relations de transitions* représentant les comportements décrits par les termes et sur la notion de *bisimulation* des automates.

Pour les termes finis, nous nous inspirons des preuves de complètude de l'axiomatisation de la congruence observationnelle des termes de l'algèbre avec la signature $\{P_\tau, nil, +, ||\}$ données en [HM 85], pour caractériser des formes normales pour l'équivalence et pour la congruence observationnelle. Sur la base de cette caractérisation, nous définissons une procédure de décision pour l'équivalence et pour la congruence observationnelle.

I.- CCS: "Calcul of Communicating Systems".

Nous rappelons dans ce chapitre les principales notions et notations du calcul CCS sans transmission de valeurs.

CCS est une algèbre de termes munie d'une relation de congruence. La sémantique opérationnelle de son langage permet d'interpréter les termes comme des processus (systèmes de transitions communicants). La relation de congruence permet de comparer les termes.

Dans ce travail, on se restreint à l'algèbre de processus qui représente des comportements réguliers.

I.1.- Algèbre pour CCS.

Soient:

- D, un ensemble de noms *d'actions* ou *ports de communication* dont les éléments sont désignés par des lettres minuscules,
- ~D, un ensemble de noms de ports, appelés *complémentaires,* dont les éléments sont les éléments de D précédés par le symbol "~",
- une bijection:

  α <---> ~α, où: $\alpha \in D$,

  $\sim\alpha \in \sim D$,

  on dit que "~α" est le *port complémentaire* de "α",
  ceci nous permet de définir l'interaction entre processus,
- τ, ($\tau \notin D$ et $\tau \notin \sim D$) une étiquette représentant une *action non observable,* résultant d'une communication interne entre α et ~α, $\alpha \in D$,
- P, l'ensemble des étiquettes des ports de communication, $P = D \cup \sim D$,
- P_τ $P_\tau = P \cup \{\tau\}$,

- F, l'ensemble de fonctions de redéfinition f telles que:
  $f: L \subseteq P \dashrightarrow M \subseteq P$
  f est une bijection,
  f respecte les compléments définis par la bijection,
- V, l'ensemble de variables de comportement,
- les signatures, $\Sigma_1 = (P_\tau, nil, +)$

  $\Sigma_2 = (P_\tau, nil, +, ||, [\,], \backslash)$

où:

nil est une constante,
+ et || sont des opérateurs binaires,
$\forall \alpha \in P_\tau$ α, $\forall f \in F$ [f], $\forall \alpha \in P$ $\backslash\alpha$, sont des opérateurs unaires.

Les opérateurs + et || sont des opérateurs infixés, α est un opérateur préfixé, [f] et $\backslash\alpha$ sont des opérateurs postfixés. Le symbole "." est utilisé comme séparateur après l'opérateur α, par exemple on écrit α.nil et non αnil.

La priorité est croissante pour les opérateurs '+' et '||', '[f]' et '$\backslash\alpha$', 'α'. Les opérateurs '+', '||', '[f]' et '$\backslash\alpha$' sont associatifs à gauche et l'opérateur 'α' est associatif à droite.

Nous représentons par:

T_f, l'algèbre avec la signature Σ_2. Les éléments de T_f sont appelés *termes finis*.
On dit qu'un terme fini t est sous *la forme d'une somme* si :
- soit t=nil,
- soit $t=\Sigma_{i \in I} \alpha_i.t_i$ et chaque t_i est un terme fini sous la forme d'une somme.

T_s, est le sous-ensemble de termes finis sous la forme d'une somme. ($T_s \subseteq T_f$).

$T[\Sigma_1,V]$, l'algèbre avec signature $\Sigma_1 \cup V$. ($T_s \subseteq T[\Sigma_1,V]$).

$T[\Sigma_2,V]$, l'algèbre avec signature $\Sigma_2 \cup V$. ($T[\Sigma_1,V] \subseteq T[\Sigma_2,V]$ et $T_f \subseteq T[\Sigma_2,V]$).

Les éléments de $T[\Sigma_1,V]$ et ceux de $T[\Sigma_2,V]$ sont des expressions à partir desquelles les termes définissant des comportements infinis sont construits .

T'_r, l'ensemble des paires de la forme: $u = (t,E)$

où $t \in T[\Sigma_2,V]$,

E est un ensemble d'équations de la forme: $\{v_i=t_i\}_{i=1,\ldots,r}$ où $v_i \in V$ et $t_i \in T[\Sigma_1,V]$ et $\forall i,j\ i\neq j \Rightarrow v_i \neq v_j$. On dit que t_i est la *définition* de v_i.

Les éléments de T'_r définissent des comportements infinis. Les paires $(t,\varnothing) \in T'_r$ (*termes finis*) sont notées simplement t. ($T_f \subseteq T'_r$).

Pour les termes de T'_r nous introduisons les notions de *variable utile, variable inutile, variable libre, terme fermé , variable non gardée, terme gardé et de terme bien gardé.* Ces notions sont utilisées pour définir les termes réguliers de CCS et leurs formes normales.

Pour $t \in T[\Sigma_2,V]$, soit var(t) l'ensemble de variables de t.

Pour un terme u=(t,E), on dit qu'une variable v est *utile* si et seulement si $v \in \bigcup_{j=0}^{card(E)} \rho^j(var(t))$, où $\rho(X)$ est défini par:

$\rho(X)=\{z \mid v_i \in X$ et $v_i=t_i$ et $z \in var(t_i)\}$,
ρ^0 est l'identité
$\rho^{k+1}(X) = \rho(\rho^k(X))$, pour $k>1$.

On dit qu'une variable v est *inutile* dans un terme u=(t,E) si et seulement si $\exists t_i$ tel que $v=t_i$ appartient à E et v n'est pas une variable utile du terme u.

On dit qu'une variable v est *libre* dans un terme $(t,E) \in T'_r$ si et seulement si il n'existe pas de j tel que $v=t_j \in E$.

On dit qu'un terme est *fermé* si et seulement si il ne contient pas de variables libres.

On dit qu'une variable v est *non gardée* dans un terme u=(t,E) si et seulement si v apparaît dans t ou dans la définition d'une variable utile de u sans être précédée par une action.

On dit qu'un terme u=(t,E) est *gardé* si et seulement si toute variable non gardée du terme peut être remplacée par nil ou par une expression de la forme $\Sigma \alpha_i.t_i$, par substitution répétée des variables par leurs définitions.

On dit qu'un terme u=(t,E) est *bien gardé* si et seulement s'il est gardé et toute variable gardée par τ dans le terme peut être remplacée par nil ou par une expression de la forme $\Sigma_{i=1,\ldots,n} \alpha_{i1}. \ldots . \alpha_{ik}.t_i$, $\alpha_{ik} \neq \tau$.

Sur les termes de T'_r nous définissons les opérateurs α, +, ||, \, [f] de la manière suivante:
$\alpha(t,E) = (\alpha t, E)$

$(t_1,E_1) + (t_2,E_2) = (t_1+t_2, E_1 \cup E_2)$, si $\nexists x_i \in V$ tel que $x_i=t_i \in E_1$ et $x_i=t'_i \in E_2$

$(t_1,E_1) \,||\, (t_2,E_2) = (t_1||t_2, E_1 \cup E_2)$, si $\nexists x_i \in V$ tel que $x_i=t_i \in E$ et $x_i=t'_i \in E_2$

$(t,E)\backslash\alpha = (t\backslash\alpha, E)$

$(t,E)[f] = (t[f], E)$.

- T_r, est le sous-ensemble de termes fermés et gardés de T'_r. Les éléments de T_r sont les termes réguliers de CCS.
- T_{rs}, est le sous-ensemble de termes $u=(t,E) \in T_r$ tels que $t \in T[\Sigma_1,V]$.

I.2.- Sémantique opérationnelle des opérateurs.

La sémantique opérationnelle [Pl 81] de CCS est donnée à l'aide des relations de transition $\xrightarrow{\alpha}$, $\alpha \in P_\tau$. Ces relations de transition sont les plus petites relations binaires sur T_r définies par les règles:

$$\frac{t \xrightarrow{\alpha} t'}{(t,E) \xrightarrow{\alpha} (t',E)}$$

$$\frac{v=t' \in E \text{ et } t[^{t'}/_{v}] \xrightarrow{\alpha} t''}{(t,E) \xrightarrow{\alpha} (t'',E)}$$

(α): $\alpha.t \xrightarrow{\alpha} t$ (ACTION)

(+):

(1) $$\frac{t_1 \xrightarrow{\alpha} t'_1}{t_1+t_2 \xrightarrow{\alpha} t'_1}$$

(2) $$\frac{t_2 \xrightarrow{\alpha} t'_2}{t_1+t_2 \xrightarrow{\alpha} t'_2}$$

(CHOIX NON DETERMINISTE)

(||)

(1) $$\frac{t_1 \xrightarrow{\alpha} t'_1}{t_1 \| t_2 \xrightarrow{\alpha} t'_1 \| t_2}$$

(2) $$\frac{t_2 \xrightarrow{\alpha} t'_2}{t_1 \| t_2 \xrightarrow{\alpha} t_1 \| t'_2}$$

(COMPOSITION PARALLELE)

(3) $$\frac{t_1 \xrightarrow{\alpha} t'_1 \, , \, t_2 \xrightarrow{\sim\alpha} t'_2}{t_1 \| t_2 \xrightarrow{\tau} t'_1 \| t'_2}$$

(\ ) $$\frac{t \xrightarrow{\alpha} t' \, , \, \alpha \neq \beta}{t \backslash \beta \xrightarrow{\alpha} t'}$$

(EFFACEMENT)

([ ]) $$\frac{t \xrightarrow{\alpha} t'}{t[f] \xrightarrow{f(\alpha)} t'[f]}$$ (REDEFINITION)

Notation:

- Soit $s = \alpha_1 . \ldots . \alpha_n$, $n>0$ et $\alpha_i \in P$
  alors $p \ (\xrightarrow{\tau})^* \xrightarrow{\alpha_1} (\xrightarrow{\tau})^* \cdots \xrightarrow{\alpha_n} (\xrightarrow{\tau})^* p'$
  est noté $p \Rightarrow s \Rightarrow p'$.
- $t \ (\xrightarrow{\tau})^* \ t'$ est noté $t \Rightarrow t'$.
- $t \ (\xrightarrow{\tau})^+ \ t'$ est noté $t \Rightarrow \tau \Rightarrow t'$.

I.3.- Relations d'équivalence et de congruence.

R. Milner définit des relations d'équivalence et de congruence sur T'_r. Ces relations définissent des critères de comparaison entre termes. Par la suite nous présentons trois relations définies pour T'_r. Ces relations sont:

- la *congruence forte* .
- l'*équivalence observationnelle*
- la *congruence observationnelle*

Nous présentons également une relation auxiliaire utilisée pour comparer des termes appartenant à T_s.

I.3.1. *Congruence Forte.*

Soient: $p, q \in T'_r$,

R une relation binaire sur T'_r et F(R) la fonction:

$(p,q) \in F(R)$ ssi (i) $p \xrightarrow{\alpha} p'$ implique $\exists q'$ $(q \xrightarrow{\alpha} q'$ et $(p',q') \in R)$ et
(ii) $q \xrightarrow{\alpha} q'$ implique $\exists p'$ $(p \xrightarrow{\alpha} p'$ et $(p',q') \in R)$,

Définition 1.1.1.

On dit que R est une *bisimulation* si et seulement si $R \subseteq F(R)$.

Définition 1.1.2.

L'*équivalence forte* (notée ~) est définie comme la plus grande bisimulation, c'est-à-dire:

$$\sim = \cup \{R \subseteq T'_f \times T'_f \mid R \subseteq F(R)\}.$$

La relation ~ existe et elle est unique parce que F est monotone dans le treillis des relations binaires avec l'inclusion d'ensembles ($\subseteq$).

Pour les termes appartenant à T_f, l'*équivalence forte* est axiomatisée par: (A_1)-(A_{16}) de la Table 1. Cette équivalence est une relation de congruence pour les opérateurs α, + et ||. Par la suite nous l'appelons *congruence forte.*

I.3.2.- *Equivalence observationnelle.*

L'équivalence observationnelle est définie exactement comme la congruence forte, sauf que les relations $=\alpha\Rightarrow$ et $\Rightarrow$ sont utilisées à la place de $\xrightarrow{\alpha}$ et $\xrightarrow{\tau}$. L'équivalence observationnelle est notée $\approx$.

Par exemple, les termes $\alpha.nil$, $\tau.\alpha.nil$, $\tau.\alpha.\tau.nil$, $\tau.\alpha.nil+\alpha.nil$ sont observationnellement équivalents.

Proposition 1.I.1.

Pour deux termes $t_1, t_2 \in T_r$, $t_1 \approx t_2$ si et seulement si les deux conditions suivantes sont vérifiées:

(i) $t_1 = \alpha \Rightarrow t'_1$ implique $(\alpha = \tau$ et $t'_1 \approx t_2)$

ou $\exists t'_2 (t_2 = \alpha \Rightarrow t'_2$ et $t'_1 \approx t'_2)$

(ii) $t_2 = \alpha \Rightarrow t'_2$ implique $(\alpha = \tau$ et $t'_2 \approx t_1)$

ou $\exists t'_1 (t_1 = \alpha \Rightarrow t'_1$ et $t'_1 \approx t'_2)$.

♦

Proposition 1.I.2.

$p \sim q$ implique $p \approx q$. ♦

La relation d'équivalence observationnelle n'est pas une relation de congruence. Par exemple: $a.nil \approx \tau.a.nil$ mais, $b.nil + a.nil$ *n'est pas observationnellement équivalent à* $b.nil + \tau.a.nil$.

Pour les termes appartenant à T_f, l'équivalence observationnelle est axiomatisée par: (A_1)-(A_{20}) de la Table 1.

I.3.3.- *Congruence Observationnelle.*

La *congruence observationnelle* (notée $\approx^c$) est définie comme la plus grande relation de congruence pour les opérateurs α, + et || incluse dans l'équivalence observationnelle.

Proposition 1.I.3.

Pour deux termes $t_1, t_2 \in T_r$, $t_1 \approx^c t_2$ si et seulement si $\forall \alpha \in P$, les deux conditions suivantes sont vérifiées:

(1) $t_1 \xrightarrow{\alpha} t'_1$ implique $\exists t'_2$ $(t_2 = \alpha \Rightarrow t'_2$ et $t'_1 \approx t'_2)$

(2) $t_2 \xrightarrow{\alpha} t'_2$ implique $\exists t'_1$ $(t_1 = \alpha \Rightarrow t'_1$ et $t'_1 \approx t'_2)$

♦

Dans la proposition suivante nous donnons une nouvelle caractérisation de la congruence observationnelle. Cette caractérisation est utilisée dans la définition de la procédure de décision pour la congruence observationnelle (chapitre III).

Proposition 1.I.4.

Pour deux termes $t_1, t_2 \in T_r$, $t_1 \approx^c t_2$ si et seulement si les trois conditions suivantes sont vérifiées:

- $t_1 \approx t_2$,
- $t_1 \xrightarrow{\tau} t'_1$ implique $\exists\, t'_2$ tel que $t_2 = \tau \Rightarrow t'_2$ et $t'_1 \approx t'_2$,
- $t_2 \xrightarrow{\tau} t'_2$ implique $\exists\, t'_1$ tel que $t_1 = \tau \Rightarrow t'_1$ et $t'_1 \approx t'_2$.

Preuve.

La preuve est facile à établir par l'absurde à partir des propositions 1.I.1 et 1.I.3.

♦

Pour les termes appartenant à T_f, l'équivalence observationnelle est axiomatisée par: (A_1)-(A_{19}) de la Table 1.

On définit aussi une autre relation de congruence appelée *congruence* (A_1)-(A_2) notée $\approx^S$ entre les termes finis comme la congruence induite par les axiomes (A_1) et (A_2) de la Table 1.

Milner a montré la complétude d'une axiomatisation pour la congruence observationnelle pour les termes récursifs. Cette axiomatisation est définie pour des termes récursifs sous une forme différente de la notre, mais les axiomes et règles d'inférence de notre formalisme peuvent être obtenus des siens par des transformations syntaxiques. Ainsi, une axiomatisation complète pour la congruence observationnelle des termes de T_r est définie par les axiomes (A_1)-(A_{19}) de la Table 1 et les axiomes et règles d'inférence donnés dans la Table 2.

Nous remarquons qu'en fait, les axiomes (A_6), (A_7), (A_{12}), (A_{14}), (A_{16}) exprimant des propriétés d'associativité, de commutativité et de distributivité pour les opérateurs ||, \ et [ ] ne peuvent pas être déduits des autres axiomes; mais dans la suite on n'utilise jamais ces axiomes parce que toutes les instances de ces axiomes peuvent être déduites.

TABLE 1:

(+):	(A_1):	$t_1 + (t_2 + t_3) = (t_1 + t_2) + t_3$
	(A_2):	$t_1 + t_2 = t_2 + t_1$
	(A_3):	$t + t = t$
	(A_4):	$t + nil = t$
(||):	(A_5):	Si t_1 et t_2 sont de la forme: $t_1 = \sum_{i=1,\ldots,n} \alpha_i . t'_i$ et $t_2 = \sum_{j=1,\ldots,n} \beta_j . t'_j$ alors: $t_1 \| t_2 = \sum \alpha_i . (t'_i \| t_2) + \sum \beta_j . (t_1 \| t'_j) + \sum \tau . (t'_i \| t'_j)$ $\alpha_i = \sim\beta_j \wedge \sim\alpha_i = \beta_j \wedge i=1,\ldots,n \wedge j=1,\ldots,n$
	(A_6):	$t_1 \| t_2 = t_2 \| t_1$
	(A_7):	$t_1 \| (t_2 \| t_3) = (t_1 \| t_2) \| t_3$
	(A_8):	$t \| nil = t$
([]):	(A_9):	$(\alpha.t)[f] = f(\alpha).(t[f])$
	(A_{10}):	$(t_1 + t_2)[f] = t_1[f] + t_2[f]$
	(A_{11}):	$nil[f] = nil$
	(A_{12}):	$(t_1 \| t_2)[f] = t_1[f] \| t_2[f]$
(\):	(A_{13}):	$(\alpha.t)\backslash\beta = \begin{cases} nil & \text{si } \alpha = \beta \text{ ou } \alpha = \sim\beta \\ \alpha.(t\backslash\beta) & \text{sinon} \end{cases}$
	(A_{14}):	$(t_1+t_2)\backslash\alpha = t_1\backslash\alpha + t_2\backslash\alpha$
	(A_{15}):	$nil\backslash\alpha = nil$
	(A_{16}):	$t\backslash\alpha_1\backslash\alpha_2 = t\backslash\alpha_2\backslash\alpha_1$
(τ):	(A_{17}):	$t+\tau.t = \tau.t$
	(A_{18}):	$\alpha.(t_1+\tau.t_2) = \alpha.(t_1+\tau.t_2)+\alpha.t_2$
	(A_{19}):	$\alpha.\tau.t = \alpha.t$
	(A_{20}):	$\tau.t = t$

TABLE 2:

---

(1): $$\frac{t_1 = t_2}{(t,E\cup\{x_1=t_1\}) = (t,E\cup\{x_2=t_2\})}$$

(2): $$(t, E\cup\{x_1=t_1, x_2=t_2\}) = (t, E\cup\{x_1=t_1, x_2=t_2[^{t}1/x_1]\})$$

(3): $$\frac{(z_i,E) = (t_i[^{z}j/x_j]_{j=1,\ldots,n}, E)}{(z_i,E) = (x_i, \{x_j=t_j\}_{i=1,\ldots,n})} \quad \text{var}(t_i)\ \{x_1,\ldots,x_n\}$$

(4): $$\frac{x \notin \text{utile}(t,E)}{(t,E) = (t,E\cup\{x=t_1\})}$$

(5): $$(t,E\cup\{x=\tau.x+t_1\}) = (t, E\cup\{x=\tau.t_1\})$$

(6): $$(t,E\cup\{x=\tau.(x+t_1)+t_2\}) = (t, E\cup\{x=\tau.x+t_1+t_2\})$$

II.- Vérification de termes finis.

Dans le cas des termes décrivant des comportements finis (*termes finis*), nous décidons de l'équivalence et de la congruence observationnelle par construction de formes normales modulo $\approx^S$ des termes. C'est-à-dire, à tous les termes $t_1, t_2 \in T_f$ on associe des termes $t_{n1}, t_{n2} \in T_S$ tels que $t_1 \approx^C t_2$ si et seulement si $t_{n1} \approx^S t_{n2}$. La forme normale choisie correspond au terme minimal de la classe d'équivalence ou de congruence. Donc, pour comparer les termes, nous vérifions si leurs formes normales sont équivalentes via la relation de congruence induite par les axiomes (A1)-(A2) (associativité et commutativité du "+"), ce qui est facile à faire.

Par exemple, considérons les termes:

(1) $\tau.(a.(\tau.nil+nil)+a.nil)+b.nil$

(2) $b.nil + \tau.(nil+\tau.(nil+\tau.a.nil))$

leurs formes normales pour la congruence observationnelle sont:

(1) $\tau.a.nil + b.nil$

(2) $b.nil + \tau.a.nil.$

Comme ces formes normales sont congrues modulo (A_1)-(A_2), les termes donnés sont observationnellement congrus.

II.1.- Forme normale pour la congruence observationnelle.

Pour définir la forme normale pour la congruence observationnelle d'un terme de T_f, nous montrons que tout terme décrivant un comportement fini est observationnellement congru à un autre terme écrit sous la forme d'une somme.

Ensuite, nous montrons qu'il n'est pas possible de construire directement à partir des axiomes de la congruence observationnelle, un système de réécriture nœthérien et confluent permettant d'obtenir la forme normale d'un terme.

Afin d'éviter cet inconvenient, nous introduisons la notion de *comportement dérivé* d'une expression; cette notion permet de construire un axiome conditionnel et de caractériser une relation de congruence et la *forme normale* des termes. Nous montrons que la relation de congruence définie à l'aide de l'axiome conditionnel est exactement la relation de congruence observationnelle. Cette nouvelle axiomatisation permet d'obtenir un système de réécriture nœthérien et confluent.

Nous utilisons cette forme normale pour la construction d'une procédure de décision pour la congruence observationnelle des termes finis.

Proposition 1.II.1.

Tout terme $t \in T_f$ est congru à un terme $t_s \in T_s$.

Preuve.

Effectivement, par l'application exhaustive des axiomes (A_4), (A_5), (A_8), (A_9), (A_{10}), (A_{11}), (A_{13}), (A_{14}), (A_{15}) dans le sens gauche-droite et en appliquant aussi les axiomes (A_1),(A_2) et (A_6) dans les deux sens on peut transformer tout terme de T_f en un terme de T_S.

♦

Pour construire la forme normale d'un terme, il est souhaitable de construire un ensemble de règles de réécriture nœthérien et confluent définissant une procédure automatisable, à partir des axiomes de la somme et ceux de "τ".

Mais, il n'est pas possible de construire un ensemble de règles de réécriture directement à partir des axiomes de la congruence observationnelle puisque l'axiome:

(A_{18}): $\alpha(t_1 + \tau t_2) + \alpha t_2 = \alpha(t_1 + \tau t_2)$

est nécessairement utilisé dans les deux sens pour certaines preuves.

Par exemple, pour construire la forme normale du terme:

a. nil + τ. (c. nil + τ. a. nil)

à l'aide des axiomes de la congruence observationnelle, nous utilisons d'abord l'axiome (A_{18}) dans le sens droite-gauche (<--) pour obtenir le terme:

a. nil + τ. (c. nil + τ. a. nil) + τ. a. nil,

ensuite nous éliminons le sous-terme "a.nil" grace à l'axiome (A_{17}), et finalement nous réutilisons l'axiome (A_{18}), cette fois dans le sens gauche-droite (-->) pour éliminer le sous-terme "τ. a. nil" introduit précédemment, et obtenir la forme normale:

τ. (c. nil + τ. a. nil).

Vue l'impossibilité de construire un ensemble de règles de réécriture nœthérien et confluent directement à partir des axiomes de "+" et ceux de "τ", pour la définition de la procédure de construction de la forme normale d'un terme, nous introduisons un axiome conditionnel:

(A_C): $\alpha.t_1 + t = t$ si t_1 est un "α–dérivé" de t.

Intuitivement, si α est une action appartenant à P_τ, les α-dérivés d'une expression correspondent aux comportements possibles lorsqu'il y a eu une α-communication.

Par exemple, la forme normale du terme a. nil + τ. (c. nil + τ. a. nil) est construite par l'application de l'axiome (A_C). En effet, comme "nil" est un a-dérivé de "τ.(c.nil+τ.a.nil)" nous appliquons l'axiome (A_C) dans le sens gauche-droite (-->) pour

éliminer le sous-terme "a.nil" et obtenir la forme normale $\tau.(c.nil+\tau.a.nil)$.

A l'aide de cet axiome conditionnel, nous définissons une nouvelle relation d'équivalence entre termes. Nous caractérisons une forme normale pour cette relation et finalement nous montrons que cette nouvelle relation coïncide avec la congruence observationnelle. En effet, les axiomes (A_3), (A_{17}) et (A_{18}) peuvent être déduits à partir de l'axiome (A_C).

Définition 1.II.1.

Pour tout $\alpha \in P_\tau$, les *α-dérivés* d'un terme $t = \sum_{i=1,\ldots,n} \alpha_i.t_i$ sont définis de la manière suivante:

(i) t_i est α_i-dérivé de t,

(ii) tout τ-dérivé de t_i est un α_i-dérivé de t,

(iii) si $\alpha_i = \tau$ alors tout α-dérivé de t_i est un α-dérivé de t,

(iv) t' est α-dérivé de t si et seulement si t' est un α-dérivé de t par (i) à (iii).

Des résultats prouvés par la suite, on peut déduire que les α-dérivés d'un terme t sont les termes t' tels que:

$$t \Rightarrow\alpha\Rightarrow t'.$$

Par exemple, considérons les termes suivants:

- $\tau.(a.nil + \tau.(b.nil + c.(e.nil + \tau.(d.nil + \tau.nil))))$

les τ -dérivés sont:

$a.nil + \tau.(b.nil + c.(e.nil + \tau.(d.nil + \tau.nil)))$

$b.nil + c.(e.nil + \tau.(d.nil + \tau.nil))$

les c-dérivés sont:

$e.nil + \tau.(d.nil + \tau.nil)$, $d.nil + \tau.nil$, nil

- $a.nil + \tau.(b.nil + a.(\tau.nil + b.nil)) + b.\tau.nil$

les τ -dérives sont:

$b.nil + a.(\tau.nil + b.nil)$

les a-dérivés sont:

nil, $\tau.nil + b.nil$

les b-dérivés sont:

nil, $\tau.nil$

Définition 1.II.2.

La relation de congruence sur les termes de T_S induite par les axiomes (A_1), (A_2), (A_{19}) et (A_c) est notée $\simeq^c$.

Remarque: On prouve dans le paragraphe II.1.2 que les congruences $\simeq^c$ et $\approx^c$ coïncident sur T_S.

Pour la caractérisation de la forme normale modulo $\approx^s$ des termes finis, et afin de garantir que l'axiome $\alpha.\tau.t = \alpha.t$ n'est plus applicable à un terme sous forme normale, il est nécessaire d'introduire les notions de *forme normale propre* et de *forme normale impropre* .

Definitions 1.II.3.

On dit qu'un terme t est sous forme normale propre si et seulement si:

- soit t=nil,
- soit $t = \sum_{i=1,\ldots,n} \alpha_i.t_i$ et les conditions suivantes sont vérifiées:
  - i) t n'est pas de la forme $\tau.t'$,
  - ii) chaque t_i est sous forme normale propre,
  - iii) pour $i \neq j$, il n'existe pas de α_i-dérivé t' de $\alpha_j.t_j$ tel que $t_i \approx^S t'$.

On dit qu'un terme $t = \tau.\ t'$ est sous *forme normale impropre* si et seulement si t' est une forme normale propre.

On dit qu'un terme est sous *forme normale* si et seulement si il est sous forme normale propre ou impropre.

Remarque: Le théorème d'unicité prouvé par la suite justifie le nom *forme normale* donné à cette forme particulière de termes.

Exemples.

Expression	*Forme normale*
a.nil +a.(b.nil+ τ.nil)	a.(b.nil+ τ.nil)
a.nil + τ .(b.nil+ τ.(d.nil+a.nil))+d.nil	τ.(b.nil+ τ. (d.nil+a.nil))
a.nil+ τ.(nil+ τ.(nil+ τ.a.nil))	τ.a.nil
(a.nil || na.nil)+ τ.a.na.nil	τ.a.na.nil+na.a.nil+ τ.nil

II.1.1.- Existence et unicité de la forme normale.

Théorème d'existence.

Tout terme $t \in T_f$ est congru à une forme normale t_n.

Preuve.

D'après la proposition 1.II.1., il suffit de considérer les termes appartenant à T_s. La preuve est établie par induction sur la structure des termes.

- Pour t=nil, rien n'est à prouver.

- Supposons que $t = \Sigma_{i=1,\ldots,n} \alpha_i.t_i$ tel que pour tout i, il existe un t_{ni} sous forme normale tel que $t_{ni} \approx^C t_i$. Ceci implique $t \approx^C \Sigma_{i=1,\ldots,n} \alpha_i.t_{ni}$ ($\approx^C$ est une congruence). Sans perte de généralité, on peut supposer que chaque tn_i est une forme normale propre (sinon on applique l'axiome (A_{19}). S'il existe i, j (i≠j) tels que tn_i est congru (A_1)-(A_2) à un α_i-dérivé de $\alpha_j.tn_j$, alors l'axiome (A_c) (en appliquant eventuellement aussi les axiomes (A_1) et (A_2)) permet d'absorber le terme $\alpha_i.tn_i$. Après un nombre fini d'absorptions on obtient une forme normale.

♦

Théorème d'unicité.

Pour tout $t_1, t_2 \in T_f$ sous forme normale, $t_1 \approx^C t_2$ si et seulement si $t_1 \approx^S t_2$.

Preuve.

(<==): Directe du fait que l'axiomatisation de $\approx^S$ est incluse dans celle de $\approx^C$.

(==>): La preuve est établie par induction sur la structure des termes.

Pour ceci, il suffit de prouver que pour tout terme t_n sous forme normale, aucun des axiomes (A_{19}) et (A_c) n'est applicable et que l'application des axiomes (A_1) et (A_2) ne les rend pas de nouveau applicables (condition 1).

- Pour t_n = nil la condition 1 est vérifiée.
- Soit $t_n = \tau.t'_n$ tel que t'_n est une forme normale propre et vérifie la condition 1 (hypothèse d'induction).

Ces conditions impliquent que le terme t_n satisfait lui aussi la condition 1.

- Soit $t_n=\Sigma\alpha_i.t'_i$ une forme normale propre telle que chaque t'_i est une forme normale propre satisfaisant la condition 1.

(1): Puisque chaque t_i est une forme normale propre, (A_{19}) n'est pas applicable sur t_n.

(2): L'axiome (A_c) n'est pas applicable d'après la condition (iii) de la définition de forme normale propre.

