ANDRÉ CARON

TRANSFORMATION D'EXPRESSIONS RELATIONNELLES SANS VARIABLES

Mémoire
présenté
à la Faculté des études supérieures
de l'Université Laval
pour l'obtention
du grade de maître ès sciences (M. Sc.)

Département d'informatique
FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL

JUILLET 1997

Résumé

Ce mémoire porte sur l'automatisation de la transformation d'expressions relationnelles sans variables. De telles expressions sont utilisées dans le calcul de programmes à partir de spécifications relationnelles. Dans le cas particulier de la construction de programmes parallèles, de tels calculs impliquent la manipulation de matrices d'expressions relationnelles sans variables.

L'approche que nous proposons consiste à appliquer les transformations lors de la construction des expressions. Nous avons nommé cette approche « la transformation constructive ». Il s'agit d'effectuer certaines simplifications à chaque fois que l'on exécute une opération. Cette approche minimise la longueur des expressions résultantes et, de ce fait, améliore les performances d'un éventuel logiciel de réduction d'expressions algébriques.

Il existe aujourd'hui une multitude de produits capables d'effectuer des manipulations algébriques. Cela va des applications mathématiques sophistiquées aux langages spécialisés en passant par divers produits intermédiaires. Cependant, aucun des logiciels étudiés n'offre la possibilité d'effectuer les transformations souhaitées sur des matrices d'expressions relationnelles sans variables. Un prototype a donc été développé à l'aide du logiciel *Scilab*. Le prototype a fourni les résultats attendus et ouvre la voie au développement d'une application qui pourrait intégrer l'ensemble des règles de simplification.

Jules Desharnais
Directeur de recherche

André Caron
Étudiant gradué

Avant-propos

Je désire d'abord remercier mon directeur de recherche, Monsieur Jules Desharnais, qui a su m'encourager et me motiver dans l'avancement de mes travaux. J'ai grandement apprécié sa disponibilité, sa facilité à partager ses connaissances et la pertinence de ses remarques.

Je tiens aussi à remercier Ridha Khédri, étudiant au doctorat, qui m'a fourni les documents préliminaires à ses recherches sur la composition parallèle.

J'aimerais également exprimer ma reconnaissance à Monsieur Brahim Chaib-draa et Monsieur Mourad Debbabi qui ont accepté d'évaluer ce mémoire.

Enfin, je désire remercier mes proches pour leur patience et leur appui constant.

Table des matières

Chapitre 1

Introduction

1.1 Le contexte

Dès ses débuts, l'informatique s'est heurtée au problème de l'exactitude des programmes. Les seules méthodes connues et utilisées à l'époque étaient basées sur la création de bancs d'essai que l'on espérait complets. L'augmentation de la complexité et de l'envergure des programmes fit rapidement ressortir la faiblesse des méthodes empiriques. L'avènement de nouvelles technologies telle le parallélisme n'a fait que rendre plus évidente encore cette faiblesse.

De nos jours, l'ordinateur étant présent dans toutes les sphères d'activités de notre société, les erreurs dans les logiciels peuvent avoir des conséquences monétaires, sociales, politiques et humaines. Cependant, malgré le développement phénoménal qu'a connu l'informatique, l'exactitude des programmes mis sur le marché n'est vérifiée qu'à l'aide de méthodes ou d'outils empiriques. D'ailleurs, l'incapacité à assurer des logiciels exempts d'erreurs est presque devenue la tare de l'industrie du logiciel. L'expression « un logiciel vient toujours avec des bogues » est considérée par plusieurs comme une incontournable évidence.

La recherche de méthodes formelles permettant d'assurer l'exactitude des programmes est aujourd'hui un enjeu primordial [60]. La recherche sur ce sujet est depuis quelques années déjà un des domaines les plus actifs du génie logiciel. Ce sont les approches mathématiques et logiques qui ont actuellement la faveur des chercheurs. En utilisant l'outil mathématique, les chercheurs héritent de tout un bagage de lois et de règles connues et démontrées.

Les méthodes formelles actuelles introduisent le formalisme dès le début du cycle de création d'un programme, *i.e.* lors de la spécification du problème. Une caractéristique très recherchée est la possibilité de pouvoir exécuter directement ces spécifications en utilisant l'ordinateur pour les manipuler. Diverses manipulations algorithmiques deviennent alors possibles, allant de la vérification de la cohérence des spécifications jusqu'au prototypage du système pour validation avec le client.

1.2 Délimitation du propos

Un des outils mathématiques utilisés par les chercheurs ces dernières années est l'algèbre des relations binaires (de type Tarski) qui prend son origine de la théorie des ensembles. Les travaux de Schmidt et Ströhlein [52], Mili *et al.* [44, 45], ainsi que ceux de Desharnais *et al.* [12, 13, 14, 16, 17] introduisent l'algèbre des relations comme outil de spécification de programmes.

Il est possible d'écrire des spécifications relationnelles sans utiliser de variables puisque l'algèbre relationnelle permet d'exprimer certaines propriétés des relations par des formules qui n'utilisent pas de variables de niveau objet (i.e. des variables qui font référence aux paires qui appartiennent à la relation). Par exemple, une relation R est transitive si et seulement si

$$R^2 \subseteq R$$

La caractérisation conventionnelle de cette propriété est

$$\forall s : \forall s' : \forall s'' : (s, s') \in R \wedge (s', s'') \in R \Rightarrow (s, s'') \in R$$

(dans cette dernière formule, s, s' et s'' sont les variables).

Il est ainsi possible de définir des types de données (entiers, piles, séquences, ...) par des axiomes ne contenant pas de variables, puis de produire des spécifications de programmes qui utilisent ces types de données. Ces spécifications se font également sans utiliser de variables de niveau objet. L'idée d'éviter les variables de niveau objet n'est pas vraiment nouvelle. Par exemple, Backus [3] la promeut fortement dans le contexte de la programmation fonctionnelle, avec son langage FP. L'usage de relations, plutôt que de fonctions, permet une étude plus facile du non-déterminisme dans les spécifications et les programmes [46].

Toutefois, l'utilisation de spécifications relationnelles sans variables pour définir un problème, même de petite taille, nécessite la manipulation d'une quantité importante de formules, d'où le besoin de posséder un outil capable de manipuler ces formules.

Initialement, l'objet de ce mémoire était orienté vers l'utilisation de systèmes de réécriture. Mais lorsque les travaux du groupe de recherche en génie logiciel du département d'informatique de l'Université Laval ont été étendus à la programmation parallèle, il a été convenu d'insister davantage sur la transformation d'expressions relationnelles utilisées dans ce contexte. L'outil recherché devait permettre la manipulation non seulement de formules, mais également de matrices de formules utilisées dans la programmation parallèle.

Ce mémoire se veut donc une contribution bien délimitée à tout un ensemble de travaux de recherche, sous la direction de Jules Desharnais, dont le but ultime est la construction systématique de programmes corrects par rapport à leur spécification. Certains de ces travaux portent sur la création de spécifications relationnelles sans variables, d'autres sur le raffinement de ces spécifications. Le présent mémoire traite de la transformation d'expressions relationnelles sans variables dans un contexte de parallélisme. Il s'agit d'étudier la possibilité de transformer directement ces expressions au moyen d'un outil informatique.

1.3 Organisation

Le chapitre deux de ce mémoire présente les fondements de l'algèbre relationnelle nécessaires à la compréhension de ce mémoire. On y retrouve également plusieurs simplifications applicables aux formules relationnelles sans variables.

Une partie des travaux réalisés dans le cadre de ce projet a porté sur la recherche d'un logiciel capable d'effectuer la transformation de matrices d'expressions relationnelles sans variables. La recherche a mené au constat qu'aucun des produits actuellement disponibles ne répondait directement au besoin. Cependant, certains d'entre eux offrent les fonctionnalités nécessaires au développement d'une application permettant de telles manipulations. Le troisième chapitre rend compte de l'information recueillie sur les différents logiciels étudiés.

Le quatrième chapitre, intitulé *La transformation constructive*, présente l'approche que nous avons adoptée pour effectuer la simplification des expressions relationnelles sans variables. On y retrouve également les algorithmes qui ont servi à programmer le prototype.

Le chapitre cinq décrit les fonctions et les possibilités du prototype, nommé $\mathcal{SR}\circ\Psi lab$ (SR pour symbolisme relationnel), qui a été développé avec le logiciel public Ψlab. Ce prototype permet de transformer des matrices d'expressions relationnelles sans variables. Il peut exécuter les fonctions élémentaires de l'algèbre relationnelle et des fonctions associées à la

composition parallèle. Ce prototype intègre également un environnement de travail permettant, entre autres, l'édition de matrices et la définition de nouveaux opérateurs évolués à partir des opérateurs disponibles.

En conclusion, le chapitre 6 présente les bilans et perspectives de cette recherche sur la transformation d'expressions relationnelles sans variables.

On retrouve en annexe le code source des fonctions du prototype $\mathcal{SR}\circ\Psi lab$.

Chapitre 2

L'algèbre des relations

Plusieurs lois et propriétés découlent des nombreux travaux effectués dans le domaine de l'algèbre relationnelle. Il serait trop long d'en faire ici une présentation exhaustive. Le contenu de ce chapitre se limitera en un rappel des notions de l'algèbre relationnelle pertinentes dans le cadre de cette recherche. Le lecteur pourra trouver les démonstrations et les autres lois dans Mili [44], Chin et Tarski [11, 58], Schmidt et Ströhlein [52], ainsi que Brink *et al.* [6].

2.1 La théorie des ensembles

L'algèbre des relations concrètes est basée sur la théorie des ensembles. Aussi est-il opportun de rappeler les principales définitions, propriétés et opérateurs de la théorie des ensembles.

2.1.1 Définitions et propriétés

Le mathématicien Cantor (1845-1918) a formulé la définition intuitive d'un *ensemble* comme étant une collection d'objets distincts et identifiables, appelés les *éléments* de l'ensemble. Nous disons que b *appartient* à l'ensemble B, ou bien que b *est un membre* de B si et seulement si b est un élément de B. Cela est noté par $b \in B$. De manière inverse, on note par $b \notin B$ si b n'est pas un élément de B.

Un ensemble est dit fini s'il possède un nombre fini d'éléments. Il est dit infini dans le cas contraire. Un ensemble qui ne possède pas d'éléments est appelé un *ensemble vide* et est identifié par $\emptyset$. Deux ensembles sont dits égaux ou identiques s'ils contiennent les mêmes éléments. On écrit alors $A = B$ ou dans le cas contraire $A \neq B$. Les objets d'un ensemble peuvent

aussi être des ensembles. Dans ce cas, on parle alors d'un ensemble d'ensembles, de famille d'ensembles ou d'ensembles imbriqués. Nous excluons ici les imbrications infinies (on dit que les ensembles sont *bien fondés*).

Si tous les éléments de A sont aussi des éléments de B, on dit alors que A est un *sous-ensemble* de B ou que A est *inclus* dans B . Plus formellement, A est un sous-ensemble de B si

$$\forall x : x \in A \Rightarrow x \in B$$

L'inclusion de A dans B est notée par $A \subset B$ ou bien par $A \subseteq B$ si A est inclus ou égal à B. L'ensemble B est alors considéré comme un *super-ensemble* de A. De plus, A est dit *sous-ensemble propre* de B si $A \subset B$ et $A \neq B$. L'ensemble de tous les sous-ensembles d'un ensemble donné B est appelé l'*ensemble puissance* de B et est dénoté par 2^B. Ainsi $A \in 2^B$ si et seulement si $A \subseteq B$.

L'inclusion satisfait les propriétés suivantes :

- réflexivité $A \subseteq A$
- transitivité $A \subseteq B \wedge B \subseteq C \Rightarrow A \subseteq C$
- antisymétrie $A \subseteq B \wedge B \subseteq A \Rightarrow A = B$

L'ensemble $\emptyset$ est considéré comme un sous-ensemble de chaque ensemble. Habituellement, dans un contexte donné, on fixe un ensemble U, appelé *ensemble universel*. Dans ce contexte, tous les ensembles considérés sont des sous-ensembles de U. C'est donc dire que $\forall S : \emptyset \subseteq S \subseteq U$.

2.1.2 Les opérations

L'*union* de deux ensembles A et B est l'ensemble de tous les éléments qui appartiennent à A ou à B ou bien aux deux. L'union de A et B est indiquée par $A \cup B$. Il s'ensuit directement de la définition que $A \cup B = B \cup A$. L'*intersection* de deux ensembles A et B est l'ensemble de tous les éléments qui appartiennent à la fois à A et à B. L'intersection de A avec B est notée par $A \cap B$. L'union et l'intersection satisfont les propriétés suivantes :

- associativité
  $(A \cup B) \cup C = A \cup (B \cup C)$
  $(A \cap B) \cap C = A \cap (B \cap C)$

- commutativité
  $A \cup B = B \cup A$
  $A \cap B = B \cap A$

- distributivité
  $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
  $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

- idempotence
  $A \cup A = A$
  $A \cap A = A$

- absorption
  $A \cup (A \cap B) = A$
  $A \cap (A \cup B) = A$
  $\emptyset \cup A = A$
  $\emptyset \cap A = \emptyset$
  $U \cup A = U$
  $U \cap A = A$

Le *complément* d'un ensemble A (noté $\overline{A}$) est l'ensemble de tous les éléments qui n'appartiennent pas à A, *i.e.* $\overline{A} = \{b : b \in U \wedge b \notin A\}$. Le complément satisfait les propriétés suivantes :

- $\overline{A} \cup A = U$
  $\overline{A} \cap A = \emptyset$

- involution
  $\overline{\overline{A}} = A$

- lois de De Morgan
  $\overline{A \cup B} = \overline{A} \cap \overline{B}$
  $\overline{A \cap B} = \overline{A} \cup \overline{B}$

La *différence* de deux ensembles A et B est l'ensemble des éléments qui appartiennent à A mais qui n'appartiennent pas à B et est notée $A - B$. Donc $A - B = A \cap \overline{B}$.

Le *produit cartésien* de deux ensembles A et B, que l'on note $A \times B$, est l'ensemble de toutes les paires ordonnées (a, b) où $a \in A$ et $b \in B$, *i.e.*

$$A \times B = \{(a, b) : a \in A \wedge b \in B\}$$

Puisque les paires sont ordonnées, alors $A \times B \neq B \times A$. Le produit cartésien est toutefois distributif sur les opérations intersection, union et différence.

2.2 Les relations

Une *relation* entre les éléments d'un même ensemble est appelée une *relation homogène* et une relation entre les éléments de deux ensembles distincts est appelée une *relation*

hétérogène. Nous nous limiterons ici aux propriétés touchant les relations homogènes.

2.2.1 Définitions et propriétés

Une relation R sur un ensemble S est un sous-ensemble de $S \times S$, *i.e.* $R \subseteq S \times S$. Soit $s, s' \in S$ et $(s, s') \in R$, alors s est dit un *antécédent* de s' par R et s' est une *image* de s par R.

On définit les relations suivantes sur S :

- la relation universelle, $L = S \times S$
- la relation vide, $O = \{\}$
- la relation identité, $I = \{(s, s') : s' = s\}$
- la relation diversité, $V = \{(s, s') : s \neq s'\}$

Et les ensembles suivants :

- le domaine de R, $dom(R) = \{s : \exists s' : (s, s') \in R\}$
- l'ensemble des images de s par R, $s.R = \{s' : (s, s') \in R\}$
- le codomaine de R, $cod(R) = \{s' : \exists s : (s, s') \in R\}$
- l'ensemble des antécédents de s' par R, $R.s' = \{s : (s, s') \in R\}$

De plus :

- R est dite réflexive ssi (si et seulement si) $\forall s : (s, s) \in R$
- R est dite symétrique ssi $\forall s, s' : (s, s') \in R \Rightarrow (s', s) \in R$
- R est dite transitive ssi $\forall s, s', s'' : (s, s') \in R \wedge (s', s'') \in R \Rightarrow (s, s'') \in R$

2.2.2 Les opérations

Les opérations ensemblistes telles que l'union, l'intersection, la différence et le complément s'appliquent également aux relations binaires. S'ajoutent les définitions suivantes pour des relations quelconques Q et R sur un ensemble S :

- L'*inverse*, $\widehat{R} = \{(s', s) : (s, s') \in R\}$. Par convention, nous écrirons $(R)\widehat{\ }$ plutôt que $\widehat{(R)}$ pour les expressions munies de parenthèses.
- La *composition* ou le *produit relationnel*, $Q \circ R = \{(s, s') : \exists s'' \in S : (s, s'') \in Q \wedge (s'', s') \in R\}$. S'il n'y a pas de confusion, le symbole de composition $\circ$ sera omis et nous écrirons simplement QR pour $Q \circ R$.
- La *puissance relationnelle*, $R^0 = I$ et $R^i = R^{i-1}R$, pour $i \geq 1$.
- La *fermeture transitive*, $R^+ = \{(s, s') : \exists i \geq 1 : (s, s') \in R^i\} = \bigcup_{i \geq 1} R^i$.
- La *fermeture transitive réflexive*, $R^* = I \cup R^+$.
- Le *résidu à gauche*, $Q/R = \{(s, s') : \forall s'', (s', s'') \in R \Rightarrow (s, s'') \in Q\} = \overline{\overline{Q}\widehat{R}}$.
- Le *résidu à droite*, $Q\backslash R = \{(s, s') : \forall s'', (s'', s) \in Q \Rightarrow (s'', s') \in R\} = \widehat{\overline{\widehat{Q}\overline{R}}}$.