♦

II.1.2.- Coincidence de $\underline{\approx}^c$ et $\approx^c$.

Afin de prouver que pour des termes finis écrits sous la forme d'une somme, la congruence observationnelle est exactement la congruence utilisée dans la caractérisation de la forme normale, nous prouvons que:

$$\forall\ t_1, t_2 \in T_S,\ t_1 \approx^c t_2 \text{ si et seulement si } t_1 \underline{\approx}^c t_2.$$

La preuve dans le sens droite à gauche peut être déduite facilement de la proposition suivante.

Proposition 1.II.2.

Si t' est un α-dérivé de t, alors $\alpha.t' + t \approx^c t$ (condition 1)

Preuve.

La preuve est réalisée par induction sur la structure des termes:

- Pour t = nil rien n'est à prouver.
- Supposons que pour $t = \Sigma_{i=1,\ldots,n}\ \alpha_i.t_i$, la condition 1 soit satisfaite pour tous les t_i (hypothèse d'induction).

Considérons tous les cas pour lesquels t' est un α-dérivé de t.

Cas 1: $\exists\, i : \alpha_i = \alpha$ et $t_i = t'$

On peut déduire les égalités suivantes:

$$\begin{aligned} \alpha.t' + t &\approx^c \alpha.t' + \Sigma\, \alpha_i.t_i \\ &\approx^c \Sigma\, \alpha_i.t_i \qquad \text{par } (A_3) \\ &\approx^c t \end{aligned}$$

Cas 2: $\exists i : \alpha_i = \alpha$ et t' est un τ-dérivé de t_i.

On peut déduire les égalités suivantes:

$\tau.t' + t_i \approx^c t_i$ (2) par hypothèse d'induction

d'où:

$\alpha.t_i \approx^c \alpha.(\tau.t'+t_i)$

$\approx^c \alpha.t' + \alpha.(\tau.t'+t_i)$ par (A_2) et (A_{18})

$\approx^c \alpha.t' + \alpha.t_i$ par (2)

donc, $\alpha.t' + t \approx^c t$.

Cas 3: $\exists i : \alpha_i = \tau$ et t' est un α-dérivé de t_i.

On peut déduire les égalités suivantes:

$\alpha.t' + t_i \approx^c t_i$ (3) par hypothèse d'induction

d'où:

$\tau.t_i \approx^c \tau.(\alpha.t' + t_i)$

$\approx^c \alpha.t' + \tau.(\alpha.t' + t_i)$ par (A_{17}), (A_1), (A_2), (A_3)

$\approx^c \alpha.t' + \tau.t_i$ par (3)

donc,

$\alpha.t' + t \approx^c t$

♦

Théorème 1.II.1.

Pour tous les termes $t_1, t_2 \in T_s$, $t_1 \approx^c t_2$ implique $t_1 \simeq^c t_2$.

Preuve.

Pour prouver ce théorème, il suffit de montrer que les axiomes (A_3), (A_{17}) et (A_{18}) de la congruence observationnelle peuvent être déduits à partir de l'axiome (A_c).

Ces déductions peuvent être faites par application répétée de l'axiome (A_c) à des termes de la forme t+t, t+τ.t et α(t+τ.t')+α.t'.

♦

II.2.- Forme normale pour l'équivalence observationnelle.

Définition 1.II.4.

On dit qu'un terme fini t est sous forme normale pour l'équivalence observationnelle si et seulement si il est sous forme normale propre (définitions 1.II.3)

Remarque: Les théorèmes d'existence et d'unicité montrés par la suite justifient le nom de *forme normale* donné à cette forme de termes.

Théorème d'existence.

Tout terme $t \in T_f$ est équivalent à un terme t_n sous forme normale.

Preuve.

Comme l'axiomatisation de la congruence observationnelle est incluse dans celle de l'équivalence observationnelle, nous utilisons les résultats présentés dans le paragraphe II.1.1. En effet, nous avons prouvé que pour tout terme de T_f il existe un terme équivalent de la forme $t=\Sigma\alpha_i.t_i$ ou $t=\tau.t_n$.

Mais, un terme de la forme $t=\tau.t_n$, peut être simplifié par application de l'axiome (A_{20}). Dans ce cas t_n est effectivement sous forme normale d'après la définition précédente.

♦

Théorème d'unicité.

Pour tout terme t_1, t_2 de T_f sous forme normale, $t_1 \approx t_2$ si et seulement si $t_1 \approx^S t_2$.

Preuve.

La preuve est analogue à celle du théorème d'unicité pour la congruence observationnelle.

♦

II.3.- Algorithmes de calcul des formes normales.

Nous présentons dans cette partie les algorithmes de calcul des formes normales des termes finis pour la congruence et pour l'équivalence observationnelle.

II.3.1.- Congruence Observationnelle.

L'algorithme qui calcule la forme normale d'un terme fini pour la congruence observationnelle est déduit directement de la définition de la forme normale donnée précédemment. Cet algorithme procède en deux étapes:

1.- Calcul du terme sous la forme d'une somme.
2.- Calcul de la forme normale d'un terme écrit sous la forme d'une somme.

II.3.1.1.- Calcul du terme sous la forme d'une somme.

Le calcul du terme sous la forme d'une somme, congru à un terme donné, est réalisé par application exhaustive de l'ensemble de règles de réécriture suivant:

$$t + nil \quad \text{-----}> \quad t$$
$$nil + t \quad \text{-----}> \quad t$$

$$\Sigma\ \alpha_i . t'_i \ \|\Sigma\ \beta_j . t'_j \ \text{--->} \Sigma\ \alpha_i . (t'_i \| \Sigma\ \beta_j . t'_j) + \Sigma\ \beta_j . (\Sigma\ \alpha_i . t'_i \| t'_j) + \Sigma\ \tau . (t'_i \| t'_j)$$
$$\alpha_i = \sim\beta_j \wedge \sim\alpha_i = \beta_j$$

$$nil \| t \quad \text{-----}> \quad t$$
$$t \| nil \quad \text{-----}> \quad t$$

$$(\alpha.t)[f] \quad \text{--->} \quad f(\alpha).(t[f])$$
$$(t_1+t_2)[f] \quad \text{--->} \quad t_1[f] + t_2[f]$$
$$nil[f] \quad \text{--->} \quad nil$$

$$\alpha.t\backslash\beta \quad \text{--->} \quad \begin{cases} nil, & \text{si } \alpha = \beta \\ \alpha.(t\backslash\beta), & \text{sinon} \end{cases}$$
$$(t_1+t_2)\backslash\alpha \quad \text{--->} \quad t_1\backslash\alpha + t_2\backslash\alpha$$
$$nil\backslash\alpha \quad \text{--->} \quad nil$$

L'application de ces règles tient compte des priorités et de l'associativité des opérateurs.

Cet ensemble de règles de réécriture est construit à partir des axiomes de la Table 1.

II.3.1.2.- Calcul de la forme normale d'un terme sous la forme d'une somme.

Proposition 1.II.3.

(i) nil est une expression sous forme normale,

(ii) $\forall \alpha \in P \cup \{\tau\}$, $\alpha.t$ est sous forme normale ssi t est sous forme normale et t n'est pas de la forme $\tau.t'$,

(iii) t_1+t_2 est sous forme normale ssi t_1 et t_2 sont sous forme normale et il n'existe aucun α-dérivé de t_1 qui est aussi un α-dérivé de t_2 et vice-versa.

(iv) toute expression sous forme normale est obtenue à partir de (i), (ii) et (iii).

Preuve.

La preuve est facilement établie par induction sur la structure des termes.

♦

L'algorithme qui calcule la forme normale d'un terme mis sous forme d'une somme consiste à appliquer la définition de la forme normale d'une manière récursive sur chaque sous-terme; pour ceci, on part des sous-termes les plus imbriqués en appliquant de manière exhaustive les règles de réécriture:

- $\alpha.\ \tau.B$ ---> $\alpha.B$,	obtenue à partir de l'axiome (A_{19}) de la Table 1.
- $t' + t$ ---> t	où: t, t' sont deux formes normales et t' est un comportement dérivé de t. (Axiome (A_c)).

Pour pouvoir appliquer la deuxième règle, il peut être nécessaire d'appliquer d'abord $t+t'$ ---> $t'+t$ (Axiome (A_2)).

Définition 1.II.5.

Pour deux termes t,t' sous forme normale, on dit que t' est un *comportement dérivé* de t si et seulement si $t' + t \approx t$

Proposition 1.II.4.

(i) nil est un comportement dérivé de nil ,

(ii) $\alpha.t$ est un comportement dérivé de $\alpha.t'$ ssi
soit $t \approx^S t'$
soit t' est de la forme $\Sigma\alpha_i.t_i + \tau.t''$ et $\alpha.t$ est un comportement dérivé de $\alpha.t''$

(iii) t est un comportement dérivé de $\tau.t'$ ssi
soit $t \approx^S t'$
soit t est un comportement dérivé de t' ,

(iv) $\alpha.t$ est un comportement dérivé de $t' + t''$ ssi $\alpha.t$ est un comportement dérivé de t' ou de t" ,

(v) si t, t' sont deux comportements dérivés de t" alors t+t' est un comportement dérivé de t"

(vi) tout comportement dérivé d'un terme sous forme normale peut être obtenu a partir de (i)-(v).

Preuve.

La preuve est établie par induction sur la structure des termes. ♦

II.3.2.- Equivalence Observationnelle.

L'algorithme qui calcule la forme normale d'un terme fini pour l'équivalence observationnelle procède en deux étapes:

1.- Calcul de la forme normale du terme pour la congruence observationnelle.

2.- Si il y a lieu, application de la règle de réécriture $\tau.t$ --> t (axiome (A_{20})).

III. Vérification des termes réguliers.

Dans cette partie, nous présentons une procédure de décision pour la congruence forte, pour l'équivalence et pour la congruence observationnelle de termes décrivant des comportements réguliers (*termes réguliers*).

Cette procédure utilise la notion de *système de transitions* pour représenter les dérivations des termes et la notion de *bisimulation* pour les comparer. A partir de ces notions et sur la base de la proposition 1.I.1., on obtient directement une procédure de décision pour la congruence forte. A l'aide des propositions 1.I.2. et 1.I.4. nous définissons les *transformations EO* et *CO* permettant de décider de l'équivalence et de la congruence observationnelle par*simulation*.

Pour décider chacune des équivalences sur T_r, nous procédons en trois étapes:

1. Construction des *systèmes de transitions* associés aux termes à comparer.
2. Application de transformations ad hoc sur ces systèmes de transitions .
3. Décision de l'équivalence par bisimulation entre les systèmes de transitions transformés.

III.1.- Construction des systèmes de transitions.

Définition 1.III.1.

Un *système de transitions* S_u *associé à un terme* $u=(t,E)\in T_r$ est défini de la manière suivante:

$$S_u = (Q, \ \cup_{\alpha \in P\tau} R_\alpha , t)$$

où:

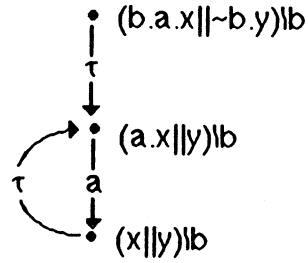
Q est un ensemble d'états correspondant à l'ensemble de termes obtenus par dérivation à partir de u,

$R_\alpha \subseteq Q \times Q$ est définie par : $(q',q'') \in R_\alpha$ si et seulement si $(t',E) \xrightarrow{\alpha} (t'',E)$ et $q'=(t',E)$ et $q''=(t'',E)$.

Pour noter $(p,q) \in R_\alpha$ on écrit aussi $p \xrightarrow{\alpha} q$ et de manière analogue on utilise les notations $=\alpha\Rightarrow$ et $\Rightarrow$.

u est l'état initial du système.

Par exemple, pour le terme ((b.a.x||~b.y)\b,{x=b.a.x, y=~b.y}), le système de transitions est:



On étend les relations $\sim$, $\approx$ et $\approx^c$ définies sur les termes, aux systèmes de transitions correspondants; c'est-à-dire, pour tous termes $t_1, t_2 \in T_r$,

(i) $t_1 \sim t_2$ si et seulement si $S_{t1} \sim S_{t2}$

(ii) $t_1 \approx t_2$ si et seulement si $S_{t1} \approx S_{t2}$

(iii) $t_1 \approx^c t_2$ si et seulement si $S_{t1} \approx^c S_{t2}$

III.2.- Transformations sur des systèmes de transitions.

Dans cette partie, nous définissons les transformations *EO* et *CO*. Nous montrons quelques proprietés intéressantes de ces transformations et leur utilisation pour décider de l'équivalence et de la congruence observationnelle des termes de T_r.

2.1.- Transformation *EO*.

Théoriquement, pour décider de l'équivalence observationnelle de termes réguliers, il suffit de construire pour chaque terme le système de transitions O(S) représentant les relations $=\alpha\Rightarrow$ pour $\alpha\in P$ et $\Rightarrow$ pour l'action τ et décider de l'équivalence par bisimulation entre les systèmes de transitions construits (proposition 1.I.2).

Définition 1.III.2.

Si $S=(Q,\cup_{\alpha\in P\tau} R_\alpha, q_0)$ alors $O(S)=(Q,\cup_{\alpha\in P\tau} R'_\alpha, q_0)$ est le système de transitions tel que pour tout $\alpha\in P$ les deux conditions suivantes sont vérifiées:

- $q\xrightarrow{\alpha}q'$ est une transition de $O(S)$ si et seulement si $q=\alpha\Rightarrow q'$ est une dérivation de S
- $q\xrightarrow{\tau}q'$ est une transition de$O(S)$ si et seulement si $q\Rightarrow q'$ est une dérivation de S ($\xrightarrow{\tau}$ est réflexive en $O(S)$).

En suivant la terminologie de Bergstra et Klop [BK 85], nous appelons cette transformation de S en $O(S)$, *saturation de* S.

Par exemple, si l'on veut décider de l'équivalence observationnelle des termes u et v définis par:

u= ((x||y)\d , {x=d.~d.x+a.c.nil , y=~d.(b.nil+d.y)})
v= ((x||y)\d , {x=d.(b.nil+a.~d.nil) , y=~d.d.c.nil})

- On construit les systèmes de transitions S_u et S_v (Figures S_u et S_v).

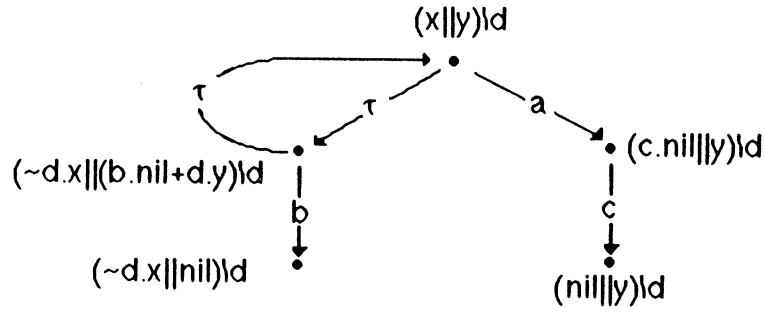


Figure S_u

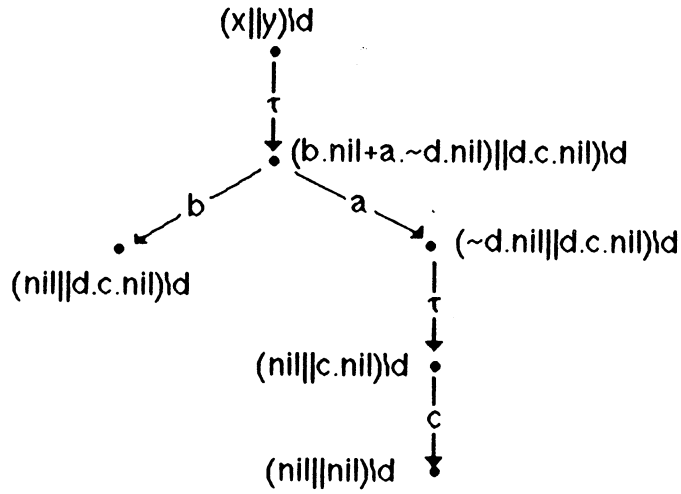


Figure S_v

- A partir de S_u et de S_v, on construit les systèmes de transitions $O(S_u)$ et $O(S_v)$ (Figure R).

- On calcule la relation de bisimulation R entre les systèmes $O(S_u)$ et $O(S_v)$ illustrée dans la figure R:

R = { ((x||y)\d , (x||y)\d) ,
((x||y)\d , ((b.nil+a.~d.nil)||d.c.nil)\d) ,
((c.nil||y)\d , (~d.nil||d.c.nil)\d) ,
((c.nil||y)\d , (nil ||c.nil)\d),
((nil||y)\d , (nil||nil)\d) ,
((nil||y)\d , (nil||d.c.nil)\d),
((~d.x||(b.nil+d.y))\d , (x||y)\d),
((~d.x||(b.nil+d.y))\d , ((b.nil+a.~d.nil)||d.c.nil)\d) ,
((~d.x||nil)\d , (nil||nil)\d) ,
((~d.x||nil)\d , (nil||d.c.nil)\d)}, pour finalement conclure que u≈v.

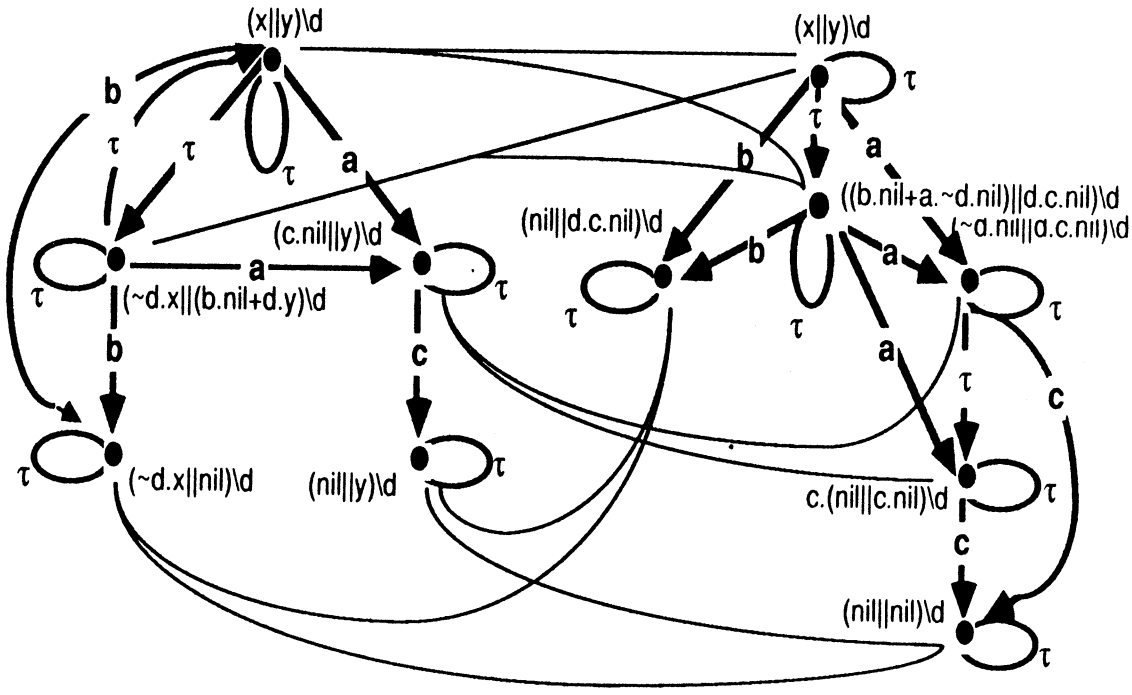


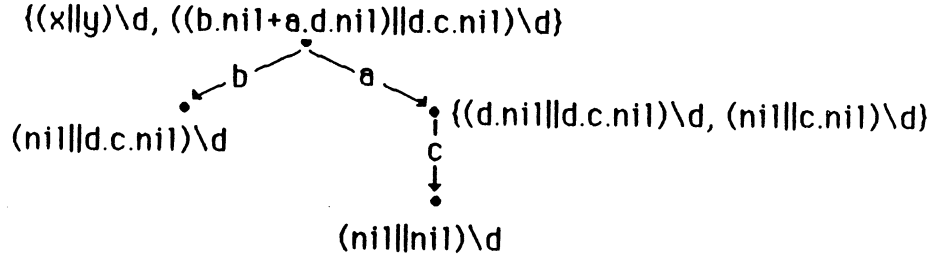
Figure R.

Etant donné que la transformation observationnelle ajoute des transitions au système, il est intéressant d'essayer de réduire le système initial avant l'application de cette transformation.

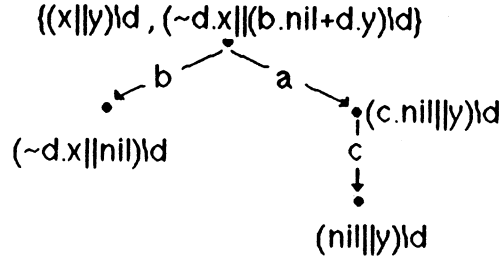
En effet, dans certains cas, il est possible de réduire le nombre d'états (sous-termes) et de transitions (dérivations) à traiter. Ces réductions peuvent être définies d'une manière syntaxique à l'aide des règles de réécriture sur les systèmes de transitions associés aux termes. Elles permettent ainsi de réduire dans certains cas considérablement la quantité de calcul nécessaire pour décider de l'équivalence observationnelle de termes.

Les réductions que nous proposons correspondent à:

- l'élimination des *chaînes* d'actions non observables (*τ-chaînes*): ie. dans l'exemple précédent, pour le système de transitions associé au terme v cette réduction produit le système de transitions:



- l'élimination des *circuits* d'actions non observables (*τ–circuits*): ie. dans l'exemple précédent, pour le système de transitions associé au terme u, cette réduction produit le système de transitions:



De cette manière, pour la vérification de l'équivalence observationnelle de l'exemple antérieur:

- nous construisons les systèmes de transitions montrés dans la Figure R'.

- nous calculons la relation de bisimulation R' illustrée dans la figure R'.

R'= { ((x||y)\d , (x||y)\d) ,
((c.nil||y)\d , (~d.nil||d.c.nil)\d) ,
((nil||y)\d , (nil||nil)\d) ,
((nil||y)\d , (nil||d.c.nil)\d),
((~d.x||nil)\d , (nil||nil)\d) ,
((~d.x||nil)\d , (nil||d.c.nil)\d)}, permettant de conclure que u≈v.

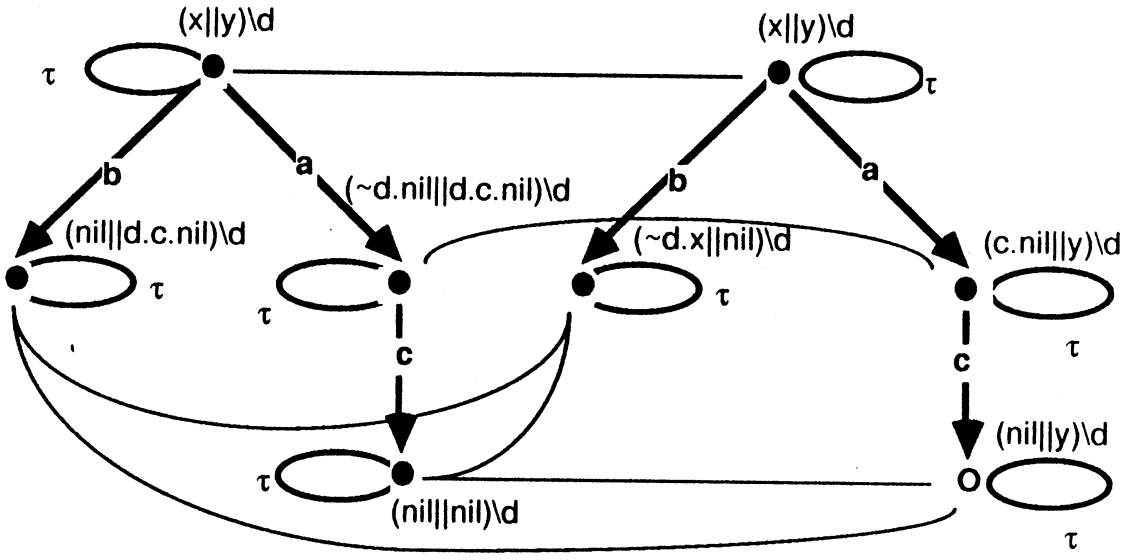


Figure R'

Par la suite, nous définisons les transformations des *τ–chaînes* et *τ-circuits* sur les systèmes de transitions associés aux termes. Nous montrons que chacune de ces transformations préserve l'équivalence observationnelle. Nous définissons la transformation *EO* sur la base de ces transformations et nous montrons qu'elle permet de décider de l'équivalence observationnelle par *bisimulation.*

Définitions 1.III.3.

Pour un système de transitions $S = (Q, \cup_{\alpha \in P\tau} R_\alpha, t)$,

- On dit qu'une transition $p \xrightarrow{\tau} q$ est une *chaîne de* τ dans S si et seulement si elle est la seule transition possible à partir de l'état p.
- On dit que $p_1 \xrightarrow{\tau} p_2, \ldots, p_n \xrightarrow{\tau} p_1 \in R_\tau$ et $p_1 \xrightarrow{\tau} p_1 \in R_\tau$ constituent des *circuits de* τ dans S.

Propriété 1.III.1.

Pour tous les systèmes de transitions S_1, S_2, la propriété suivante est vérifiée:

$S_1 \approx S_2$ si et seulement si $O(S_1) \sim O(S_2)$.

Preuve: Evidente.

♦

Définition 1.III.4.

Pour un système de transitions $S_u = (Q, \cup_{\alpha \in P\tau} R_\alpha, u)$, l'élimination des chaînes de τ (transformation *τ-chaînes*) consiste en:

- l'élimination de toute transition $t \xrightarrow{\tau} t' \in R_\tau$, si elle est la seule transition possible à partir de l'état t,
- la substitution de l'état t' par l'état t dans le système résultant.

Propriété 1.III.2.

La transformation τ-chaînes préserve l'équivalence observationnelle. C'est-à-dire, deux systèmes de transitions S_1 et S_2 sont observationnellement équivalents si et seulement si après l'application de la transformation τ-chaînes ils sont observationnellement équivalents.

Preuve.

Soit $S=(Q\cup\{p,p'\}, \cup_{\alpha\in P\tau} R_\alpha, q_0)$, un système de transitions contenant la chaîne de τ: $p\xrightarrow{\tau}p'$,

$S'=(Q\cup\{p\}, \cup_{\alpha\in P\tau} R'_\alpha, q_0)$ le système obtenu à partir de S en appliquant la transformation τ-circuits seulement à la chaîne $p\xrightarrow{\tau}p'$.

Par construction de S', on obtient que: $\forall q,q'\in Q\cup\{p\}\ \forall\alpha\in P$

$q=\alpha\Rightarrow q'$ est une transition de S si et seulement si $q=\alpha\Rightarrow q'$ est une dérivation de S'

et $p'=\alpha\Rightarrow q$ est une transition de S si et seulement si $p=\alpha\Rightarrow q$ est une dérivation de S'

et $q=\alpha\Rightarrow p'$est une transition de S si et seulement si $q=\alpha\Rightarrow p$ est une dérivation de S'.

A partir de ce résultat il est facile à vérifier que $R=\{(q,q)|\ q\in Q\cup\{p\}\}$ est une bisimulation entre $O(S)$ et $O(S')$.

Par généralisation de ces résultats, on conclut que la transformation τ-chaînes préserve l'équivalence observationnelle.

♦

Définition 1.III.5.

Pour un système de transitions $S_u = (Q, \cup_{\alpha \in P\tau} R_\alpha, u)$, l'élimination des circuits de τ

$$t_1 \xrightarrow{\tau} t_2, \ldots, t_n \xrightarrow{\tau} t_1 \in R_\tau \quad \text{ou} \quad t_1 \xrightarrow{\tau} t_1 \in R_\tau$$

(transformation *τ-circuits*) consiste en:

- l'élimination de toute transition $t_i \xrightarrow{\tau} t_j$ du circuit,
- la substitution de chaque état du circuit par t_1 dans le système résultant.

Propriété 1.III.3.

La transformation τ-circuits préserve l'équivalence observationnelle.

Preuve.

(i) Soit $S=(Q\cup\{p\}, \cup_{\alpha\in P\tau} R_\alpha, q_0)$,

$p \xrightarrow{\tau} p$ un circuit de τ dans S,

$S'=(Q\cup\{p\}, \cup_{\alpha\in P\tau} R'_\alpha, q_0)$ le système obtenu à partir de S en appliquant la transformation τ-circuits seulement au circuit $p \xrightarrow{\tau} p$. C'est-à-dire, $R'_\alpha = R_\alpha \ \forall \alpha \in P$ et $R'_\tau = R_\tau - \{p \xrightarrow{\tau} p\}$.

Evidemment $\forall q,q' \in Q\cup\{p\}$, $q \stackrel{\alpha}{\Rightarrow} q'$ est une dérivation de S si et seulement si $q \stackrel{\alpha}{\Rightarrow} q'$ est une dérivation de S' et ceci implique $S \approx S'$.

(ii) Soit $S=(Q\cup\{p_1,\ldots,p_n\}, \cup_{\alpha\in P\tau} R_\alpha, q_0)$, un système de transitions contenant le τ-circuit $p_1 \xrightarrow{\tau} p_2, \ldots, p_n \xrightarrow{\tau} p_1$,

$S'=(Q\cup\{p_1\}, \cup_{\alpha\in P\tau} R'_\alpha, q_0)$ le système obtenu à partir de S en appliquant la transformation τ-circuits seulement au circuit $p_1 \xrightarrow{\tau} p_2, \ldots, p_n \xrightarrow{\tau} p_1$.

On obtient aisement que:

$\forall q,q'\in Q\cup\{p_1\}$ $q=\alpha\Rightarrow q'$ est une dérivation de S si et seulement si $q=\alpha\Rightarrow q'$ est une dérivation de S'

et $\forall p_i, p_j\in\{p_1, \ldots, p_n\}$ $\forall q\in Q\cup\{p_1\}$ $\forall\alpha\in P$

$p_i=\alpha\Rightarrow p_j$ est une dérivation de S si et seulement si $p_1=\alpha\Rightarrow p_1$ est une dérivation de S' et

$p_i=\alpha\Rightarrow q$ est une dérivation de S si et seulement si $p_1=\alpha\Rightarrow q$ est une dérivation de S' et

$q=\alpha\Rightarrow p_i$ est une dérivation de S si et seulement si $q=\alpha\Rightarrow p_1$ est une dérivation de S'.

A partir de ce résultat, il est facile à vérifier que la relation $R=\{(q,q)|\ q\in Q\}\cup\{(p_i,p_1)\ i\in\{1, \ldots, n\}\}$ est une bisimulation entre $O(S)$ et $O(S')$.

Par généralisation de ces résultats, on conclut que la transformation τ-circuits préserve l'équivalence observationnelle.

♦

Définition 1.III.6.