La priorité des opérateurs relationnels est la suivante : le complément ($\overline{\ }$) et l'inverse ($\widehat{\ }$) ont la priorité la plus élevée; viennent en deuxième le produit ($\circ$), le résidu à gauche ($/$) et le résidu à droite ($\backslash$); suit en troisième l'intersection ($\cap$); finalement l'union ($\cup$) possède la priorité la plus basse.

2.3 Le calcul des relations

L'origine du calcul des relations remonte au siècle dernier avec les travaux de De Morgan, Peirce, Dedekind et Schröder. Leur étude a été ravivée par les articles de Chin et Tarski [11, 58] qui axiomatisent la notion d'algèbre de relations homogènes.

2.3.1 Définitions et propriétés

Une *algèbre de relations homogènes* [52] est une structure $(\mathcal{R}, \cup, \cap, \overline{\ }, \circ, \widehat{\ })$ sur un ensemble $\mathcal{R}$ non vide dont les éléments sont appelés *relations*. Les conditions suivantes sont satisfaites :

1. $(\mathcal{R}, \cup, \cap, \overline{\ })$ est une algèbre booléenne complète, avec élément *zéro* O et élément *universel* L. Les éléments de $\mathcal{R}$ sont ordonnés par *inclusion*, notée $\subseteq$.
2. La composition est associative et a pour identité I :

   (a) $P(QR) = (PQ)R$

(b) $IR = RI = R$.

3. $PQ \subseteq R \Leftrightarrow \widehat{P}\overline{R} \subseteq \overline{Q} \Leftrightarrow \overline{R}\widehat{Q} \subseteq \overline{P}$ (*règle de Schröder*).

4. Si $R \neq O$, alors $LRL = L$ (*règle de Tarski*). □

De plus :

- R est réflexive $\Leftrightarrow I \subseteq R$
- R est symétrique $\Leftrightarrow R \subseteq \widehat{R}$
- R est transitive $\Leftrightarrow RR \subseteq R$
- R est déterministe $\Leftrightarrow \widehat{R}R \subseteq I$
- R est totale $\Leftrightarrow L = RL \Leftrightarrow I \subseteq R\widehat{R}$
- R est injective $\Leftrightarrow R\widehat{R} \subseteq I$
- R est surjective $\Leftrightarrow LR = L \Leftrightarrow I \subseteq \widehat{R}R$

Il est à noter que les définitions de réflexivité, de symétrie et de transitivité sont équivalentes à celles de la section 2.2.1 pour les relations sur un ensemble.

Voici quelques règles habituelles du calcul des relations. La démonstration de ces règles est faite, entre autres, dans Chin et Tarski [11, 58], Brink *et al.* [6], ainsi que Schmidt et Ströhlein [52]. Soient P, Q, R des relations.

Commutativité

$Q \cap R = R \cap Q$ $\qquad$ $Q \cup R = R \cup Q$

Associativité

$(P \cap Q) \cap R = P \cap (Q \cap R)$ $\qquad$ $(P/Q)/R = P/(RQ)$
$(P \cup Q) \cup R = P \cup (Q \cup R)$ $\qquad$ $P\backslash(Q\backslash R) = (QP)\backslash R$
$(PQ)R = P(QR)$

Distributivité

$P(Q \cap R) \subseteq PQ \cap PR$ $\qquad$ $(P \cup Q)R = PR \cup QR$
$(P \cap Q)R \subseteq PR \cap QR$ $\qquad$ $P \cup (Q \cap R) = (P \cup Q) \cap (P \cup R)$
$P(Q \cup R) = PQ \cup PR$ $\qquad$ $P \cap (Q \cup R) = (P \cap Q) \cup (P \cap R)$
$(P \cap Q)/R = (P/R) \cap (Q/R)$ $\qquad$ $P\backslash(Q \cap R) = (P\backslash Q) \cap (P\backslash R)$
$P/(Q \cup R) = (P/Q) \cap (P/R)$ $\qquad$ $(P \cup Q)\backslash R = (P\backslash R) \cap (Q\backslash R)$

Absorption et identité

$R \cap O = O$ $\quad RO = O$

$R \cup L = L$ $\quad OR = O$

$R \cup O = R$ $\quad RI = R$

$R \cap L = R$ $\quad IR = R$

$L/R = L$ $\quad R\backslash L = L$

$R/O = L$ $\quad O\backslash R = L$

$R/I = R$ $\quad I\backslash R = R$

$O/L = O$ $\quad L\backslash O = O$

Inversion

$\hat{\hat{R}} = R$ $\quad \hat{I} = I$

$(QR)^{\wedge} = \hat{R}\hat{Q}$ $\quad \hat{L} = L$

$(Q \cup R)^{\wedge} = \hat{Q} \cup \hat{R}$ $\quad \hat{O} = O$

$(Q \cap R)^{\wedge} = \hat{Q} \cap \hat{R}$

Monotonie

$P \subseteq Q \Rightarrow PR \subseteq QR$

$Q \subseteq R \Rightarrow PQ \subseteq PR$

Complémentation

$\overline{\overline{R}} = R$ $\quad \overline{Q \cup R} = \overline{Q} \cap \overline{R}$

$\overline{L} = O$ $\quad \overline{Q \cap R} = \overline{Q} \cup \overline{R}$

$\overline{O} = L$ $\quad \overline{\hat{R}} = \hat{\overline{R}}$

Équivalences et implications

$Q \subseteq R \Leftrightarrow \overline{R} \subseteq \overline{Q}$ $\quad P \cap Q \subseteq R \Leftrightarrow P \subseteq \overline{Q} \cup R$

$Q \subseteq R \Leftrightarrow \hat{Q} \subseteq \hat{R}$ $\quad PQ \subseteq R \Leftrightarrow \hat{P}\overline{R} \subseteq \overline{R} \Leftrightarrow \overline{R}\hat{Q} \subseteq \overline{P}$

P déterministe $\Rightarrow P(Q \cap R) = PQ \cap PR$ $\quad P$ injective $\Rightarrow P(\hat{P}Q \cap R) = Q \cap PR$

P déterministe $\Rightarrow (Q \cap R\hat{P})P = QP \cap R$ $\quad P$ injective $\Rightarrow (Q \cap R)P = QP \cap RP$

2.3.2 Les matrices de relations

Un ensemble de matrices de dimensions appropriées et dont les entrées sont des relations constitue une algèbre de relations [52], en définissant les opérateurs relationnels comme suit

(R_{ij} désigne l'entrée i, j de la matrice R) :

$$\begin{aligned}
(Q \cup R)_{ij} &= Q_{ij} \cup R_{ij} \\
(Q \cap R)_{ij} &= Q_{ij} \cap R_{ij} \\
(\overline{R})_{ij} &= \overline{R_{ij}} \\
(\widehat{R})_{ij} &= (R_{ji})\widehat{} \\
(QR)_{ij} &= \bigcup_k Q_{ik} R_{kj} \\
(Q/R)_{ij} &= \bigcap_k Q_{ik}/R_{jk} \\
(Q\backslash R)_{ij} &= \bigcap_k Q_{ki}\backslash R_{kj}
\end{aligned}$$

Dans cette algèbre, la matrice universelle est la matrice dont chaque entrée est la relation universelle L et la matrice vide est la matrice dont chaque entrée est la relation vide O. Finalement, la matrice identité est la matrice dont la diagonale contient la relation identité I et les autres entrées la relation vide O.

Cette dernière définition introduit donc des opérations relationnelles entre des matrices de tailles différentes ce qui en fait une algèbre hétérogène plutôt qu'homogène. Cette algèbre satisfait les lois de la section 2.3.1 sauf que :

- Les opérations union et intersection ne sont plus toujours définies et ne s'appliquent que sur des matrices de tailles identiques.
- L'inverse d'une matrice de dimension $(n \times m)$ produit une matrice de dimension $(m \times n)$.
- Le produit entre des matrices de dimensions différentes est possible seulement si le nombre de lignes de la première matrice est identique au nombre de colonnes de la seconde.
- Le résidu à gauche entre des matrices de dimensions différentes est possible seulement si le nombre de colonnes de la première matrice est identique au nombre de colonnes de la seconde.
- Le résidu à droite entre des matrices de dimensions différentes est possible seulement si le nombre de lignes de la première matrice est identique au nombre de lignes de la seconde.
- Il peut exister plusieurs matrices vides et plusieurs matrices universelles. Plus précisément, pour chaque couple (n, m), on peut former une matrice vide et une matrice universelle de dimension $(n \times m)$.

- Pour chaque couple (n, n), on peut former une matrice identité de dimension $(n \times n)$.

L'hétérogénéité se situe au niveau matriciel seulement. Les opérations sur les entrées des matrices satisfont totalement les lois d'une algèbre de relations homogènes.

Chapitre 3

Les logiciels

En 1959, des chercheurs du MIT (Massachusetts Institute of Technology) entreprirent le développement d'un logiciel capable de résoudre des équations différentielles élémentaires. Le logiciel fut baptisé *MACSYMA* et peut être considéré comme le premier système de manipulations algébriques (Computer Algebra System).

Il existe aujourd'hui une multitude de produits capables d'effectuer des manipulations algébriques. Cela va des applications mathématiques sophistiquées aux langages spécialisés en passant par divers produits intermédiaires.

La question est de savoir si, parmi ces produits, on en trouve qui peuvent effectuer les transformations sur des matrices d'expressions relationnelles sans variables, c'est-à-dire des matrices dont les entrées sont des expressions relationnelles. Notre recherche nous a mené au constat suivant : aucun des produits étudiés n'offre directement la possibilité d'effectuer les transformations souhaitées sur des matrices d'expressions relationnelles sans variables. Cependant, plusieurs d'entre eux possèdent les fonctionnalités nécessaires au développement d'une application répondant à ce besoin. Nous avons choisi d'utiliser le logiciel public *Scilab* (Ψlab) comme plate-forme de développement d'un prototype car les particularités de ce logiciel favorisaient le développement rapide du prototype souhaité.

Ce chapitre présente succintement les différents logiciels qui ont été étudiés dans le cadre de cette recherche. Ils ont été regroupés sous trois grands types: les applications symboliques, les logiciels de manipulation de règles et les langages spécialisés (algébriques, fonctionnels, etc.).

3.1 Les applications symboliques

Les applications symboliques sont des logiciels qui permettent la manipulation symbolique d'expressions mathématiques. Ces produits sont plus évolués que des langages au sens où ils intègrent un environnement relativement convivial. Les plus répandus sont *Maple*, *Mathematica* et *Matlab*.

Maple

Maple [9, 24] est un logiciel de manipulation mathématique développé par le « Symbolic Computation Group » de l'Université de Waterloo et commercialisé par la compagnie Waterloo Maple Software. Il est disponible pour les environnements UNIX, PC et Macintosh.

Ce logiciel possède une multitude de fonctions mathématiques dont les fonctions trigonométriques, le calcul différentiel et intégral, l'algèbre linéaire et non linéaire, la manipulation d'êtres mathématiques complexes et autres. Il offre aussi une série de fonctions graphiques rendant possible la représentation sous forme de graphique à deux ou trois dimensions. *Maple* permet d'effectuer des simplifications d'expressions ou d'autres tranformations sur des expressions utilisant des éléments qui ne sont pas évalués.

La compagnie Waterloo Maple Software commercialise aussi *Theorist* qui est un programme interactif de manipulation symbolique et graphique avec une interface graphique et *Mathedge* qui est la librairie permettant d'utiliser le moteur symbolique de *Maple* avec une application maison.

Maple n'a pas été retenu principalement parce que son éditeur de matrices était trop rudimentaire et l'environnement très fermé. Mais son moteur symbolique est l'un des plus complets et des plus répandus. Il est, entre autres, utilisé par *MathCad*, *Matlab* et *PV-Wave*.

Des informations sur *Maple* sont disponibles sur les sites internet suivants : `ftp.maplesoft.com`, `ftp.maplesoft.on.ca` ainsi que `www.maplesoft.com`.

Mathematica

Mathematica [62] est un des systèmes algébriques les plus répandus. Il offre une librairie variée de fonctions mathématiques et graphiques ainsi que plusieurs librairies spécialisées. L'utilisateur peut définir ses propres fonctions et ses types de données. Il permet la programmation logique, procédurale, fonctionnelle et orientée objet. De plus, il supporte la manipu-

lation de l'arbre d'une expression symbolique et il existe une librairie pour les algèbres non commutatives (*NCAlgebra*).

Mathematica n'a pas été notre premier choix pour le développement du prototype en raison de son environnement fermé et de l'absence d'un éditeur de matrices. Toutefois, dans l'éventualité du développement d'une application complète, *Mathematica* présenterait un intérêt certain étant donné ses possibilités de programmation symbolique.

Mathematica est disponible sur les plate-formes PC, Macintosh et UNIX. Des informations supplémentaires sont disponibles sur les sites `ftp.wolfram.com` et `www.wolfram.com`. De plus, il existe un groupe public de discussion soit le `comp.soft-sys.math.mathematica`.

Matlab

Matlab [31] est un système interactif et un langage de programmation utilisés pour les calculs et la représentation graphique de problèmes scientifiques et techniques. Le logiciel vient avec une multitude de fonctions intégrées et il existe diverses librairies pour des besoins spécifiques. Le type de données élémentaires de *Matlab* est la matrice numérique, mais il supporte aussi la matrice caractère. Cependant, il ne possède pas à proprement dit un éditeur de matrices pouvant répondre adéquatement à notre besoin, soit l'édition en mode « plein écran ».

Les fonctions symboliques de *Maple* ont été intégrées dans *Matlab* sous la forme de deux librairies : la « Standard Symbolic Toolbox » qui contient près de 125 fonctions et la « Extended Symbolic Toolbox » qui contient toutes les fonctions de *Maple*, incluant la capacité de programmation.

Il existe une version de *Matlab* pour les principales plate-formes. Des informations supplémentaires sont disponibles sur les sites suivants : `ftp.mathworks.com` et `www.mathworks.com`. De plus, le groupe public de discussion `comp.soft-sys.matlab` est dédié à *Matlab* et il existe des fichiers `FAQ.text` et `FAQ.html` au site `csi.jpl.nasa.gov` dans le répertoire `/pub/matlab`.

Scilab

Le logiciel *Scilab* (aussi écrit comme Ψlab) [28] est une application « freeware » dont la syntaxe et les fonctionnalités sont similaires au logiciel *Matlab*. Il a été développé à l'Institut

national de recherche en informatique et en automatique (INRIA) de France. *Scilab* est un interpréteur qui permet la manipulation de matrices et l'exécution de fonctions matricielles élémentaires. Il permet également la manipulation symbolique de polynômes, de matrices de polynômes, de systèmes linéaires et non linéaires. Il supporte le traitement de chaînes de caractères et la manipulation de matrices de chaînes de caractères.

Scilab offre un environnement propice au développement d'applications où il est possible d'y définir de nouveaux types de données, de créer des fonctions et des librairies de fonctions. Plus encore, ces fonctions peuvent être considérées et traitées comme des objets. *Scilab* reconnaît également des fonctions développées en FORTRAN ou en langage C, ce qui permet d'élargir les possibilités des applications au-delà des limites du produit.

Scilab est le logiciel que nous avons retenu pour le développement du prototype, principalement à cause de son environnement de programmation ouvert, de ses fonctions d'édition de matrices de nombres et de caractères, ainsi que de la possibilité de définir de nouveaux types de données.

Ce logiciel a d'abord été conçu pour les plate-formes UNIX. Il existe maintenant une version bêta pour Macintosh et une version pour Windows 95 est prévue prochainement. Ce logiciel est disponible sur le site `ftp.inria.fr` dans le répertoire `/INRIA/Scilab` . De plus, il existe une groupe restreint de discussion à l'adresse `scilab@inria.fr` .

Rlab

Un autre émule de *Matlab* est *Rlab* [54]. Ce logiciel est utilisé pour l'analyse numérique sur des matrices, plus particulièrement pour le prototypage et l'expérimentation d'algorithmes. La manipulation symbolique étant absente, ce produit n'a pas été retenu. Il existe des versions pour UNIX et Macintosh et les sites ftp pour ce logiciel sont `csi.jpl.nasa.gov` dans le répertoire `/pub/matlab/RLab` ainsi que `evans.ee.adfa.oz.au` dans le répertoire `/pub/RLaB` .

Matcalc

Matcalc [27] est un système interactif de calcul matriciel conçu principalement pour la solution de problèmes matriciels avec nombres réels ou complexes, la solution de systèmes d'équations linéaires et l'enseignement des techniques modernes des algèbres linéaire et matricielle. Il offre la possibilité d'intégrer des routines externes en C, mais les possibilités symboliques sont absentes.

Il existe une version pour UNIX et Macintosh. La version Macintosh se trouve sur le site ftp `hwarang.postech.ac.kr` dans le répertoire `/pub/mac/info-mac/sci` .

Reduce

Reduce [43, 57] est un système interactif et un langage de programmation conçus pour effectuer des manipulations symboliques et des calculs numériques. Parmi les possibilités offertes par cet outil, on retrouve l'expansion de polynômes et de fonctions rationnelles, la simplification automatique ou contrôlée des expressions et les calculs avec des matrices symboliques. Il existe plusieurs librairies de programmes dont une pour les algèbres commutatives (*CALI*).

Toutefois, *Reduce* ne possède pas d'éditeur de matrices « plein écran ». Ce logiciel est disponible sur plus d'une plate-forme et des informations, ainsi que des logiciels d'essai, peuvent être trouvés au site web `www.rrz.uni-koeln.de` à la page `/REDUCE` et au site ftp `bath.ac.uk` dans le répertoire `/pub/jpff/reduce` .

SyMan

SyMan [63] pour SYmbolic MANipulator est un outil de manipulation algébrique simple, principalement axé vers les mathématiques de niveau collégial. Cependant, cet outil n'est pas assez puissant pour effectuer le traitement d'expressions relationnelles sans variables.

Il est utilisable dans l'environnement Macintosh. On le trouve sur le site web `archives.math.utk.edu` dans le répertoire `/software/mac/collegeAlgebra` à la page `.directory.html` , ainsi que sur le site ftp `archives.math.utk.edu` dans le répertoire `/software/mac/` et sous-répertoire `collegeAlgebra/sysman` .

Octave

Octave [21] est un langage interactif évolué, principalement axé vers les calculs numériques et presque totalement compatible avec *Matlab*. Ce produit peut exécuter des opérations arithmétiques sur des matrices de nombres réels et complexes et résoudre des ensembles d'équations non linéaires. Il peut traiter des intervalles finis ou infinis, ainsi que des équations différentielles ordinaires et algébriques. Toutefois, il n'offre pas toutes les fonctionnalités nécessaires aux manipulations symboliques. Il existe sur la plate-forme UNIX et est disponible sur le site `ftp.che.utexas.edu` dans le répertoire `/pub/octave` .

3.2 Logiciels de manipulation de règles

Sont regroupés ici les logiciels spécialisés dans la manipulation de règles logiques et/ou axiomatiques. Ces logiciels sont utilisés pour effectuer la démonstration de théorèmes ou la réduction d'expressions. De manière générale, ils procèdent en effectuant toutes les substitutions possibles et conservent seulement les chemins qui ont donné des résultats. Certains nécessitent aussi l'intervention de l'usager dans le processus de décision des chemins à parcourir.

Ces logiciels ne répondaient pas à notre besoin premier, soit de pouvoir construire des expressions relationnelles suite aux opérations matricielles. De plus, aucun d'entre eux ne permet l'édition de matrices. Cependant, leur capacité à réduire des expressions les rend particulièrement intéressants comme outils complémentaires.

Les logiciels étudiés sont : *CoCoa*, *Pari*, *Isabelle*, *IMPS*, *LP* et *Otter*.

CoCoa

CoCoa [1] est un logiciel spécialisé dans les calculs sur des algèbres commutatives. Son nom vient de COmputations in COmmutative Algebra. Il a été concu pour la plate-forme Macintosh et il existe maintenant une version MS-DOS. Il est capable d'effectuer les opérations de base sur les polynômes et les fonctions rationnelles (somme, produit, puissance, dérivée), sur les nombres parfaits (somme, produit, puissance) et sur les matrices (somme, produit, puissance, déterminant). Il peut également exécuter des opérations plus avancées comme la fonction de Hilbert, les séries de Poincaré, etc.

Pari

Le logiciel *PARI* [4] est axé principalement sur la théorie des nombres. Il est très performant pour effectuer des calculs sur des types récursifs. Il est capable d'effectuer du calcul symbolique avec diverses entités mathématiques telles que des polynômes, des séries, des matrices, etc. Même s'il est capable d'effectuer des manipulations symboliques, il est cependant moins efficace que des produits comme *Maple* et *Mathematica*. Il est disponible pour plusieurs plateformes et son site ftp est `megrez.math.u-bordeaux.fr`, répertoire `/pub/pari` .

Isabelle

Isabelle [33, 49] est un système générique pour la démonstration de théorèmes. Il supporte le raisonnement avec plusieurs types d'objets logiques comme la logique du premier ordre, la

logique d'ordre supérieur, les logiques modales T, S4 et S3, etc. Une particularité intéressante de ce produit est qu'il supporte la théorie des ensembles de Zermelo-Fraenkel. Des connaissances de *Standard ML* sont nécessaires pour utiliser ce logiciel puisque *ML* est l'interface d'*Isabelle*.

IMPS

Le produit *IMPS* (Interactive Mathematical Proof System) [25] a été développé comme support informatique dans l'utilisation des techniques traditionnelles du raisonnement mathématique. Ce système est constitué d'une base de données mathématiques (axiomes et règles d'interprétation) et d'un ensemble d'outils pour l'exploration, l'application, l'expansion et la propagation des règles mathématiques de la base de données. Il est particulièrement utile pour le développement de preuves formelles.

Ce logiciel est disponible sur le site ftp `math.harvard.edu` dans le répertoire `/imps` . Il existe aussi une page web `imps.html` sur le site `math.harvard.edu` dans le répertoire `imps/imps-html` .

LP - Larch Prover

LP [26] est un système interactif de démonstration de théorèmes pour les logiques du premier ordre. Il est utilisé surtout lors de la conception de circuits, d'algorithmes parallèles, d'équipements et de logiciels. *LP* est surtout un outil pour aider l'utilisateur à trouver et corriger les incohérences des systèmes en cours de développement.

Ce logiciel fait partie de la famille de langages de spécifications *LARCH* qui sont basés sur une approche algébrique et axiomatique. Le module « Larch shared language » est la base de toute la famille et il existe un module « Larch interface language » pour chacun des langages de programmation suivants : Ada, C, C++, ML, Modula-3, Smaltalk, CLU. Il existe aussi un module « Larch interface language » universel qui peut être adapté pour un langage de programmation particulier ou utilisé pour faire le lien entre des programmes de différents langages. Les fonctionnalités de l'interface sont modelées sur celles du langage de programmation, incluant les types abstraits s'ils sont supportés par le langage de programmation.

On peut obtenir de l'information sur *LP* au site ftp `larch.lcs.mit.edu` dans le répertoire `/pub/Larch/lp`.

Otter

Otter (Organized Techniques for Theorem-proving and Effective Research) [42] effectue la démonstration de théorèmes pour les logiques du premier ordre avec égalité. Quelques-unes de ses possibilités sont la conversion de clauses à partir de formules du premier ordre, la vérification arrière et avant, la factorisation, la pondération, l'ordonnancement de termes, l'évaluation de fonctions et de prédicats et la complétion de Knuth-Bendix. *Otter* est écrit en langage C et est portable sur différentes plate-formes.

3.3 Langages spécialisés

Même s'ils n'ont pas été retenus pour le développement du prototype parce qu'ils sont trop de bas niveau, certains langages spécialisés sont présentés ici car ils possèdent des particularités intéressantes qui pourraient être utiles dans le contexte des relations.

SETL, ISETL et SETLS

SETL est un langage basé sur la théorie des ensembles. *ISETL* est la version interactive de *SETL* et *SETLS* [18] est une version allégée de *SETL*. L'exécutabilité est ce qui distingue *SETL* du langage mathématique conventionnel. Afin de garantir qu'une spécification *SETL* est exécutable, seuls des objets finis peuvent être créés. Les mécanismes de boucle et de récursivité sont implantés à la manière de Pascal, mais sans la structure de bloc. Les ensembles, les tuples, et les « maps » sont entièrement supportés par *SETL*. Ce langage traite les ensembles seulement. Il ne traite pas les représentations symboliques, ni les relations. Il est disponible sur les plate-formes UNIX, MS-DOS et Macintosh.

IFP

IFP (Illinois Functional Programming) [50] est un langage fonctionnel interactif développé pour les systèmes UNIX et MS-DOS. L'utilisateur peut interactivement créer et exécuter des programmes fonctionnels. La sémantique de *IFP* est presque identique au langage *FP*, même si la syntaxe est différente. Il offre donc les avantages de la programmation modulaire, des processus parallèles, de la vérification et de l'optimisation des programmes. En plus des fonctionnalités du langage *FP*, *IFP* offre une organisation hiérarchique et modulaire des fonctions, ainsi qu'une syntaxe de structure de bloc. *IFP* est constitué d'objets, de fonctions et de formes fonctionnelles.

ASpecT

ASpecT [61] est un langage fonctionnel développé à l'Université de Bremen originellement pour la manipulation de spécifications algébriques pour des types de données abstraits. Ce produit a évolué avec l'ajout d'options, incluant le polymorphisme restreint. Le site ftp est `gatekeeper.dec.com` dans le répertoire `/.e/language/aspect`.

Scheme

Scheme [19, 34] est un langage simple, avec peu d'instructions. Il est considéré comme un langage fonctionnel pur. Il est basé sur le modèle du lambda-calcul. Il possède plusieurs propriétés utiles pour les théoriciens et il offre des outils de développement à l'utilisateur. Son site ftp est `nexus.yorku.ca` dans le répertoire `/pub/scheme`.

JACAL

JACAL [32] est un outil de manipulation symbolique. Il permet la simplification et la manipulation d'équations, l'évaluation d'expressions algébriques simples ou multiples composées de nombres, de variables ou de radicaux. Il permet le traitement de fonctions algébriques, différentielles ou holonomiques. Il traite également les vecteurs et les matrices. *JACAL* est développé avec *Scheme* et est disponible sur le site ftp `prep.ai.mit.edu` dans le répertoire `/pub/gnu/jacal`.

Aucun des produits analysés ne répondant à notre besoin spécifique, soit la transformation de matrices d'expressions relationnelles sans variables, nous avons opté pour le développement d'un prototype à l'aide du logiciel *Scilab*. L'approche utilisée pour la transformation des expressions relationnelles, ainsi que les algorithmes implantés dans le prototype, sont présentés dans le chapitre suivant.

Chapitre 4

La transformation constructive

Dans le contexte du parallélisme, des opérateurs relationnels complexes sont appliqués sur des matrices d'expressions relationnelles sans variables (ERSV) plutôt que sur des ERSV simples. Au-delà des cas théoriques où les matrices sont habituellement de dimensions réduites, un outil informatique s'impose pour effectuer les manipulations symboliques. Par exemple, un simple produit parallèle de deux matrices (10×10) génère comme résultat une matrice (100×100). Or, une matrice (10×10) ne représente qu'un petit système.

L'utilisation d'un logiciel de réduction pour simplifier les ERSV est toujours possible. Cependant, la difficulté première n'est pas de simplifier les expressions mais bien de les construire. Une approche possible pourrait être d'utiliser d'abord un logiciel pour construire les expressions, puis d'utiliser un logiciel de réduction pour simplifier les résultats. La faiblesse majeure de cette approche est qu'elle favorise la création d'expressions inutilement longues, alourdissant ainsi à la fois la tâche du logiciel qui construit les expressions et celle du logiciel qui les réduit.

L'approche que nous proposons consiste à appliquer les transformations lors de la construction des expressions. Nous avons nommé cette approche « la transformation constructive ». Il s'agit d'effectuer certaines simplifications à chaque fois que l'on exécute une opération. Cette approche minimise donc la longueur des expressions résultantes, accélérant ainsi le traitement de la prochaine opération et facilitant la tâche d'un éventuel logiciel de réduction.

4.1 Principes généraux

Les simplifications sont effectuées lors de l'application de chaque opérateur relationnel. Dans le cas où aucune simplification n'est applicable, l'expression résultante est alors construite. À cause de la priorité des opérateurs, une gestion des parenthèses doit être faite afin d'assurer la cohérence des expressions. Une solution facile pour éviter la gestion des parenthèses aurait été de toujours mettre les expressions entre parenthèses avant de construire l'expression résultante. Toutefois, l'insertion automatique de parenthèses aurait alourdi inutilement l'expression résultante et le nombre de parenthèses aurait augmenté avec le nombre d'opérations effectuées.

La règle de gestion des parenthèses peut s'exprimer ainsi : on doit mettre entre parenthèses les expressions qui contiennent un opérateur de priorité inférieure ou un opérateur différent mais de priorité équivalente à l'opération que l'on effectue. Prenons l'exemple de l'intersection de deux expressions: $expr1$ et $expr2$. Dans le cas où $expr1$ correspond à P et $expr2$ à R, il est correct d'écrire le résultat comme $P \cap R$ et les parenthèses ne sont pas utiles. Dans le cas où $expr1 = P \cap S$ et $expr2 = R \cup S$, alors le résultat attendu est $P \cap S \cap (R \cup S)$. Ici, les parenthèses sont nécessaires afin de ne pas perdre la priorité d'évaluation car l'expression $P \cap S \cap R \cup S$ n'est pas équivalente à $P \cap S \cap (R \cup S)$. Il aurait été possible de mettre les deux expressions entre parenthèses avant d'effectuer l'intersection et obtenir ainsi $(P \cap S) \cap (R \cup S)$. Cependant, même si cela n'est pas faux, l'insertion de parenthèses aurait contribué à alourdir l'expression.

Les simplications et l'ajout de parenthèses se font selon un ordre déterminé. Il faut d'abord effectuer les simplifications avant d'ajouter les parenthèses. Supposons le cas où $expr1$ et $expr2$ correspondent toutes les deux à l'expression $P \cup R$. Si les parenthèses étaient traitées avant les simplifications, nous obtiendrions d'abord $(P \cup R) \cap (P \cup R)$, *i.e.* une expression de la forme $expr1 \cap expr1$, où $expr1 = (P \cup R)$, ce qui nous donnerait $(P \cup R)$ après application de l'idempotence. Des parenthèses auraient ainsi été introduites inutilement. Par contre, en effectuant d'abord les simplifications, la résultante obtenue ici est $P \cup R$.

Dans les sections suivantes sont décrits les algorithmes de transformation constructive pour chaque opérateur. Les règles de simplification étant presque mutuellement exclusives, la transformation constructive s'imbrique naturellement dans une instruction de la forme `SI ... SINON SI`. Il est à noter que pour les besoins du prototype, nous n'avons utilisé que quelques-unes des règles de simplification. Dans le cadre du développement d'une application, il suffirait d'ajouter les autres règles de simplification.

4.2 Les opérations élémentaires

4.2.1 L'union

Les règles suivantes du calcul des relations nous permettent de simplifier l'union de deux ERSV :

$$expr \cup O = expr \qquad expr \cup L = L$$
$$O \cup expr = expr \qquad L \cup expr = L$$
$$expr \cup expr = expr$$

où *expr* représente une ERSV quelconque.

L'application des simplifications précédentes s'effectue selon un algorithme déterminé. Dans cet algorithme (et tous ceux qui suivent), *expr*1 signifie l'expression à gauche et *expr*2 l'expression à droite de l'opérateur considéré. S'il s'agit d'un opérateur monadique (*i.e.* s'appliquant sur une seule expression), *expr* est alors utilisé. De plus, les conventions suivantes s'appliquent :

- La fonction « présence » est une fonction booléenne qui sert à déterminer la présence d'opérateurs relationnels à l'intérieur d'une expression; dans cette fonction, les opérateurs sont transmis sous forme d'un ensemble d'opérateurs.
- Le symbole (•) est un opérateur de concaténation d'expressions.
- Le symbole (==) est un opérateur qui vérifie l'égalité syntaxique de deux expressions.
- La forme R_{ij} désigne l'entrée i, j de la matrice R.
- La forme $R_{i \times j}$ indique une matrice de dimensions $(i \times j)$.
- Afin d'alléger la description des algorithmes, les symboles $(L, O, I, \cup, \cap, \circ, /, \backslash)$ représentent leurs équivalents entre apostrophes. Par exemple, L représente "L" et $\cap$ représente "$\cap$".

Dans le prototype, en raison de certaines contraintes techniques, l'union et l'intersection ont la même priorité. L'algorithme de transformation doit donc ajouter des parenthèses à chaque expression contenant un opérateur intersection.

Voici l'algorithme utilisé pour la transformation constructive de l'union de deux ERSV :

UNION($expr1$;$expr2$)

1. SI $expr1 == expr2$ ALORS *résultat* := $expr1$
2. SINON SI $expr1 == L$ OU $expr2 == L$ ALORS *résultat* := L
3. SINON SI $expr1 == O$ ALORS *résultat* := $expr2$
4. SINON SI $expr2 == O$ ALORS *résultat* := $expr1$
5. SINON
6. SI présence($expr1$; $\{\cap\}$) ALORS $expr1$:= "(" • $expr1$ • ")" FIN SI
7. SI présence($expr2$; $\{\cap\}$) ALORS $expr2$:= "(" • $expr2$ • ")" FIN SI
8. *résultat* := $expr1 \bullet \cup \bullet expr2$
9. FIN SI
10. RETOURNER *résultat*

Par exemple, si on désire effectuer l'union de P avec P, alors l'idempotence de l'union (ligne 1 de l'algorithme UNION) permet de simplifier la résultante comme P seulement plutôt que $P \cup P$. De la même façon, si l'une des deux expressions correspond à la relation universelle L, alors la règle d'absorption (ligne 2) permet de simplifier la résultante de cette union comme étant L. La relation vide O étant l'élément nul de l'union et l'union étant un opérateur commutatif, si l'expression de gauche est O, alors le résultat de l'union est l'expression de droite (ligne 3) et si l'expression de droite est O, alors le résultat est l'expression de gauche (ligne 4). La gestion des parenthèses (lignes 6 à 8) s'effectue selon la règle définie à la section 4.1. Ici, l'intersection est l'unique opérateur dont la priorité est inférieure ou équivalente à celle de l'union.

Tel que spécifié à la section 2.3.2, l'opération d'union de deux matrices de dimensions identiques contenant des ERSV est définie comme suit : $(Q \cup R)_{ij} = Q_{ij} \cup R_{ij}$.

Ainsi $$\begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix} \cup \begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix} = \begin{pmatrix} Q_{11} \cup R_{11} & Q_{12} \cup R_{12} \\ Q_{21} \cup R_{21} & Q_{22} \cup R_{22} \end{pmatrix}$$

On constate que chaque cellule de la matrice résultante est formée de l'union de deux ERSV.