Pour tout système de transitions S, la transformation *EO* est définie comme *EO* (S) = *O* (S') où S' est le système obtenu par application des transformations τ–chaînes et τ-circuits sur S.

Proposition 1.III.1.

Pour tous les termes t_1, t_2 de T_r
$t_1 \approx t_2$ si et seulement si $EO(S_{t1}) \sim EO(S_{t2})$.

Preuve.

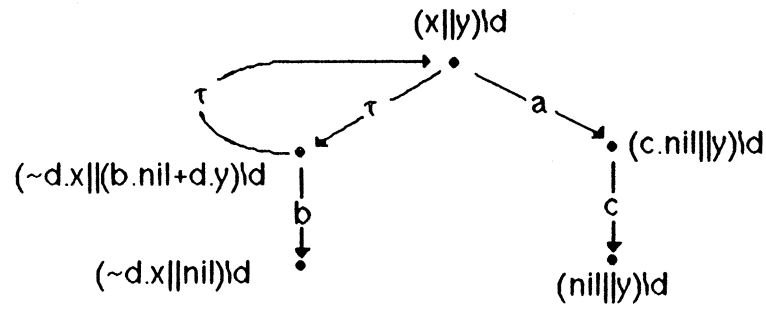
Conséquence immédiate des propriétés 1.III.1. à 1.III.3.

♦

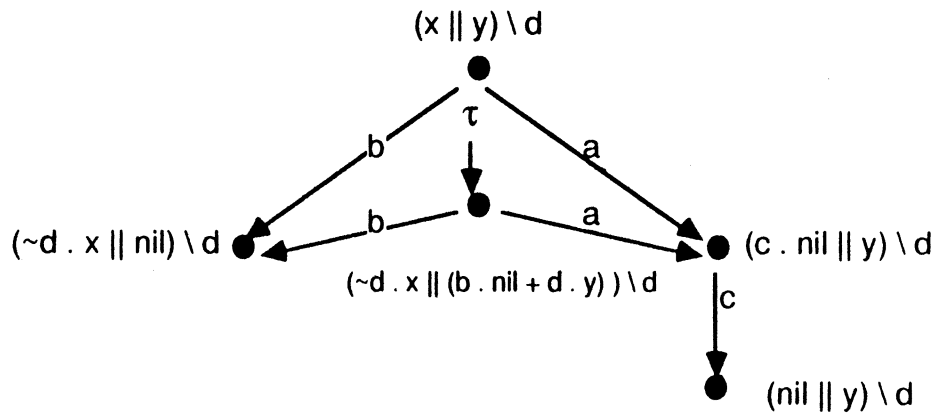
III.2.2.- Transformation *CO.*

De manière analogue, nous définissons une transformation (*transformation CO*) sur les systèmes de transitions pour décider de la congruence observationnelle. Nous montrons que cette transformation préserve la congruence observationnelle. Ainsi, la congruence observationnelle est décidée par simulation entre les systèmes de transitions obtenus par l'application de la *transformation CO*.

Par exemple, le système de transitions:



associé au terme u= ((x||y)\d , {x=d.~d.x+a.c.nil , y=~d.(b.nil+d.y)}) est transformé en:



par application de la transformation *CO* .

Définition 1.III.7.

Pour tout système de transitions $S=(Q, \cup_{\alpha\in P\tau} R_\alpha, q_0)$, la *transformation CO* procède en trois étapes:

1.- Construction du système de transitions $S'=(Q', \cup_{\alpha\in P\tau} R'_\alpha, q'_0)$ par application des transformations τ-chaînes et τ-circuits sur le système S.

2.- Si pendant la construction du système S', nous avons éliminé des τ-transitions partant de l'état initial q_0, nous ajoutons un nouveau état initial $q\notin Q'$, ainsi que la transition $q \xrightarrow{\tau} q'_0$ pour obtenir le système de transitions S".

3.- Transformation de S" en un système S'" tel que $\forall\alpha\in P_\tau$ $q\xrightarrow{\alpha} q'$ est une transition de S'" si et seulement si $q=\alpha\Rightarrow q'$ est une dérivation de S". Nous rémarquons que la relation $\xrightarrow{\tau}$ n'est pas rendue réflexive comme dans le cas de l'équivalence observationnelle.

Proposition 1.III.2.

La transformation *CO* préserve la congruence observationnelle.

Preuve.

Par la proposition 1.III.3 et par la définition de la transformation *CO*, on obtient facilement:

$S_{t1} \approx S_{t2}$ si et seulement si $CO(S_{t1}) \approx CO(S_{t2})$

c'est-à-dire, la transformation *CO* préserve l'équivalence observationnelle.

Soit $S_{ti} = (Q_{ti}, \cup_{\alpha \in P_\tau} R_{\alpha\, ti}, q_{io})$ pour i= 1, 2

$CO(S_{ti}) = (Q'_{ti}, \cup_{\alpha \in P_\tau} R'_{\alpha\, ti}, q'_{io})$ pour i= 1, 2

Si dans S_{ti} il y a une τ-transition $q_{io} \xrightarrow{\tau} q_{ti}$ qui fait partie d'un circuit ou qui est une chaîne de τ, c'est à dire qui est éliminée dans le pas 1 de la transformation *CO*, une transition $q'_{io} \xrightarrow{\tau} q'_{ti}$ est ajoutée au pas 2, où $q'_{ti} \in Q'$ est l'état initial du système après la transformation *EO*. Donc par construction de $CO(S_{ti})$ pour i=1,2 on sait que:

(1) $q_{io} \xrightarrow{\tau} q_{ti} \in R_{\tau\, ti}$ implique $\exists q'_{ti} \in Q'_{ti}\ q'_{io} \xrightarrow{\tau} q'_{ti} \in R'_{\tau\, ti}$ et $q'_{ti} \approx q_{ti}$

$q'_{io} \xrightarrow{\tau} q'_{ti} \in R'_{\tau\, ti}$ implique $\exists q_{ti} \in Q_{ti}\ q_{io} \xrightarrow{\tau} q_{ti} \in R_{\tau\, ti}$ et $q'_{ti} \approx q_{ti}$

Par (1) et la proposition 1.I.4. on obtient que $S_{ti} \approx^c CO(S_{ti})$,
donc $S_{t1} \approx^c S_{t2}$ si et seulement si $CO(S_{t1}) \approx^c CO(S_{t2})$.

♦

Pour décider de la congruence observationnelle, il ne suffit pas de tester si les systèmes $CO(S_{ti})$ obtenus à partir de S_{ti} se bisimulent; en fait, la relation de transitions doit parfois être réflexive. Par exemple, pour décider de la congruence observationnelle des termes b.a.nil et b.(τ.a.nil+a.nil) par bisimulation il faut considérer la relation $\stackrel{\tau}{\rightarrow}$ réflexive. Mais, il est facile de voir que si l'on considère la relation $\stackrel{\tau}{\rightarrow}$ réflexive on ne peut pas distinguer par bisimulation les termes a.nil et τ.a.nil; ainsi elle ne doit pas être considérée comme réflexive pour l'état initial. De la même manière si l'on suppose la relation $\stackrel{\tau}{\rightarrow}$ réflexive pour tous les états sauf pour l'état initial, il n'est pas possible de vérifier par bisimulation que les termes (a.b.x,x=a.b.x) et (x,x=a.b.x) sont observationnellement congrus.

Pour cette raison, nous introduisons la notion de *r-bisimulation* entre systèmes de transitions. Cette notion est une variante de la notion de bisimulation; elle se distingue uniquement par le fait que l'on rend la relation $\stackrel{\tau}{\rightarrow}$ réflexive d'une manière asymétrique:

- pour l'état initial q_0 on n'ajoute jamais une boucle $q_0 \stackrel{\tau}{\rightarrow} q_0$,
- à chaque étape, pour la vérification de la condition (i) de la bisimulation on considère comme réflexive seulement la relation $\stackrel{\tau}{\rightarrow}$ du deuxieme système de transitions sauf pour l'état initial et inversement pour la vérification de la condition (ii).

Définition 1.III.8.

Pour $S_i = (Q_i, \cup_{\alpha \in P\tau} R\alpha, q_{i0})$, i=1,2 on dit que S_1 et S_2 *r-bisimulent* (noté $S_1 \sim^r S_2$) si et seulement si le plus grand point fixe $R \subseteq Q_1 x Q_2$ de la fonctionnelle F_r contient la paire (q_{1o}, q_{2o}).

La fonctionnelle F_r est définie comme:

$F_r(R)=\{(p,q)|$ pRq et

et $\forall\alpha\neq\tau$ (si $p\xrightarrow{\alpha}p'$ alors $\exists q'\in Q_2(q\xrightarrow{\alpha} q'$et $p'Rq')$

et si $q\xrightarrow{\alpha}q'$ alors $\exists p'\in Q_1$ $(p\xrightarrow{\alpha}p'$ et $p'Rq'))$

et si $p\xrightarrow{\tau}p'$ alors $((p\neq q_{1o}$ et $(p',q)\in R)$ ou

$\exists q'\in Q_2(q=\tau\Rightarrow q'$et $p'Rq'))$

et si $q\xrightarrow{\tau}q'$ alors ($(q\neq q_{2o}$ et $(p,q')\in R)$ ou

$\exists p'\in Q_1(p=\tau\Rightarrow p'$ et $p'Rq')\}$.

Proposition 1.III.4.

Pour tous les termes t_1,t_2 de T_r

$t_1 \approx^c t_2$ si et seulement si $CO(S_{t1}) \sim^r CO(S_{t2})$.

Preuve.

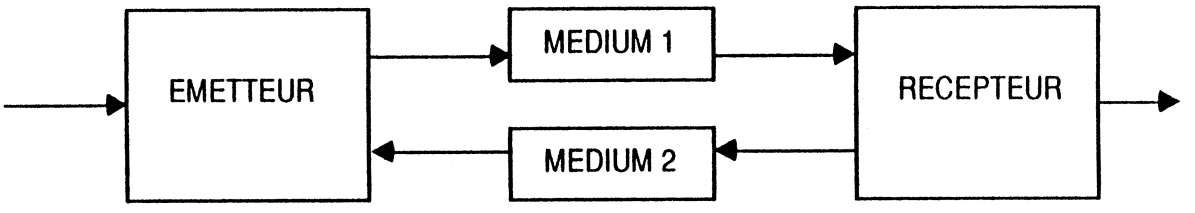
Conséquence directe de la proposition précédente, de la définition de r-bisimulation.

♦

III.3.- Exemple: Le protocole de communication du bit alterné.

Nous effectuons la comparaison de la description d'un protocole de communication défini entre deux localités qui utilisent un médium non fiable, avec sa spécification. Nous traitons une variante simplifiée du protocole de communication du bit alterné.

Le protocole de communication du bit alterné peut être schématisé par:



- L'émetteur est un processus qui reçoit un message. Il ajoute un bit de contrôle au message reçu et l'envoie au medium 1 jusqu'à ce qu'un acquittement soit reçu via le medium 2.

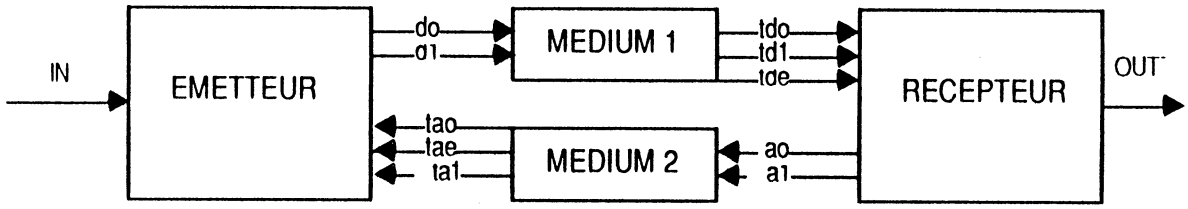
- Le medium 1 est un canal non fiable de transmission de messages. Il communique donc soit des messages corrects, soit une valeur erronée 'e'.

- Le recepteur est un processus qui reçoit les messages envoyés par le medium 1, les transmet vers l'extérieur et envoie un acquittement via le medium 2.

- Le medium 2 est un canal non fiable de transmission d'acquittements. Il transmet soit des acquittements corrects, soit une valeur erronée 'e'.

Nous donnons la structure du protocole et une description du comportement de chacune des composants. Seulement, dans cette version nous faisons abstraction des messages transmis en simplifiant le message à deux signaux indiquant la bonne transmission.

La structure du protocole est la suivante:



où: d_i sont les messages envoyés par l'émetteur (EM),

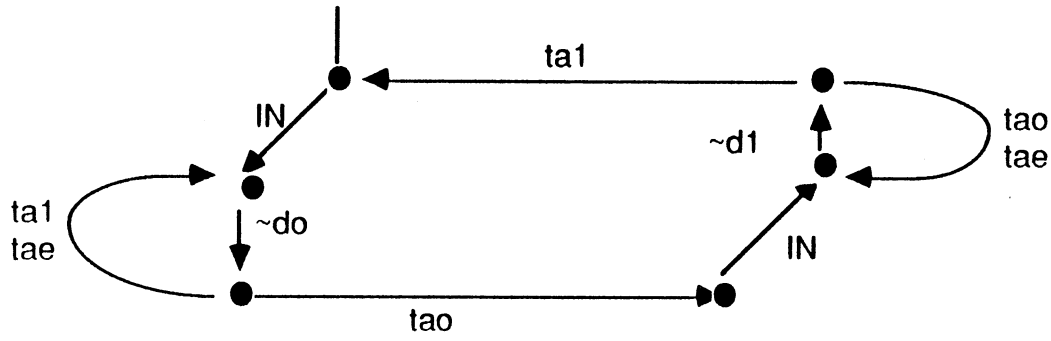
a_i les acquitements envoyés par R,

t_{di} et t_{ai} les signaux transmis correctement par les canaux M_1 et M_2,

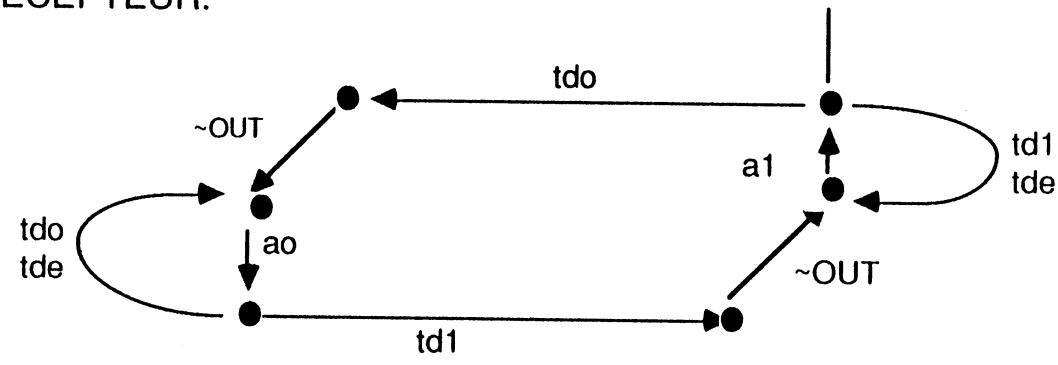
t_{de} et t_{ae} les messages d'erreur respectifs.

Nous décrivons le comportement des composants du protocole par les diagrammes de transitions suivantes:

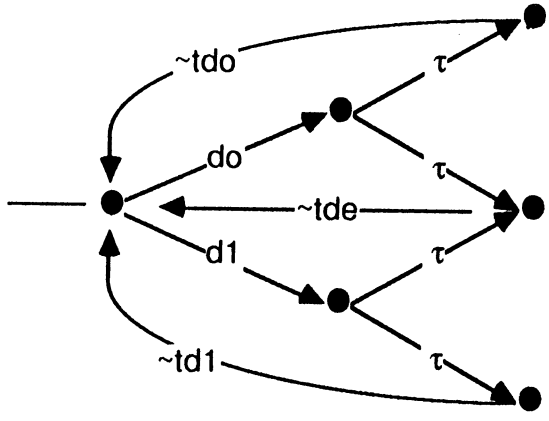
EMETTEUR:



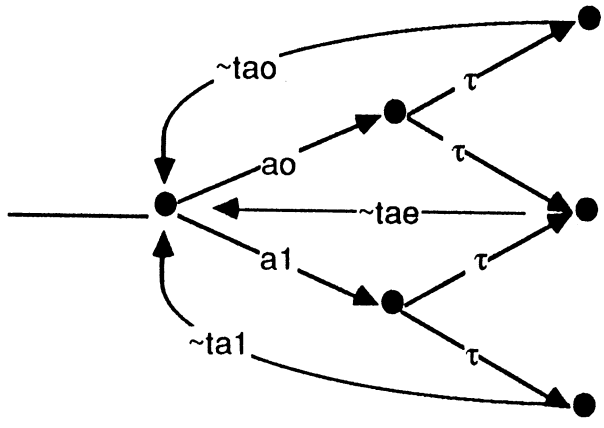
RECEPTEUR:



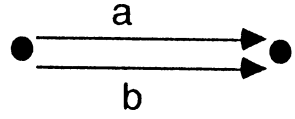
MEDIUM 1:



MEDIUM 2:



Notation: Par convention, une flèche avec des étiquettes multiples <u>a b</u>> est utilisée à la place de:



Le terme CCS obtenu à partir des descriptions précédentes est le suivant:

$$PROT = ((EM \,||\, R \,||\, M1 \,||\, M2) \backslash d_0 \backslash d_1 \backslash td_0 \backslash td_1 \backslash td_e \backslash a_0 \backslash a_1 \backslash ta_0 \backslash ta_1 \backslash ta_e \,,\, E)$$

où:

$$E = \{\, EM = in.E_1,$$

$$E_1 = \sim d_0.(ta_1.E_1 + ta_e.E_1 + ta_0.in.E_2)$$

$$E_2 = \sim d_1.(ta_0.E_2 + ta_e.E_2 + ta_1.EM)$$

$$R = td_0.\sim out.R_1$$

$$R_1 = a_0.(td_0.R_1 + td_e.R_1 + td_1.\sim out.R_2)$$

$$R_2 = a_1.(td_1.2_1 + td_e.R_2 + R)$$

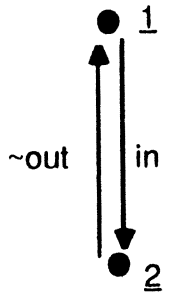
$$M_1 = d_0(\tau.\sim td_0.M_1 + \tau.\sim td_e.M_1) + d_1(\tau.\sim td_1.M_1 + \tau.\sim td_e.M_1)$$

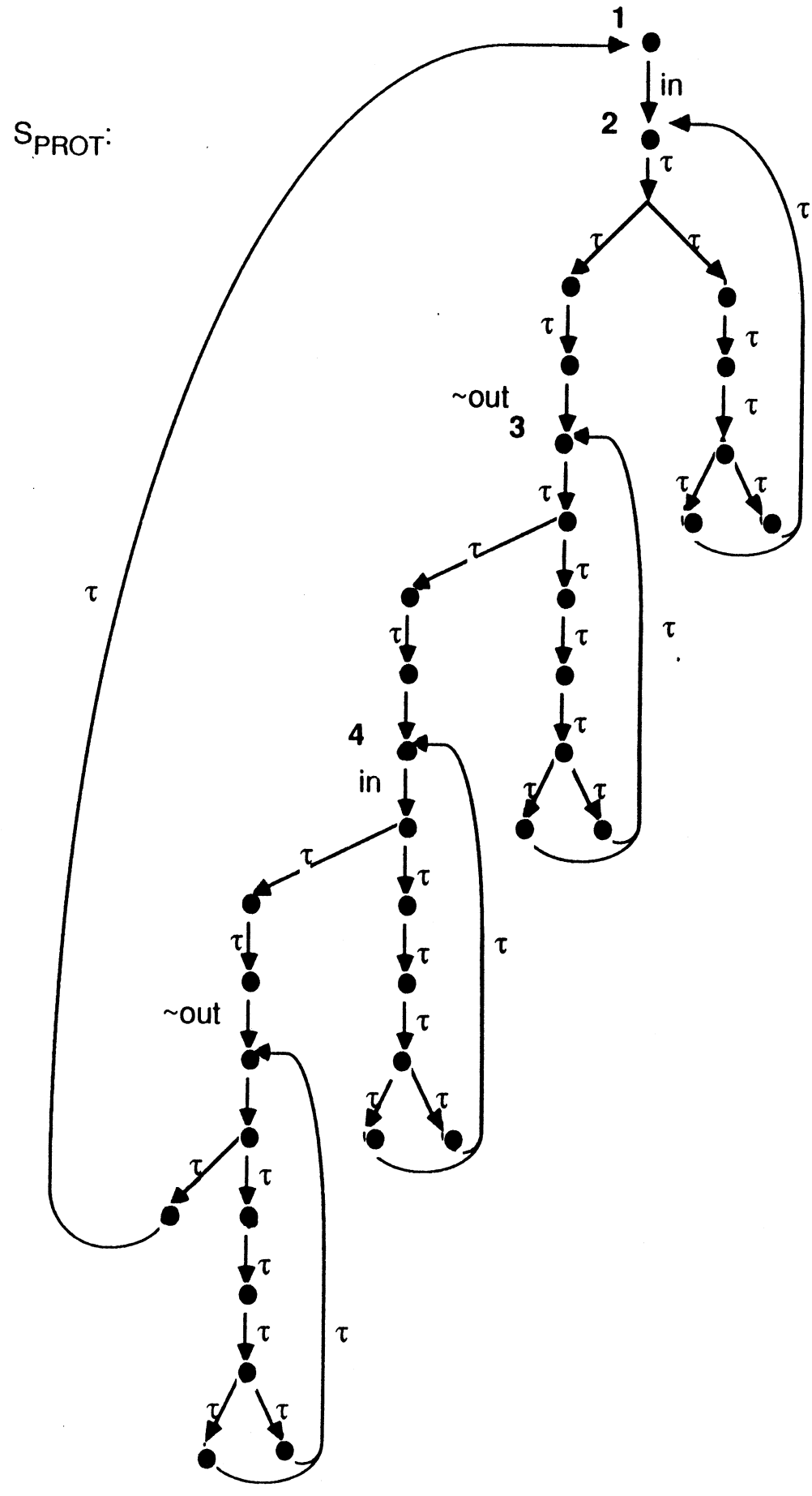
$$M_2 = a_0(\tau.\sim ta_0.2_1 + \tau.\sim ta_e.M_2) + a_1(\tau.\sim ta_1.2_1 + \tau.\sim ta_e.M_2)\,\}.$$

Nous vérifions PROT est observationnellement congru au terme SPEC=(SP,SP=in.~out.SP) qui est la spécification du comportement désiré, en appliquant la méthode décrite précédemment:

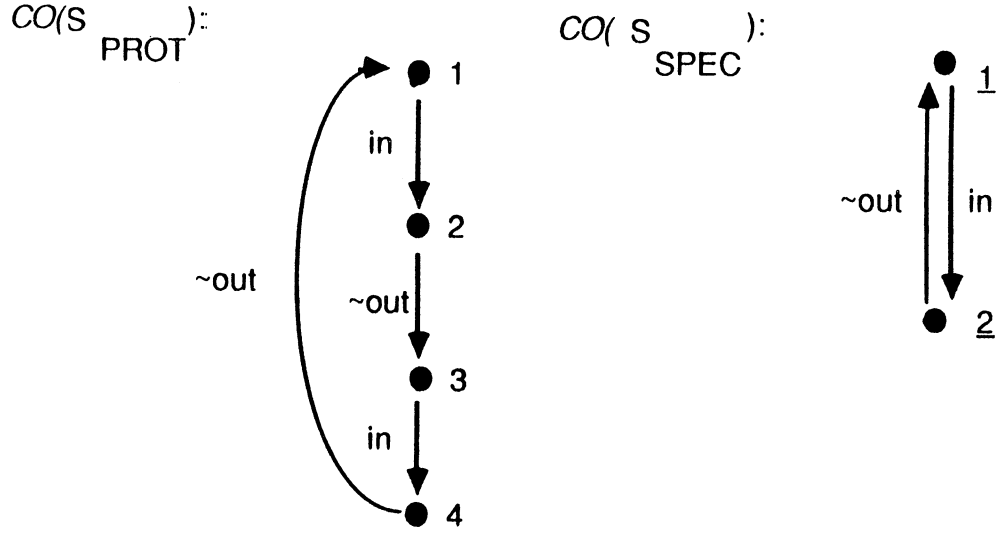
- Construction des systèmes de transitions associés aux termes.

S_{SPEC} :




S_{PROT}:
1
in
2
τ
~out
3
4
in
~out

- Par application de la transformation *CO* aux systèmes de transitions S_{SPEC} et S_{PROT}, on obtient :



- Construction d'une relation de bisimulation entre les systèmes de transitions obtenus.

  R = {(1,1) , (2,2) , (3,1) , (4,2)}

  Puisque les états initiaux des systèmes se bisimulent, nous concluons que les termes CCS donnés sont observationnellement congrus.

III.4.- Algorithme de vérification.

L'algorithme qui permet de décider chacune des équivalences choisies procède en trois étapes:

1.- Construction des systèmes de transitions associés aux termes.
2.- Application de la transformation ad-hoc pour la vérification.
3.- Essai de construction d'une bisimulation entre les systèmes transformés.

III.4.1.- Algorithme de construction des systèmes de transitions.

L'algorithme de construction du système de transitions associé à un terme régulier procède par induction sur la structure du terme, en utilisant la relation de transitions définie dans le paragraphe 1.1.2. et les priorités et les règles d'associativité des opérateurs.

Dans le cas où le terme contient l'opérateur de composition parallèle et celui d'effacement, i.e: $(t_1 || t_2 || \ldots || t_n) \backslash \alpha$, nous construisons uniquement les transitions de $t_1 || t_2 || \ldots || t_n$ qui restent après l'application de l'opération d'effacement. Les propriétés des opérateurs $||$ et $\backslash\alpha$ permettent de les considérer simultanément lors de la construction du système de transitions. En effet, on a la transition:

$$(t_1 || t_2 || \ldots || t_n) \backslash \alpha_1 \backslash \alpha_2 \backslash \ldots \backslash \alpha_n \xrightarrow{\beta} (t'_1 || t'_2 || \ldots || t'_n) \backslash \alpha_1 \backslash \alpha_2 \backslash \ldots \backslash \alpha_n$$

seulement dans le deux cas suivants:

1.- Action observable $(\beta, \sim\beta \notin \{\alpha_1, \alpha_2, \ldots, \alpha_n\})$:

$\exists j \ (1 \leq j \leq n \text{ et } t_j \xrightarrow{\beta} t'_j)$ et $\forall k \ 1 \leq k \leq n, k \neq j \ (t'_k = t_k))$,

2.- Action non observable $(\beta = \tau)$:

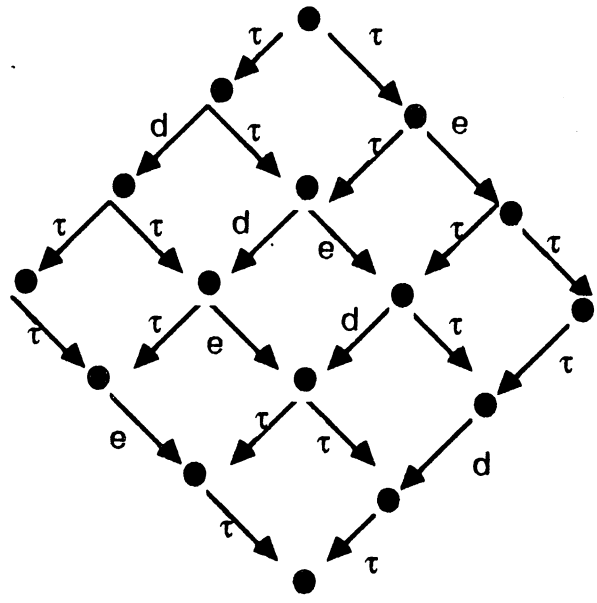
$\exists \delta \in P \ \exists i,j \ 1 \leq i \leq n$ et $1 \leq j \leq n \ \exists t'_i, t'_j \ (t_i \xrightarrow{\delta} t'_i$ et $t_j \xrightarrow{\sim\delta} t'_j)$

et $\forall k \ 1 \leq k \leq n, k \neq j, k \neq i \ (t'_k = t_k)$.

Par exemple, pour le terme:

(a.d.b.nil||f.e.c.nil||~a.~b.nil||~f.~c.nil)\a\b\c\f

nous construisons directement le système de transitions:



Il est important de noter que si nous n'utilisons pas cette optimisation, il faut construire d'abord le système de transitions associé au terme a.d.b.nil||f.e.c.nil||~a.~b.nil||~f.~c.nil. Ce système de transitions a 144 états et 456 transitions; ensuite on élimine 128 états et 432 transitions par application de l'opération d'effacement.

III.4.2.- Algorithmes d'application des transformations.

III.4.2.1.- Algorithme d'application de la transformation *EO*

L'algorithme d'application de la transformation *EO*, sur un système de transitions procède en quatre étapes:

1.- Appliquer la transformation *τ-chaînes* .

2.- Appliquer la transformation *τ-circuits*. Nous utilisons l'algorithme proposé par Tarjan [Se 83] pour trouver les composantes fortement connexes des graphes associés aux τ transitions, puisque la complexité de cet algorithme est linéaire; en effet sa complexité est O(n+m) où n est le nombre d'états et m est le nombre de transitions.

3.- Application de la transformation *O*:

- Construire des nouvelles relations $\xrightarrow{\alpha}$ pour $\alpha \in P_\tau$, par calcul de la fermeture transitive de la relation $\xrightarrow{\tau}$. Nous utilisons une adaptation de l'algorithme de Warshall [Se 83]:

  pour toute transition $x \xrightarrow{\alpha} y$ **faire**

  si $\alpha=\tau$ **alors pour** toute transition $y \xrightarrow{\beta} z$ **faire** $x \xrightarrow{\beta} z$

  sinon pour toute transition $y \xrightarrow{\tau} z$ **faire** $y \xrightarrow{\alpha} z$

  Cet algorithme a besoin d'un acces direct aux transitions, il construit la fermeture transitive de la relation $\xrightarrow{\tau}$, en parcourant une seule fois toutes les transitions de l'automate.
- Rendre réflexive la relation $\xrightarrow{\tau}$.

III.4.2.2.- Algorithme d'application de la transformation *CO*.

L'algorithme d'application de la transformation *CO* , sur un système de transitions procède en quatre étapes:

1.- Appliquer la transformation *τ-chaînes* et *τ-circuits*.

2.- Si pendant l'application des transformations τ-chaînes et τ-circuits nous avons éliminé une τ-transition partant de l'état initial, nous ajoutons un état "0" représentant un sous-terme n'appartenant pas à l'ensemble d'états du système, et une τ-transition de l'état 0 à l'état initial du système obtenu précédemment. Ainsi l'état 0 est le nouveau état initial du système de transitions.

3.- Construire des nouvelles relations $\xrightarrow{\alpha}$ pour $\alpha \in P_\tau$, par calcul de la fermeture transitive de la relation $\xrightarrow{\tau}$.

III.4.3.- Algorithme de construction d'une bisimulation.