L'union de deux matrices ERSV s'effectue selon l'algorithme suivant :

MUNION($A_{i \times j}$;$B_{i \times j}$)

1. FAIRE POUR $n = 1$ à i
2. FAIRE POUR $m = 1$ à j
3. $résultat_{nm}$:= UNION(A_{nm};B_{nm})
4. FIN FAIRE

5. FIN FAIRE
6. RETOURNER $résultat_{i \times j}$

4.2.2 L'intersection

Les règles suivantes du calcul des relations nous permettent de simplifier l'intersection de deux ERSV :

$$expr \cap O = O \qquad O \cap expr = O$$
$$expr \cap L = expr \qquad L \cap expr = expr$$
$$expr \cap expr = expr$$

L'intersection ayant la même priorité que l'union dans le prototype, des parenthèses sont donc ajoutées à chaque expression contenant un opérateur d'union.

Voici l'algorithme utilisé pour la transformation constructive de l'intersection de deux ERSV :

INTER($expr1$;$expr2$)
1. SI $expr1 == expr2$ ALORS $résultat := expr1$
2. SINON SI $expr1 == O$ OU $expr2 == O$ ALORS $résultat := O$
3. SINON SI $expr1 == L$ ALORS $résultat := expr2$
4. SINON SI $expr2 == L$ ALORS $résultat := expr1$
5. SINON
6. SI présence($expr1$; {∪}) ALORS $expr1$:= "(" •$expr1$ •")" FIN SI
7. SI présence($expr2$; {∪}) ALORS $expr2$:= "(" •$expr2$ •")" FIN SI
8. $résultat := expr1$ •∩• $expr2$
9. FIN SI
10. RETOURNER $résultat$

Tel que spécifié à la section 2.3.2, l'opération d'intersection de deux matrices de dimensions identiques contenant des ERSV est définie comme suit : $(Q \cap R)_{ij} = Q_{ij} \cap R_{ij}$.

Ainsi $$\begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix} \cap \begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix} = \begin{pmatrix} Q_{11} \cap R_{11} & Q_{12} \cap R_{12} \\ Q_{21} \cap R_{21} & Q_{22} \cap R_{22} \end{pmatrix}$$

On constate que chaque cellule de la matrice résultante est formée de l'intersection de deux ERSV.

L'intersection de deux matrices ERSV s'effectue selon l'algorithme suivant :

MINTER(A_{ixj};B_{ixj})

1. FAIRE POUR $n = 1$ à i
2. FAIRE POUR $m = 1$ à j
3. $résultat_{nm}$:= INTER(A_{nm};B_{nm})
4. FIN FAIRE
5. FIN FAIRE
6. RETOURNER $résultat_{ixj}$

4.2.3 Le produit

Les règles suivantes nous permettent de simplifier le produit (ou la composition) de deux ERSV :

$$expr \circ I = expr \qquad I \circ expr = expr$$
$$expr \circ O = O \qquad O \circ expr = O$$
$$L \circ L = L$$

Si aucune simplification n'est applicable, on effectue le produit. Auparavant, des parenthèses sont ajoutées pour chaque expression contenant un opérateur de priorité inférieure (*i.e.* union, intersection) ou un opérateur de priorité équivalente (*i.e.* résidu à gauche, résidu à droite).

Voici l'algorithme utilisé pour la transformation constructive du produit de deux ERSV :

PROD($expr1$;$expr2$)

1. SI $expr1 == O$ OU $expr2 == O$ ALORS $résultat := O$
2. SINON SI $expr1 == L$ ET $expr2 == L$ ALORS $résultat := L$
3. SINON SI $expr1 == I$ ALORS $résultat := expr2$
4. SINON SI $expr2 == I$ ALORS $résultat := expr1$
5. SINON
6. SI présence($expr1$; {∪, ∩, /, \}) ALORS $expr1$:= "(" • $expr1$ • ")" FIN SI
7. SI présence($expr2$; {∪, ∩, /, \}) ALORS $expr2$:= "(" • $expr2$ • ")" FIN SI
8. $résultat := expr1$ •∘• $expr2$
9. FIN SI
10. RETOURNER $résultat$

Tel que spécifié à la section 2.3.2, l'opération du produit de deux matrices de dimensions adéquates, *i.e.* dont le nombre de colonnes de la première matrice est identique au nombre de lignes de la seconde matrice, est définie comme suit : $(QR)_{ij} = \bigcup_k Q_{ik} R_{kj}$.

Ainsi $$\begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix} = \begin{pmatrix} Q_{11}R_{11} \cup Q_{12}R_{21} & Q_{11}R_{12} \cup Q_{12}R_{22} \\ Q_{21}R_{11} \cup Q_{22}R_{21} & Q_{21}R_{12} \cup Q_{22}R_{22} \end{pmatrix}$$

On constate que chaque cellule de la matrice résultante est formée de l'union de deux produits ERSV.

Le produit de deux matrices ERSV s'effectue selon l'algorithme qui suit. Dans cet algorithme, la matrice $résultat_{i \times k}$ est initialisée avec la relation vide, car celle-ci est l'élément neutre de l'union. Par ailleurs, l'algorithme PROD est défini ci-dessus tandis que l'algorithme UNION a été défini précédemment à la section 4.2.1.

MPROD($A_{i \times j}$;$B_{j \times k}$)

1. FAIRE POUR $n = 1$ à i
2. FAIRE POUR $m = 1$ à k
3. $résultat_{nm} := O$
4. FIN FAIRE
5. FIN FAIRE
6. FAIRE POUR $n = 1$ à i
7. FAIRE POUR $m = 1$ à k
8. FAIRE POUR $r = 1$ à j
9. $résultat_{nm}$:= UNION($résultat_{nm}$;PROD(A_{nr};B_{rm}))
10. FIN FAIRE
11. FIN FAIRE
12. FIN FAIRE
13. RETOURNER $résultat_{i \times k}$

4.2.4 L'inverse

Les règles suivantes du calcul des relations nous permettent de simplifier l'inverse d'une ERSV :

$$\widehat{O} = O \qquad \widehat{I} = I$$
$$\widehat{L} = L \qquad \widehat{\widehat{expr}} = expr$$

L'algorithme utilisant une notation postfixe de l'inverse et ce dernier partageant la priorité la plus haute avec le complément, des parenthèses sont ajoutées lors de la présence de tout autre opérateur (*i.e.* union, intersection, produit, résidu à gauche, résidu à droite et complément).

Voici l'algorithme utilisé pour la transformation constructive de l'inverse d'une ERSV :

INV(*expr*)

1. SI $expr == O$ OU $expr == L$ OU $expr == I$ ALORS *résultat* := *expr*
2. SINON SI (*expr* a la forme $expr1^\frown$) ALORS *résultat* := $expr1$
3. SINON SI (*expr* a la forme $(expr1)^\frown$) ALORS *résultat* := $expr1$
4. SINON
5. SI présence(*expr* ; $\{\cup, \cap, \circ, \bar{}, /, \backslash\}$) ALORS *expr* := "(" $\bullet$ *expr* $\bullet$ ")" FIN SI
6. *résultat* := $expr \bullet ^\frown$
7. FIN SI
8. RETOURNER *résultat*

Tel que spécifié à la section 2.3.2, l'opération inverse d'une matrice contenant des ERSV est définie comme suit : $(\widehat{R})_{ij} = (R_{ji})^\frown$.

Ainsi $$\begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix}^\frown = \begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{21} \\ \widehat{R}_{12} & \widehat{R}_{22} \end{pmatrix}$$

On constate que chaque cellule de la matrice résultante est formée de l'inverse d'une ERSV.

L'inverse d'une matrice ERSV s'effectue selon l'algorithme suivant :

MINV($A_{i \times j}$)

1. FAIRE POUR $n = 1$ à i
2. FAIRE POUR $m = 1$ à j
3. $résultat_{nm}$:= INV(A_{mn})
4. FIN FAIRE
5. FIN FAIRE
6. RETOURNER $résultat_{j \times i}$

4.2.5 Le complément

Les règles suivantes du calcul des relations nous permettent de simplifier la complémentation d'une ERSV :

$$\overline{O} = L$$
$$\overline{L} = O$$
$$\overline{\overline{expr}} = expr$$

L'algorithme utilisant une notation postfixe du complément et ce dernier partageant la priorité la plus haute avec l'inverse, des parenthèses sont ajoutées lors de la présence de tout autre opérateur (*i.e.* union, intersection, produit, résidu à gauche, résidu à droite et inverse).

Voici l'algorithme pour la transformation constructive du complément d'une ERSV :

COMP(*expr*)
1. SI $expr == O$ ALORS $résultat := L$
2. SINON SI $expr == L$ ALORS $résultat := O$
3. SINON SI (*expr* a la forme $expr1^{-}$) ALORS $résultat := expr1$
4. SINON SI (*expr* a la forme $(expr1)^{-}$) ALORS $résultat := expr1$
5. SINON
6. SI présence(*expr* ; $\{\cup, \cap, \circ, \hat{}, /, \backslash\}$) ALORS $expr :=$ "(" $\bullet expr \bullet$ ")" FIN SI
7. $résultat := expr \bullet^{-}$
8. FIN SI
9. RETOURNER *résultat*

Tel que spécifié à la section 2.3.2, le complément d'une matrice ERSV est un opérateur monadique qui est défini comme suit : $(\overline{R})_{ij} = \overline{R_{ij}}$.

Ainsi $$\overline{\begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix}} = \begin{pmatrix} \overline{R}_{11} & \overline{R}_{12} \\ \overline{R}_{21} & \overline{R}_{22} \end{pmatrix}$$

On constate que chaque cellule de la matrice résultante est formée du complément d'une ERSV.

Le complément d'une matrice ERSV s'effectue selon l'algorithme suivant :

MCOMP($A_{i \times j}$)
1. FAIRE POUR $n = 1$ à i
2. FAIRE POUR $m = 1$ à j
3. $résultat_{nm} :=$ COMP(A_{nm})
4. FIN FAIRE
5. FIN FAIRE
6. RETOURNER $résultat_{i \times j}$

4.2.6 Le résidu à gauche

Les règles suivantes nous permettent de simplifier le résidu à gauche de deux ERSV :

$$L/expr = L \qquad expr/I = expr$$
$$expr/O = L \qquad O/L = O$$

Des parenthèses sont ajoutées pour chaque expression contenant un opérateur de priorité inférieure (*i.e.* union, intersection) ou un opérateur de priorité équivalente (*i.e.* résidu à droite, produit). De plus, le résidu à gauche n'étant pas un opérateur associatif, *i.e.* que $(P/Q)/R \neq P/(Q/R)$, les expressions contenant cet opérateur sont aussi mises entre parenthèses.

Voici l'algorithme utilisé pour la transformation constructive du résidu à gauche de deux ERSV :

RESIDUG($expr1$;$expr2$)

1. SI $expr1 == L$ ALORS $résultat := L$
2. SINON SI $expr2 == I$ ALORS $résultat := expr1$
3. SINON SI $expr2 == O$ ALORS $résultat := L$
4. SINON SI $expr1 == O$ ET $expr2 == L$ ALORS $résultat := O$
5. SINON
6. SI présence($expr1$; $\{\cup, \cap, \circ, \backslash, /\}$) ALORS $expr1$:= "(" • $expr1$ • ")" FIN SI
7. SI présence($expr2$; $\{\cup, \cap, \circ, \backslash, /\}$) ALORS $expr2$:= "(" • $expr2$ • ")" FIN SI
8. $résultat := expr1$ •/• $expr2$
9. FIN SI
10. RETOURNER $résultat$

Tel que spécifié à la section 2.3.2, l'opération résidu à gauche de deux matrices de dimensions adéquates, *i.e.* dont le nombre de colonnes de la première matrice est identique au nombre de colonnes de la seconde matrice, est définie comme suit : $(Q/R)_{ij} = \bigcap_k Q_{ik}/R_{jk}$.

$$\text{Ainsi } \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix} / \begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix} = \begin{pmatrix} Q_{11}/R_{11} \cap Q_{12}/R_{12} & Q_{11}/R_{21} \cap Q_{12}/R_{22} \\ Q_{21}/R_{11} \cap Q_{22}/R_{12} & Q_{21}/R_{21} \cap Q_{22}/R_{22} \end{pmatrix}$$

On constate que chaque cellule de la matrice résultante est formée de l'intersection de deux résidus à gauche d'ERSV.

Le résidu à gauche de deux matrices ERSV s'effectue selon l'algorithme qui suit. Dans cet algorithme, la matrice $résultat_{i \times j}$ est initialisée avec la relation universelle L, car celle-ci est l'élément neutre de l'intersection. Par ailleurs, l'algorithme RESIDUG est défini ci-dessus tandis que l'algorithme INTER a été défini précédemment à la section 4.2.2.

MRESIDUG($A_{i\times k}$;$B_{j\times k}$)

1. FAIRE POUR $n = 1$ à i
2. FAIRE POUR $m = 1$ à j
3. $résultat_{nm} := L$
4. FIN FAIRE
5. FIN FAIRE
6. FAIRE POUR $n = 1$ à i
7. FAIRE POUR $m = 1$ à j
8. FAIRE POUR $r = 1$ à k
9. $résultat_{nm}$:= INTER($résultat_{nm}$;RESIDUG(A_{nr};B_{mr}))
10. FIN FAIRE
11. FIN FAIRE
12. FIN FAIRE
13. RETOURNER $résultat_{i\times j}$

4.2.7 Le résidu à droite

Les règles nous permettant de simplifier le résidu à droite de deux ERSV sont les suivantes :

$$expr\backslash L = L \qquad I\backslash expr = expr$$
$$O\backslash expr = L \qquad L\backslash O = O$$

Des parenthèses sont ajoutées pour chaque expression contenant un opérateur de priorité inférieure (*i.e.* union, intersection) ou un opérateur de priorité équivalente (*i.e.* résidu à gauche, produit). De plus, le résidu à droite n'étant pas un opérateur associatif, *i.e.* que $(P\backslash Q)\backslash R \neq P\backslash(Q\backslash R)$, les expressions contenant cet opérateur sont aussi mises entre parenthèses.

Voici l'algorithme utilisé pour la transformation constructive du résidu à droite de deux ERSV :

RESIDUD($expr1$;$expr2$)

1. SI $expr2 == L$ ALORS $résultat := L$
2. SINON SI $expr1 == I$ ALORS $résultat := expr2$
3. SINON SI $expr1 == O$ ALORS $résultat := L$
4. SINON SI $expr1 == L$ ET $expr2 == O$ ALORS $résultat := O$
5. SINON
6. SI présence($expr1$; $\{\cup, \cap, \circ, /, \backslash\}$) ALORS $expr1$:= "(" • $expr1$ • ")" FIN SI

7. SI présence($expr2$; $\{\cup, \cap, \circ, /, \backslash\}$) ALORS $expr2$:= "(" • $expr2$ • ")" FIN SI
8. $résultat := expr1 \bullet\backslash\bullet expr2$
9. FIN SI
10. RETOURNER *résultat*

Tel que spécifié à la section 2.3.2, l'opération résidu à droite de deux matrices de dimensions adéquates, *i.e.* dont le nombre de lignes de la première matrice est identique au nombre de lignes de la seconde matrice, est définie comme suit : $(Q\ R)_{ij} = \cap_k Q_{ki} \backslash R_{kj}$.

Ainsi $\begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix} \backslash \begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix} = \begin{pmatrix} Q_{11}\backslash R_{11} \cap Q_{21}\backslash R_{21} & Q_{11}\backslash R_{12} \cap Q_{21}\backslash R_{22} \\ Q_{12}\backslash R_{11} \cap Q_{22}\backslash R_{22} & Q_{12}\backslash R_{12} \cap Q_{22}\backslash R_{22} \end{pmatrix}$

On constate que chaque cellule de la matrice résultante est formée de l'intersection de deux résidus à droite d'ERSV.

Le résidu à droite de deux matrices ERSV s'effectue selon l'algorithme qui suit. Dans cet algorithme, la matrice $résultat_{i \times j}$ est initialisée avec la relation universelle, car celle-ci est l'élément neutre de l'intersection. Par ailleurs, l'algorithme RESIDUD est défini ci-dessus tandis que l'algorithme INTER a été défini précédemment à la section 4.2.2.

MRESIDUD($A_{k \times i}$;$B_{k \times j}$)
1. FAIRE POUR $n = 1$ à i
2. FAIRE POUR $m = 1$ à j
3. $résultat_{nm} := L$
4. FIN FAIRE
5. FIN FAIRE
6. FAIRE POUR $n = 1$ à i
7. FAIRE POUR $m = 1$ à j
8. FAIRE POUR $r = 1$ à k
9. $résultat_{nm}$:= INTER($résultat_{nm}$;RESIDUD(A_{rn};B_{rm})
10. FIN FAIRE
11. FIN FAIRE
12. FIN FAIRE
13. RETOURNER $résultat_{i \times j}$

4.3 La composition ou le produit parallèle

Les opérateurs définis pour la composition parallèle sont des opérateurs très complexes, mais qui sont tous construits à partir des opérateurs élémentaires de l'algèbre des relations. Dans ce contexte, les règles de transformation constructive définies pour les opérateurs élémentaires sont automatiquement appliquées lors de l'utilisation des opérateurs de composition parallèle.

4.3.1 Définitions

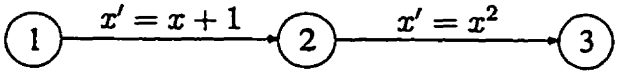
Dans un article portant sur une approche relationnelle à la décomposition parallèle, Chaib-draa *et al.* [8] définissent la sémantique d'un opérateur de composition parallèle à partir de la notion de diagramme de programme telle que définie par Schmidt et Ströhlein [52] pour l'étude des programmes séquentiels. Les définitions qui suivent sont tirées de l'article de Chaib-draa *et al.* [8].

4.3.1.1 Définition. Soient les ensembles S (l'ensemble des *états*) et V (l'ensemble des *points de contrôle*). Un *diagramme de programme* sur S et V est une relation P sur $S \times V$ (*i.e.* $P \subseteq (S \times V) \times (S \times V)$). □

Le terme *diagramme de programme* vient du fait que de telles relations servent à modéliser les programmes. Considérons, par exemple, le programme suivant :

$$x := x + 1;$$
$$x := x^2;$$

où x prend ses valeurs sur l'ensemble des naturels **N**, qui est l'ensemble des états S. Un tel programme est conventionnellement représenté par le diagramme suivant sur l'ensemble des points de contrôle $V = \{1, 2, 3\}$:



Ce diagramme peut être représenté par la matrice suivante :

$$\begin{pmatrix} O & x' = x+1 & O \\ O & O & x' = x^2 \\ O & O & O \end{pmatrix}$$

où O correspond à l'absence d'arc entre deux sommets.

Ce même diagramme peut aussi être représenté par la relation suivante P sur $\mathbf{N} \times \{1,2,3\}$:

$$\{((v,s),(v',s')) : (v = 1 \wedge v' = 2 \wedge x' = x+1) \vee (v = 2 \wedge v' = 3 \wedge x' = x^2)\}$$

4.3.1.2 Définition. Un n-uplet de relations $(\pi_1, \ldots, \pi_n)$ est un *produit direct* ssi

$$\hat{\pi}_i \pi_i = I \quad (i = 1 \ldots n), \qquad \bigcap_{i=1}^{n} \pi_i \hat{\pi}_i = I.$$

On dit que le produit est *plein* si $i \neq j \Rightarrow \hat{\pi}_i \pi_j = L \quad (i,j = 1 \ldots n)$. Les relations π_i sont appelées *projections*. □

4.3.1.3 Définition. Soient les diagrammes de programme P_1 et P_2, tels que P_1 est une relation sur $T \times S_1 \times V_1$ et P_2 est une relation sur $T \times S_2 \times V_2$ (l'ensemble des états de P_i est $T \times S_i$ et l'ensemble de ses points de contrôle est V_i, pour $i = 1, 2$). Les composantes S_1 et S_2 sont les composantes propres à P_1 et P_2, respectivement, alors que T est une composante partagée. Soit le produit direct (π_1, π_2), où π_1 et π_2 sont les projections

$$\pi_1 : T \times S_1 \times S_2 \times V_1 \times V_2 \rightarrow T \times S_1 \times V_1$$

$$\pi_2 : T \times S_1 \times S_2 \times V_1 \times V_2 \rightarrow T \times S_2 \times V_2$$

La *composition parallèle* de P_1 et P_2, notée $P_1 \parallel P_2$, est la relation sur $T \times S_1 \times S_2 \times V_1 \times V_2$ définie par $P_1 \parallel P_2 = \pi_1 P_1 \hat{\pi}_1 \cap \pi_2 P_2 \hat{\pi}_2 \cup \pi_1 \hat{\pi}_1 \cap \pi_2 P_2 \hat{\pi}_2 \cup \pi_1 P_1 \hat{\pi}_1 \cap \pi_2 \hat{\pi}_2$ □

L'algorithme du produit parallèle présenté ci-dessous utilise cette définition et les algorithmes subséquents utilisent des variantes de cette définition. Il est à noter qu'aucune simplification n'est effectuée dans les algorithmes associés aux différents produits parallèles. Chacun de ces algorithmes ne fait qu'appeler ceux définis à la section 4.2. Puisque la transformation constructive est effectuée pour chacun des opérateurs élémentaires, le résultat final sera automatiquement simplifié.

4.3.2 Le produit parallèle

Le produit parallèle de deux matrices est un opérateur complexe tel que défini en 4.3.1.3. Pour plus d'explications sur cet opérateur, le lecteur est invité à consulter Chaib-draa *et al.* [8].

Le produit parallèle de deux matrices ERSV s'effectue selon l'algorithme qui suit. Afin de simplifier la notation, la dimension des matrices π_1, π_2, $x1$, $x2$, $x3$, $x4$ et $x5$ n'est pas inscrite. Ces matrices sont de dimension $(k \times k)$ où k est le produit de i par j.

MPARAL($A_{i\times i}$;$B_{j\times j}$;π_1;π_2)

1. $x1$:= MPROD(π_1;MPROD($A_{i\times i}$;MINV(π_1)))
2. $x2$:= MPROD(π_2;MPROD($B_{j\times j}$;MINV(π_2)))
3. $x3$:= MPROD(π_1;MINV(π_1))
4. $x4$:= MPROD(π_2;MINV(π_2))
5. $x5$:= MUNION(MINTER($x1$;$x4$);MINTER($x3$;$x2$))
6. $résultat_{k\times k}$:= MUNION(MINTER($x1$;$x2$);$x5$)
7. RETOURNER $résultat_{k\times k}$

4.3.3 Le produit parallèle synchrone

Le produit parallèle synchrone de deux matrices est un opérateur complexe défini comme suit :

$$P_1 \parallel_S P_2 = \pi_1 P_1 \widehat{\pi}_1 \cap \pi_2 P_2 \widehat{\pi}_2$$

où π_1 et π_2 sont les matrices des projections rattachées à P_1 et P_2 et satisfaisant les propriétés de la définition 4.3.1.3.

Le produit parallèle synchrone de deux matrices ERSV s'effectue selon l'algorithme qui suit. Afin de simplifier la notation, la dimension des matrices π_1, π_2, $x1$ et $x2$ n'est pas inscrite. Ces matrices sont de dimension $(k \times k)$ où k est le produit de i par j.

MPARALS($A_{i\times i}$;$B_{j\times j}$;π_1;π_2)

1. $x1$:= MPROD(π_1;MPROD(A;INV(π_1)))
2. $x2$:= MPROD(π_2;MPROD(B;INV(π_2)))
3. $résultat_{k\times k}$:= MINTER($x1$;$x2$)
4. RETOURNER $résultat_{k\times k}$

4.3.4 Le produit parallèle entrelaçant

Le produit parallèle entrelaçant [35, 36] de deux matrices est un opérateur complexe défini comme suit :

$$P_1 \parallel_E P_2 = \pi_1 P_1 \widehat{\pi}_1 \cap \pi_{P2} \widehat{\pi}_{P2} \cup \pi_{P1} \widehat{\pi}_{P1} \cap \pi_2 P_2 \widehat{\pi}_2$$

où π_1 et π_2 sont les matrices des projections rattachées à P_1 et P_2 et satisfaisant les propriétés de la définition 4.3.1.3, tandis que π_{P1} et π_{P2} sont les matrices des projections rattachées à P_1 et P_2 et satisfaisant les propriétés suivantes :

$$\pi_{P1} : T \times S_1 \times S_2 \times V_1 \times V_2 \rightarrow S_1 \times V_1$$

$$\pi_{P2} : T \times S_1 \times S_2 \times V_1 \times V_2 \rightarrow S_2 \times V_2$$

Le produit parallèle entrelaçant de deux matrices ERSV s'effectue selon l'algorithme qui suit. Afin de simplifier la notation, la dimension des matrices π_1, π_2, π_{P1}, π_{P2}, $x1$, $x2$, $x3$ et $x4$ n'est pas inscrite. Ces matrices sont de dimension $(k \times k)$ où k est le produit de i par j.

MPARALE($A_{i\times i}$;$B_{j\times j}$;π_1;π_2;π_{P1};π_{P2})

1. $x1$:= MPROD(π_1;MPROD(A;MINV(π_1)))
2. $x2$:= MPROD(π_2;MPROD(B;MINV(π_2)))
3. $x3$:= MPROD(π_{P1};MINV(π_{P1}))
4. $x4$:= MPROD(π_{P2};MINV(π_{P2}))
5. $résultat_{k\times k}$:= MUNION(MINTER($x1$;$x4$);MINTER($x3$;$x2$))
6. RETOURNER $résultat_{k\times k}$

4.3.5 Le produit parallèle entrelaçant particulier

Le produit parallèle entrelaçant particulier [35, 36] est défini comme suit :

$$P_1 \parallel_{EP} P_2 = \pi_1 P_1 \widehat{\pi}_1 \cap \pi_2 \widehat{\pi}_2 \cup \pi_1 \widehat{\pi}_1 \cap \pi_2 P_2 \widehat{\pi}_2$$

où π_1 et π_2 sont les matrices des projections rattachées à P_1 et P_2 et satisfaisant les propriétés de la définition 4.3.1.3.

Le produit parallèle entrelaçant particulier de deux matrices ERSV s'effectue selon l'algorithme qui suit. Afin de simplifier la notation, la dimension des matrices π_1, π_2, π_{P1}, π_{P2}, $x1$, $x2$, $x3$ et $x4$ n'est pas inscrite. Ces matrices sont de dimension $(k \times k)$ où k est le produit de i par j.

MPARALEP($A_{i\times i}$;$B_{j\times j}$;π_1;π_2)

1. $x1$:= MPROD(π_1;MPROD(A;MINV(π_1)))
2. $x2$:= MPROD(π_2;MPROD(B;MINV(π_2)))
3. $x3$:= MPROD(π_1;MINV(π_1))
4. $x4$:= MPROD(π_2;MINV(π_2))
5. $résultat_{k\times k}$:= MUNION(MINTER($x1$;$x4$);MINTER($x3$;$x2$))
6. RETOURNER $résultat_{k\times k}$

Les algorithmes associés aux opérations élémentaires ainsi qu'aux produits parallèles ont été implantés dans un prototype dont les fonctionnalités d'utilisation sont présentées dans le chapitre suivant.

Chapitre 5

Présentation du prototype

5.1 Introduction

SR∘Ψlab est le nom donné à un prototype développé pour être utilisé dans l'environnement du logiciel Ψlab (ou *Scilab*). Ces fonctions ajoutent à l'environnement de Ψlab la capacité d'effectuer du calcul symbolique sur des expressions relationnelles sans variables (ERSV). Plus particulièrement, ces fonctions manipulent des matrices de ERSV, *i.e.* des matrices dont le contenu de chaque cellule est une ERSV.

Les sections suivantes décrivent les fonctionnalités d'utilisation du prototype développé à l'aide du logiciel Ψlab. La lecture des chapitres 1 et 2, ainsi que les sections 4.1 à 4.2 du manuel « Introduction to Ψlab », fourni avec le logiciel, est recommandée avant d'utiliser la librairie *SR∘Ψlab*. Le lecteur pourra alors se familiariser avec le fonctionnement de Ψlab ainsi qu'aux fonctions élémentaires dont la fonction `who` qui affiche la liste des variables, ainsi que la fonction `clear` qui élimine toutes les variables définies par l'usager.

5.2 Démarrage de l'application

Sous UNIX, dans un environnement avec fenêtrage, le logiciel Ψlab est activé en tapant la commande `scilab` (en minuscule). Par la suite, toutes les commandes se font à l'intérieur de la fenêtre de Ψlab. Pour utiliser la librairie *SR∘Ψlab*, il faut d'abord posséder dans son

répertoire une copie du fichier *SymRel.bin* qui contient les fonctions compilées et activer la librairie $\mathcal{SR}$∘Ψ*lab* de l'une ou l'autre des façons suivantes :

- au « prompt » de Ψlab, tapez directement la commande `load("SymRel.bin")`

  ou

- chargez le fichier *SymRel.bin* en utilisant le menu **File Operations** de la fenêtre Ψlab.

Les fonctions pour effectuer le calcul symbolique sur des ERSV sont maintenant disponibles.

En l'absence d'un environnement avec fenêtrage, le logiciel Ψlab est activé en tapant la commande `scilab -nw` .

5.3 Fermeture de l'application

Pour terminer, il suffit de taper la commande `quit` ou bien de cliquer sur le menu **File Quit**. Les données créées par l'utilisateur ne sont cependant pas conservées. Si l'utilisateur désire conserver les données qu'il a créées, il peut procéder de différentes façons (pour plus de détails, voir « Introduction to Ψlab »). Une façon simple consiste à sauvegarder tout l'environnement de travail, *i.e.* fonctions et données[1]. Pour ce faire, l'utilisateur doit taper la commande `save("`*nom*`.bin")` , *nom* correspondant à un nom choisi par l'utilisateur sous lequel sera conservé l'environnement de travail. Attention cependant, Ψlab donne un message d'erreur si le fichier existe déjà. Il faut donc détruire ce fichier avant de faire la commande `save` si ce n'est pas la première fois qu'une sauvegarde est faite sous ce nom. Pour contourner ce problème, la procédure suivante est suggérée :

1. `save("SRscilab.bak")`
2. `unix("rm `*nom*`.bin")`
3. `save("`*nom*`.bin")`
4. `unix("rm SRscilab.bak")`

Par la suite, le fichier *nom.bin* devra être chargé au lieu de *SymRel.bin*. Si, par erreur, une fonction de la librairie $\mathcal{SR}$∘Ψ*lab* est modifiée et qu'on désire la retrouver, il n'y a qu'à faire la commande `load("SymRel.bin")` . Ce deuxième chargement consécutif n'affecte que les variables et fonctions de la librairie.

[1] Pour les familiers du langage APL, cela est similaire à sauvegarder le bloc de travail.

5.4 Fonction d'aide - *SRinfo*

Une fonction d'aide est incluse dans la librairie $\mathcal{SR}\circ\Psi lab$. Elle permet d'obtenir le nom de toutes les fonctions et opérations disponibles avec la librairie, ainsi que le mode d'utilisation pour chacune de ces fonctions. Pour appeler cette fonction d'aide, il suffit de taper la commande `SRinfo()` et de suivre les indications.

Le menu **Help** de Ψlab est aussi très utile pour obtenir de l'information sur les fonctions et possibilités du logiciel Ψlab.

5.5 Fonctions élémentaires et le type SRE

Afin de manipuler les ERSV (expressions relationnelles sans variables), un nouveau type de données a été créé. Il s'agit d'une liste de deux éléments, dont le premier est la chaîne de caractères *sre* (d'après Symbolisme RElationnel) qui sert comme identifiant pour le nouveau type de données. Le deuxième élément est une matrice caractère de dimension $(m \times n)$ pour $m, n \geq 1$, dont chaque cellule contient une ERSV. La gestion de ce type particulier est prise en charge par les fonctions de la librairie. C'est aussi le seul type accepté par ces mêmes fonctions. Pour simplifier, les données de ce type seront appelées des matrices SRE.

5.5.1 Expressions relationnelles sans variables (ERSV)

Ψlab ne permet pas l'utilisation des symboles usuels de l'algèbre relationnelle. Des caractères ASCII sont donc utilisés pour les représenter, à savoir :

relation universelle
: La relation universelle que l'on identifie souvent par L est représentée par le chiffre 1. Ainsi $A \cup L$ s'écrit `A+1`

relation vide
: La relation vide que l'on identifie par O ou bien par le symbole O est représentée par le chiffre 0. Ainsi $A \cup O$ s'écrit `A+0`

relation identité
: La relation identité que l'on identifie par I est représentée par la lettre i majuscule[2]. Ainsi $A \cup I$ s'écrit `A+`I

[2]Pour éviter toute confusion avec le chiffre 1, le i majuscule sera écrit en italique dans les exemples illustrant l'utilisation de $\mathcal{SR}\circ\Psi lab$.

union

L'union de 2 relations, telles A et B, que l'on indique habituellement par $A \cup B$, est représentée par `A+B`

intersection

L'intersection $A \cap B$ est représentée par `A.B`

produit

Le produit relationnel (ou la composition) que l'on indique par $A;B$, $A \circ B$, ou AB devient `A*B`

inverse

L'inverse (ou la transposée) que l'on indique par A^{-1}, $\widehat{A}$ ou A^T est représenté par `A^`

complément

Le complément $\overline{A}$ est représenté par `A'`

résidu à gauche

Le résidu à gauche A/B est représenté par `A/B`

résidu à droite

Le résidu à droite $A\backslash B$ est représenté par `A\B`

Le tableau suivant résume les notations :

Relation universelle	1
Relation vide	0
Relation identité	I
Union de 2 relations	+
Intersection de 2 relations	.
Produit de 2 relations	*
Inverse (transposée) d'une relation	^
Complément d'une relation	'
Résidu à gauche de 2 relations	/
Résidu à droite de 2 relations	\

L'utilisateur a la possibilité de modifier cette convention (voir section 5.9).

En algèbre relationnelle, les lettres identifient les diverses relations. Avec $\mathcal{SR}\circ\Psi lab$, le nom des relations doit répondre aux règles de construction suivantes :

- le nom doit être composé de caractères majuscules et/ou minuscules ;

- le premier caractère doit être alphabétique ;
- les autres caractères peuvent être alphabétiques ou numériques ;
- il n'y a pas de longueur maximum pour un nom.

Par exemple, les noms suivants : `a`, `A`, `ABC`, `projA`, `proja`, `w1` et `W2` sont tous des noms valides et différents.

À l'intérieur d'une ERSV, les parenthèses s'utilisent de la façon habituelle. Cependant, la priorité des opérateurs n'est pas tout à fait la même. En algèbre relationnelle, l'intersection est traitée avant l'union alors qu'avec $\mathcal{SR}\circ\Psi lab$, ces deux opérateurs ont une priorité identique. Aussi, pour éviter toute mauvaise interprétation, $A \cap B \cup C$ sera écrit `(A.B)+C` et l'expression $A \cap (B \cup C)$ sera écrite `A.(B+C)` .