L'algorithme que nous utilisons pour construire une bisimulation entre deux systèmes de transitions calcule itérativement le plus grand point fixe de la fonctionnelle F définie dans le chapitre I.

L'algorithme s'arrête:

- soit au moment où on a obtenu une relation R où les états initiaux des systèmes ne sont plus en relation. Dans ce cas les systèmes ne sont pas équivalents.
- soit à la stabilisation de la relation, en concluant que les systèmes sont équivalents.

Nous avons préféré cet algorithme à celui basé sur la notion de retour arrière [Sa 82] parce qu'il nous permet de conserver moins d'information sur le déroulement de la construction de la relation de bisimulation et d'utiliser moins de mémoire dans des cas critiques.

L'algorithme pour décider de la congruence observationnelle est le même, sauf que l'on utilise la fonctionnelle F_r définie précédemment à la place de F.

Cet algorithme est aussi utilisé pour construire la relation de bisimulation nécessaire pour la construction des formes normales des termes réguliers décrites dans le chapitre IV.

IV. Formes normales des termes réguliers.

Dans ce chapitre, nous construisons les formes normales des termes réguliers pour la congruence forte, et pour l'équivalence et pour la congruence observationnelle. Ces formes normales sont les termes dont les systèmes de transitions associés ont un nombre minimal d'états et de transitions. Dans le cas de termes finis, ces formes normales pour l'équivalence et pour la congruence observationnelle coïncident avec les formes normales présentées aux paragraphes II.1 et II.2.

Pour la construction des formes normales, nous utilisons les transformations *τ-chaînes et τ-circuits* définies au paragraphe 1.III.2 et la notion de *bisimulation.*

Pour construire chacune des formes normales nous procédons en trois étapes:

1.- Construction du système de transitions associé au terme.
2.- Application de transformations ad-hoc à ce système de transitions (réduction).
3.- Construction du terme de T_r associé au système de transitions transformé.

Par la suite, on décrit les étapes 2 et 3 des algorithmes de construction des formes normales. Ces étapes son presentées respectivement dans IV.1 et IV.2.

IV.1. Transformations sur les systèmes de transitions.

Dans cette partie, nous définissons les transformations *NCF, NEO* et *NCO.* Nous montrons que ces transformations permettent de construire les formes normales des systèmes de transitions pour chacune des équivalences.

IV.1.1.- Transformation *NCF*.

Définition 1.IV.1.

Pour tout système de transitions $S=(Q, \cup_{\alpha \in P\tau} R_\alpha, q_0)$, *NCF*(S) est le système de transitions quotient par rapport à la congruence forte (les classes d'équivalence sont obtenues en calculant la plus grande bisimulation contenue dans QxQ).

Proposition 1.IV.1.

Pour tout système de transitions S, *NCF* (S) est une forme normale pour la congruence forte.

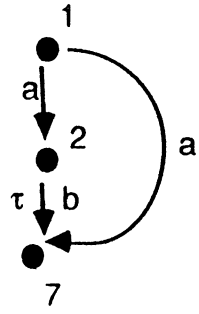
Preuve.

L'automate quotient a un nombre minimal d'états puisqu'il ne contient pas deux états équivalents (Théorème 3.6 [Gi 62]).

♦

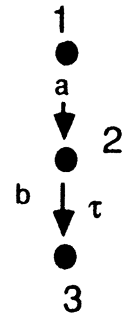
IV.1.2.- Transformations *NEO* et *NCO*.

Dans le cas de l'équivalence et de la congruence observationnelle, le système de transitions quotient a un nombre minimal d'états, mais il n'a pas toujours un nombre minimal de transitions. Par exemple, le système de transitions réduit:



a un nombre minimal d'états, mais il n'est pas minimal par rapport au nombre de transitions. En fait, ce système est observationnellement

équivalent au système:



Ce système est la forme normale (système bien gardé contenant un minimum d'états et de transitions).

Définition 1.IV.2.

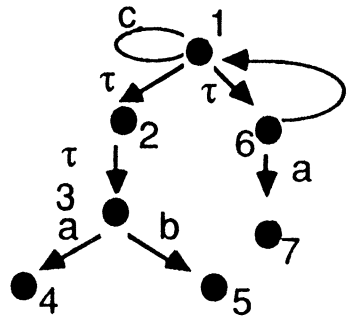
Pour tout système de transitions S on obtient le système de transitions *NEO*(S) par l'application de la suite de transformations:

1.- Transformations τ-chaînes et τ-circuits.

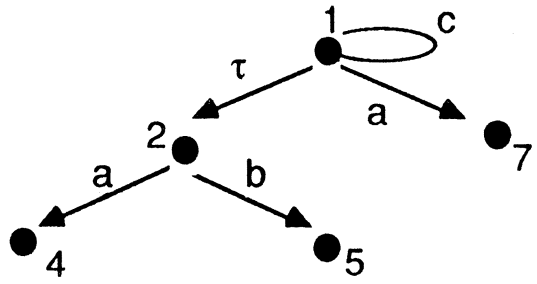
2.- Réduction par rapport à l'équivalence observationnelle.

3.- Elimination de toute transition $p \xrightarrow{\alpha} q$ dans le cas où $\exists s \in \{\tau^*\alpha\tau^+, \tau^+\alpha\tau^*\}$ tel que $p \xrightarrow{s} q$.

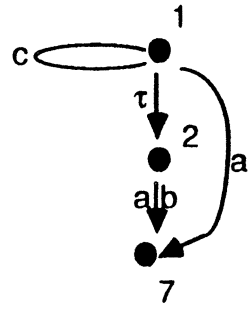
Par exemple, soit le système de transitions:



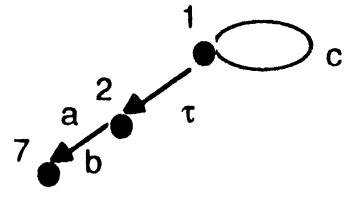
- Par application des transformations τ-chaînes et τ-circuits, on obtient le système de transitions:



- Le système de transitions quotient par rapport à l'équivalence observationnelle est le suivant:



- Par élimination de transitions superflues (la transition $1 \xrightarrow{a} 3$ est éliminée puisque $1 \xrightarrow{\tau a} 3$) on obtient le système *NEO* (S):



Proposition 1.IV.2.

Pour tout système de transitions S, *NEO*(S) est une forme normale pour l'équivalence observationnelle.

Preuve.

1) La transformation *NEO* préserve l'équivalence observationnelle. En fait, il a été démontré que les pas 1 et 2 préservent l'équivalence observationnelle. Par définition de l'équivalence observationnelle il est évident que le pas 3 appliqué sur

un système sans circuits de τ préserve l'équivalence observationnelle puisque nous éliminons seulement les transitions $p \xrightarrow{\alpha} q$ pour les quelles la condition ($\exists s \in \{\tau^*\alpha\tau^+, \tau^+\alpha\tau^*\}$ tel que $p \xrightarrow{s} q$) est vérifiée.

2) Comme pour la transformation *NCF*, le nombre d'états de *NEO*(S) est minimal. Par construction, si $p \xrightarrow{\alpha} q$ est dans *NEO*(S) alors $\nexists s \in \{\tau^*\alpha\tau^+, \tau^+\alpha\tau^*\}$ tel que $p \xrightarrow{s} q$. Donc le système de transitions *NEO*(S) est minimal pour l'équivalence observationnelle.

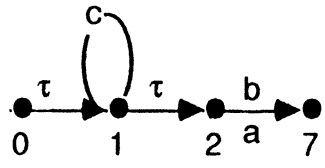
♦

Définition 1.IV.3.

Pour tout système de transitions S, *NCO* (S) est le système de transitions obtenu de la manière suivante:

1.- Construction du système $S'=(Q, \cup_{\alpha \in P\tau} R_\alpha, q_0)$ par application de la transformation *NEO* sur S.
2.- Si pendant la construction du système *NEO* (S) nous avons éliminé une τ-transition partant de l'état initial, nous ajoutons un nouvel état initial q'_0, ainsi que la τ-transition $q \xrightarrow{\tau} q'_0$.

Par exemple, pour le système de transitions S de l'exemple précédent, le système de transitions *NCO* (S) est le suivant:



Proposition 1.IV.3.

Pour tout système de transitions S, *NCO* (S) est une forme normale pour la congruence observationnelle.

Preuve: Analogue à celle de la proposition 1.IV.2.

♦

IV.2.- Construction du terme associé à un système de transitions.

Définition 1.IV.4.

Le terme régulier associé à un système de transitions $S=(Q, \cup_{\alpha\in P_\tau} R_\alpha, q_0)$ est $u_S=(C(q_0),E)$ où:

$E=\{x_p=D(p)\}_{p\in Q'}$, avec $Q'\subseteq Q$, $D: Q \rightarrow T[\Sigma_1,V]$, $C: Q \rightarrow T[\Sigma_1,V]$ définis par,

$q_0\in Q'$ ssi $\exists q\in Q, \exists\alpha\in P_\tau$ tels que: $q \xrightarrow{\alpha} q_0$,

$q\in Q'$, $q\neq q_0$ ssi $(\exists q_1,q_2,q_3\in Q, \exists\alpha,\beta,\gamma\in P_\tau$ tels que: $q_1 \xrightarrow{\alpha} q$, $q_2 \xrightarrow{\beta} q$ et $q \xrightarrow{\gamma} q_3)$

$D(p)=nil$, si $\nexists q\in Q, \nexists\alpha\in P_\tau$ tels que $p \xrightarrow{\alpha} q$

$D(p)=\Sigma_{i\in I}\ \alpha_i.C(q_i)$, ssi $p \xrightarrow{\alpha_i} q_i$ pour $i\in I$ et pour tout j tel que $p \xrightarrow{\alpha_j} q_j$, $j\in I$,

$C(q)=x_q$, si x_q est une variable de E.

$C(q)=D(q)$, sinon.

Proposition 1.IV.4.

Si S est un système de transitions sous forme normale pour la congruence forte ou l'équivalence ou la congruence observationnelle alors le terme u=(t,E) associé à S défini précédemment est lui aussi sous forme normale pour la congruence forte ou l'équivalence ou la congruence observationnelle.

V.- Vérification des termes de $T[\Sigma_1,V]$.

Pour les termes de $T[\Sigma_1,V]$, il est possible de construire des formes normales et de les comparer par rapport à la congruence et à l'équivalence observationnelle. La construction des formes normales est faite par extension des résultats présentés pour les termes finis dans la chapitre I.

Pour comparer des termes de $T[\Sigma_1,V]$, on suppose que V définit un ensemble de constantes de comportements différentes, c'est-à-dire pour x, et y appartenant à V, x est équivalent à y si et seulement si x = y.

Par ailleurs, il est nécessaire d'étendre la notion de comportement dérivé aux termes de $T[\Sigma_1,V]$.

Définition 1.V.1.

(i) $\forall x \in V$, x est un comportement dérivé de x.
(ii) nil est un comportement dérivé de nil ,
(iii) $\alpha.t$ est un comportement dérivé de $\alpha.t'$ ssi
soit $t \approx^S t'$
soit t' de la forme $\Sigma\alpha_i.t_i+\tau.t''$ et $\alpha.t$ un comportement dérivé de $\alpha.t''$,
(iv) t est un comportement dérivé de $\tau.t'$ ssi
soit $t \approx^S t'$
soit t est un comportement dérivé de t' ,
(v) $\alpha.t$ est un comportement dérivé de t' + t'' ssi $\alpha.t$ est un comportement dérivé de t' ou de t'' ,
(vi) si t, t' sont deux comportements dérivés de t'' alors t+t' est un comportement dérivé de t'' ,
(vii) tout comportement dérivé d'un terme de $T[\Sigma_1,V]$ peut être obtenu a partir de (i)-(vi).

L'existence et l'unicité des formes normales des termes de $T[\Sigma_1,V]$ pour la congruence et pour l'équivalence observationnelle, est une conséquence directe des résultats montrés pour les termes finis.

Par la suite nous caractérisons la forme normale pour la congruence et pour l'équivalence observationnelle.

V.1.- Forme normale pour la congruence observationnelle.

Définition 1.V.2.

(i) $\forall x \in V$, x est une expression sous forme normale,
(ii) nil est une expression sous forme normale,
(iii) $\forall \alpha \in P \cup \{\tau\}$, $\alpha.t$ est sous forme normale ssi t est sous forme normale et t n'est pas de la forme $\tau.t'$,
(iv) t_1+t_2 est en la forme normale ssi t_1 et t_2 sont deux formes normales et il n'existe aucun comportement dérivé de t_1 qui est aussi un comportement dérivé de t_2 et vice-versa,
(v) toute expression sous forme normale est obtenue à partir de (i) à (iv).

Par exemple les expressions a.x, y, τ.y+b.z sont sous forme normale; par contre les expressions x+τ.x et a.x+τ.(τ.a.x+b.nil) ne sont pas sous forme normale.

V.2.- Forme normale pour l'équivalence observationnelle.

Définition 1.V.3.

(i) $\forall x \in V$, x est une expression sous forme normale,
(ii) nil est une expression sous forme normale,
(iii) $\forall \alpha \in P$, $\alpha.t$ est sous forme normale ssi t est sous forme normale et t n'est pas de la forme $\tau.t'$,
(iv) t_1+t_2 est sous forme normale ssi t_1 et t_2 sont deux formes normales et il n'existe aucun α-dérivé de t_1 qui est aussi un α-dérivé de t_2 et vice-versa,
(v) toute expression sous forme normale est obtenue à partir de (i) à (iv).

En fait, ces définitions sont analogues aux définitions données pour des termes finis (chapitre II) sauf que nous ajoutons à chaque fois une clause pour tenir en compte des termes avec des variables libres. Les algorithmes pour calculer ces formes normales sont analogues aux algorithmes de calcul de la forme normale pour des termes finis.

PARTIE II: Le système VENUS.

Un des problèmes de la vérification est la grande quantité de calcul exigée pour le traitement des systèmes ayant un certain niveau de complexité. Dans la plupart des cas, ce problème rend la vérification manuelle difficile sinon impossible, d'où la nécessité d'automatisation de cette tâche.

Dans ce sens, plusieurs systèmes informatiques ont été réalisés, comme par exemple les systèmes CESAR [Qu 82] [QS 82] [Sc 83], OGIVE/OVIDE [Mo 82], SCAN [Na 86] et ECRINS [MV 86] [Ve 86].

En CESAR la spécification d'un programme est exprimée par un ensemble de formules d'une logique et ce programme vérifie ces spécifications s'il satisfait chacune de ses formules.

Le système OGIVE/OVIDE sert à la définition et à la vérification des réseaux de Petri. Dans OGIVE/OVIDE l'analyse des réseaux est faite par vérification de propriétés de base (propriétés telles que la "vivacité" et la propriété de "borné"), détection de cycles et réduction des réseaux en préservant les propriétés de base.

Le système SCAN permet de comparer des systèmes communicants décrits par des systèmes de transitions en utilisant une notion d'équivalence observationnelle.

Le système ECRINS est conçu pour la manipulation de termes dans des calculs de processus communicants, la comparaison des termes est faite par une notion d'équivalence observationnelle.

Nous proposons le système VENUS qui est un outil d'aide à la vérification de systèmes communicants. Il permet de décrire, transformer et comparer des termes modulo differentes relations d'équivalence.

La présentation de VENUS est faite en cinq chapitres. Le premier chapitre contient une déscription générale du système. Le deuxième chapitre est le manuel d'utilisation. Dans le troisième chapitre nous donnons la description et la vérification d'un protocole de transport défini par l'ISO OSI [Ta 81] [Na 87] à l'aide de VENUS. Le quatrième chapitre donne une description de la réalisation du système. Dans le cinquième chapitre nous proposons une évaluation du système.

I.- Description générale de VENUS.

VENUS est un outil interactif d'aide à la conception et à la vérification de processus communicants. Il a été conçu sur la base des résultats présentés dans la partie I de cette thèse, et il a les caractéristiques suivantes:

- *Modularité des descriptions:* Possibilité de description de systèmes par composition de modules définis séparément.

- *Facilité d'utilisation des possibilités offertes par le calcul:* Vérification, simplification et en général transformation et comparaison des définitions de systèmes.

- *Facilités de modification et de mémorisation des comportements:* Possibilité d'archiver les descriptions des systèmes en vue de leur réutilisation ultérieure, ainsi que des résultats obtenus par transformation des définitions. Possibilité de modifier la définition d'un système en gardant toute l'information que n'est pas affectée par la modification.

- *Construction des systèmes de transitions et des termes:* Possibilité de transformer des termes du calcul en systèmes de transitions et vice-versa ainsi que d'appliquer des transformations sur ces systèmes de transitions.

Pour définir, vérifier et archiver les définitions des systèmes communicants, VENUS utilise des *environnements de travail.*

Par la suite, nous présentons le langage de description des environnements de travail, les facilités de traitement et gestion des environnements ainsi que les facilités de vérification et de normalisation des systèmes définis dans les environnements de travail.

I.1.- Le langage de description des environnements de travail.

Le langage de description des environnements de travail permet de définir les systèmes communicants d'une manière modulaire par un ensemble de *composantes*; la structure de l'application est reflétée directement par la structure de sa description.

Les spécifications des systèmes sont écrites dans le même langage. La vérification d'un système est effectuée par comparaison des termes (description et spécification), en tenant compte d'une notion d'équivalence.

Dans ce langage, deux types de variables sont proposées: le type action et le type comportement.

Les comportements sont décrits par des variables de comportement déclarées dans une composante et définies dans cette composante par une équation lui associant un terme de $T[\Sigma_2,V]$. Nous distingons entre une variable u désignant un terme $u=(t,E)$ de T_r, et les variables de var(t) définies dans E.

Nous présentons par la suite la syntaxe du langage de description des environnements de travail. Nous définissons la portée des identificateurs et les vérifications contextuelles faites sur les descriptions des environnements. Nous donnons une description en VENUS, du protocole de communication du bit alterné décrit dans la partie I.

I.1.1.- Syntaxe du langage de description.

Le langage de description des environnements de travail est défini par la grammaire ci-dessous, dont l'axiome est le non terminal <env>. Les notations utilisées sont:

<xxx>	symbole non terminal;
xxx	mot clé;
[xxx]	occurrence éventuelle de xxx;
[xxx]*	occurrence éventuelle de xxx, un nombre quelconque de fois (0,1,...);
[xxx]+	occurrence éventuelle de xxx, un nombre positif de fois (au moins une fois);

Comme d'habitude, le caractère "|" est utilisé pour délimiter les alternatives des parties droites des règles. Les symboles terminaux [et] de la grammaire sont écrits respectivement "[" et "]" Les symboles terminaux sont écrits en gras.

<env> ::[**composante** <idf>
<type> = <liste-idf> [; <type>=<liste-idf>]*
[**definitions**
<idf> = <part-d-def> [; <idf>=<part-d-def>]*
fin-definitions]
[**equations**
<idf> = <part-d-eq> [; <idf>=<part-d-eq>]*
fin-equations]
fin-composant]*

<type> ::= **comportement** | **action**

<liste-idf> ::= <idf >[, <idf>]*

<part-d-def> ::= <s-terme> [<redef-eff>]* [<op> <s-terme> [<redef-eff>]*]

<s-terme> ::= <idf> [<s-terme>] | (<part-d-def>) | **in** <idf> | **nil**

<redef-eff> ::= \ "[" <liste_idf> "] " | # "["<liste-idf> "]" / "[" <liste-idf>"]"

<op> ::= + | &

<part-d-eq> ::= <sous-terme-add> [+ <sous-terme-add>]

<sous-terme-add> ::= <idf> [<sous-terme-add>] | (<partie-d-eq>) | nil

Remarques: - les opérateurs + et & correspondent aux opérateurs + et || de CCS.
- les opérateurs $\backslash \delta_1 \backslash \ldots \backslash \delta_m$ et [f] de CCS, sont réalisés dans VENUS par $\backslash [\delta_1,\ldots,\delta_m]$ et $\#[\alpha_1,\ldots,\alpha_n] \backslash [\beta_1,\ldots,\beta_n]$, où $\delta_1,\ldots,\delta_m$, $\alpha_1,\ldots,\alpha_n$ et $\beta_1,\ldots,\beta_n$ sont des actions telles que $\beta_i=f(\alpha_i)$ pour i=1,...n.
- $n\alpha$ est l'action complementaire de l'action α.
- l'action *t* symbolise l'action τ.

I.1.2.- Portée des identificateurs.

La portée des identificateurs dans la définition d'un environnement de travail est donnée par l'ensemble suivant de règles:

1.- La portée d'une **composante** est l'environnement de travail dans lequel elle est définie.

2.- La portée d'une équation de la partie **definition** est la partie **definition** de la composante dans laquelle elle est déclarée et les parties **definition** de toutes les composantes de l'environnement.

3.- La portée d'une équation de la partie **equation** est la

composante dans laquelle elle est déclarée et la partie **definition** de toute **composante** de l'environnement.

4.- La portée d'une **action** est la **composante** dans laquelle elle est déclarée.

I.1.3.- Vérifications contextuelles.

Les vérifications contextuelles réalisées sur les définitions des environnements de travail sont données par l'ensemble suivant de règles:

1.- A propos des déclarations de variables:

Tout identificateur de comportement <idf> utilisé dans la partie gauche d'une équation est:
- soit déclaré dans la **composante**,
- soit declaré dans une autre **composante** et dans ce cas il est suivi de: **in** <idf de la composante>,
- soit **nil**.

Toute action utilisée dans une **composante** est déclarée dans la même **composante**.

2.- A propos des appels récursifs:

Dans les équations de la partie **definition** d'une composante, il n'y a pas d'appels récursifs à des variables de comportement définies dans une partie **definition**.

Dans les équations de la partie **equation**, il n'y a pas d'appels récursifs non gardés.

I.1.4.- Exemple.

Nous présentons ici une description en VENUS du protocole du bit alterné décrit dans la partie I paragraphe III.3. Elle comporte cinq composantes:

- La composante *bit-alterné* contient la description globale du protocole de communication du bit alterné et de sa spécification.
- Les composantes *émetteur* et *récepteur* décrivent les comportements des localités émetteur et récepteur.
- Les composantes *médium1* et *médium2*: décrivent le comportement des mediums non fiables utilisés pour la transmission de messages et d'acquittements entre l'émetteur et le récepteur.

composante bit-alterné
comportement = ba, s;
action = d_0, d_1, td_0, td_1, td_e, a_0, a_1, ta_0, ta_1, ta_e, input, output
definitions
ba = (em **in** émetteur **&** r **in** récepteur **&** m_1 **in** médium1 **&**
m_2 **in** médium2) \[d_0,d_1,td_0,td_1,td_e,a_0,a_1,ta_0,ta_1,ta_e]
fin-definitions
equations
s = in nout s
fin-equations
fin-composante

composante émetteur
comportement = em, e_1, e_2;
actions = d_0, d_1, ta_0, ta_1, ta_e, input
equations
em= input e_1
e_1 = nd_0 (ta_1 e_1 + ta_e e_1 + ta_0 input e_2)
e_2 = nd_1 (ta_0 e_2 + ta_e e_2 + ta_1 em)
fin-equations
fin composante

composante récepteur
comportement = r, r_1, r_2;
actions = $a_0, a_1, td_0, td_1, td_e$, output
equations
$r = td_0$ nout put r_1
$r_1 = a_0 (td_0\ r_1 + td_e\ r_1 + td_1$ noutput $r_2)$
$r_2 = a_1 (td_1\ r_1 + td_e\ r_2 + r)$
fin-equations
fin-composante

composante médium1
comportement = m_1;
actions = $d_0, d_1, td_0, td_1, td_e$, t
equations
$m_1 = d_0 (t\ ntd_0\ m_1 + t\ ntd_e\ m_1) +$
$d_1 (t\ ntd_1\ m_1 + t\ ntd_e\ m_1)$
fin-equations
fin-composante

composante médium2
comportement = m_2;
actions = $a_0, a_1, ta_0, ta_1, ta_e$,t
equations
$m_2 = a_0 (t\ nta_0\ m_2 + t\ nta_e\ m_2) +$
$a_1 (t\ nta_1\ m_2 + t\ nta_e\ m_2)$
fin-equations
fin-composante

I.2.- Gestion des environnements.

VENUS permet à l'utilisateur de définir des environnements de travail ainsi que de les modifier et de les détruire.

Un environnement temporaire de travail appelé *courant* est toujours présent dans le système. Au debut d'une session, cet environnement est vide. C'est dans cet environnement courant que l'utilisateur définit ses applications et réalise les transformations et vérifications des comportements définis.

Création des environnements de travail.

L'utilisateur peut définir l'environnement courant de travail à l'aide de l'éditeur de texte de sa préférence en donnant la définition de l'application écrite dans le langage présenté précédemment. Il est aussi possible de donner une valeur initiale à l'environnement courant en l'initialisant avec un environnement sauvegardé précédemment.

Dans VENUS l'utilisateur peut créér d'autres environnements de travail par sauvegarde de l'environnement courant.

Modification d'un environnement de travail

Le seul environnement de travail susceptible de modification est l'environnement courant. Les modifications sont faites:
- soit par modification de la définition de l'environnement,
- soit par transformation d'un autre comportement.

La modification de la définition de l'environnement courant est faite à l'aide d'un éditeur de texte choisi par l'utilisateur parmi ceux disponibles sous le système. Il est important de signaler qu'une modification de la définition de l'environnement courant produit la destruction dans l'environnement courant de tout comportement qui a été construit à partir des comportements dont les définitions ont été modifiées.

Destruction d'un environnement de travail.

L'utilisateur de VENUS peut détruire des environnements de travail créés précédemment.

La destruction de l'environnement courant efface toutes les informations contenues dans cet environnement.

La destruction d'un environnement de travail créé par sauvegarde de l'environnement courant produit la destruction de l'information contenue dans l'environnement et l'élimination de cet environnement.

VENUS offre aussi un certain nombre de primitives permettant de connaître les noms et les définitions des environnements existants et les noms et les définitions des comportements dans l'environnement courant.

I.3.- Fonctions permettant la vérification et la normalisation des systèmes.

VENUS offre des fonctions pour la vérification et pour la normalisation des systèmes. L'utilisateur peut choisir de réaliser les vérifications et les normalisations soit d'une manière automatique, soit d'une manière controlée.

Pour normaliser et vérifier une description de manière automatique, l'utilisateur doit préciser une relation d'équivalence. Le système lui propose le choix parmi les relations suivantes: congruence forte, equivalence et congruence observationnelle.

Pour controler les vérifications et les normalisations, le système offre à l'utilisateur des transformations des termes en automates et vice-versa, et un ensemble de transformations applicables aux comportements. Les transformations proposées dans

VENUS sont entre autres:

- Eliminer les chaînes et les circuits de τ.
- Réduire le système (par construction du quotient) tenant compte de la congruence forte ou l'équivalence ou la congruence observationnelle.
- Rendre réflexive la relation $\xrightarrow{\tau}$ (en considérant ou non l'état initial).
- Saturer le système c'est-à-dire ajouter des transitions $p \xrightarrow{\alpha} p''$ si $p \xrightarrow{\tau} p' \xrightarrow{\alpha} p''$ ou $p \xrightarrow{\alpha} p' \xrightarrow{\tau} p''$ (voir définition 1.III.2.).

II.- Quelques exemples d'analyse.

Nous présentons dans ce chapitre diverses utilisations du système VENUS montrant:

- les possibilités offertes par l'outil, pour l'étude du comportement des systèmes et en particulier les possibilités d'analyse de sous-comportements (exemple 1), en effet en limitant le nombre d'actions significatives "visibles" (les autres étant "invisibles" et représentées par τ), on peut "visualiser" un graphe plus petit décrivant un comportement local.
- les possibilités d'adaptation de l'outil à l'analyse de graphes produits à partir d'autres algèbres que CCS (exemples 2),
- les limites de la maquette réalisée en Prolog sur VAX (exemple 3).

Exemple 1: Vérification des propriétés des systèmes

Nous considérons un protocole de communication, inspiré par le modèle de réference ISO [Ta 81]. Nous considérons une description de "haut niveau" qui fait intervenir la couche "transport" de la norme ISO [Ga 86] [Na 87].

Le système considéré est composé de deux interlocuteurs, les sitesT_a et T_i, et de deux mediums T_{ia} et T_{ai}. Initialement, il n'y a pas de communication; pour l'établir, un des interlocuteurs (T_a, T_i), appelé *initiateur* doit envoyer au réseau (T_{ia}, T_{ai}) une *demande de connexion*. Le réseau transmet à l'autre interlocuteur que nous appelons *accepteur* (T_a, T_i) une *indication de connexion* et celui-ci émet une *réponse de connexion.* Le réseau transmet à l'initiateur une *confirmation de connexion.* La communication est ainsi établie.

Chacun des interlocuteurs peut transmettre des messages. Le contenu des messages est non significatif dans cette modélisation. L'interlocuteur qui veut transmettre un message doit émettre une

demande de données. Le réseau transmet à l'autre interlocuteur une *indication de données.*

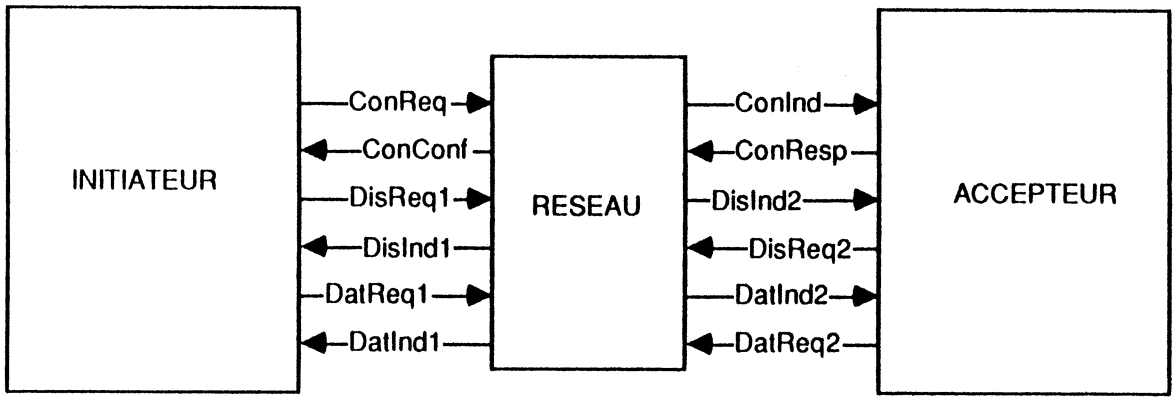
A tout instant, la communication peut être rompue. Il y a deux sortes de ruptures possibles:

- une demande de déconnexion est émise par l'un des interlocuteurs. L'interlocuteur qui veut rompre la communication émet une *demande de déconnexion* et le réseau transmet à l'autre interlocuteur une *indication de déconnexion.*
- une demande de déconnexion est émise par le réseau. Dans ce cas chaque interlocuteur reçoit une *indication de déconnexion.*

Une demande de connexion n'implique pas une connexion; en effet, la connexion peut être refusée par le réseau ou pour l'accepteur. Dans ce cas, le réseau envoie une *indication de déconnexion* à l'initiateur.

Nous considérons un medium de communication sûr (sans perte de messages). La tolérance aux pannes n'est pas prise en compte dans cet exemple. Les délais de reémission ne sont pas pris en compte.

Le protocole peut être schématisé par la figure suivante:

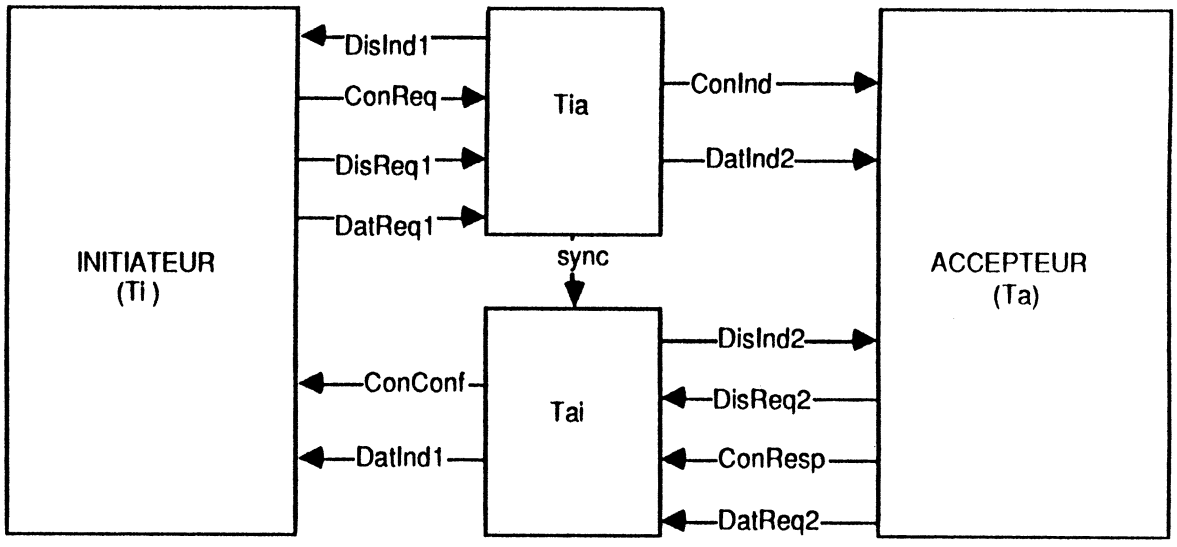


avec:

"**Con**nection **Req**uest" *(demande de connexion)*
"**Con**nection **Ind**ication" *(indication de connexion)*

"**Con**nection **Resp**onse" *(réponse de connexion)*
"**Con**nection **Conf**irmation" *(confirmation de connexion)*
"**Data Req**uest **i**", i = 1, 2 *(demande de données)*
"**Data Ind**ication **i**", i = 1, 2 *(indication de données)*
"**Dis**connect **Req**uest **i**", i = 1, 2 *(demande de déconnexion)*
"**Dis**connect **Ind**ication **i**", i = 1, 2 *(indication de déconnexion).*

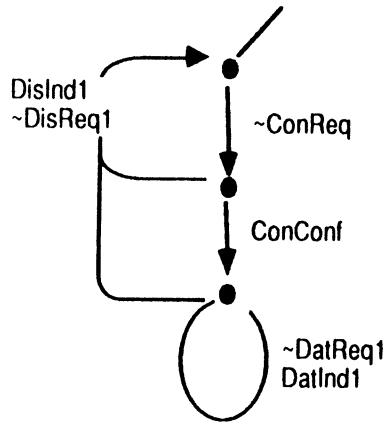
La structure du protocole est la suivante:



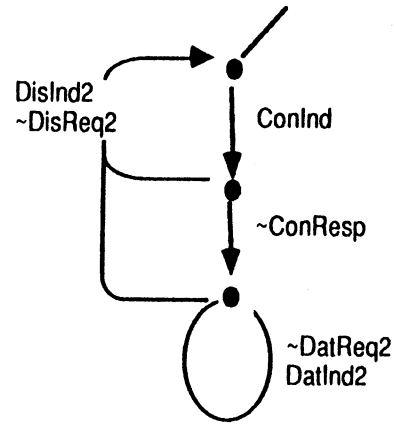
L'action **sync** sert à synchroniser la déconnexion de T_i et T_a, et le retour à l'état initial de T_{ai} et de T_{ia}.

Nous décrivons le comportement de chacun des composants du protocole par les graphes de transitions suivants:

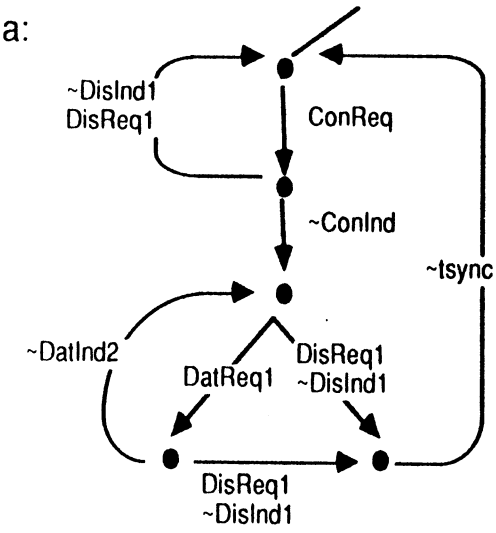
Ti:



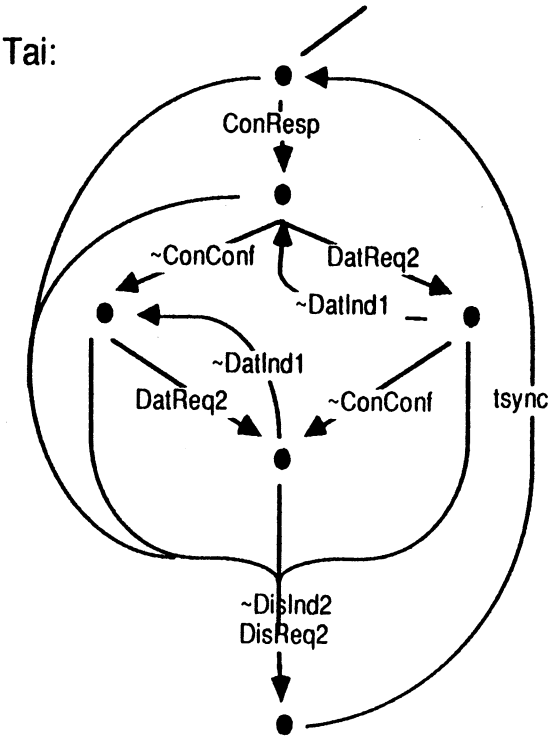
Ta:



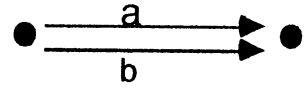
Tia:



Tai:



Notation: Par convention, un arc orienté avec des étiquettes multiples $\xrightarrow{a\,b}$ est utilisée à la place de:



Comme le système décrit, est un système fermé dans la mesure où il n'a pas d'interactions avec l'extérieur, les actions qui nous intéressent pour analyser son comportement, sont celles produites par des communications internes associées à certaines actions (conreq, conind, conresp, conconf, disreq, disind, datareq et dataind). En CCS toutes les transitions internes sont étiquettées par des τ, il y a perte d'une partie de l'information nécessaire pour l'analyse de ces systèmes. Pour remédier ceci nous utilisons un étiquettage de chaque τ-transition par le nom de l'action qui l'a produite.

Pour mieux comprendre et pour analyser le comportement du protocole réalisé, nous voulons construire les graphes de transitions réduits correspondant:

- au protocole décrit en étiquettant les τ-transitions associées à toute action différente de **sync**; ce graphe est appelé par la suite graphe 1,
- aux sous-comportements: connexion (graphe 2), déconnexion (graphe 3) et transmission de données (graphe 4).

Nous créons l'environnement de travail **haut_niveau**, à l'aide de la commande **cre** (voir annexe 1).

Nous construisons chaque graphe i, i = 1, 2, 3 et 4 en utilisant les commandes ci-dessous correspondant à la définition de l'étiquettage des τ-transitions et à l'application de la transformation *NEO* sur le graphe associé au terme **spec in specificat** décrivant le protocole de "haut niveau":

- **etiquettage**(E_i),

  avec E_1= **[conreq,conind,conresp,conconf,disreq1, disreq2,disind1,disind2,datreq1,datreq2, datind1,datind2]**

  E_2 = **[conreq,conind,conresp,conconf]**

E_3 = **[disreq1,disreq2,disind1,disind2]**

E_4 = **[datreq1,datreq2,datind1,datind2]**

- **neo(spec in specification),**

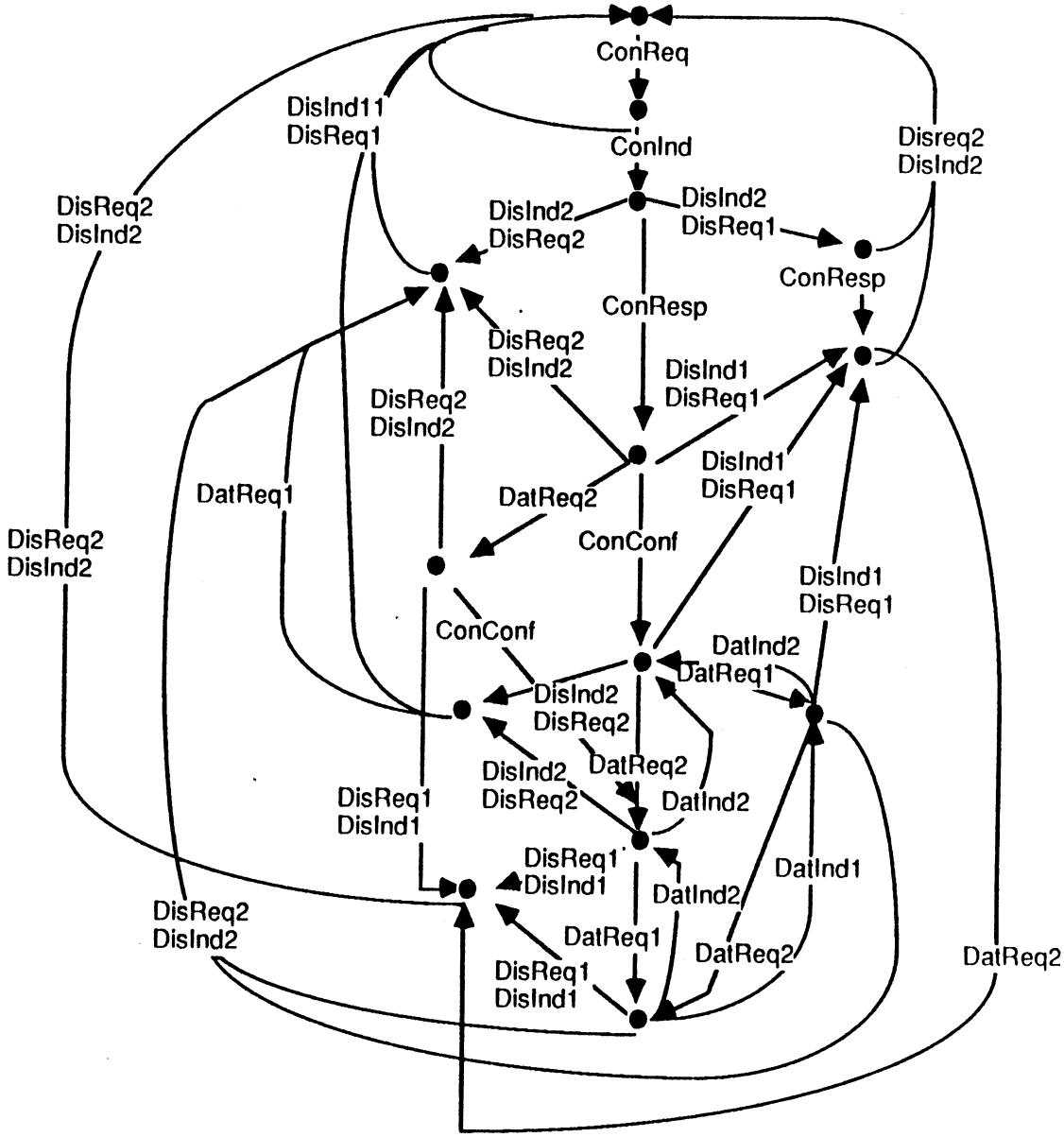
Les graphes obtenus sont:

Graphe 1: Général.

Nombre d'états: 14
Nombre de transitions: 57

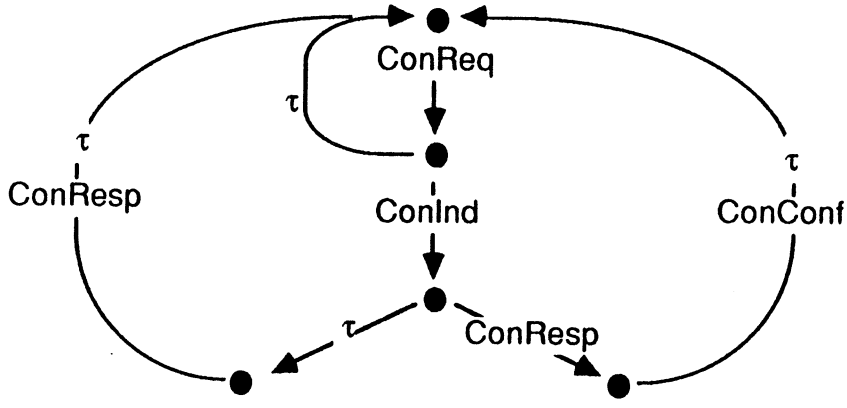
Détail des opérations réalisées:
Graphe initial: Nombre d'états: 18
Nombre de transitions: 65
Temps de construction: 15.18 sec.

Transformation *NEO*:	Nombre d'états	Nombre de transitions	temps [sec]
Pas 1: Elimination de chaînes et circuits de τ	17	64	0.25
Pas 2:			
- Saturation (définition 1.III.2)	17	64	0.54
- Construction des classes d'équivalence	-	-	5.65
- Construction du quotient	14	57	1.23
Pas 3: Réduction de transitions redondantes	14	57	0.44


ConReq
DisInd11
DisReq1
ConInd
Disreq2
DisInd2
DisReq2
DisInd2
DisInd2
DisReq2
DisInd2
DisReq1
ConResp
ConResp
DisReq2
DisInd2
DisInd1
DisReq1
DisReq2
DisInd2
DisInd1
DisReq1
DatReq1
DatReq2
DisReq2
DisInd2
ConConf
DisInd1
DisReq1
ConConf
DatInd2
DatReq1
DisInd2
DisReq2
DisInd2
DisReq2
DatReq2
DatInd2
DisReq1
DisInd1
DisReq1
DisInd1
DatInd2
DatInd1
DisReq2
DisInd2
DatReq1
DisReq1
DisInd1
DatReq2
DatReq2

Graphe 2: Connexion.

Nombre d'états: 5
Nombre de transitions: 9

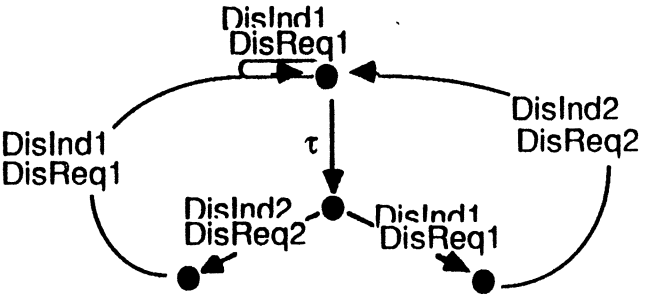


détail des opérations réalisées:
Graphe initial: Nombre d'états: 18
Nombre de transitions: 42
Temps de construction: 14.28 sec.

Transformation *NEO*:	Nombre d'états	Nombre de transitions	temps [sec]
Pas 1: Elimination de chaînes et circuits de τ	7	14	6.01
Pas 2:			
- Saturation (définition 1.III.2)	7	33	0.43
- Construction des classes d'équivalence	-	-	1.23
- Construction du quotient	5	10	0.24
Pas 3: Réduction de transitions redondantes	5	9	0.45

Graphe 3: Déconnexion.

Nombre d'états: 4
Nombre de transitions: 11

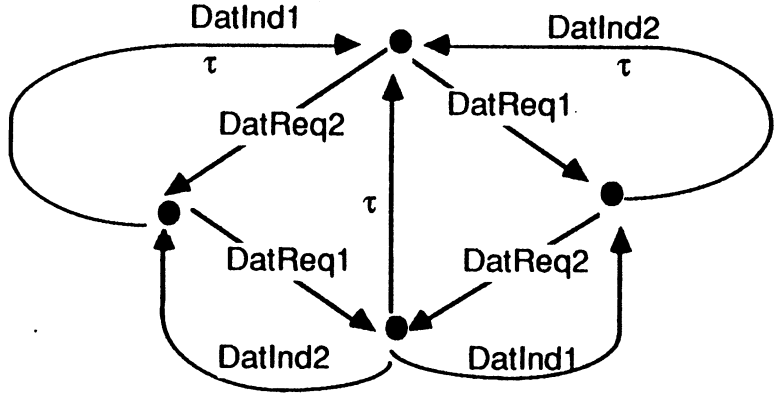


détail des opérations réalisées:
Graphe initial: Nombre d'états: 18
Nombre de transitions: 65
Temps de construction: 13.18 sec.

Transformation *NEO*:	Nombre d'états	Nombre de transitions	temps [sec]
Pas 1: Elimination de chaînes et circuits de τ	13	47	2.93
Pas 2:			
- Saturation (définition 1.III.2)	13	171	5.16
- Construction des classes d'équivalence	-	-	11.5
- Construction du quotient	4	11	1.55
Pas 3: Réduction de transitions redondantes	4	11	3.8

Graphe 4: Transmission de données.

Nombre d'états: 4
Nombre de transitions: 16



détail des opérations réalisées:
Graphe initial: Nombre d'états: 18
Nombre de transitions: 42
Temps de construction: 13.31 sec.

Transformation *NEO*:	Nombre d'états	Nombre de transitions	temps [sec]
Pas 1:			
Elimination de chaînes et circuits de τ	5	16	4.68
Pas 2:			
- Saturation (définition 1.III.2)	5	40	0.56
- Construction des classes d'équivalence	-	-	0.6
- Construction du quotient	4	13	0.15
Pas 3:			
Réduction de transitions redondantes	4	11	0.56

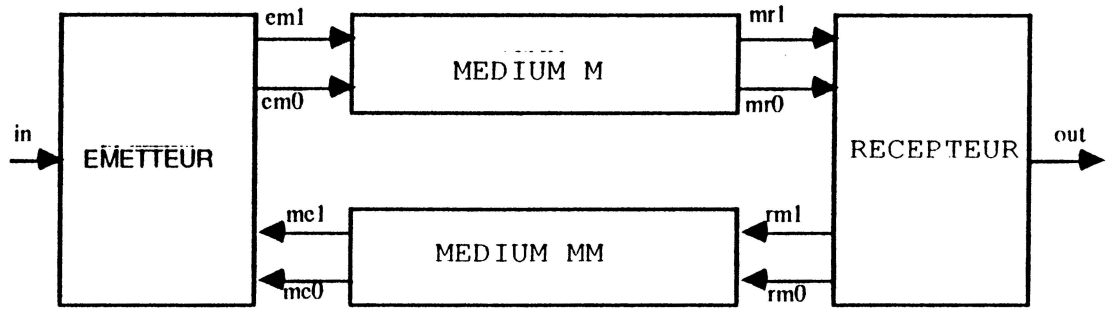
Exemple 2: Utilisation de VENUS pour réduire des graphes temporisés.

Cet exemple est une version simplifiée du protocole de communication du bit alterné (niveau OSI-4, canal D du protocole de transmission). La description de ce protocole a été donnée dans l'algèbre ATP [RVS 86], permettant de décrire des processus en tenant compte du temps; un graphe d'états a été produit par le système XESAR [Xe 86], et ce graphe a été réduit en utilisant le système VENUS.

Le système considéré est composé d'un émetteur, d'un récepteur et de deux mediums **M** et **MM**. L'emetteur envoie un message par le medium **M**, il ne renvoie pas de message tant qu'il n'a pas reçu par le medium **MM** un acquittement validant la réception, ou qu'un délai fini **de** se soit écoulé.

Les mediums sont des canaux non fiables de transmission de messages et d'acquittements. Un bit de contrôle est utilisé, d'une manière standard, pour la détection de perte de messages et d'acquittements. Le délai **de** entre l'instant d'émission d'un message et l'instant de reémission du même message est supérieur à la somme des délais de transmission **dm** par **M**, et **dmm** par **MM** (**dmm+dm<de**). Cette contrainte garantit qu'un message n'est retransmis que si lui-même ou son acquittement ont été perdus.

Ce système peut être schématisé par la figure suivante:



L'émetteur a cinq actions:

? in message à transmettre,

!em0, **!em1** transmission d'un message avec bit de contrôle 0 ou 1,

?me0, **?me1** réception d'un acquittement avec bit de contrôle 0 ou 1.

Le medium **M** a quatre actions:

?em0, **?em1** réception d'un message avec bit de contrôle 0 ou 1 envoyé par l'émetteur,

?mr0, **?mr1** transmission d'un message avec bit de contrôle 0 ou 1 au récepteur.

Le medium **MM** a quatre actions:

?rm0, **?rm1** réception d'un acquitement avec bit de contrôle 0 ou 1 envoyé par le récepteur,

?mr0, **?mr1** transmission d'un acquittement avec bit de contrôle 0 ou 1 à l'émetteur.

Le récepteur a cinq actions:

!out message reçu,

?mr0, ?mr1 réception d'un message envoyé par le medium **M**,

!rm0, !rm1 transmission d'un acquittement avec bit de contrôle 0 ou 1 au medium **MM**.

Le protocole est défini dans l'algèbre ATP par (t,E)

où $t=(E0 \parallel M0 \parallel MM0 \parallel R0)\backslash\Gamma$,

E est l'ensemble d'équations définissant les comportements de l'émetteur, les mediums et le récepteur,

E0, M0, MM0, R0 correspondent aux états initiaux de l'émeteur,des mediums M, MM et du récepteur,

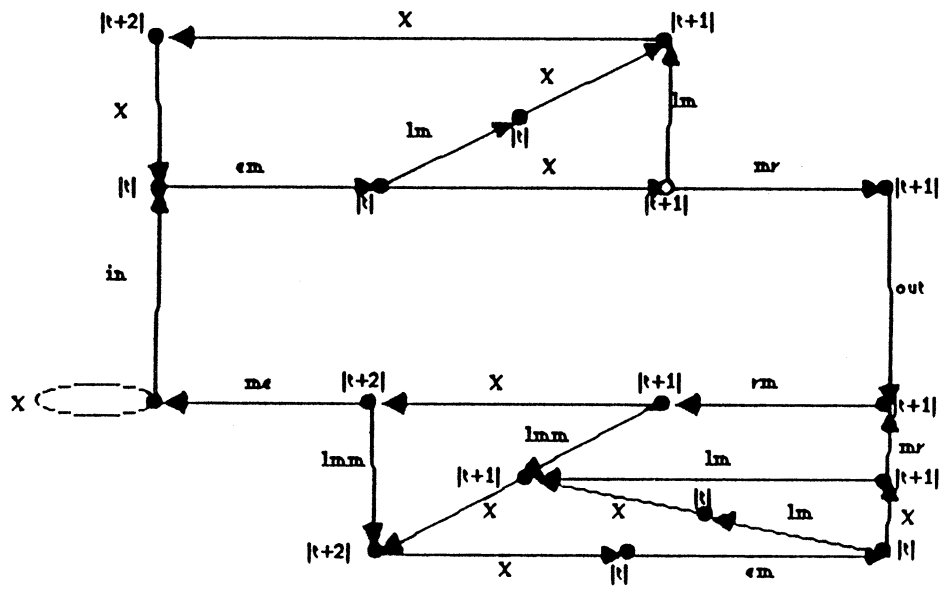
$\backslash\Gamma$ est l'opérateur de restriction de CCS,

$\Gamma=\{\$em0,\$em1,\$me0,\$me1,\$mr0,\$mr1,\$rm0,\$rm1 | \$\in\{?,!\}\}$.

Un graphe d'états pour ce système avec **de** = 3 et **dm** = **dmm** = 1 a été généré en utilisant le système XESAR. Le graphe obtenu comporte 74 etats et 91 transitions.

La figure suivante représente le graphe d'états associé à (t[Φ], E), où Φ est la fonction de redéfinition: (em, em, mr, mr, rm, rm, me, me) / (em0, em1, mr0, mr1, rm0, mr1, me0, me1). L'étiquette χ est associée aux transitions représentant l'écoulement du temps (les autres transitions ont une durée nulle).

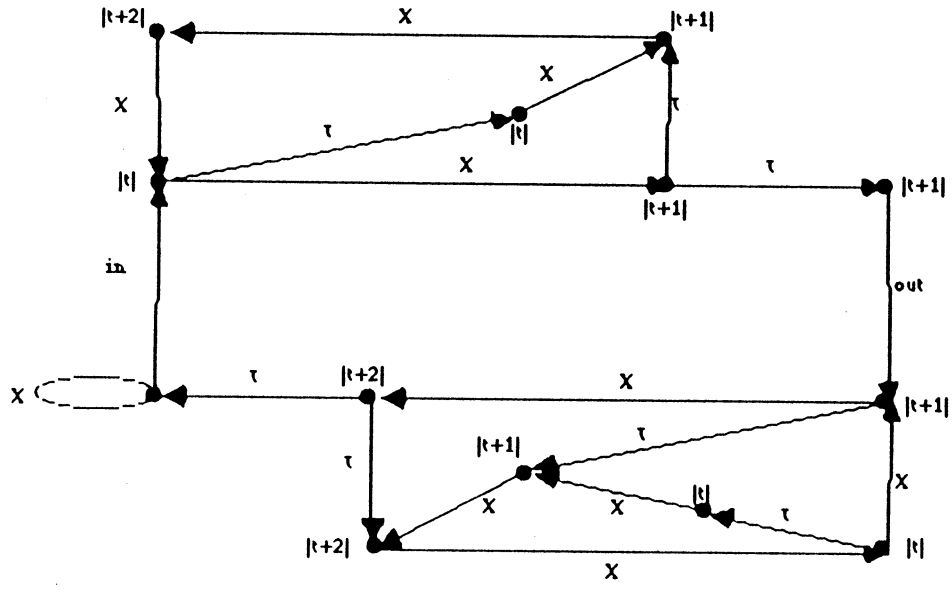
Pour faciliter l'interprétation de ce graphe, nous ajoutons aux τ-transitions des étiquettes lm, lmm associées à la perte d'un message et d'un acquittement par les mediums **M** et **MM.** A chaque état nous associons également une étiquette représentant la date modulo **de** (3 dans cet exemple), relative à l'instant de début d'émission d'un message ou d'un acquittement. Cette étiquette correspond au nombre de χ-transitions apparaissant dans le chemin qui lie l'état courant à l'état d'émission ou de reémission.



En utilisant le système VENUS, un graphe d'états minimal (nombre d'états et de transitions) pour l'équivalence observationnelle a été produit, les χ-transitions ont été laissées visibles. Ce graphe minimal comporte 39 états et 53 transitions. Pour cette minimisation, nous avons appliqué la transformation *NEO* :

Transformation *NEO*:	Nombre d'états	Nombre de transitions	temps [sec]
Pas 1: Elimination de chaînes et circuits de τ	47	64	5.31
Pas 2:			
- Saturation (définition 1.III.2)	47	88	0.96
- Construction des classes d'équivalence	-	-	18.33
- Construction du quotient	39	53	1.64
Pas 3: Réduction de transitions redondantes	39	53	1.51

La figure suivante correspond au graphe d'états associé à (t[Φ],E) avec $\Phi = (\tau, \tau, \tau, \tau, \tau, \tau, \tau, \tau)$ /(em0, em1, mr0, mr1, rm0, rm1, me0, me1) réduit en tenant compte de l'équivalence observationnelle de CCS.



Exemple 3: Utilisation de VENUS pour la génération et la réduction de graphes et limites de la maquete de VENUS pour le traitement d'exemples de taille réelle.

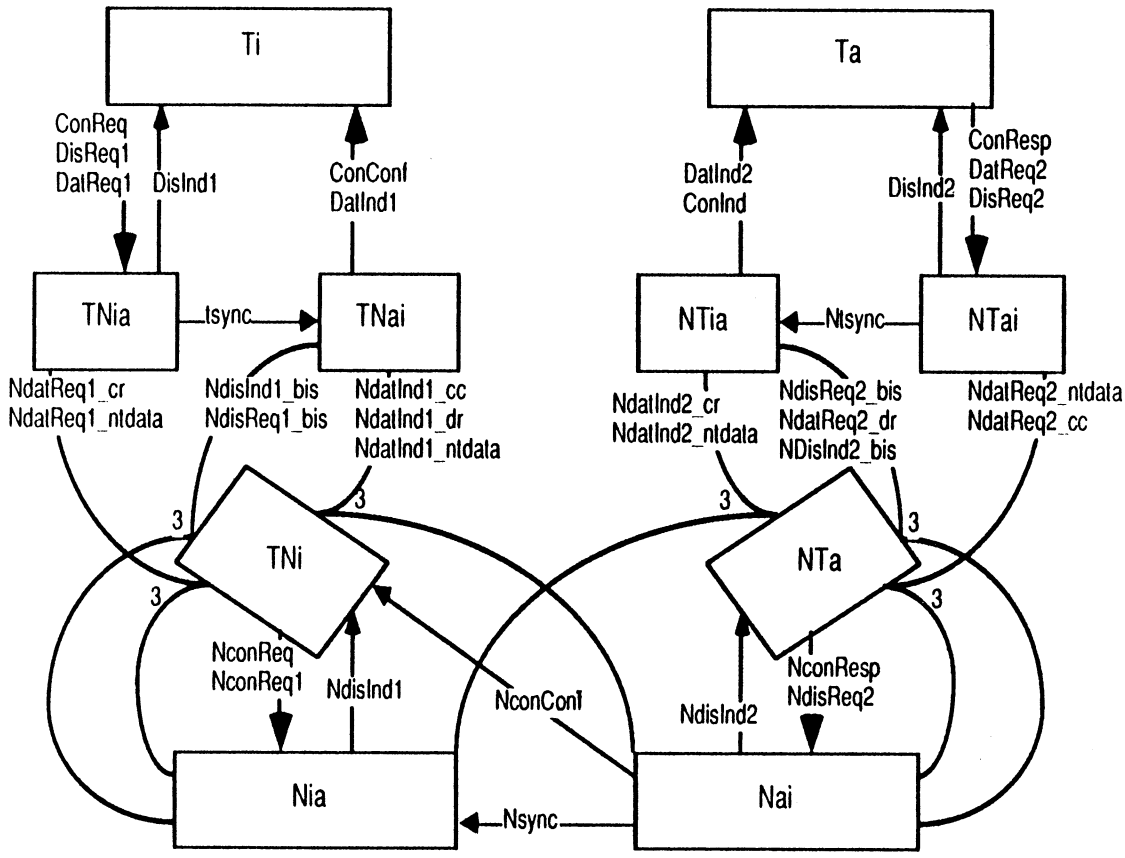
Nous considérons un protocole de communication, inspiré par le modèle de réference ISO [Ta 81]. Nous considérons une description de "haut niveau" (exemple 1) qui fait intervenir la couche "transport" de la norme ISO, et une description de "bas niveau" qui fait intervenir les couches "network" et "transport" [Ga 86] [Na 87].