Le système fait une certaine gestion des parenthèses. Par exemple, le complément de `(A+B)'` donne `A+B` et non pas `(A+B)` . Dans certains cas, l'utilisateur peut vouloir conserver les parenthèses. Pour ce faire, il n'a qu'à utiliser les parenthèses carrées. Ainsi, le complément de `[A+B]'` donne `[A+B]` et non pas `A+B` .

5.5.2 Création - *SRcreer*

Cette fonction sert à créer des matrices SRE, *i.e.* des matrices symboliques dont chaque cellule est une ERSV.

Appels de la fonction :

```
var = SRcreer(li,co,"texte")
var = SRcreer(li,co)
```

Paramètres :

`li` : nombre de lignes
`co` : nombre de colonnes
`texte` : expression à insérer dans chaque cellule
`var` : variable qui reçoit le résultat

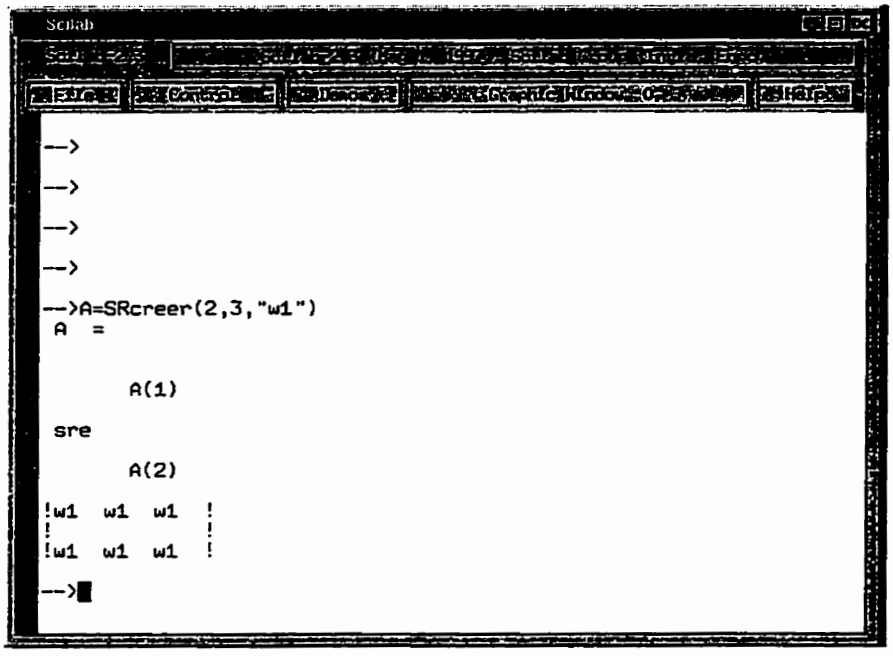
Exemples :

La commande `A=SRcreer(2,3,"w1")` assigne à A la matrice suivante :

$$\begin{vmatrix} w1 & w1 & w1 \\ w1 & w1 & w1 \end{vmatrix}$$

tandis que l'exécution de la commande `A=SRcreer(2,3)` provoque un appel à l'éditeur de matrices pour permettre la saisie des entrées.

Affichage à l'écran :



5.5.3 Édition de matrices SRE - *SRedit*

Ψlab possède un éditeur de matrices qui a été adapté pour faire l'édition des matrices SRE. Cela permet d'effectuer la modification ou la saisie à l'aide d'une fenêtre quadrillée. Cet éditeur est utilisable seulement dans un environnement avec fenêtrage.

L'éditeur permet l'édition de matrices de très grande dimension. Toutefois, pour obtenir un affichage adéquat, le gestionnaire de fenêtre doit permettre le chevauchement de la fenêtre d'édition sur plus d'un écran virtuel.

L'éditeur est appelé directement par la fonction *SRedit* ou, au besoin, par la fonction *SRcreer*. Pour modifier ou saisir le contenu d'une cellule, il faut d'abord déplacer le curseur dans la cellule en question. Lorsque la saisie est terminée, il suffit de sélectionner le choix **OK**.

Appel de la fonction :

```
var = SRedit(mat)
```

Paramètres :

`mat` : matrice symbolique à modifier

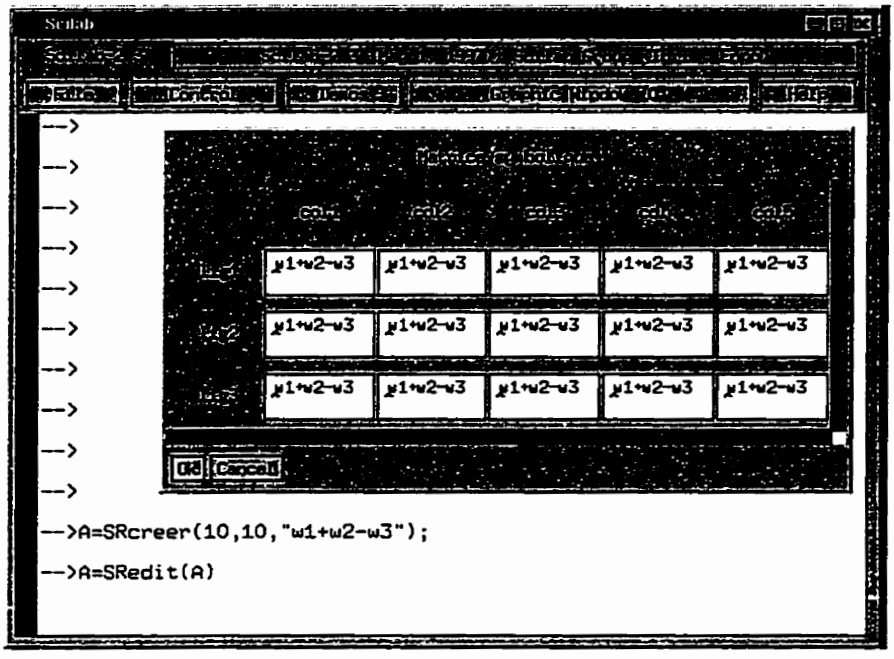
`var` : variable qui reçoit le résultat

Exemples :

Pour modifier le contenu de *A*, on tape la commande `A=SRedit(A)` .

Pour créer *B* à partir de modifications à la matrice *A*, il suffit de taper la commande `B=SRedit(A)` .

Affichage à l'écran :



5.6 Les opérateurs relationnels de base

Les opérateurs relationnels de base sont en fait des fonctions. Ψlab offre la possibilité de redéfinir l'appel de fonctions à l'aide des opérateurs mathématiques. Cette opportunité à été utilisée afin de simplifier le plus possible l'écriture des expressions matricielles. Les opérateurs disponibles dans Ψlab ont permis de redéfinir les opérations matricielles suivantes :

Union de 2 matrices	+
Intersection de 2 matrices	-
Produit de 2 matrices	*
Inverse (transposée) d'une matrice	^(-1)
Complément d'une matrice	'
Résidu à gauche de 2 matrices	/
Résidu à droite de 2 matrices	\

La priorité d'exécution de ces opérateurs est définie par Ψlab et n'est pas tout à fait la même que celle de l'algèbre relationnelle. L'exécution se fait de la gauche vers la droite selon les priorités suivantes : l'inverse et le complément ont la priorité la plus élevée; viennent ensuite le produit et les résidus à gauche et à droite; et finalement l'union et l'intersection possèdent la priorité la plus basse.

5.6.1 L'union - *SRunion*

L'union de 2 matrices SRE s'effectue à l'aide de la fonction *SRunion* ou de l'opérateur plus (+). Les matrices doivent être de même dimension puisque l'union se fait position par position.

Appels de la fonction :

```
var = m1 + m2
var = SRunion(m1,m2)
```

Paramètres :

`m1` : première matrice symbolique
`m2` : deuxième matrice symbolique
`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \end{vmatrix}$ et $B = \begin{vmatrix} a1 & x & y \\ 1 & 0 & I \end{vmatrix}$

alors l'exécution de la commande `C=A+B` ou bien `C=SRunion(A,B)` assigne à C la matrice suivante :

$$\begin{vmatrix} a1 & a2+x & a3+y \\ 1 & a5 & a6+I \end{vmatrix}$$

5.6.2 L'intersection - *SRinter*

L'intersection de 2 matrices SRE se fait à l'aide de la fonction *SRinter* ou de l'opérateur moins (-). Les matrices doivent être de même dimension puisque l'intersection se fait position par position.

Appels de la fonction :

```
var = m1 - m2
var = SRinter(m1,m2)
```

Paramètres :

`m1` : première matrice symbolique
`m2` : deuxième matrice symbolique
`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \end{vmatrix}$ et $B = \begin{vmatrix} a1 & x & y \\ 1 & 0 & I \end{vmatrix}$

alors l'exécution de la commande `C=A-B` ou bien `C=SRinter(A,B)` assigne à C la matrice suivante :

$$\begin{vmatrix} a1 & a2 \cdot x & a3 \cdot y \\ a4 & 0 & a6 \cdot I \end{vmatrix}$$

5.6.3 Le produit - *SRproduit*

Le produit de 2 matrices SRE se fait à l'aide de la fonction *SRproduit* ou de l'opérateur produit (*). Ce produit est similaire au produit matriciel, sauf que l'addition des produits est remplacée par l'union des compositions. Les dimensions des matrices doivent donc être adéquates, *i.e.* que le nombre de colonnes de la première matrice doit être identique au nombre de lignes de la seconde matrice pour que le produit soit possible.

Appels de la fonction :

```
var = m1 * m2
var = SRproduit(m1,m2)
```

Paramètres :

`m1` : première matrice symbolique
`m2` : deuxième matrice symbolique
`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \end{vmatrix}$ et $B = \begin{vmatrix} a1 & 1 \\ x & 0 \\ y & I \end{vmatrix}$

alors l'exécution de la commande `C=A*B` ou bien `C=SRproduit(A,B)` assigne à *C* la matrice suivante :

$$\begin{vmatrix} a1*a1+a2*x+a3*y & a1*1+a3 \\ a4*a1+a5*x+a6*y & a4*1+a6 \end{vmatrix}$$

5.6.4 L'inverse - *SRinv*

L'inverse d'une matrice SRE s'obtient avec la fonction *SRinv* ou en élevant la matrice à la puissance (-1).

Appels de la fonction :

```
var = SRinv(mat)
var = mat^(-1)
```

Paramètres :

`mat` : matrice symbolique à inverser

`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} a1 & a2 & a3 \\ 1 & 0 & I \end{vmatrix}$

alors l'exécution de la commande `C=A^(-1)` ou bien de la commande `C=SRinv(A)` assigne à *C* l'inverse de *A*, *i.e.* la matrice suivante :

$$\begin{vmatrix} a1^{} & 1 \\ a2^{} & 0 \\ a3^{} & I \end{vmatrix}$$

5.6.5 Le complément - *SRcomp*

Le complément d'une matrice SRE s'obtient avec la fonction *SRcomp* ou à l'aide du signe apostrophe (').

Appels de la fonction :

```
var = mat'
var = SRcomp(mat)
```

Paramètres :

`mat` : matrice symbolique à complémenter

`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} a1 & a2 & a3 \\ 1 & I & 0 \end{vmatrix}$

alors l'exécution de la commande `C=A'` ou bien de la commande `C=SRcomp(A)` assigne à C la matrice suivante :

$$\begin{vmatrix} a1' & a2' & a3' \\ 0 & I' & 1 \end{vmatrix}$$

5.6.6 La puissance

Il est possible d'élever une matrice carrée SRE à la puissance n. Pour cela, il suffit d'utiliser le signe exposant (^) suivi d'un entier. Il est à noter que si l'entier est négatif, l'inverse de la matrice est alors utilisé.

Appel de la fonction :

```
var = mat^n
```

Paramètres :

`n` : valeur de l'exposant (à mettre entre parenthèses si négatif)

`mat` : matrice carrée symbolique

`var` : variable qui reçoit le résultat

Exemples :

La commande `C=A^3` est équivalente à `C=A*A*A` .

La commande `C=A^(-1)` permet d'obtenir l'inverse de A et la commande `A=C^(-2)` est l'équivalent de `A=SRinv(C)*SRinv(C)` .

5.6.7 Le résidu à gauche - *SRresg*

Le résidu à gauche de 2 matrices SRE s'obtient avec la fonction *SRresg* ou l'opérateur barre oblique (/). Les dimensions des matrices doivent être adéquates, *i.e.* que le nombre de colonnes de la première matrice doit être identique au nombre de colonnes de la seconde matrice pour que le résidu à gauche soit possible.

Appels de la fonction :

```
var = m1 / m2
var = SRresg(m1,m2)
```

Paramètres :

`m1` : première matrice symbolique
`m2` : deuxième matrice symbolique
`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \end{vmatrix}$ et $B = \begin{vmatrix} a1 & x & y \\ 1 & 0 & I \end{vmatrix}$

alors l'exécution de la commande `C=A/B` ou bien `C=SRresg(A,B)` assigne à C la matrice suivante :

$$\begin{vmatrix} a1/a1 \,.\, a2/x \,.\, a3/y & a1/1 \,.\, a3 \\ a4/a1 \,.\, a5/x \,.\, a6/y & a4/1 \,.\, a6 \end{vmatrix}$$

5.6.8 Le résidu à droite - *SRresd*

Le résidu à droite de 2 matrices SRE s'obtient avec la fonction *SRresd* ou l'opérateur barre oblique inversée (\). Les dimensions des matrices doivent être adéquates, *i.e.* que le nombre de lignes de la première matrice doit être identique au nombre de lignes de la seconde matrice pour que le résidu à droite soit possible.

Appels de la fonction :

```
var = m1 \ m2
var = SRresd(m1,m2)
```

Paramètres :

`m1` : première matrice symbolique
`m2` : deuxième matrice symbolique
`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \left| \begin{array}{ccc} a & a & a \\ 1 & 1 & 1 \end{array} \right|$ et $B = \left| \begin{array}{ccc} a & x & y \\ 1 & 0 & I \end{array} \right|$

alors l'exécution de la commande `C=A\B` ou bien `C=SRresd(A,B)` assigne à C la matrice suivante :

$$\left| \begin{array}{ccc} a\backslash a & 0 & a\backslash y \,.\, 1\backslash I \\ a\backslash a & 0 & a\backslash y \,.\, 1\backslash I \\ a\backslash a & 0 & a\backslash y \,.\, 1\backslash I \end{array} \right|$$

5.7 Les opérateurs relationnels évolués

Chaque opérateur relationnel évolué représente un ensemble plus ou moins complexe d'opérations relationnelles élémentaires (union, intersection, produit, complément et inverse) qui sont appliquées sur les matrices SRE données comme arguments.

5.7.1 Le résidu à gauche développé - *SRresidug*

Le résidu à gauche développé de 2 matrices SRE s'obtient avec la fonction *SRresidug*. Le résidu à gauche développé est un opérateur composé à partir d'opérations relationnelles de base. Ainsi, la commande `SRresidug(A,B)` est l'équivalent de $\overline{\overline{A} \circ \hat{B}}$.

Appel de la fonction :

```
var = SRresidug(m1,m2)
```

Paramètres :

`m1` : première matrice symbolique
`m2` : deuxième matrice symbolique
`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \left| \begin{array}{ccc} a1 & a2 & a3 \\ a4 & a5 & a6 \end{array} \right|$ et $B = \left| \begin{array}{ccc} a1 & x & y \\ 1 & 0 & I \end{array} \right|$

alors l'exécution de la commande `C=SRresidug(A,B)` assigne à C la matrice suivante :

$$\left| \begin{array}{cc} (a1' * a1\hat{} + a2' * x\hat{} + a3' * y\hat{})' & (a1' * 1 + a3')' \\ (a4' * a1\hat{} + a5' * x\hat{} + a6' * y\hat{})' & (a4' * 1 + a6')' \end{array} \right|$$

5.7.2 Le résidu à droite développé - *SRresidud*

Le résidu à droite développé de 2 matrices SRE s'obtient avec la fonction *SRresidud*. Le résidu à droite développé est un opérateur composé à partir d'opérations relationnelles de base. Ainsi, la commande `SRresidud(A,B)` est l'équivalent de $\overline{\widehat{A} \circ \overline{B}}$.

Appel de la fonction :

```
var = SRresidud(m1,m2)
```

Paramètres :

`m1` : première matrice symbolique
`m2` : deuxième matrice symbolique
`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} a & a & a \\ 1 & 1 & 1 \end{vmatrix}$ et $B = \begin{vmatrix} a & x & y \\ 1 & 0 & I \end{vmatrix}$

alors l'exécution de la commande `C=SRresidud(A,B)` assigne à C la matrice suivante :

$$\begin{vmatrix} (a\hat{} * a')' & 0 & (a\hat{} * y' + 1 * I')' \\ (a\hat{} * a')' & 0 & (a\hat{} * y' + 1 * I')' \\ (a\hat{} * a')' & 0 & (a\hat{} * y' + 1 * I')' \end{vmatrix}$$

5.7.3 Le produit booléen - *SRprodBool*

Le produit booléen de 2 matrices SRE se fait à l'aide de la fonction *SRprodBool*. Le produit booléen ressemble au produit, sauf qu'il fait l'union des intersections au lieu de l'union des compositions. Les dimensions des matrices doivent être adéquates, *i.e.* que le nombre de colonnes de la première matrice doit être identique au nombre de lignes de la seconde matrice pour que le produit booléen soit possible.