La description de "bas niveau" est constituée par:

- une description NN ("Network-Network") qui décrit la communication entre les sites initiateur et accepteur au niveau "réseau", cette description est composée de deux processus $\mathbf{N_{ia}}$ et $\mathbf{N_{ai}}$ qui communiquent avec $\mathbf{T_i}$ et $\mathbf{T_a}$ à l'aide des entités TN et NT,

- les entités TN ("Transport-Network") et NT ("Network-Transport") qui réalisent les interfaces entre les couches "transport" et "réseau" sur le site initiateur et accepteur; ces entités sont composées par les processus TN_{ia}, TN_{ai}, TN_i et NT_{ai}, NT_{ia}, NT_a.

La structure du protocole est la suivante:

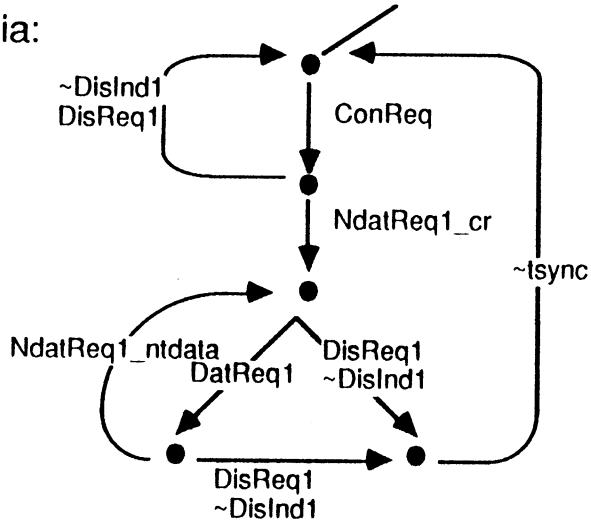


où: ConReq, CinInd, ConResp, ConConf, DisReq1, DisReq2, SisInd1, DisInd2, DatReq1, DatReq2,DatInd1, DatInd2 sont les actions définies dans la description de "haut niveau",
NconReq, NconInd, NconResp, NconConf, NdisReq1, NdisReq2, NdisInd1, NdisInd2 sont les actions de connexion et de déconnexion du niveau réseau,
les autres actions sont utilisées pour le transfert de données dans le réseau.

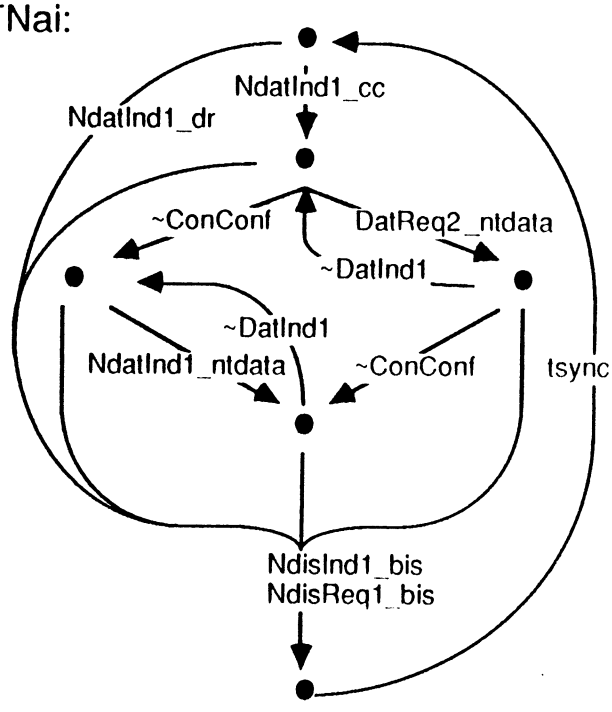
Notation: le numero 3 dans les arcs indique une communication à trois.

Nous décrivons le comportement de chacune des composantes du protocole par les graphes de transitions suivants:

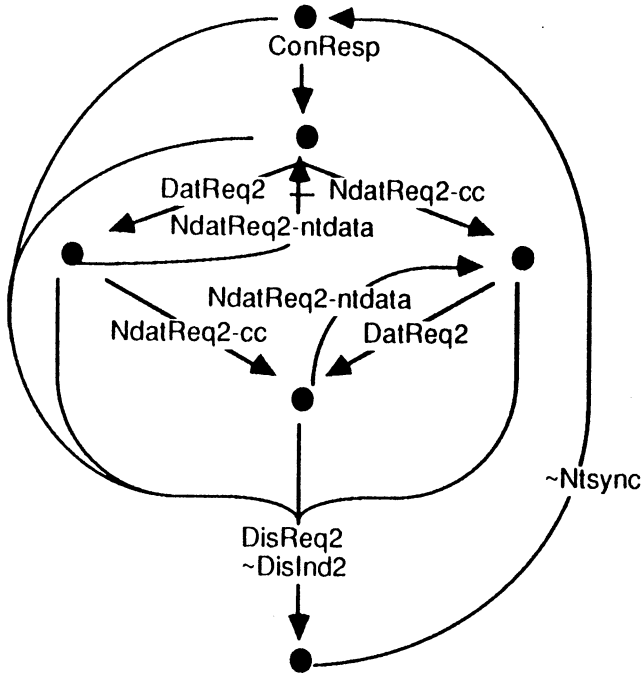
TNia:


~DisInd1
DisReq1
ConReq
NdatReq1_cr
~tsync
NdatReq1_ntdata
DatReq1
DisReq1
~DisInd1
DisReq1
~DisInd1

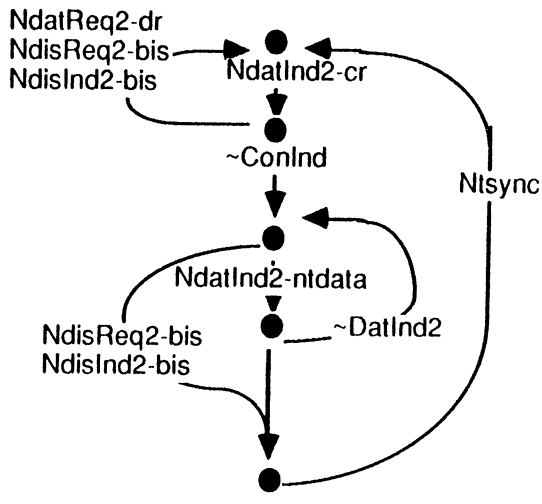
TNai:


NdatInd1_cc
NdatInd1_dr
~ConConf
DatReq2_ntdata
~DatInd1
~DatInd1
NdatInd1_ntdata
~ConConf
tsync
NdisInd1_bis
NdisReq1_bis

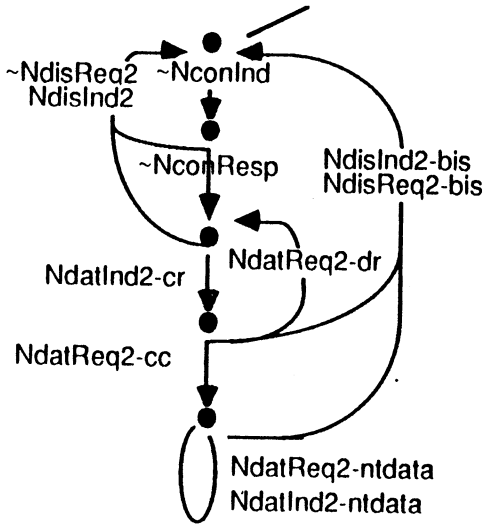
NTai:


ConResp
DatReq2
NdatReq2-cc
NdatReq2-ntdata
NdatReq2-ntdata
NdatReq2-cc
DatReq2
~Ntsync
DisReq2
~DisInd2

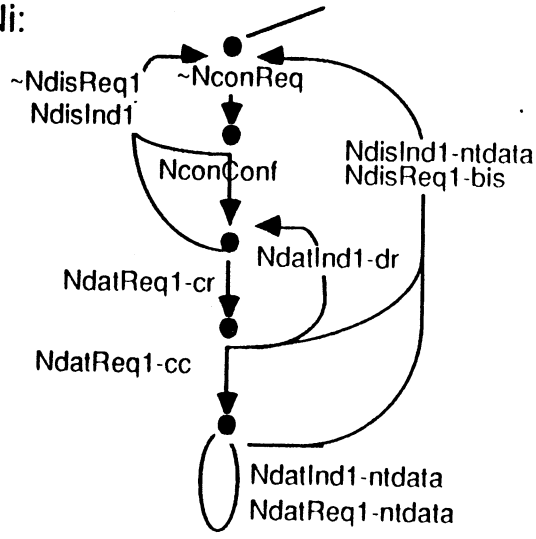
NTia:


NdatReq2-dr
NdisReq2-bis
NdisInd2-bis
NdatInd2-cr
~ConInd
Ntsync
NdatInd2-ntdata
~DatInd2
NdisReq2-bis
NdisInd2-bis

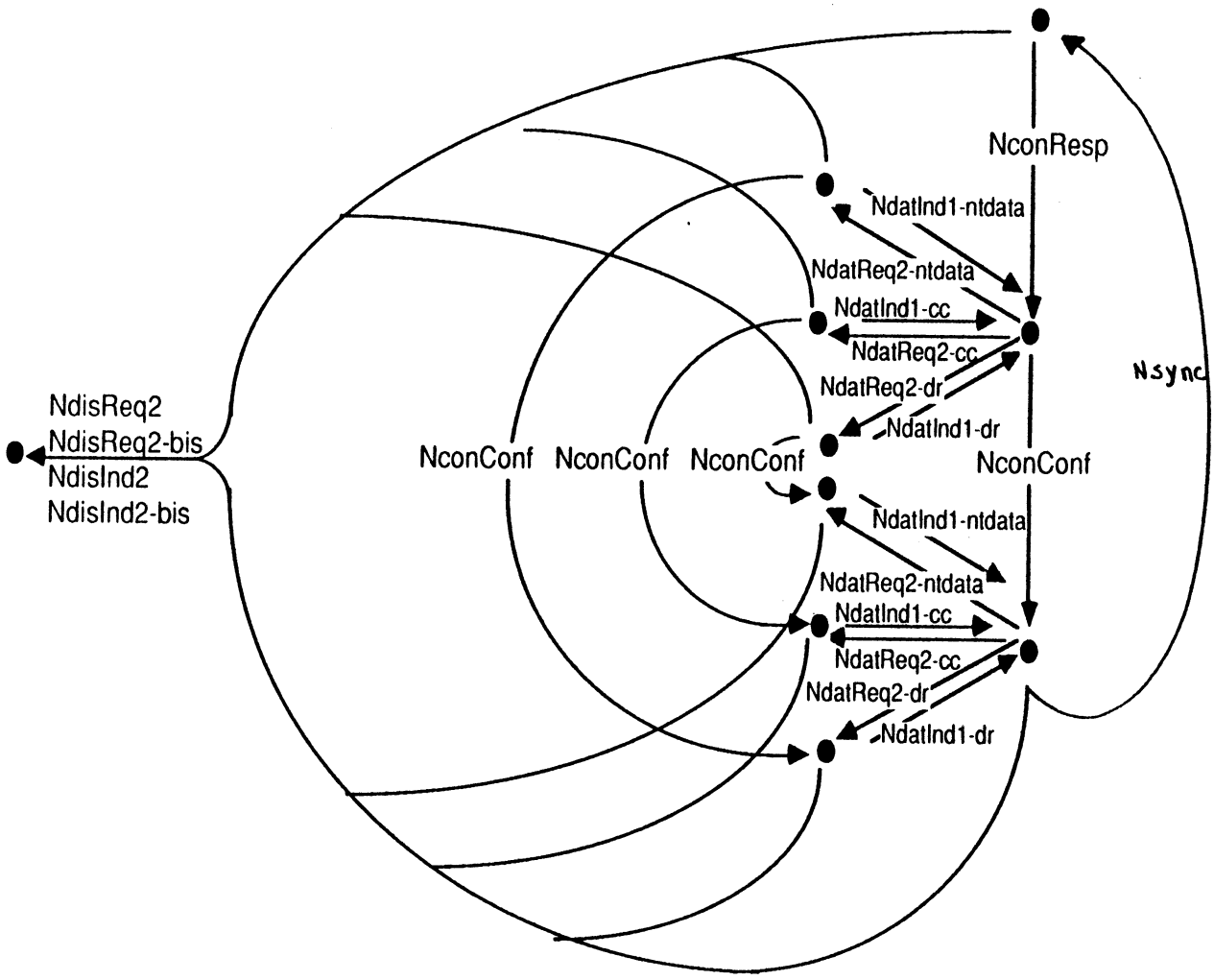
TNa:


~NdisReq2
NdisInd2
~NconInd
~NconResp
NdisInd2-bis
NdisReq2-bis
NdatReq2-dr
NdatInd2-cr
NdatReq2-cc
NdatReq2-ntdata
NdatInd2-ntdata

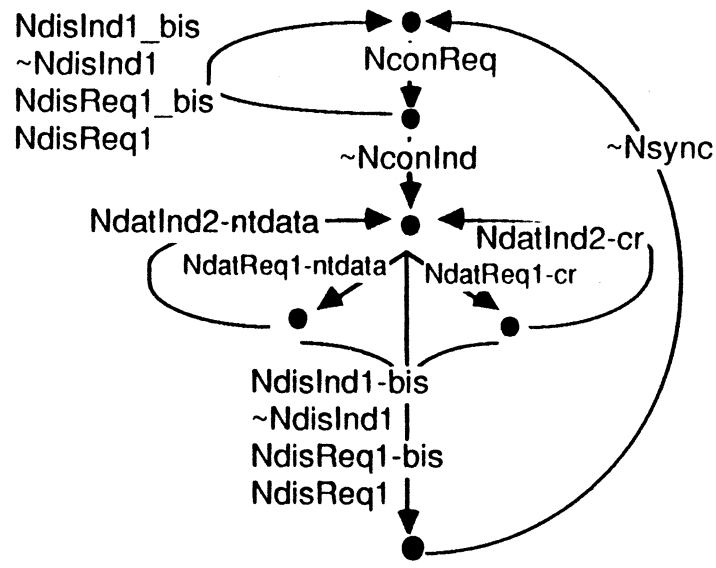
TNi:


~NdisReq1
NdisInd1
~NconReq
NconConf
NdisInd1-ntdata
NdisReq1-bis
NdatInd1-dr
NdatReq1-cr
NdatReq1-cc
NdatInd1-ntdata
NdatReq1-ntdata

Nai:


NconResp
NdatInd1-ntdata
NdatReq2-ntdata
NdatInd1-cc
NdatReq2-cc
NdatReq2-dr
NdatInd1-dr
Nsync
NdisReq2
NdisReq2-bis
NdisInd2
NdisInd2-bis
NconConf
NconConf
NconConf
NconConf
NdatInd1-ntdata
NdatReq2-ntdata
NdatInd1-cc
NdatReq2-cc
NdatReq2-dr
NdatInd1-dr

Nia:

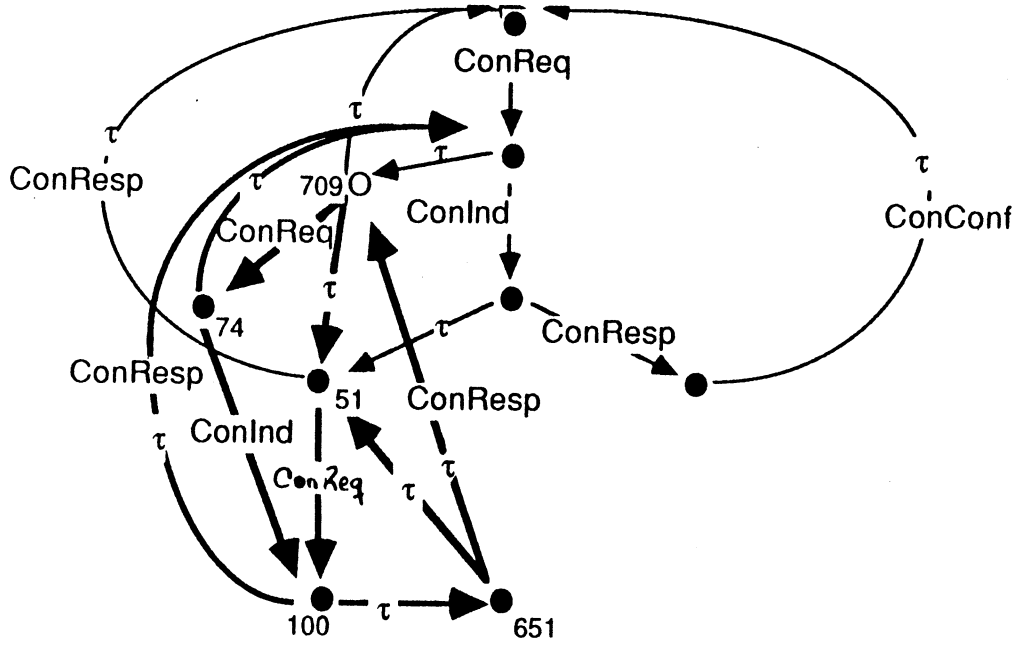


Nous créons un environnement de travail contenant la description de "bas niveau" donnée ci-dessus. Nous construisons le système de transitions associé, à l'aide de la commande **étiquettage** qui étiquete par "t-" suivi par le nom de l'action les communications internes τ produites à partir des actions ConReq, ConInd, ConResp, ConConf, DisReq1, DisReq2, DisInd1, DisInd2, DataReq1, DataReq2, DataInd1, DataInd2. Ce système de transitions a été construit par utilisation de la commande **add** (voir annexe 1) dans laquelle nous réalisons le rendez-vous à trois. L'automate construit contient 790 états et 4265 transitions. Nous avons vérifié que les termes correspondant à la spécification ("haut niveau") et à la réalisation ("bas niveau") de ce protocole ne sont pas observationnellement équivalents.

En effet, en faisant *abstraction* [BK 84] de toute action n'appartenant pas à la phase de connexion du protocole, nous construisons un automate décrivant la phase de connexion du protocole de "bas niveau" et sa forme normale à partir de l'automate anterieure. Cette forme normale contient 9 états et 21 transitions.

Il est possible d'observar que la phase de connexion de "bas niveau" n'est pas elle non plus observationnellement équivalent à la phase de connexion de "haut niveau".

L'automate correspondant à la forme normale de la phase de connexion du protocole de "bas niveau" est le suivant:



Dans cet automate, nous avons noirci la partie de l'automate qui fait la différence par rapport à la phase de connexion du protocole de "haut niveau".

Du point de vue performance, on peut signaler que le prototype de VENUS construit en PROLOG-C atteind ses limites quand nous traitons des exemples de grande taille (des milliers de transitions). En effet, nous avons généré des graphes correspondant à des sous-parties de cette description du protocole, par exemple le réseau de bas niveau sans les deux comportements T_i et T_a, et sans étiquetter les τ-transitions; il peut ainsi communiquer avec l'extérieur.

Le graphe généré comporte environ 2000 états et 17000 transitions; par saturation après élimination des chaînes et des circuits de τ, nous avons ajouté environ 50000 transitions.

Ceci nous permet de conclure que les limites réelles ne sont pas imposées par le stockage des transitions, mais par leur traitement. En fait, chaque fois qu'une résolution doit être faite, PROLOG doit parcurir une grande quantité de transitions ce qui alourdit le temps de calcul nécessaire. Une solution possible pour des systèmes comportant des milliers de transitions serait peut être l'utilisation d'un PROLOG compilé.

III.- Réalisation d'un prototype de VENUS.

Pour la réalisation de VENUS, nous avons cherché à construire un prototype modulaire. Ce prototype de VENUS a été programmé en PROLOG (version 1.5) [Pe 84] [CM 81] sous UNIX 4.2 sopporté par un VAX 790. Le programme contient environ 400 prédicats PROLOG.

Par la suite, nous présentons l'architecture du prototype et une description de chacun des modules qui le composent.

III.1.- Architecture générale du prototype.

Ce prototype du système VENUS est composé de cinq modules:

- Module *moniteur*: Ce module est le moniteur de VENUS.
- Module *gestion des environnements:* Ce module réalise le type abstrait environnements, en permettant la définition et la gestion des environnements de travail.
- Module *automates:* Ce module contient la réalisation de toutes les fonctions concernant les automates.
- Module *termes:* Ce module contient la réalisation de toutes les fonctions concernant les termes.
- Module *gestion des écrans:* Ce module sert d'interface de communication entre l'utilisateur et le système.

Les rélations entre modules sont montrées dans la figure III.1.

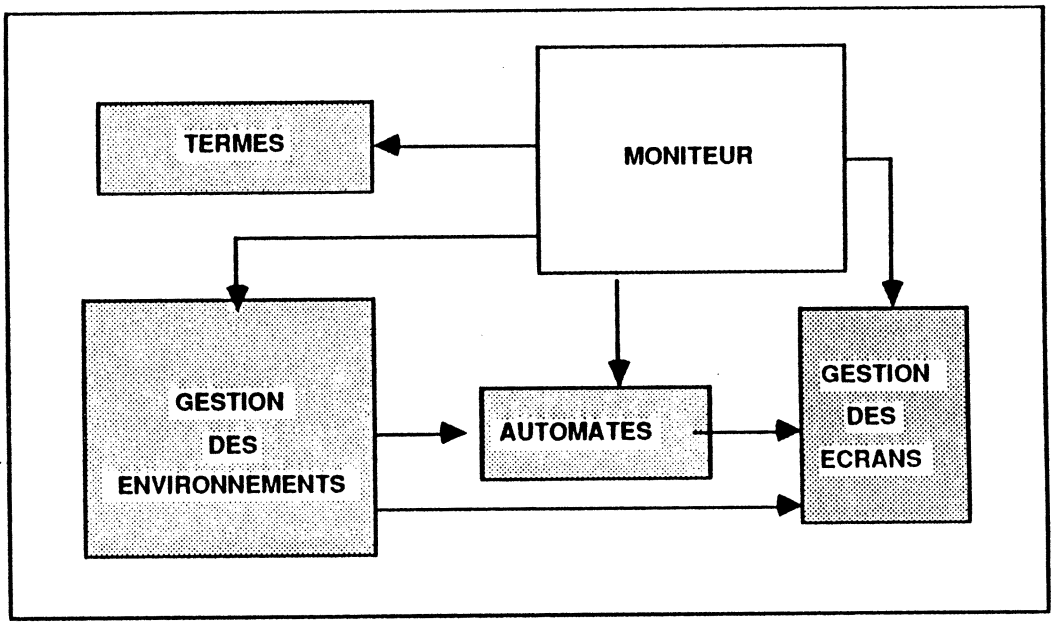


Figure III.1: Diagramme de Relations des Modules

III.2.- Module *moniteur.*

Le module *moniteur* coordonne les niveaux de commandes, analyse les commandes fournies par l'utilisateur et dirige leur exécution.

Nous présentons par la suite la liste des prédicats prolog réalisant les fonctions définies dans le module moniteur. Cette liste est présentée dans un ordre logique d'exécution.

- **commencer:** démarre VENUS au le niveau de commandes 1.

- **travail**: contrôle le niveau de commandes 2 et coordonne l'exécution des commandes fournies par l'utilisateur.

- **traiter_commande(<commande>)** coordonne le traitement d'une commande, c'est-à-dire l'analyse syntaxique et l'exécution de la commande <commande>.

- **analyse_commande(<commande>,<clause-commande>)** réalise la fonction d'analyse syntaxique de la commande <commande> et produit une clause <clause-commande> correspondant à l'appel de la commande fournie par l'utilisateur.

- pour chaque commande utilisateur il y a une clause du même nom et des mêmes arguments décrivant l'exécution de la commande.

III.3.- Module *gestion des environnements.*

Le module *gestion des environnements* définit et réalise le type abstrait *environnement.*

Un objet du type environnement est un environnement de travail, c'est-à-dire une définition d'environnement faite en tenant compte de la syntaxe définie dans le chapitre I, et un ensemble de comportements (termes et automates) obtenus par transformation d'autres comportements de l'environnement.

Les opérations et fonctions définies sur les environnements sont:

- **cre(<éditeur>)**	création d'un environnement en utilisant l'éditeur indiqué.
- **le**	lister les noms des environnements déjà sauvegardés.
- **che(<environnement>)**	charger <environnement> dans l'environnement courant.
- **me(<éditeur>)**	modifier la définition de l'environnement courant en utilisant l'éditeur spécifié.
- **de(<environnement>)**	détruire l'environnement indiqué.
- **se(<environnement>)**	sauvegarder l'environnement courant sous le nom <environnement>.
- **ee(<environnement>)**	écrire la définition de l'environnement indiqué.
- **lc**	lister les comportements de l'environnement courant.

- **chc(<comportement>)** charger le comportement indiqué comme le comportement courant.
- **eec** lister la définition de l'environnement courant.
- **alias(<comportement>,<idf>)** renommer <comportement> par <idf>.
- **lv** lister les variables de comportement. déclarées dans l'environnement courant.
- **ec(<comportement>)** afficher le comportement indiqué.
- **ecc** afficher le comportement courant.
- **dcc** détruire le comportement courant.
- **dc(<comportement>)** détruire le comportement indiqué.
- **sc** sauvegarder le comportement courant.

Un environnement est représenté par un catalogue UNIX. Ce catalogue est identifié par le nom de l'environnement. Dans ce catalogue on définit:

- un fichier *source* contenant la définition de l'environnement,

- un fichier *courant* contenant toute l'information actuelle sur les comportements composant l'environnement; c'est-à-dire courant est un fichier descripteur de l'environnement,

- un catalogue *objets* contenant l'ensemble de fichiers contenant les comportements sauvegardés.

L'ensemble de catalogues contenant les définitions des environnements sont réunis dans un catalogue appelé *env.*

III.3.1.- Description du fichier *courant.*

Le fichier *courant* joue le rôle de *descripteur* de l'environnement; c'est-à-dire dans ce fichier l'information générale permettant d'identifier et de localiser les comportements de l'environnement est enregistrée sous la forme des clauses prolog suivantes:

(i) Pour chaque variable de comportement déclarée dans la définition de l'environnement, il y a une clause de la forme:

env_def(<composante>,<idf>,<ty>,<description>,<dd>).
où:

<composante> indique la composante dans laquelle la variable est déclarée,

<idf> est le nom de la variable de comportement,

<ty> est un indicateur du type de comportement, c'est-à-dire

- ty = d indique que la variables est définie en **definition**,
- ty = e indique que la variable est définie en **equation**,
- ty = i indique que la variable n'est pas définie,

<description> contient la définition de la variable, c'est-à-dire le terme de la partie droite de l'équation définissant la variable de comportement,

<dd> donne l'information sur les variables utilisées en <description>, c'est-à-dire

- dd =sv <description> ne contient pas de variables,
- dd =cd <description> contient uniquement des variables complètement définies,
- dd = sd <description> contient des variables non définies.

(ii) Pour chaque comportement construit par transformation d'autres comportements, il y a une clause jouant le rôle de descripteur. Cette clause est de la forme:

env_ob(<composante>,<idf>,<transform>,<ni>,<to>).
où:

<composante> et **<idf>** identifient le comportement (la **composante** et le nom de la variable de comportement **<idf>**) transformé,

<transform> indique la suite de transformations réalisées,

<to> indique la représentation du comportement:
- to = a indique automate,
- to = sv indique terme sans variables (terme fini),
- to = cd indique terme récursif completement défini,
- to = sd indique terme récursif semi-défini,

<ni> contient le terme ou le nom de l'automate selon la valeur de <to>.

(iii) S'il existe un comportement courant dans l'environnement, le fichier courant contient une clause utilisée comme descripteur du comportement courant. Cette clause est de la forme:

objet_courant(<composante>,<idf>,<transf>,<ni>,<to>).
où: **<composante>, <idf>, <transf>, <ni>, <to>** ont la signification donnée précédemment.

(iv) Pour chaque variable définie dans les parties **definition** ou **equation**, il existe un ensemble de clauses indiquant les variables utilisées dans sa définition (une clause pour chaque variable utilisée dans la définition). Ces clauses ont la forme suivante:

comp_utilisé(<idf1>,<compos1>,<idf2>,<compos2>).
où:
<idf1> et <compos1> identifient le nom de la variable définie et la composante dans laquelle se trouve sa définition,
<idf2> et <compos2> donne de la même manière le nom et l'origine d'une des variables utilisées dans la partie droite de la définition de la variable.

(v) Pour chaque comportement pour lequel l'utilisateur a donné un nom par utilisation de l'opérateur *alias*, il y a une clause de la forme:

comp_alias(<comp>,<idf>,<transf>,<alias>).
où:
<comp>, <idf> et **<transf>** identifient un comportement pour lequel il existe une clause **env_ob**,
<alias> est le nom que l'utilisateur donne à ce comportement.

III.3.2.- Description du catalogue *objets.*

Dans le catalogue *objets,* chaque comportement construit par transformation d'un autre comportement est enregistré dans un fichier (comportement pour lequel il existe une clause **env_ob**).

Chaque fichier est identifié par un nom constitué de la manière suivante:

<comp>-<idf>-<transf>

où: **<comp>**, **<idf>** et **<transfo>** sont les informations données par une clause **env_ob(<comp>,<idf>,<transf>,<ni>,<to>)** décrite précédemment.

Chaque fichier comporte un ensemble de clauses prolog permettant d'identifier et de décrire le comportement:

(i) Une clause pour identifier le comportement et pour le signaler comme comportement courant quand il sera chargé **objet_courant(<comp>,<idf>,<transf>,<ni>,<to>).**

(ii) Un ensemble de clauses pour décrire le comportement. Cette ensemble de clauses correspond à la représentation d'un automate ou celle d'un terme selon le type de comportement (Ces représentations sont données dans les paragraphes III.4 et III.5).

III.4.- Module *automates.*

Un automate est constitué d'un ensemble de transitions et d'un état initial. Ce module réalise toutes les fonctions concernant les automates.

Les automates sont représentes par un ensemble de clauses prolog. Les états sont numérotés par des entiers et nous enregistrons la correspondence entre les variables du terme qui a produit l'automate et le nombre de l'état. Les clauses utilisés dans la représentation sont:

1.- Un ensemble de clauses pour représenter les transitions de la forme:

<aut>(<état1>,<étiquette>,<état2>).

décrivant une transition **<état1>** —**<étiquette>**→ **<état2>** du système **<aut>.**

2.- Une clause pour indiquer l'état initial du système. Cette clause est de la forme:

etat_initial(<aut>,<état>).

3.- Pour chaque état correspondant à une variable de comportement, nous avons une clause de la forme:

rel_etats(<aut>,<état>,<variable>).

4.- Une clause pour indiquer le numero maximal associé à un état de l'automate. Cette clause est de la forme:

max_numero(<aut>,<nombre>).

Cette clause est nécéssaire pour connaître les numéros pouvant être utilisés comme noms de nouveaux états.

Les fonctions et les opérations sur les automates sont réalisées par des prédicats prolog. Ces prédicats sont:

- **initialiser(<aut>,<terme>)** génération des clauses permettant la construction de l'automate **<aut>** construit à partir du terme **<terme>**; ces clauses sont:
  etat(<aut>,<terme>,1) indique que l'état **1** de l'automate <aut> correspond au terme **<terme>**,
  etat_initial(<aut>,1) indique que l'état initial de l'automate **<aut>** est **1**,
  expression(<aut>,1,1,d) indique que les transitions à générer ont comme etat initial 1 et sont obtenues à partir du terme numéroté par 1 dans une clause **etat**,

max_numero(<aut>,1) indique que le nombre maximal associé à un état de l'automate **<aut>** est **1**,

max_numero_aux(<aut>,1) indique que le prochain nombre à asigner à un terme intermédiaire nécéssaire pour la construction de l'automate **<aut>** est **1**.

Les termes intermédiaires sont enregistrés au cours de la construction de l'automate dans des clauses de la forme **etat_p(<aut>,<terme>,<numero>)**,

- **st(<aut>)** construction de l'automate <aut> associé au terme signalé par **<n-terme>** dans la clause **expression(<aut>,<état>,<n-terme>,<type>),** où:
  <aut> est le nom de l'automate,
  <état> est l'état de départ des transitions à générer,
  <n-terme> est le numero du terme à traiter,
  <type> = d indique terme définitif,
  <type> = p indique terme intermédiaire,
- **const_terme(<aut>)** construction d'un terme décrivant un automate **<aut>**; le terme est représenté par un ensemble de clauses prolog indiqué dans le paragraphe IV.5
- **saturation(<aut>,<aux>)** ajoute dans <aux> les transitions résultantes de la saturation de <aut>,
- **pgb(<aut1>,<aux1>,<aut2>,<aux2>)** construire la plus grande bisimulation entre les automates <aut1> ∪ <aux1> et <aut2> ∪ <aux2> ,
- **pgb_eo(<aut1>,<aux1>,<aut2>,<aux2>)** construire la plus grande bisimulation entre les automates <aut1> ∪ <aux1> et <aut2> ∪ <aux2> en supposant la relation de τ-transitions réflexive pour tous les états,
- **pgb_co(<aut1>,<aux1>,<aut2>,<aux2>)** construire la plus grande r-bisimulation entre les automates <aut1> ∪ <aux1> et <aut2> ∪ <aux2>,
- **cf_système_minimal(<aut>)** construire le quotient de l'automate <aut> pour la congruence forte,
- **co_système_minimal(<aut>)** construire le quotient de

l'automate <aut> pour la congruence observationnelle,

- **eo_système_minimal(<aut>)** construire le quotient de l'automate <aut> pour l'équivalence observationnelle,
- **reduction_trans(<aut>,<aux>)** éliminer des transitions pour obtenir un automate avec un nombre minimal de transitions à partir de l'automate <aut> $\cup$ <aux> (où l'automate donné ne contient pas de circuits de τ),
- **reflex(<aut>,<étiquette>)** rendre réflexive la relation étiqueté par <étiquette> dans l'automate <aut>,
- **reflex_sauf_initial(<aut>,<étiquette>)** rendre réflexive la relation étiqueté par <étiquette> dans l'automate <aut> sauf dans l'état initial,
- **t_classes_co(<aut>)** applique les transformations élimine τ-chaînes et τ-circuits à l'automate **<aut>**. Une clause **non_congru** est générée si la congruence n'est pas garantie,
- **garantir_congruence(<aut>)** si la clause **non_congru** est présente, on ajoute un nouveau état initial et une τ-transition de cet état à l'état initial de <aut> et détruit la clause **non_congru**,
- **nil_classes(<aut>)** construit le quotient de <aut> par rapport à la classe de congruence observationnelle [nil],

Ces prédicats sont définis dans des sous-modules organisés selon la nature de l'opération qu'ils réalisent. Les sous-modules sont:

- **creation**: contient les predicats **st** et **initialiser**.

- **transformations**: contient les prédicats de transformation des automates: **nil_classes**, **saturation**, **cf_système_minimal**, **co_système_minimal**, **t_classes_co**, **garantir_congruence** **reflex**, **reflex_sauf_initial**, **eo_système_minimal** et **reduction_trans**,

- **bisimulation:** contient la définition des opérations de comparaison des automates: **pgb**, **pgb_eo** et **pgb_co**,

- **décompilation:** contient le prédicat **const_terme**.

III.5.- Module *termes.*

Un *terme* est un terme de la forme (t,E) de l'algèbre T_r définie dans la partie I chapitre I. Ce module réalise toutes les fonctions concernant les termes.

Les termes de la forme (t,Ø) (termes finis) sont représentés par l'arbre d'opérateurs associé à t. Vue le grand nombre d'opérateurs α, $\backslash\alpha$ et [f] définis dans CCS, nous utilisons par des raisons pratiques des opérateurs *, \ et # de la manière suivante:

Soient: T l'ensemble de termes,
F l'ensemble de fonctions de redéfinition,
Parties(P_τ) l'ensemble parties de P_τ,

les opérateurs sont donc,

$* : P_\tau \times T \dashrightarrow T$

$\backslash : T \times$ **P**arties(P_τ) -----> T

$\# : T \times F$ -----> T

Les termes (v,E), $v \in V$ et $E \neq \emptyset$ sont représentes par un ensemble de clauses prolog. Les clauses utilisées dans la représentation sont de deux types:

(i) Une clause **terme(<v>)** indiquant quelle est la variable v définissant le terme.

(ii) Un ensemble de clauses représentant E. Les clauses sont de la forme **def_eq(<état>,<var>,<terme>)**, où: **<var>** est la variable associée à l'état **<état>** et **<terme>** est une liste d'arbres d'opérateurs représentant les sous-termes de la partie droite de l'équation.

Les termes u = (t,E), où t n'est pas une variable sont représentés par le terme u' = (v,{v=t}∪E) où v∉var(u).

Les prédicats prolog réalisant des fonctions et des opérations définies sur des termes finis sont:

- **formesomme(<terme1>,<terme2>)** construction du terme <terme2> écrit sous la forme d'une somme à partir de <terme1>.
- **expansion(<terme1>,<terme2>)** construire le terme <terme2> par application du théorème d'expansion sur le terme <terme1>,
- **normale(<terme1>,<terme2>)** construction de la forme normale <terme2> associée au terme sous la forme d'une somme <terme1>,
- **propre(<terme1>,<terme2>)** construire le terme <terme2> par application (si possible) de la règle $\tau.\alpha.x \rightarrow \alpha.x$
- **sommecongru(<terme1>,<terme2>)** est la fonction booléenne qui est vrai si les termes <terme1> et <terme2> sont égaux modulo l'associativité et la commutativité de l'opérateur +.

Ces prédicats sont définis dans des sous-modules organisés selon la nature de l'opération qu'ils réalisent. Les sous-modules sont:

- **somme**: ce module contient la définition des prédicats **formesomme** et **expansion**.

- **normal**: ce module contient la définition des prédicats **normale** et **propre** pour la construction des formes normales des termes écrits sous la forme d'une somme,

- **comparaison:** ce module contient la définition du prédicat **sommecongru** pour la comparaison de termes.

III.6.- Module gestion des écrans.

Le module Gestion des écrans réalise l'interface de communication entre VENUS et son utilisateur. Dans ce module les différents niveaux de commandes sont définis.

Les prédicats définis dans ce module sont:
- **ecran_nc1** réalisant le niveau de commandes 1,
- **ecran_nc2** réalisant le niveau de commandes 2.

Ce module contient le sous-module **CCS-VT100** réalisant des fonctions élémentaires de gestion d'écran pour un terminal VT100.

III.7.- Quelques aspects de la réalisation en PROLOG-C.

Nous nous sommes fixés une contrainte pour la réalisation du prototype de VENUS: possibilité de traitement de grands systèmes; ainsi le choix des structures de données et des algorithmes utilisées pour enregistrer, modifier et comparer des comportements doit tenir compte des critères de la mémoire utilisée pour stocker le comportement, de l'accesibilité et de la mémoire nécessaire pour l'exécution des algorithmes. Ces choix ne permettent pas toujours la transportabilité du prototype.

Représentation des comportements.

PROLOG-C offre des possibilités pour la déclaration d'opérateurs (priorité et associativité); cette possibilité est non négligeable pour la représentation des arbres d'opérateurs associés aux termes. En fait, la recherche de sous-arbres, nécessaire pour la construction des automates associés aux termes comme pour les transformations des termes finis, est réalisée par résolution.

Pour décider quelles sont les structures de données les mieux adaptées pour la représentation des automates et pour résoudre les problèmes liés à la gestion de la mémoire, nous rappelons que PROLOG-C dispose entre autres, des zones de

mémoire suivantes pour stocker les informations nécessaires pour l'exécution d'un programme:

- le "heap" contient les clauses du programe PROLOG,
- le "global stack" contient les termes construits pendant l'exécution,
- le "trail" contient la liste de variables à restaurer pendant le retour arrière,
- le "local stack" contient l'information de contrôle pendant l'exécution.

Les tailles de ces zones de mémoire, allouées de manière standart sur le VAX 790 est de 256 K, 256 K, 64 K et 128 K. En PROLOG-C, l'utilisateur peut augmenter la taille de ces zones.

Nous considérons que les automates peuvent être représentés en PROLOG-C:

- soit par des listes d'arcs,
- soit à l'aide de la base de données fournie par PROLOG-C,
- soit par des clauses chacune correspondant à un arc.

Nous discutons chacune de ces trois possibilités de représentations en tenant compte des critères donnés précédemment:

1.- *Listes d'arcs:* ces listes sont pasées comme parametres dans les appels des prédicats, donc enregistrées dans le global stack. Cette représentation a été écartée puisqu'elle pose deux problèmes:
   - lorsque l'automate est modifié, une copie de la liste est générée, ce qui produit une rapide saturation du "global stack";
   - les techniques pour la recherche d'information en listes, nécessaires pour accéder aux arcs de l'automate sont lentes et coûteuses en mémoire, puisque des copies de listes sont faites à chaque appel d'un prédicat utilisant la structure.

2.- *Base de données fournie par PROLOG-C:* les arcs de l'automate sont enrégistrés dans le "heap", chaque arc peut être un terme de la forme "etat*etiquette*etat" et il occupe environ 40 mots. La mémoire occupée par la représentation d'un arc peut être liberée lorsqu'on sort d'un prédicat contenant la commande "effacer un arc". Cette représentation des automates permet de laisser les

autres zones de travail libres pour leur utilisation par le système pour d'autres informations nécessaires pendant l'exécution. Le problème apparaît au moment où l'on veut enregistrer plusieurs automates; en effet, la zone "heap" peut être saturée rapidement.

3.- *Clauses:* cette solution a été retenue puisque l'accessibilité aux arcs de l'automate est "directe" (par utilisation de la résolution); les clauses sont rangées dans le "heap" et la quantité de mémoire nécessaire pour enregistrer un arc est de même ordre que dans la base de données; par contre, cette structure présente l'avantage de permettre la sauvegarde des automates hors de l'espace de travail d'une façon rapide en fichiers UNIX, grace à la notion d'environnement de travail que nous avons introduit. Le temps pour enregistrer les automates en fichiers UNIX et le temps pour les acceder est d'environ 100 arcs par seconde.

Un problème peut surgir quand on traite des automates de très grande taille: la zone "heap" peut être saturée. En effet, pendant la construction de l'automate associé à un terme, des clauses temporaires sont générées pour indiquer les termes déjà traités et les états correspondants de l'automate; ainsi quand on traite des termes produisant des automates de grande taille, la zone "heap" peut être saturée.

Choix des algorithmes.

Nous présentons dans cette partie, les choix des algorithmes sur la base des problèmes de la récursion en PROLOG-C et de la stratégie "depth first".

L'utilisation de la récursion en PROLOG-C pour décrire des schémas répétitifs peut conduire à épuiser la mémoire disponible dans le "local stack" lors d'un nombre considérable d'appels récursifs; pour cette raison, nous avons préféré l'utilisation de schémas itératifs basés sur le retour arrière plutôt que des appels récursifs. En effet, les schémas itératifs n'ont pas besoin d'enregistre les environnements d'appel dans le "local stack"; en plus, l'utilisation du rétour arrière dans les schémas itératifs permet de libérer de la mémoire utilisée par les termes construits pendant l'exécution (zone

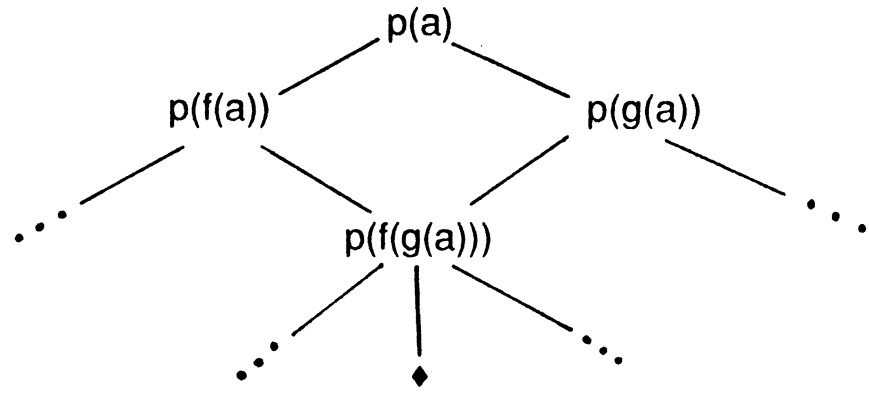
"global stack") à chaque fois que l'on fait un retour arrière.

PROLOG-C utilise "depth-first" comme stratégie de résolution. La stratégie "depth-first" consiste à choisir une branche et à l'explorer jusqu'à ce que la résolution aboutisse soit à échec, soit à succès; dans le cas d'échec, un retour arrière se produit et une autre branche est traitée. Le problème de cette stratégie est que les autres branches ne sont prises en compte qu'au moment de l'échec de la résolution; ainsi, il est possible de tomber sur une branche infinie (ou trop longue) épuisant les zones "local stack" et "trail" et ne conduisant pas à un succès alors qu'une autre branche aurait pu fournir la solution.

Considérons le programme [Af 85]:

```
p(f(g(a))).
p(X):-p(f(X)).
p(X):-p(g(X)).
```

et supposons que nous voulons prouver p(a). En utilisant la stratégie "depth-first" de PROLOG-C, le programme épuise la mémoire assignée au "local stack" et n'aboutit pas à une solution, alors que l'arbre de recherche suivant montre qu'une solution est obtenue après trois pas en utilisant une autre stratégie:



Ainsi, la réalisation d'un algorithme de bisimulation basé sur la stratégie "depht-first" et l'idée de retour arrière (i.e: celui proposé par Sanderson[Sa 82]) dans certains cas peut conduire à une rapide saturation du "local stack". Pour cette raison nous avons préféré de

réaliser un algorithme basé sur le calcul de point fixe; cet algorithme a le désavantage d'être plus complexe mais il permet de séparer en étapes la comparaison.

III.8.- Evaluation du prototype.

Dans cette partie nous présentons les possibilités et les limites du prototype réalisé; de plus nous discutons quelques problèmes pouvant apparaître lors de son utilisation et nous donnons les solutions envisagées.

Le prototype de VENUS réalisé en PROLOG-C est composé d'environ 4000 clauses occupant environ 200 K du "heap". Pour son utilisation il est recommandé d'étendre le "heap" de 256 K (quantité allouée automatiquement par PROLOG-C sur VAX 790) à au moins 300 K. Ceci permet d'enregistrer un automate composé d'environ 2000 arcs.

Les clauses utilisées pour la représentation des automates sont enregistrées dans le "heap"; ainsi pour traiter des termes produisant des automates de grande taille, il faut étendre cette zone (sur le VAX 790 dont nous disposons, il est possible d'allouer jusqu'à environ 3500 K).

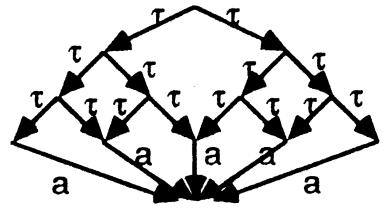
L'utilisation des environnements de travail permet de stocker une grande quantité de termes et d'automates en fichiers pour ne pas saturer la mémoire utilisée par le programme. La limite de fichiers ains créés est imposée par le système hôte (UNIX).

Les algorithmes utilisés pour la réalisation des transformations τ-chaînes, τ-circuits, saturation et minimisation sont assez performants comme témoignent les exemples présentés dans l'annexe 2. Par exemple, considérons l'automate arcs2 de l'annexe 2 comportant 68 états et 82 transitions; la forme normale pour l'équivalence observationnelle de cet automate comporte 12 états et 16 transitions et a été obtenue en moins de 2 minutes, ainsi:

- l'application des transformations τ-chaînes et τ-circuits élimine 44 états et 50 transitions en moins de 10 secondes;

- la saturation construit 28 transitions en environ 2 secondes;
- la relation de bisimulation pour cet exemple est obtenue en environ 80 secondes.

Il est important d'indiquer qu'ils existent des cas critiques, par exemple des automates contenant un grand nombre de τ-transitions de la forme suivante que nous appelons *araignées de τ:*



Dans ce cas, les τ-transitions ne peuvent pas être réduites par l'application des transformations τ-chaînes et τ-circuits (transformations pour lesquelles les algorithmes sont assez rapides). Donc la réduction doit être réalisée par bisimulation; dans ce cas, le nombre d'itérations pour le calcul de la relation de bisimulation est relativement grand et pour chaque état plusieurs comparaisons doivent être effectuées, et le temps nécessaire pour la normalisation dans certains cas peut représenter une limitation pour l'utilisation du système (ie: exemple arcs4 de l'annexe 2).

CONCLUSION.

L'objectif de ce travail a été la définition de méthodes de vérification de systèmes communicants décrits en CCS, et la conception d'un outil d'aide à la vérification.

Nous avons d'abord étudié le calcul CCS sans transmission de valeur. Pour les termes réguliers, nous avons défini des procédures de décision pour la congruence forte, l'équivalence et la congruence observationnelle. Nous avons aussi proposé des procédures de construction de formes normales pour ces équivalences. Ces procédures de décision sont basées sur des transformations et des comparaisons de systèmes de transitions. Pour les termes finis, nous avons donné une nouvelle caractérisation axiomatique de l'équivalence et de la congruence observationnelle en remplaçant un certain nombre d'axiomes ((A_3), (A_{17}), (A_{18})) par un axiome conditionnel (A_c). Ces nouvelles axiomatisations ont conduit à définir des procédures de décision basées sur la construction et la comparaison des formes normales des termes; ces formes normales sont construites par réécriture. Les résultats obtenus pour les termes finis ont été étendus à des termes contenant des variables libres; les procédures de décision pour l'équivalence et la congruence observationnelle ont été adaptés à ces termes.

A partir de ces résultats, nous avons conçu le système VENUS qui permet de réaliser des descriptions modulaires de comportements et possède des fonctionnalités pour réaliser aisément des vérifications des descriptions, pour enregistrer des définitions et des transformations de comportements dans des environnements de travail et pour la gestion des environnements de travail.

Les vérifications et les normalisations des systèmes pour les équivalences mentionnées précédemment peuvent être réalisées automatiquement ou sous le contrôle de l'utilisateur. En effet, VENUS fournit un ensemble de commandes permettant de transformer et

comparer des termes et des systèmes de transitions. Il est possible de définir des procédures de décision et de normalisation à partir de ces commandes pour d'autres relations d'équivalence.

Un prototype de VENUS a été écrit en PROLOG-C. Cette réalisation a permis de constater l'utilité des méthodes proposées pour la vérification et la normalisation des systèmes décrits en CCS. Nous avons pu vérifier une couche d'un protocole de communication défini par l'ISO OSI et minimiser des graphes temporisés significatifs correspondant à des termes de l'algèbre ATP [RSV 86] et produits par le générateur de graphes du système Xesar [Xe 86].

Les extensions de VENUS envisagées doivent permettre d'une part d'enrichir l'interface de communication, d'autre part d'étendre les possibilités de vérification offertes. Pour enrichir l'interface de communication de VENUS, il est intéressant d'introduire un langage de composition et de définition de commandes. L'utilisateur poura ainsi définir des suites de transformations, des procédures de preuve et de normalisation automatiques pour d'autres relations d'équivalence. Dans cette réalisation, nous nous sommes restreints à la comparaison de comportements décrits en CCS. Il est cependant possible de l'étendre à d'autres algèbres comme par exemple SCCS [Mi 83] et MEIJE [Bo 85].

REFERENCES

[Ab 79] Abrahamson K.
Modal Logic of concurrent non deterministic programs.
Semantics of Concurrent Computation Proceedings.
Lecture Notes in Computer Science 70. Springer-Verlag. 1979.

[AB 84] Austry D., Boudol G.
Algèbre de processus et synchronisation.
TCS 30 pp. 91-131.

[Af 85] groupe PROLOG de l'AFCET
PROLOG: FONDEMENTS ET APPLICATIONS
16ème Ecole Internationnale d'informatique de l'AFCET, 15-17 juillet 1985.

[AS 79] Abrial J. R., Schuman S. A.
Non deterministic system verification.
Lecture Notes in Computer Science 70. Springer-Verlag. 1979.

[BH 84] Brookes S. D., Hoare C. A. R. , Roscoe A. W.
Theory of communicating sequential processes.
JACM. 31 (3). Juillet 1984.

[BK 84] Bergstra J. A., Klop J. W.
Verification of an alternating bit protocol by means of process algebra.
Centre for Mathematics and Computers.
Departement of Computer Science.
Rapport CS-R8404. Mars 1984. Amsterdam.

[BK 85] Bergstra J. A., Klop J. W.
A complete inference system for regular processes with silent moves.
Centre for Mathematics and Computers.
Departement of Computer Science.
Rapport CS-R8420. Mars 1984. Amsterdam.

[Bo 85] Boudol G.
Le calcul MEIJE
Parallélisme, Communication et Synchronisation
Edité par J.P. Verjus et G. Roucairol
Editions du Centre National de la Recherche Scientifique.
1985. pp. 23-45

[CM 81] Clocksin, W. F., Mellish, C. S.
PROGRAMMING IN PROLOG
Springer-Verlag. 1981.

[Ga 87] Garavel, H.
Utilisation du système CESAR pour la vérification de protocoles spécifiés en LOTOS.
Rapport Technique. IMAG-LGI. 1987.

[Gi 62] Gill A.
Introduction to the theory of finite state machines.
Mc. Graw-Hill Book Company, Inc. 1962.

[HM 85] Hennessy M., Milner R.
Algebraic laws for nondeterminism and concurrency.
JACM. 32(1). pp. 137-161. Janvier 1985.

[Ko 85] Koomen C.J.
Algebraic specification and verification of communication protocols.
Science of Computer Programming 5. pp 1-36. 1985.

[MV 86] Madelaine E., Vergamini D.
Ecrins - Manuel d'utilisation.
Journées du Pole "Sémantique et Vérification" de C-Cube.
Grenoble, mars1986.

[Mi 80] Milner R.
A calculus of communicating systems.
Lecture Notes in Computer Science 92.
Springer-Verlag. 1980.

[Mi 83] Milner R.
Calculi for synchrony and asynchrony.
T.C.S. 25; pp. 267-310. 1983.

[Mi 85] Milner R.
A complete axiomatisation for observational congruence of finite state behaviors.
Manuscript non publié, décembre 1985.

[Mo 83] Montel B. et al.
OVIDE, a software package for the validation of systems represented by Petri net based models.
Computer Networks, December 1982.

[Na 86] Najm, E., Budkowski, S., Gilot, T., Lumbroso, L.
Generql presentation of SCAN.
A distributed systems modelling and validation tool.
Protocol, specification and verification. M. Diaz (editor)
Elsevier Science Publishers. B. V. (North Holland).
IFIP. 1986.

[Na 87] Najm E.
A verification oriented specification in LOTOS of the transport protocol.
Project ESPRIT/SEDOS/C2, Novembre 1986. Submitted to Distibuted Computing Systems, Berling, 1987.

[Pe 84] Pereira, F.
C-PROLOG USER'S MANUAL
SRI International, Menlo Park
California. 1984.

[Pl 81] Plotkin G.
A structural approach to operational semantics.
Aarhus University. Denmark.
DAIMI FN-19. Septembre 1981.

[Pn 77] Pnueli A.
The temporal logic of concurrent programs.
19 th. FOCS. 1977.

[QS 82] Queille J.P., Sifakis J.
Specification and verification of concurrent systems in CESAR.
Lecture Notes in Computer Science 137. Springer-Verlag 1982.

[Qu 82] Queille J.P.
Le système CESAR: description, spécification et analyse des applications réparties.
Thèse de Docteur Ingénieur.
Université Scientifique et Médicale de Grenoble. 1982.

[RSV 86] Richier J.L., Sifakis J., Voiron J.
ATP-An algebra for timed processes.
Rapport IMAG-LGI.
Octobre 1986.

[Sa 82] Sanderson M. T.
Proof Techniques for CCS.
Ph. D. Thesis CST- 19-82.
University of Edinburgh. Nov. 1982.

[Se 83] Sedgewick R.
Algorithms.
Addison Wesley Publishing Company. 1983.

[Sc 83] Schwartz, J.P.
QUASAR une réalisation du système CESAR.
Thèse de Docteur Ingénieur.
Université Scientifique et Médicale de Grenoble. 1983.

[Ta 81] Tanenbaum, A.
Networks Protocols.
Computing Surveys 13(4). 1981.

[Ve 86] Vergamini D.
Verification by means of observational equivalence on automata.
Journées du Pole "Sémantique et Vérification" de C-Cube.
Grenoble, mars1986.

[Xe 86] **Definition document of XESAR.**
Rapport de Recherche IMAG-LGI à paraitre.

Annexe 1: Manuel d'utilisation de VENUS.

Nous présentons une description générale de l'utilisation du système VENUS. Nous décrivons les facilités que VENUS met à la disposition des utilisateurs pour la conception et la vérification des systèmes communicants; ceci sans préciser les méthodes utilisées pour la construction de l'outil. Ce manuel comporte quatre parties:

- Une première partie est dédiée à la présentation des caractéristiques générales du système.
- Dans une deuxième partie nous décrivons comment l'invocation du système est faite.
- Dans une troisième partie nous présentons les commandes que VENUS met à la disposition des utilisateurs pour décrire et analyser des systèmes communicants, et nous donnons également un arbre de description de l'enchaînement des commandes.
- Une quatrième partie est constituée d'un schéma d'une session d'utilisation de VENUS.

1.- Caractéristiques générales.

VENUS est un système interactif conçu pour la conception et pour la vérification des systèmes communicants. Dans VENUS, une application répartie est définie dans un *environnement de travail*.

Toute définition et vérification d'un système communicant est faite dans l'environnement de travail que nous appelons *courant*. Cet environnement peut être sauvegardé pour une utilisation postérieure.

Dans l'environnement de travail courant il est possible de définir une application répartie à l'aide d'un langage de description (défini dans la partie II paragraphe I.1.) permettant de structurer l'application en *composantes*. Dans chaque composante, les *comportements* sont définis par des équations associant à chaque variable de comportement un terme de $T[\Sigma_2,V]$. Des nouveaux comportements peuvent aussi être construits par transformation

d'autres comportements.

VENUS dispose d'un ensemble de commandes permettant de gérer les environnements de travail définis par l'utilisateur.

Dans VENUS nous avons voulu disposer d'une notion unique de *comportement* permettant à l'utilisateur une grande flexibilité pour l'utilisation des concepts du calcul CCS et des transformations définies dans la première partie de cette thèse. Un comportement peut être représenté soit par un terme CCS, soit par un système de transitions. Pour comparer ou transformer des comportements, l'utilisateur n'est pas obligé de connaître leurs représentations, mais il peut utiliser explicitement chacune des représentations.

Pour permettre la vérification des systèmes communicants, des commandes pour la transformation et la comparaison des comportements sont définies dans VENUS. En particulier, il est possible de vérifier l'équivalence et la congruence observationnelle et la congruence forte.

Dans VENUS, nous appelons *comportement courant* le dernier comportement obtenu par utilisation d'une des commandes du système.

2.- Appel.

Pour pouvoir utiliser le système VENUS, il faut charger le module objet **venus** par de la commande UNIX ***prolog venus***. Une fois chargé ce module, l'environnement VENUS est démarré par la clause prolog ***commencer***.

Après un message de bienvenue, le système VENUS commence un dialogue permettant à l'utilisateur de consulter une notice d'utilisation du système. Le système affiche "--->" chaque fois qu'il attend une commande ou une réponse à une question.

3.- Les commandes utilisateur.

Dans cette partie, nous présentons l'ensemble de commandes disponibles dans VENUS pour décrire et pour analyser des systèmes communicants. Nous donnons tout d'abord la liste exhaustive des commandes reconnues par VENUS. Les commandes sont listées en ordre alphabétique, et pour chaque commande la syntaxe et une description sont données. Finalement, nous donnons un schéma indiquant l'enchaînement des commandes.

3.1.- Les commandes.

Annuler commande.

Commande: ***a***
format: *a*

Cette commande permet d'aller au niveau de commandes précédent. Son utilisation en cas d'erreurs utilisateur au moment de l'exécution de la commande **me**, permet de venir en arrière en réinstallant l'environnement de travail existant juste avant l'appel de la commande qui a produit l'erreur.

Construire un terme sous la forme d'une somme.

Commande: ***add***
format: *add[option](<comportement>)*

La commande *add* construit le terme écrit sous la forme d'une somme décrivant le comportement identifié par *<comportement>* dans l'environnement courant.
Options:

a	construction du systèmes de transitions.
r	construction d'un terme.

Renommer un comportement.

Commande: ***alias***
format: *alias(<comportement1>,<comportement2>)*

<comportement1> identifie un comportement dans l'environnement courant. *<comportement2>* est un idéntificateur non utilisé dans l'environnement de travail courant. La commande *alias* associe l'identificateur *<comportement2>* au comportement identifié par *<comportement1>.*

Calculer une bisimulation.

Commande: ***bis***
format: *bis(<comportement1>,<comportement2>)*

La commande *bis* calcule la plus grande relation de bisimulation entre les comportements identifiés par *<comportement1>* et *<comportement2>* dans l'environnement courant.

Charger un comportement.

Commande: ***chc***
format: *chc(<comportement>)*

La commande *chc* détruit le comportement courant et charge le comportement identifié par *<comportement>* comme comportement courant dans l'environnement courant.

Charger un environnement.

Commande: ***che***
format: *che(<environnement>)*

La commande *che* fait que l'environnement de travail *<environnement>* soit chargé comme environnement courant, en détruisant l'environnement courant actuel.

Décider de la congruence forte.