Appel de la fonction :

```
var = SRprodBool(m1,m2)
```

Paramètres :

`m1` : première matrice symbolique
`m2` : deuxième matrice symbolique
`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \end{vmatrix}$ et $B = \begin{vmatrix} a1 & 1 \\ 0 & 0 \\ 1 & a6 \end{vmatrix}$

alors l'exécution de la commande `C=SRprodBool(A,B)` assigne à C la matrice suivante :

$$\begin{vmatrix} a1 + a3 & a1 + (a3.a6) \\ (a4.a1) + a6 & a4 + a6 \end{vmatrix}$$

5.7.4 L'inverse booléen - *SRinvBool*

L'inverse booléen d'une matrice SRE a été créé pour accompagner le produit booléen, mais il n'est pas à proprement parler un opérateur complexe puisqu'il ne fait que transposer la matrice.

Appel de la fonction :

```
var = SRinvBool(mat)
```

Paramètres :

`mat` : matrice symbolique à inverser

`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \end{vmatrix}$

alors l'exécution de la commande `C=SRinvBool(A)` assigne à C la matrice suivante :

$$\begin{vmatrix} a1 & a4 \\ a2 & a5 \\ a3 & a6 \end{vmatrix}$$

5.7.5 Le produit parallèle - *SRparal*

Le produit parallèle de 2 matrices SRE s'obtient avec la fonction *SRparal*. Ce produit parallèle est un opérateur complexe composé de plusieurs opérations élémentaires que l'on définit comme suit :

$$P_1 \parallel P_2 = \pi_1 P_1 \widehat{\pi_1} \cap \pi_2 P_2 \widehat{\pi_2} \cup \pi_1 \widehat{\pi_1} \cap \pi_2 P_2 \widehat{\pi_2} \cup \pi_1 P_1 \widehat{\pi_1} \cap \pi_2 \widehat{\pi_2}$$

ce qui se traduit par l'instruction suivante en $\mathcal{SR}\circ\Psi lab$[3] :

```
 pi1*P1*pi1^(-1) - pi2*P2*pi2^(-1)
+(pi1*pi1^(-1) - pi2*P2*pi2^(-1))
+(pi1*P1*pi1^(-1) - pi2*pi2^(-1))
```

où `pi1` et `pi2` sont des projections automatiquement calculées par la fonction *SRparal* (voir fonctions utilitaires *SRpi1* et *SRpi2* aux sections 5.8.3 et 5.8.4). Pour plus d'explications sur cet opérateur, le lecteur est invité à consulter l'article de Chaib-draa *et al.* [8].

Appel de la fonction :

```
var = SRparal(m1,m2,"textePI1","textePI2")
```

Paramètres :

`m1` : première matrice symbolique

`m2` : deuxième matrice symbolique

`textePI1` : expression à insérer dans la matrice symbolique `pi1`

`textePI2` : expression à insérer dans la matrice symbolique `pi2`

`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} 0 & aI \\ Ib & 0 \end{vmatrix}$ et $B = \begin{vmatrix} 0 & Ic \\ aI & 0 \end{vmatrix}$

alors l'exécution de la commande `C=SRparal(A,B,"w1","w2")` assigne à C la matrice suivante :

$$\begin{vmatrix} \text{colonne 1} & \text{colonne 2} \\ 0 & w1 * w1\hat{} \,.\, w2 * Ic * w2\hat{} \\ w1 * w1\hat{} \,.\, w2 * aI * w2\hat{} & 0 \\ w1 * Ib * w1\hat{} \,.\, w2 * w2\hat{} & w1 * Ib * w1\hat{} \,.\, w2 * Ic * w2\hat{} \\ w1 * Ib * w1\hat{} \,.\, w2 * aI * w2\hat{} & w1 * Ib * w1\hat{} \,.\, w2 * w2\hat{} \end{vmatrix}$$

$$\begin{vmatrix} \text{colonne 3} & \text{colonne 4} \\ w1 * aI * w1\hat{} \,.\, w2 * w2\hat{} & w1 * aI * w1\hat{} \,.\, w2 * Ic * w2\hat{} \\ w1 * aI * w1\hat{} \,.\, w2 * aI * w2\hat{} & w1 * aI * w1\hat{} \,.\, w2 * w2\hat{} \\ 0 & w1 * w1\hat{} \,.\, w2 * Ic * w2\hat{} \\ w1 * w1\hat{} \,.\, w2 * aI * w2\hat{} & 0 \end{vmatrix}$$

[3] L'évaluation de gauche à droite fait en sorte qu'il n'est pas nécessaire de mettre entre parenthèses la première ligne de l'instruction.

5.7.6 Le produit parallèle synchrone - *SRparalS*

Le produit parallèle synchrone de 2 matrices SRE s'obtient avec la fonction *SRparalS*. Ce produit parallèle est un opérateur complexe composé de plusieurs opérations élémentaires que l'on définit comme suit :

$$P_1 \|_S P_2 = \pi_1 P_1 \widehat{\pi_1} \cap \pi_2 P_2 \widehat{\pi_2}$$

ce qui se traduit par l'instruction suivante en $\mathcal{SR}\circ\Psi lab$:

```
pi1*P1*pi1^(-1) - pi2*P2*pi2^(-1)
```

où `pi1` et `pi2` sont des projections automatiquement calculées par la fonction *SRparalS* (voir fonctions utilitaires *SRpi1* et *SRpi2* aux sections 5.8.3 et 5.8.4).

Appel de la fonction :

```
var = SRparalS(m1,m2,"textePI1","textePI2")
```

Paramètres :

`m1` : première matrice symbolique
`m2` : deuxième matrice symbolique
`textePI1` : expression à insérer dans la matrice symbolique `pi1`
`textePI2` : expression à insérer dans la matrice symbolique `pi2`
`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} 0 & aI \\ Ib & 0 \end{vmatrix}$ et $B = \begin{vmatrix} 0 & Ic \\ aI & 0 \end{vmatrix}$

alors l'exécution de la commande `C=SRparalS(A,B,"w1","w2")` assigne à C la matrice suivante :

colonne 1	colonne 2
0	0
0	0
0	$w1 * Ib * w1\hat{} \,.\, w2 * Ic * w2\hat{}$
$w1 * Ib * w1\hat{} \,.\, w2 * aI * w2\hat{}$	0

colonne 3	colonne 4
0	$w1 * aI * w1\hat{} . w2 * Ic * w2\hat{}$
$w1 * aI * w1\hat{} . w2 * aI * w2\hat{}$	0
0	0
0	0

5.7.7 Le produit parallèle entrelaçant - *SRparalE*

Le produit parallèle entrelaçant de 2 matrices SRE s'obtient avec la fonction *SRparalE*. Ce produit parallèle est un opérateur complexe composé de plusieurs opérations élémentaires que l'on définit comme suit :

$$P_1 \|_E P_2 = \pi_1 P_1 \widehat{\pi}_1 \cap \pi_{P2} \widehat{\pi}_{P2} \cup \pi_{P1} \widehat{\pi}_{P1} \cap \pi_2 P_2 \widehat{\pi}_2$$

ce qui se traduit par l'instruction suivante en $\mathcal{SR} \circ \Psi lab$:

```
pi1*P1*pi1^(-1) - pip2*pip2^(-1)
+(pip1*pip1^(-1) - pi2*P2*pi2^(-1))
```

où `pi1` , `pi2` , `pip1` et `pip2` sont des projections automatiquement calculées par la fonction *SRparalE* (voir fonctions utilitaires *SRpi1* et *SRpi2* aux sections 5.8.3 et 5.8.4). Pour plus d'explications sur cet opérateur, le lecteur est invité à consulter Khédri [36].

Appel de la fonction :

```
var=SRparalE(m1,m2,"textePI1","textePI2","textePIP1",
     "textePIP2")
```

Paramètres :

`m1` : première matrice symbolique
`m2` : deuxième matrice symbolique
`textePI1` : expression à insérer dans la matrice symbolique `pi1`
`textePI2` : expression à insérer dans la matrice symbolique `pi2`
`textePIP1` : expression à insérer dans la matrice symbolique `pip1`
`textePIP2` : expression à insérer dans la matrice symbolique `pip2`
`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} 0 & aI \\ Ib & 0 \end{vmatrix}$ et $B = \begin{vmatrix} 0 & Ic \\ aI & 0 \end{vmatrix}$

alors la commande `C=SRparalE(A,B,"w1","w2","wp1","wp2")`
assigne à C la matrice suivante :

colonne 1	colonne 2	colonne 3	colonne 4
0	$wp1 * wp1^{\frown} . w2 * Ic * w2^{\frown}$	$w1 * aI * w1^{\frown} . wp2 * wp2^{\frown}$	0
$wp1 * wp1^{\frown} . w2 * aI * w2^{\frown}$	0	0	$w1 * aI * w1^{\frown} . wp2 * wp2^{\frown}$
$w1 * Ib * w1^{\frown} . wp2 * wp2^{\frown}$	0	0	$wp1 * wp1^{\frown} . w2 * Ic * w2^{\frown}$
0	$w1 * Ib * w1^{\frown} . wp2 * wp2^{\frown}$	$wp1 * wp1^{\frown} . w2 * aI * w2^{\frown}$	0

5.7.8 Le produit parallèle entrelaçant particulier - *SRparalEP*

Le produit parallèle entrelaçant particulier de 2 matrices SRE s'obtient avec la fonction *SRparalEP*. Il est un cas particulier du produit parallèle entrelaçant. Ce produit parallèle est un opérateur complexe composé de plusieurs opérations élémentaires que l'on définit comme suit :

$$P_1 \|_{EP} P_2 = \pi_1 P_1 \widehat{\pi_1} \cap \pi_2 \widehat{\pi_2} \cup \pi_1 \widehat{\pi_1} \cap \pi_2 P_2 \widehat{\pi_2}$$

ce qui se traduit par l'instruction suivante en $\mathcal{SR}\circ\Psi lab$:

```
pi1*P1*pi1^(-1) - pi2*pi2^(-1)
+(pi1*pi1^(-1) - pi2*P2*pi2^(-1))
```

où `pi1` et `pi2` sont des projections automatiquement calculées par la fonction *SRparalEP* (voir fonctions utilitaires *SRpi1* et *SRpi2* aux sections 5.8.3 et 5.8.4). Pour plus d'explications sur cet opérateur, le lecteur est invité à consulter Khédri [35].

Appel de la fonction :

```
var = SRparalEP(m1,m2,"textePI1","textePI2")
```

Paramètres :

`m1` : première matrice symbolique
`m2` : deuxième matrice symbolique
`textePI1` : expression à insérer dans la matrice symbolique `pi1`

`textePI2` : expression à insérer dans la matrice symbolique `pi2`

`var` : variable qui reçoit le résultat

Exemple :

Soit $A = \begin{vmatrix} 0 & aI \\ Ib & 0 \end{vmatrix}$ et $B = \begin{vmatrix} 0 & Ic \\ aI & 0 \end{vmatrix}$

alors l'exécution de la commande `C=SRparalEP(A,B,"w1","w2")` assigne à C la matrice suivante :

colonne 1	colonne 2	colonne 3	colonne 4
0	$w1 * w1\hat{} . w2 * Ic * w2\hat{}$	$w1 * aI * w1\hat{} . w2 * w2\hat{}$	0
$w1 * w1\hat{} . w2 * aI * w2\hat{}$	0	0	$w1 * aI * w1\hat{} . w2 * w2\hat{}$
$w1 * Ib * w1\hat{} . w2 * w2\hat{}$	0	0	$w1 * w1\hat{} . w2 * Ic * w2\hat{}$
0	$w1 * Ib * w1\hat{} . w2 * w2\hat{}$	$w1 * w1\hat{} . w2 * aI * w2\hat{}$	0

5.8 Les fonctions utilitaires

Les fonctions qui suivent ne sont pas des opérateurs mais plutôt des fonctions d'ordre utilitaire. Elles servent à faciliter le travail de l'utilisateur en automatisant des tâches répétitives.

5.8.1 Menu des opérateurs - *SR*

Il est possible d'avoir sous un même menu l'ensemble des opérateurs relationnels définis dans la librairie. Cela se fait à l'aide de la fonction *SR*. Cette fonction est utilisable seulement dans un environnement avec fenêtrage.

Appels de la fonction :

```
var = SR(m1)
var = SR(m1,m2)
var = SR(m1,m2,"texte1","texte2")
var = SR(m1,m2,"texte1","texte2","texte3","texte4")
```

Paramètres :

`m1, m2` : matrices symboliques à utiliser
`texte1 - texte4` : expressions à utiliser
`var` : variable qui reçoit le résultat

Exemples :

Pour que *C* soit l'union de *A* et *B*, tapez la commande `C=SR(A,B)` et choisissez ensuite **Union**.

Pour que *C* soit l'inverse de *A*, tapez la commande `C=SR(A)` et choisissez ensuite **Inverse**.

Pour que *C* soit le résultat du produit parallèle synchrone de *A* avec *B* en utilisant *w1* dans la projection π_1 et *w2* dans la projection π_2, tapez la commande `C=SR(A,B,"w1","w2")` . Choisissez ensuite **Paral synchrone**.

5.8.2 Modification des dimensions d'une matrice SRE - *SRdim*

La fonction SRdim permet d'augmenter ou de diminuer la taille d'une matrice SRE après sa création. On peut spécifier l'expression à insérer dans les nouvelles cellules lorsque la matrice est agrandie.

Appels de la fonction :

```
var = SRdim(mat,li,co)
var = SRdim(mat,li,co,"texte")
```

Paramètres :

`mat` : matrice symbolique à modifier
`li` : nouveau nombre de lignes
`co` : nouveau nombre de colonnes
`texte` : expression à insérer dans les nouvelles cellules (si nécessaire)
`var` : variable qui reçoit le résultat

Exemples :

Soit $A = \begin{vmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \end{vmatrix}$

alors l'exécution de la commande `C=SRdim(A,2,5,"w1")` assigne à *C* la matrice suivante :

*a*1	*a*2	*a*3	*w*1	*w*1
*a*4	*a*5	*a*6	*w*1	*w*1

L'exécution de la commande `C=SRdim(A,3,5)` assigne à *C* la matrice ci-dessous (où la ligne 3, ainsi que les colonnes 4 et 5 sont vides), puis appelle la fonction `SRedit` :

*a*1	*a*2	*a*3
*a*4	*a*5	*a*6

La commande `C=SRdim(A,1,2)` assigne à *C* la matrice suivante :

*a*1	*a*2

5.8.3 Projection π_1 - *SRpi1*

La projection π_1 s'obtient avec la fonction *SRpi1*. Cette projection est celle utilisée dans les produits parallèles. Il est à noter que les opérateurs parallèles appellent automatiquement cette fonction. Il n'est donc pas nécessaire de l'utiliser avant d'effectuer des produits parallèles. Toutefois, comme cette fonction peut être utile pour d'autres besoins, elle est mise à la disposition de l'utilisateur. Dans cette fonction, le nombre de lignes doit être un multiple entier du nombre de colonnes.

Appel de la fonction :

```
var = SRpi1(li,co,"texte")
```

Paramètres :

`li` : nombre de lignes

`co` : nombre de colonnes

`texte` : expression à insérer dans les positions qui ne contiennent pas le symbole de la relation vide

`var` : variable qui reçoit le résultat

Exemple :

La commande `C=SRpi1(4,2,"w1")` assigne à C la matrice suivante :

$$\begin{vmatrix} w1 & 0 \\ w1 & 0 \\ 0 & w1 \\ 0 & w1 \end{vmatrix}$$

5.8.4 Projection π_2 - *SRpi2*

La projection π_2 s'obtient avec la fonction *SRpi2*. Cette projection est celle utilisée dans les produits parallèles. Il est à noter que les opérateurs parallèles appellent automatiquement cette fonction. Il n'est donc pas nécessaire de l'utiliser avant d'effectuer des produits parallèles. Toutefois, comme cette fonction peut être utile pour d'autres besoins, elle est mise à la disposition de l'utilisateur. Dans cette fonction, le nombre de lignes doit être un multiple entier du nombre de colonnes.

Appel de la fonction :

```
var = SRpi2(li,co,"texte")
```

Paramètres :

`li` : nombre de lignes

`co` : nombre de colonnes

`texte` : expression à insérer dans les positions qui ne contiennent pas le symbole de la relation vide

`var` : variable qui reçoit le résultat

Exemple :

L'exécution de `C=SRpi2(4,2,"w2")` assigne à C la matrice suivante :

$$\begin{vmatrix} w2 & 0 \\ 0 & w2 \\ w2 & 0 \\ 0 & w2 \end{vmatrix}$$

5.8.5 Matrice universelle - *SRL*

La fonction *SRL* permet de créer une matrice SRE remplie avec la relation universelle.

Appel de la fonction :

```
var = SRL(li,co)
```

Paramètres :

`li` : nombre de lignes

`co` : nombre de colonnes

`var` : variable qui reçoit le résultat

Exemple :

La commande `C=SRL(2,3)` assigne à *C* la matrice suivante :

$$\begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

5.8.6 Matrice vide - *SR0*

La fonction *SR0* permet de créer une matrice SRE remplie avec la relation vide.

Appel de la fonction :

```
var = SR0(li,co)
```

Paramètres :

`li` : nombre de lignes

`co` : nombre de colonnes

`var` : variable qui reçoit le résultat

Exemple :

La commande `C=SR0(2,3)` assigne à *C* la matrice suivante :

$$\begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{vmatrix}$$

5.8.7 Matrice diagonale - *SRI*

La fonction *SRI* permet de créer une matrice carrée SRE dont la diagonale est remplie avec l'expression spécifiée. Par défaut, c'est le symbole identité qui est inséré.