Commande: ***cf***

format: *cf(<comportement1>,<comportement2>)*

La commande *cf* décide de la congruence forte des comportements identifiés par *<comportement1>* et *<comportement2>* dans l'environnement courant. Le système interroge l'utilisateur pour afficher ou non des résultats intermédiaires obtenus pendant la vérification.

Décider de la congruence observationnelle.

Commande: ***co***
format: *co[option](<comportement1>,<comportement2>)*

La commande *co* décide de la congruence observationnelle des comportements identifiés par *<comportement1>* et *<comportement2>* dans l'environnement courant.
Options:

a la méthode à utiliser est basée sur la construction de systèmes de transitions.

r la méthode à utiliser est basée sur la réécriture de termes.

Le système interroge l'utilisateur pour afficher ou non des résultats intermédiaires obtenus pendant la vérification.

Définir l'environnement courant.

Commande: ***cre***
format: *cre(<éditeur>)*

Le paramètre *éditeur* désigne un éditeur de texte présente sur UNIX, c'est-à-dire: *vi, ed* ou *ex*. La commande *cre* permet la définition d'un nouveau environnement courant de travail à l'aide de l'éditeur *<éditeur>*.

La définition de l'environnement de travail est faite en utilisant le langage défini dans I.1. Une fois finie la définition du nouveau environnement de travail, le système fait l'analyse lexicographique et syntaxique de la définition d'environnement en

émettant des messages d'érreur et des avertissements s'il y a lieu.

Dans le cas où il y a des erreurs de syntaxe, il est possible soit d'annuler cette création en restaurant l'environnement de travail précédent, soit de modifier la définition faite.

Si il n'y a pas d'erreurs, le système détruit la définition précédente de l'environnement courant en la substituant par la nouvelle définition.

Détruire un comportement.

Commande: ***dc***
format: *dc(<comportement>)*

La commande *dc* détruit le comportement identifié par *<comportement>* dans l'environnement courant.

Détruire le comportement courant.

Commande: ***dcc***
format: *dcc*

La commande *dcc* détruit le comportement courant de l'environnement courant.

Détruire un environnement de travail.

Commande: ***de***
format: *de(<environnement>)*

La commande *de* détruit l'environnement de travail designé par *<environnement>*.

Ecrire une relation de bisimulation.

Commande: ***eb***
format: *eb*

Après du calcul d'une bisimulation (commande *bis* ou *rbis*), un appel de la commande *eb* produit l'affichage sur l'écran de la

relation de bisimulation calculée.

Ecrire un comportement.

Commande: ***ec***
format: *ec(<comportement>)*

La commande *ec* affiche sur l'écran le comportement désigné par *<comportement>* dans l'environnement courant.

Ecrire la définition d'un environnement.

Commande: ***ee***
format: *ee(<environnement>)*

La commande *ee* affiche sur l'écran la définition de l'environnement *<environnement>.*

Ecrire la définition de l'environnement courant.

Commande: ***eec***
format: *eec*

La commande *eec* affiche sur l'écran la définition de l'environnement courant.

Etiquetter certaines τ-transitions

Commande: etiquettage
format: *etiquettage(ensemble d'actions)*

Cette commande étiquette chaque τ-transition par "t-" suivi du nom de l'action qui l'a produit, si l'action appartient à un ensemble donné.

Décider de l'équivalence observationnelle.

Commande: ***eo***
format: *eo[option](<comportement1>,<comportement2>)*

La commande *eo* décide de l'équivalence observationnelle des comportements identifiés par *<comportement1>* et *<comportement2>* dans l'environnement courant.
Options:

a la méthode à utiliser est basée sur la construction de systèmes de transitions.

r la méthode à utiliser est basée sur la réécriture de termes.

Le système interroge l'utilisateur pour afficher ou non des résultats intermédiaires obtenus pendant la vérification.

Fin de session.

Commande: ***fin***
format: *fin*

La commande *fin* termine la session sur VENUS en retournant le contrôl au système d'exploitation UNIX.

Help.

Commande: ***h***
format: *h*

La commande *h* donne des informations sur l'utilisation du système. L'information donnée est associée à la situation particuliaire dans laquelle le système se trouve au moment de la commande.

Lister les noms des comportements.

Commande: ***lc***
format: *lc*

La commande *lc* affiche les noms des comportements existant dans l'environnement courant.

Lister les noms des environnements.

Commande: ***le***

format: *le*

La commande *le* affiche sur l'écran tous les noms d'environnements existant.

Lister les variables decomportement déclarées dans l'environnement courant.

Commande: ***lv***
format: *lv*

La commande *lv* affiche sur l'écran les variables de comportement déclarées dans l'environnement courant.

Modifier la définition de l'environnement courant.

Commande: ***me***
format: ***me****(<éditeur>)*

Le paramètre *éditeur* désigne un éditeur de texte présent sous UNIX, c'est-à-dire: *vi, ed* et *ex.* La commande *me* permet de modifier la définition de l'environnement courant. Cette modication est faite à l'aide de l'éditeur *<éditeur>.*

La modification de l'environnement courant est faite en respectant la syntaxe et sa sémantique du langage défini dans la partie I, paragraphe I.1. Une fois finies les modifications, le système analyse la nouvelle définition de l'environnement courant, en émettant des messages d'erreur et des avertissements s'il y a lieu.

Dans le cas où il y a des erreurs de syntaxe, il est possible soit d'annuler cette modification en restaurant l'environnement de travail précédent, soit de modifier la définition.

Si il n'y a pas d'erreur, le système détruit la définition précédente de l'environnement courant en lui substituant par la nouvelle définition, le système détruit aussi tout comportement qui

avait été sauve-gardé et qui est affecté par les modifications faites sur la définition de l'environnement.

Construire une forme normale pour la congruence forte.

Commande: ncf
format: *ncf(<comportement>)*

La commande *neo* construit la forme normale pour la congruence forte du comportement identifié par*<comportement>* dans la définition de l'environnement courant. Le système interroge l'utilisateur pour afficher ou non des résultats intermédiaires obtenus pendant la vérification.

Construire une forme normale pour la congruence observationnelle.

Commande: *nco*
format: *nco[option](<comportement>)*

La commande *neo* construit la forme normale pour la congruence observationnelle du comportement désigné par *<comportement>* dans la définition de l'environnement courant.
Options:

a la méthode à utiliser est basée sur la construction de systèmes de transitions.

r la méthode à utiliser est basée sur la réécriture de termes.

Le système interroge l'utilisateur pour afficher ou non des résultats intermédiaires obtenus pendant la vérification.

Construire une forme normale pour l'équivalence observationnelle.

Commande: *neo*
format: *neo[option](<comportement>)*

La commande *neo* construit la forme normale pour l'équivalence observationnelle du comportement désigné

par*<comportement>* dans la définition de l'environnement courant.
Options:

a la méthode à utiliser est basée sur la construction de systèmes de transitions.

r la méthode à utiliser est basée sur la réécriture de termes.

Le système interroge l'utilisateur pour afficher ou non des résultats intermédiaires obtenus pendant la vérification.

Réduire à un nil unique.

Commande: ***nu***
format: *nu*

La commande *nu* construit le quotient pour la classe de congruence observationnelle de [nil] de l'automate courant.

Calculer une relation de r-bisimulation

Commande: ***rbis***
format: *rbis(<comportement1>,<comportement2>)*

La commande *rbis* calcule la plus grande relation de r-bisimulation entre les comportements désignés par *<comportement1>* et *<comportement2>* dans l'environnement courant.

Construire le quotient pour la congruence forte.

Commande: ***recf***
format: *recf*

La commande *recf* construit le quotient pour la congruence forte de l'automate courant, en utilisant la relation de bisimulation precedemment calculée.

Construire le quotient pour la congruence observationnelle.

Commande: ***reco***
Format: *reco*

La commande *reco* construit le quotient pour la congruence observationnelle de l'automate courant, en utilisant la relation de bisimulation precedemment calculée.

Construire le quotient pour l'équivalence observationnelle

Commande: ***reeo***
format: *reeo*

La commande *reeo* construit le quotient pour l'équivalence observationnelle du comportement icourant.

Rendre réflexive la relation $\xrightarrow{\tau}$>.

Commande: ***rg***
format: *rg*

La commande *rg* rend réflexive la relation $\xrightarrow{\tau}$> pour l'automate courant.

Rendre réflexive la relation $\xrightarrow{\tau}$> sauf pour l'état initial..

Commande: ***rst***
format: *rst*

La commande rend réflexive la relation $\xrightarrow{\tau}$> sauf pour l'état initial de l'automate courant.

Réduire observationnellement les transitions.

Commande: ***rt***
format: *rt*

La commande *rt* réduit les transitions $\xrightarrow{\alpha}$ pouvant être obtenues par des dérivations de la forme $\xrightarrow{\tau+\alpha\ \tau^*}$ ou $\xrightarrow{\tau^*\alpha\tau+}$ dans l'automate courant.

Saturation.

Commande: ***sat***
format: *sat*

La commande *sat* calcule la fermeture transitive de la relation $\xrightarrow{\tau}$ de l'automate courant.

Sauvegarder un comportement.

Commande: ***sc***
format: *sc*

La commande *sc* sauvegarde le comportement courant.

Sauvegarder un environnement.

Commande: ***se***
format: *se(<environnement>)*

La commande *se* sauvegarde l'environnement courant sous le nom *<environnement>.*

Construire un système d'équations.

Commande: ***syseq***
format: *syseq*

La commande *syseq* construit un système d'équations décrivant le comportement courant.

Simplifier les τ pour la congruence observationnelle.

Commande: *tsco*
format: *tsco*

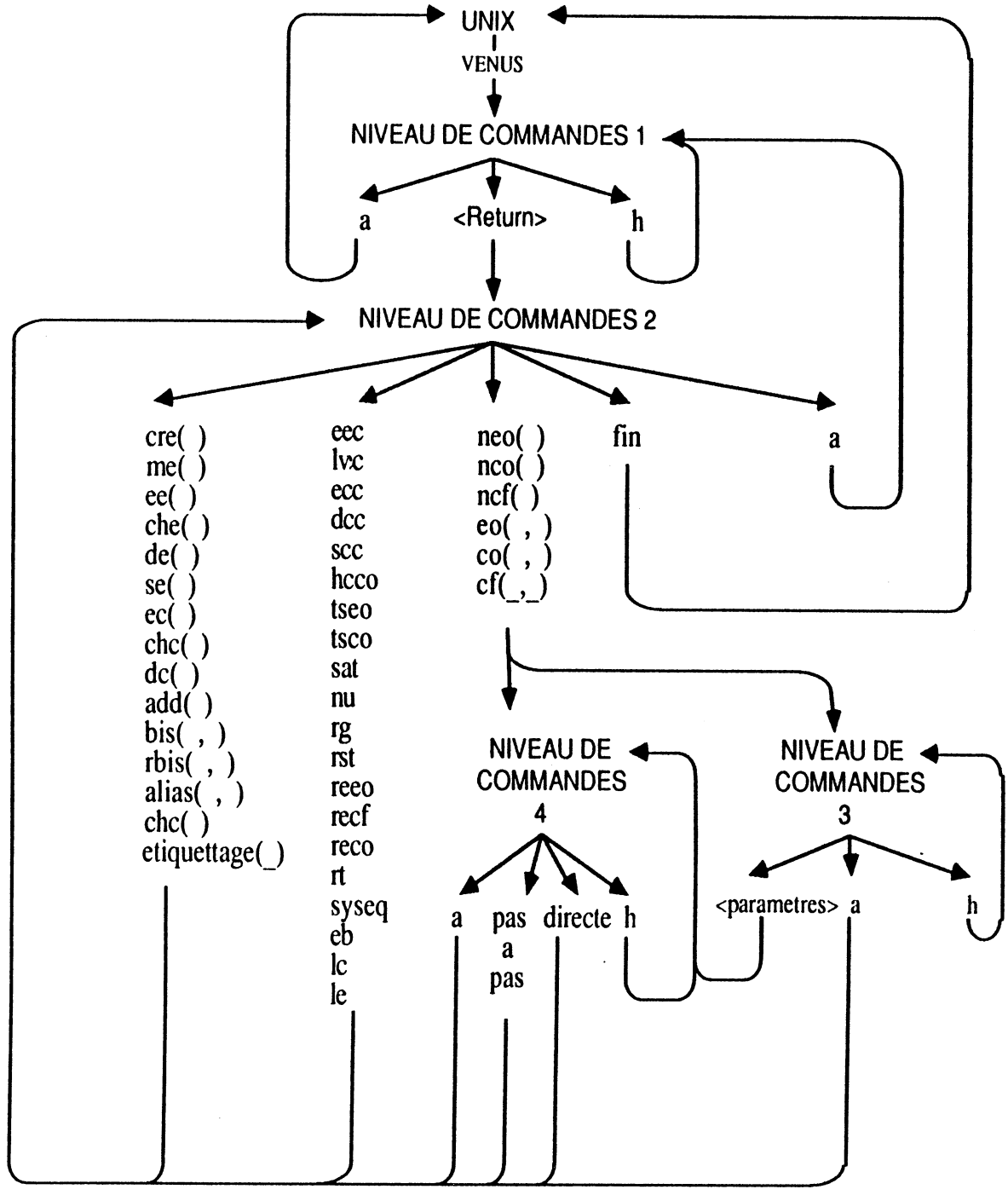
La commande *tsco* fait les τ-simplifications possibles pour la congruence observationnelle sur le comportement courant.

Simplifier les τ pour l'équivalence observationnelle.

Commande: *tseo*
format: *tseo*

La commande *tseo* fait les τ-simplifications possibles pour l'équivalence observationnelle sur le comportement courant.

3.2.- Arbre d'enchaînement des commandes.



3.3.- Commandes groupées par fonctionnalité.

Commandes générales:

h help,
a annuler commande,
fin sortir de VENUS.

Commandes relatives à la définition de l'environnement de travail:

che(_) charger un environnement,
cre(_) creer un environnement,
de(_) détruire un environnement,
ee(_) écrire la définition d'un environnement,
eec écrire la définition de l'environnement courant,
etiquettage(_) etiquetter certaines t-transitions,
lc lister les noms des comportements construits dans l'environnement courant,
le lister les noms des environnements,
lv lister les variables de comportement définies dans l'environnement courant,
me(_) modifier la définition de l'environnement courant,
se(_) sauve-garder l'environnement courant.

Commandes relatives aux comportements dans l'environnement courant:

alias(_,_) renomer un comportement,
chc(_) lire un comportement,
dc(_) detruire un comportement,
dcc detruire le comportement courant,
ec(_) ecrire un comportement,
ecc ecrire le comportement courant,
sc(_) sauve-garder le comportement courant.

Commandes relatives à la construction de formes normales de comportements donnés dans la définition de l'environnement courant:

ncf(_) pour la congruence forte,
nco(_) pour la congruence observationnelle,
neo(_) pour l'équivalence observationnelle.

Commandes relatives à la vérification de l'équivalence de termes donnés dans la définition de l'environnement courant:

cf(_,_) vérifier la congruence forte,
co(_,_) vérifier la congruence observationnelle,
eo(_,_) vérifier l'équivalence observationnelle.

Commandes relatives à des transformations sur les comportements:

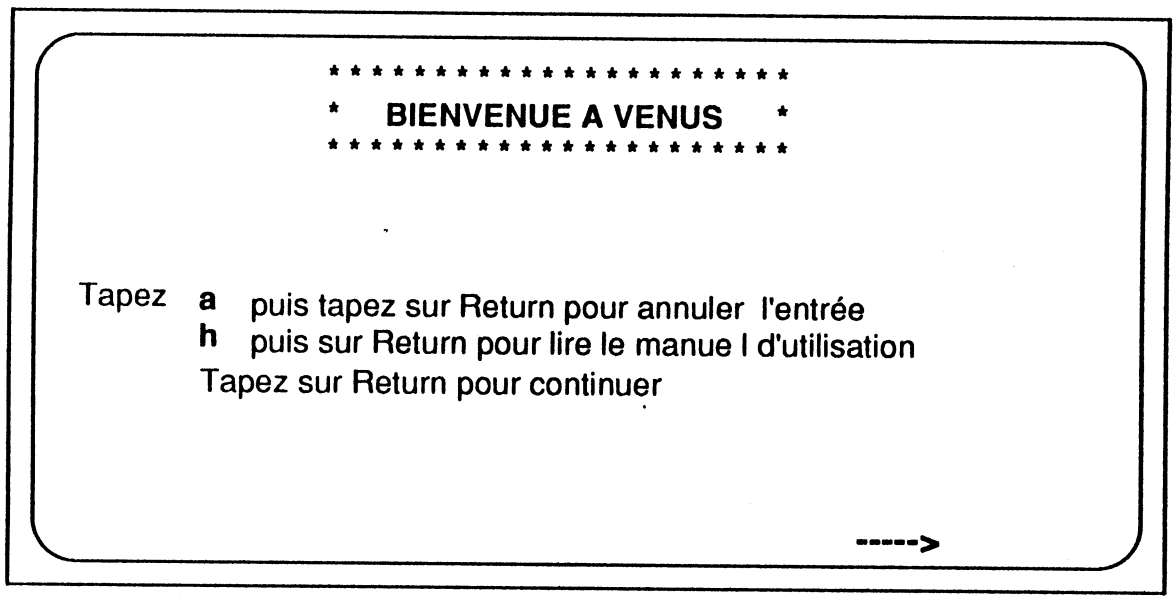
add(_) construire le comportement sous la forme d'une somme,
bis(_,_) construire la plus grande relation de bisimulation entre deux comportements,
eb ecrire la relation de bisimulation calculée,
nu nil unique,
rbis(_,_) construire la plus grande relation de r-bisimulation entre deux comportements,
recf construire le quotient pour la congruence forte, tennant compote de la relation de bisimulation calculée,
reco construire le quotient pour la congruence observationnelle, tennant compote de la relation de bisimulation calculée,
reeo construire le quotient pour l'équivalence observationnelle, tennant compote de la relation de bisimulation calculée,
rg rendre réflexive la t-relation,
rst rendre reflexive la t-relation pour tous les états sauf pour l'état initial,

rt réduire observationnellement les transitions,
sat saturation,
syseq construire un systeme d'équations décrivant le comportement courant,
tseo simplifier les "t" pour l'équivalence observationnelle,
tsco simplifier les "t" pour la congruence observationnelle.

4.- Schéma de session.

Nous présentons dans cette partie le schéma d'une session dans VENUS. Cette présentation est faite sous l'œil de l'utilisateur, c'est-à-dire en intercalant des dessins représentant l'écran à un instant donné.

Tout d'abord, nous faisons l'invocation du système en tapant la commande **prolog venus** puis **commencer**, le système VENUS demarre sur le niveau de commandes 1, en affichant l'écran suivant:



1) Dans le premier cas (réponse est **A** suivi de **Return)**, nous terminons la session et retournons au système UNIX .

2) Dans le deuxième cas (réponse est **H** suivi de **Return)**, une partie de ce manuel d'utilisation apparaît sur le terminal. A la fin de cette séquence d'écrans, le niveau de commandes 1 est affiché a nouveau. La présentation du manuel d'utilisation est faite comme une suite d'écrans controlée par la commande **more** de UNIX. C'est-à-dire, nous voyons sur le terminal des écrans de la forme:

MANUEL D'UTILISATION

VENUS est un système interactif d'aide à la conception et à la vérification des systèmes communicants. Dans VENUS, une application répartie est faite dans un environnement de travail.

Toute définition et vérification d'un système communicant est faite dans l'environnement de travail que nous appelons "courant". Cet environnement peut être sauvegardé pour une utilisation postérieure. ...

MORE

3) Dans le troisième cas (réponse est **Return**), nous entrons au niveau de commandes 2 et le message suivante est affiché sur l'écran:

Tapez une commande ----->

Dans ce niveau de commandes il est possible d'utiliser la commande **a** pour aller au niveau de commandes 1, la commande **fin** pour finir la session sur VENUS et rentrer au système UNIX ou bien taper une des commandes du système pour la gestion des environnements de travail ou pour la vérification de définitions de comportements. Une fois réalisée une commande le système émet les messages:

Temps total pour la commande: XX.XXX
Tapez une commande ----->

Le temps de la commande est exprimé en secondes.

4) Dans le cas où l'utilisateur tape une autre séquence de caractères suivie de Return, le système réagit comme dans le deuxième cas.

A titre d'exemple, nous vérifions par la suite, que la définition du protocole de communication et sa spécifications données dans la partie I de cette thèse (chapitre III), sont observationnellement congrues.

Nous faisons d'abord l'appel à VENUS:

```
(1) prolog venus
    commencer.
```

```
**************************
*   BIENVENUE A VENUS    *
**************************

Tapez  a  puis tapez sur Return pour annuler l'entrée
       h  puis sur Return pour lire le manuel d'utilisation
       Tapez sur Return pour continuer

                                          ----->
```

```
Tapez une commande ---->
```

Annexe 1: Manuel d'utilisation de **VENUS**

Pour réaliser la vérification, nous construisons un environnement de travail contenant la définition et la spécification du protocole du bit alterné et posterieurement nous verifions la congruence observationnelle entre les termes décrivant le protocole et sa spécification.

Nous construisons l'environnement de travail en utilisant la commande **cre** et à l'aide de l'éditeur de textes **vi**:

```
composante bit-alterné
 comportement = ba, s;
 action = do, d1, tdo, td1, tde, ao, a1, tao, ta1, tae, input, output
definitions
 ba = (em in émetteur & r in récepteur & m1 in médium1 &
  m2 in médium2) \[do,d1,tdo,td1,tde,ao,a1,tao,ta1,tae]
fin-definitions
equations
 s = input output s
fin-equations
fin-composante
.
.
.

composante médium2
 comportement = m2;
 actions = ao, a1, tao, ta1, tae,t
equations
 m2= ao (t ntao m2 + t ntae m2) + a1 (t nta1 m2 + t ntae m2)
fin-equations
fin-composante



 "environnement" [New File]
```

```
Temps total pour la commande: 12.4333

Tapez une commande ---->
```

Nous verifions la congruence observationnelle entre les terme ba et s défini s dans la composante bit_alterne:

```
Temps total pour la commande: 12.4333

Tapez une commande ---->        co(ba in bit_alterne,s in bit_alterne)
```

```
Temps total pour la commande: 12.4333

Tapez une commande ---->        co(ba in bit_alterne,s in bit–alterne)

        OUI,    les termes sont observationnellement congrus.

Temps total pour la commande: 20.3833

Tapez une commande ---->
```

Si la commande **fin** est tapée, se produit la sortie de VENUS et nous pasons sur UNIX.

Annexe 2: Automate arcs2

```
etat_initial( a1, 1).
a1( 1, chi, 1).
a1( 1, ins, 2).
a1( 2, t, 3).
a1( 3, t, 4).
a1( 4, t, 5).
a1( 5, chi, 6).
a1( 6, t, 7).
a1( 6, t, 8).
a1( 7, t, 9).
a1( 8, chi, 10).
a1( 9, t, 11).
a1( 10, chi, 12).
a1( 11, t, 13).
a1( 12, t, 2).
a1( 13, out, 14).
a1( 14, t, 15).
a1( 14, t, 16).
a1( 15, t, 17).
a1( 16, t, 17).
a1( 16, t, 18).
a1( 17, t, 19).
a1( 18, t, 19).
a1( 18, t, 20).
a1( 19, t, 21).
a1( 20, t, 21).
a1( 21, chi, 22).
a1( 22, t, 23).
a1( 22, t, 24).
a1( 23, t, 25).
a1( 24, chi, 26).
a1( 25, t, 27).
a1( 26, t, 28).
a1( 27, chi, 27).
a1( 27, ins, 29).
a1( 28, t, 30).
a1( 29, t, 31).
a1( 30, t, 32).
a1( 31, t, 33).
a1( 32, t, 34).
a1( 33, t, 35).
a1( 34, chi, 36).
a1( 35, chi, 37).
a1( 36, t, 38).
a1( 36, t, 39).
a1( 37, t, 40).
a1( 37, t, 41).
a1( 38, t, 42).
a1( 39, chi, 24).
a1( 40, t, 43).
a1( 41, chi, 44).
a1( 42, t, 15).
a1( 43, t, 45).
a1( 44, chi, 46).
a1( 45, t, 47).
a1( 46, t, 29).
a1( 47, out, 48).
a1( 48, t, 49).
a1( 48, t, 50).
a1( 49, t, 51).
a1( 50, t, 51).
a1( 50, t, 52).
a1( 51, t, 53).
a1( 52, t, 53).
a1( 52, t, 54).
a1( 53, t, 55).
a1( 54, t, 55).
a1( 55, chi, 56).
a1( 56, t, 57).
a1( 56, t, 58).
a1( 57, t, 59).
a1( 58, chi, 60).
a1( 59, t, 1).
a1( 60, t, 61).
a1( 61, t, 62).
a1( 62, t, 63).
a1( 63, t, 64).
a1( 64, chi, 65).
a1( 65, t, 66).
a1( 65, t, 67).
a1( 66, t, 68).
a1( 67, chi, 58).
a1( 68, t, 49).
```

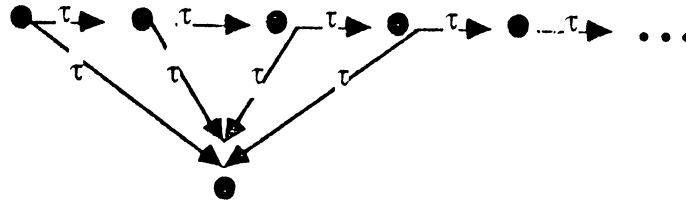
Forme normale de l'automate arcs2 pour l'équivalence observationnelle:

```
a1(10,chi,12).
a1(12,chi,6).
a1(14,chi,22).
a1(22,t,24).
a1(22,t,57).
a1(24,chi,26).
a1(26,chi,36).
a1(36,t,14).
a1(36,t,39).
a1(39,chi,24).
a1(57,chi,57).
a1(57,ins,12).
a1(6,t,7).
a1(6,t,8).
a1(7,out,14).
a1(8,chi,10).
etat_initial(a1,57).
```

Automate arcs4

```
al(10,chi,13).
al(100,t,104).
al(100,t,156).
al(102,t,106).
al(102,t,153).
al(104,t,108).
al(104,t,156).
al(106,t,110).
al(106,t,153).
al(108,t,112).
al(108,t,156).
al(11,out,23).
al(110,t,113).
al(110,t,115).
al(110,t,116).
al(112,t,118).
al(112,t,156).
al(113,t,116).
al(113,t,135).
al(114,t,125).
al(114,t,135).
al(115,t,122).
al(115,t,135).
al(116,chi,125).
al(118,t,114).
al(118,t,124).
al(118,t,125).
al(122,t,129).
al(122,t,135).
al(123,t,132).
al(123,t,135).
al(124,t,131).
al(124,t,135).
al(125,chi,132).
al(129,in,134).
al(129,t,135).
al(13,chi,16).
al(131,t,129).
al(131,t,135).
al(132,chi,136).
al(134,t,137).
al(134,t,19).
al(135,chi,135).
al(135,in,19).
al(136,chi,139).
al(137,t,140).
al(137,t,19).
al(139,chi,151).
al(139,t,150).
al(140,t,142).
al(140,t,19).
al(142,t,144).
al(142,t,145).
al(142,t,19).
al(144,t,147).
al(144,t,152).
al(145,chi,10).
al(147,t,149).
al(147,t,152).
al(149,t,152).
al(150,t,116).
al(150,t,153).
al(151,t,125).
al(151,t,156).
al(152,out,18).
al(153,chi,114).
al(153,t,113).
al(156,chi,133).
al(156,t,114).
al(16,chi,19).
al(18,t,22).
al(18,t,82).
al(19,chi,8).
al(19,t,145).
al(19,t,152).
al(22,t,25).
al(22,t,82).
al(23,t,27).
al(23,t,89).
al(25,t,29).
al(25,t,82).
al(27,t,31).
al(27,t,89).
al(29,t,33).
al(29,t,34).
al(29,t,82).
al(31,t,36).
al(31,t,89).
al(33,t,40).
al(33,t,48).
al(34,chi,43).
al(36,t,42).
al(36,t,43).
al(36,t,89).
al(39,t,43).
al(39,t,48).
al(40,t,47).
al(40,t,48).
al(42,t,48).
al(42,t,50).
al(43,chi,52).
al(47,in,54).
al(47,t,48).
al(48,chi,48).
al(48,in,96).
al(49,t,48).
al(49,t,52).
al(50,t,47).
al(50,t,48).
al(52,chi,57).
al(54,t,58).
al(54,t,96).
al(57,chi,61).
al(58,t,62).
al(58,t,96).
al(61,chi,83).
al(61,t,34).
al(61,t,82).
al(62,t,65).
al(62,t,96).
al(65,t,68).
al(65,t,96).
al(68,t,69).
al(68,t,72).
al(68,t,73).
al(69,t,73).
al(69,t,90).
al(70,t,80).
al(70,t,81).
al(72,t,78).
al(72,t,90).
al(73,chi,80).
al(78,t,84).
al(78,t,90).
al(8,t,10).
al(8,t,11).
al(80,chi,86).
al(81,out,100).
al(82,chi,39).
al(82,t,34).
al(82,t,48).
al(83,t,43).
al(83,t,89).
al(84,t,90).
al(86,chi,92).
al(89,chi,49).
al(89,t,43).
al(89,t,48).
al(90,out,95).
al(92,chi,96).
al(95,t,153).
al(95,t,99).
al(96,chi,70).
al(96,t,69).
al(99,t,102).
al(99,t,153).
etat_initial(al,135).
```

Schéma d'araignée:



VENUS: UN OUTIL D'AIDE A LA VERIFICATION DES SYSTEMES COMMUNICANTS

RESUME

VENUS est un outil d'aide à la conception et à la vérification de systèmes communicants. Il est basé sur le calcul CCS de Milner et permet de normaliser et de vérifier des descriptions de comportements réguliers en tenant compte de la congruence forte et de l'équivalence et de la congruence observationnelle définies dans CCS. Nous présentons dans ce travail les bases théoriques de VENUS, une réalisation en PROLOG et quelques exemples de vérification de protocoles de communication.

MOTS CLES:

Parallelisme, Vérification de Systemes Communicants, CCS, Protocole de Communication, PROLOG.

VENUS: AN ENVIRONNEMENT TO SPECIFY AND VERIFY COMMUNICATING SYSTEMS.

ABSTRACT

An environnement to specify and verify communicating systems is presented. This environnement (called VENUS) is based on the Milner's CCS calculus. VENUS allows the normalisation and verification of communicating systems having a regular behaviour. It is realized using CCS strong congruence and the observation equivalence and congruence. This work describes both the theoretical bases of VENUS and its implementation in PROLOG. Some protocol verification exemples are also presented.

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974

VU le rapport de présentation de Monsieur VOIRON, Maître de conférences

Madame SORIANO MONTES Amelia

est autorisée à présenter une thèse en soutenance en vue de l'obtention du titre de DOCTEUR de TROISIEME CYCLE, spécialité "Informatique".

Fait à Grenoble, le 19 décembre 1986

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,