Appels de la fonction :

```
var = SRI(nb,"texte")
var = SRI(nb)
```

Paramètres :

`nb` : nombre de lignes et de colonnes

`texte` : expression à insérer dans la diagonale

`var` : variable qui reçoit le résultat

Exemples :

L'exécution de la commande `C=SRI(3)` assigne à C la matrice suivante :

$$\left|\begin{array}{ccc} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{array}\right|$$

et la commande `C=SRI(3,"w")` assigne à C la matrice suivante :

$$\left|\begin{array}{ccc} w & 0 & 0 \\ 0 & w & 0 \\ 0 & 0 & w \end{array}\right|$$

5.8.8 Comparaison de 2 matrices - *SRegale*

La fonction *SRegale* permet de vérifier si deux matrices SRE sont identiques.

Appel de la fonction :

`SRegale(m1,m2)`

Paramètres :

`m1` : première matrice à comparer

`m2` : deuxième matrice à comparer

Exemple :

Soit $A = \left|\begin{array}{ccc} a1 & a2 & a3 \\ a4 & a5 & a6 \end{array}\right|$ et $B = \left|\begin{array}{ccc} a1 & a2 & a3 \\ a4 & a5 & 0 \end{array}\right|$

alors l'exécution de la commande `SRegale(A,B)` donne comme résultat `F` .

5.8.9 Création à partir d'une matrice quelconque - *SRmat*

La fonction *SRmat* permet de créer une matrice SRE directement à partir d'une matrice Ψlab[4]. Cette fonction est utile lorsque l'environnement de travail ne permet pas le fenêtrage.

[4]L'inverse, *i.e.* extraire la composante matrice d'une matrice SRE, s'obtient en sélectionnant le deuxième élément. Par exemple, on obtient la composante matrice de A en tapant la commande `A(2)` .

Appel de la fonction :

```
var = SRmat(z)
```

Paramètres :

`z` : matrice Ψlab

`var` : variable qui reçoit le résultat

Exemple :

Soit `y = ["a","b","c";"d","e","f"]` alors l'exécution de la commande `C=SRmat(y)` assigne à C la matrice suivante :

$$\left|\begin{array}{ccc} a & b & c \\ d & e & f \end{array}\right|$$

5.8.10 Version de la librairie - *SRversion*

La fonction *SRversion* affiche la version de la librairie $\mathcal{SR}$•Ψlab.

Appel de la fonction :

```
SRversion()
```

Paramètre :

Aucun

Exemple :

L'exécution de la commande `SRversion()` affiche

```
Version du 96-11-01 (scilab 2.2)
```

5.8.11 Substitution des ESRV - *SRremplace*

Il est possible d'effectuer la substitution de chaînes de caractères à l'intérieur de chaque ERSV d'une matrice symbolique. Pour ce faire, on utilise la fonction *SRremplace*. Cette fonction est utile pour apporter des simplifications à une matrice SRE ou pour remplacer les symboles en vue d'un transfert vers TEX.

Il est à noter que la matrice est parcourue une fois pour chaque simplification. Ainsi, la seconde simplification s'appliquera sur le résultat de la première simplification, la troisième sur le résultat de la seconde, et ainsi de suite. Il faut donc être attentif dans l'utilisation de cette fonction.

Appel de la fonction :

```
var = SRremplace(mat,table)
```

Paramètres :

`mat` : matrice symbolique à modifier

`table` : table de conversion (une matrice caractère de dimension $(n \times 2)$)

`var` : variable qui reçoit le résultat

Exemples :

Soit $A = \begin{vmatrix} a1+b & a2+c & a3+d \\ b & c & d \end{vmatrix}$

et `tab = ["a1+b","1";"a","w";"c","P5"]`

alors l'exécution de la commande `C=SRremplace(A,["a","w"])` assigne à C la matrice suivante :

$$\begin{vmatrix} w1+b & w2+c & w3+d \\ b & c & d \end{vmatrix}$$

Et l'exécution de la commande `C=SRremplace(A,tab)` assigne à C la matrice suivante :

$$\begin{vmatrix} 1 & w2+P5 & w3+d \\ b & P5 & d \end{vmatrix}$$

5.9 Modification des symboles - *SRsymb*

Il est possible de modifier les symboles utilisés pour représenter l'union, l'intersection, etc. En fait, on peut remplacer tous les symboles du tableau en page 46. Pour ce faire, on utilise la fonction *SRsymb*.

Appels de la fonction :

```
%SR = SRsymb()
%SR = SRsymb(vect)
```

Paramètres :

`vect` : vecteur caractère contenant les 10 nouveaux symboles

`%SR` : variable de système qui contient la définition des symboles

Exemples :

L'exécution de la commande `%SR=SRsymb()` ouvre une fenêtre pour effectuer les modifications (utilisable seulement dans un environnement avec fenêtrage).
Soit `v=["L","0","I","+","-","*","^","'","/","\"]` alors l'exécution de la commande `%SR=SRsymb(v)` a pour effet de modifier le symbole de la relation universelle I par L, et de remplacer le symbole d'intersection (.) par (-). Les autres symboles ne sont pas modifiés. Il est à noter qu'il faut taper deux fois l'apostrophe (') pour en obtenir une seule à l'assignation.

5.10 Exportation vers TEX

Les fonctions *texprint* et *write* de Ψlab permettent de transférer une matrice SRE sous un format TEX dans un fichier UNIX. La fonction *texprint* effectue la conversion de matrices vers le format TEX, tandis que la fonction *write* transfère le contenu d'une variable dans un fichier UNIX. Par exemple, si on désire transférer la matrice SRE A dans un fichier nommé *matrice.tex*, il suffit de taper la commande `write("matrice.tex",texprint(A(2)))` . Il est à noter que le fichier *matrice.tex* doit être inexistant avant de taper la commande.

Avant d'effectuer le transfert, il peut s'avérer avantageux d'utiliser la fonction *SRremplace* pour remplacer les opérateurs. Par exemple, soit

$$A = \left| \begin{array}{cc} a1+b & a2+I \\ a3 & 0 \end{array} \right|$$

et `tab = ["+","\cup ";"I","\mathcal{I}";"0","\Vide "]` alors, l'exécution de la commande `C=SRremplace(A,tab)` assigne à C la matrice suivante :

$$\left| \begin{array}{cc} a1\texttt{\textbackslash cup}\not\beta b & a2\texttt{\textbackslash cup}\not\beta\texttt{\textbackslash mathcal\{I\}} \\ a3 & \texttt{\textbackslash Vide}\not\beta \end{array} \right|$$

où le symbole $\not\beta$ représente le caractère *espace*. Puis, l'utilisation de la commande `write("matrice.tex",texprint(C(2)))` inscrit dans le fichier *matrice.tex* le contenu suivant :

```
{\pmatrix{a1\cup b&a2\cup \mathcal{I}\cr a3&\Vide }}
```

5.11 Création de nouvelles fonctions

Il est possible de créer de nouvelles fonctions en utilisant les opérateurs déjà définis dans la librairie. Cependant, il est nécessaire de savoir comment créer des fonctions avec Ψlab. À titre d'exemple, voici à quoi pourrait ressembler une fonction qui effectue l'opération $\overline{\widehat{A} \cup \widehat{B} \cup \widehat{C}}$ sur trois matrices SRE :

```
function [rep]=Triple(R1,R2,R3)
  n=-1
  rep=(R1^n + R2^n + R3^n)'
```

Chapitre 6

Conclusion

L'utilisation de l'algèbre des relations comme outil de spécification et de description de programmes présente des possibilités uniques et intéressantes, comme le montrent l'étude de Schmidt et Ströhlein [52] sur les programmes séquentiels ainsi que les résultats obtenus par Chaib-draa *et al.* [8] pour la décomposition parallèle.

À mesure que les travaux de recherche avancent, ils nécessitent des spécifications relationnelles de plus en plus complexes. L'utilisation d'outils informatiques capables d'effectuer des transformations algébriques sur un nombre important d'expressions relationnelles devient alors incontournable.

De tels outils spécifiques ne semblent pas être disponibles actuellement. Mais cela ne saurait tarder. Le développement important que connaissent les systèmes de manipulations algébriques favorise l'émergence d'outils propres aux calculs relationnels. Déjà, parmi les logiciels que nous avons examinés, plusieurs ont la capacité de combler certains besoins. Outre `Scilab`, que nous avons retenu pour le développement du prototype, mentionnons `Mathematica` et `Maple` pour leurs capacités de programmation et de manipulation symbolique, ainsi que `Isabelle` pour ses capacités de manipulation de règles.

Lors de l'élaboration des algorithmes nécessaires au développement du prototype, nous avons opté pour une approche de transformation dite constructive puisque cette dernière nous semblait tout à fait adaptée aux manipulations que l'on souhaitait effectuer sur les matrices d'expressions relationnelles. Cette approche a eu l'avantage de faciliter grandement la définition des règles de simplification puisque ces dernières sont définies au niveau des

opérateurs élémentaires pour être ensuite automatiquement utilisées avec les opérateurs complexes, ceux-ci étant formés au moyen des opérations élémentaires. De plus, par l'application de la transformation constructive, la longueur des expressions relationnelles sans variables est minimisée, accélérant ainsi le traitement de la prochaine opération et allégeant tout processus de réduction ultérieur.

Le logiciel *Scilab* a été utilisé pour développer un protype capable d'effectuer les transformations constructives applicables lors de la composition parallèle. *Scilab* s'est avéré un outil puissant et souple et ses fonctions d'édition et de manipulation de matrices ont facilité le développement du prototype. Par contre, l'arborescence symbolique n'étant pas supportée par *Scilab*, cela a obligé un traitement des expressions sous la forme de chaînes de caractères plutôt que comme un arbre d'expressions symboliques.

Même à l'état de protype, le produit développé a permis de valider et vérifier plusieurs résultats. Dans le cadre de travaux sur la composition parallèle, il a permis de confirmer l'exactitude de résultats obtenus suite à un processus laborieux de calculs manuels. Il permet maintenant de produire en quelques minutes les résultats de calculs qui auparavant prenaient plusieurs heures.

Les travaux réalisés dans le cadre de ce mémoire ont démontré l'intérêt d'automatiser la transformation d'expressions relationnelles sans variables. Le prototype développé a fourni les résultats attendus et ouvre la voie au développement d'une application qui pourrait intégrer l'arborescence symbolique et l'ensemble des règles de simplification. Même si les performances du prototype se sont avérées satisfaisantes, la rapidité du traitement pourrait encore être augmentée.

La transformation constructive n'élimine pas totalement le besoin de recourir à un processus de réduction, mais elle en facilite grandement le traitement. En ce sens, l'application devrait permettre le transfert des expressions symboliques vers un logiciel de manipulation de règles comme *Isabelle*.

Annexe A

Code source du prototype

Les fonctions sont présentées par ordre alphabétique.

```
function Affiche(msg)
// Affiche un message a l'ecran
write(%io(2),msg)


function [mat]=Creer(n,m,objet)
// Creation d'une matrice avec le meme d'objet
[lhs,rhs]=argn(0)
if rhs=2 then, objet=emptystr(), end
objet=objet(1)
mat=objet
for i=2:n*m
  mat=[mat,objet]
end
mat=matrix(mat,n,m)


function [m]=Edit(mat)
// Edition d'une matrice caractere ou numerique
m=mat
select typeof(mat)
  case 'character' then
     m=Xmatc('Matrice caractere',mat)
  case 'usual' then
     m=Xmatn('Matrice numerique',mat)
  else disp('Type non reconnu')
end
```

```
function [b]=Equal(mat1,mat2)
// Verifie si deux matrices sont identiques
[li1,co1]=size(mat1)
[li2,co2]=size(mat2)
if li1<>li2 | co1<>co2 then
  b=%f
else
  b=%t; li=1
  while li<=li1 & b
    co=1
    while co<=co1 & b
       b=(mat1(li,co)==mat2(li,co))
       co=co+1
    end
    li=li+1
  end
end


function [mat]=Majoute(mat1,n,objet)
// Ajoute n lignes a la fin
[li,co]=size(mat1)
mat=[]
for i=1:n*co
  mat=[mat,objet]
end
mat1=matrix(mat1',1,li*co)
mat=[mat1,mat]
mat=matrix(mat,co,li+n)'


function [mat]=Map(fn,mat1,mat2)
// Mapping de la fonction fn en utilisant les arguments mat1 et/ou mat2
[lfs,rhs]=argn(0)
[n,m]=size(mat1)
mat=[]
for i=1:n
   for j=1:m
     if rhs=3 then
        x1=fn(mat1(i,j),mat2(i,j))
      else
        x1=fn(mat1(i,j))
      end
     mat=[mat,x1]
   end
 end
mat=matrix(mat,m,n)'


function [mat]=Mcoupe(mat1,n)
// Coupe les n dernieres lignes
[li,co]=size(mat1)
```

```
mat=[]
if li-n > 0 then
  mat=matrix(mat1',1,li*co)
  mat=mat([1:co*(li-n)])
  mat=matrix(mat,co,li-n)'
end


function [pos]=Occur(mot,car)
// Premiere occurence de car dans mot
l=length(mot)
pos=0;i=1
while (i<=l) & pos==0
  if part(mot,[i:i-1+length(car)])==car then
     pos=i
  end
  i=i+1
end


function [b]=Presence(mot,vsymb)
// Verifie la presence d'un des membres de vsymb dans mot
[x,n]=size(vsymb)
b=%f;i=1
while (i<=n) & ~b
  car=vsymb(i);
  b=(Occur(mot,car)<>0)
  i=i+1
end


function [mat]=Resize(mat1,n,m,objet)
// Modifie la dimension d'une matrice quelconque
mat=[];err=%f
[lhs,rhs]=argn(0)
tmat1=typeof(mat1)
if rhs==3 then
   if tmat1=='character'
      objet=emptystr()
   elseif tmat1=='usual'
      objet=0
   else
      objet=mat1(1)
   end
end
err=~ValideArgNb(rhs,3)
if ~err then, err=~ValideArgType(n,1), end
if ~err then, err=~ValideArgType(m,1), end
if ~err & tmat1<>typeof(objet) then
   disp('Types incompatibles')
   err=%t
end
```

```
if ~err then
    mat=mat1
    objet=objet(1)
    [li1,co1]=size(mat1)
    if n>li1 then, mat=Majoute(mat,n-li1,objet), end
    if n<li1 then, mat=Mcoupe(mat,li1-n), end
    if m>co1 then, mat=Majoute(mat',m-co1,objet)', end
    if m<co1 then, mat=Mcoupe(mat',co1-m)', end
end


function [sr]=SR(sr1,sr2,s3,s4,s5,s6)
// Menu pour les operations relationnelles
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,1)
choix=['Inverse';'Complement';'Union';'Intersection';'Produit';
       'Residu a gauche';'Residu a droite';
       'Residu a gauche developpe';'Residu a droite developpe';
       'Produit parallele';'Paral. synchrone';
       'Paral. entrelacant';'Paral. entrelacant particulier';
       'Produit booleen';'Inverse booleen']
titre=['Faites un choix']
if ~err then
  no=x_choose(choix,titre)
  select (no)
    case  1 then arg=1
    case  2 then arg=1
    case  3 then arg=2
    case  4 then arg=2
    case  5 then arg=2
    case  6 then arg=2
    case  7 then arg=2
    case  8 then arg=2
    case  9 then arg=2
    case 10 then arg=4
    case 11 then arg=4
    case 12 then arg=6
    case 13 then arg=4
    case 14 then arg=2
    case 15 then arg=1
    else err=%t
  end
  if ~err then, err=~ValideArgNb(rhs,arg), end
end
if ~err then
  select (no)
    case  1 then sr=SRinv(sr1)
    case  2 then sr=SRcomp(sr1)
    case  3 then sr=SRunion(sr1,sr2)
    case  4 then sr=SRinter(sr1,sr2)
    case  5 then sr=SRproduit(sr1,sr2)
```

```
    case  6 then sr=SRresg(sr1,sr2)
    case  7 then sr=SRresd(sr1,sr2)
    case  8 then sr=SRresidug(sr1,sr2)
    case  9 then sr=SRresidud(sr1,sr2)
    case 10 then sr=SRparal(sr1,sr2,s3,s4)
    case 11 then sr=SRparalS(sr1,sr2,s3,s4)
    case 12 then sr=SRparalE(sr1,sr2,s3,s4,s5,s6)
    case 13 then sr=SRparalEP(sr1,sr2,s3,s4)
    case 14 then sr=SRprodBool(sr1,sr2)
    case 15 then sr=SRinvBool(sr1)
    else sr=[]
  end
end


function [sr]=SR0(n,m)
// Genere une matrice SRE remplie avec la relation vide
[%L,%0,%I]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,1)
if ~err & rhs==1 then, m=n, end
if ~err then, err=~ValideArgType(n,1), end
if ~err then, err=~ValideArgType(m,1), end
if ~err then
  sr=SRcreer(n,m,%0)
end


function SR0H()
// Fonction d'aide pour SR0
msg=['--------------------';
 'DESCRIPTION ';
 '      <SR0> sert a creer une matrice SRE vide ';
 ' ';
 'APPEL ';
 '      var = SR0(li,co)           ';
 ' ';
 'PARAMETRES ';
 '      li : nb de lignes';
 '      co : nb de colonnes';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLE ';
 '      A = SR0(2,3)  => ! 0 0 0 ! ';
 '                       ! 0 0 0 ! ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRcomp(sr1)
// Complement d'une matrice SRE
```

```
[%L,%0,%I,%unio,%inte,%prod,%inve,%comp,%resg,%resd]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,1)
if ~err then, err=~SRvalide(sr1), end
if ~err then
  [srm1,l1,c1]=SRxtract(sr1)
  srm1=Map(Trim,srm1)
  srm=srm1
  for i=1:l1
    for j=1:c1
      srm(i,j)=SRcompC(srm(i,j))
    end
  end
  sr=tlist('sre',srm)
end


function [c]=SRcompC(c1)
// Complement d'une expression
e=emptystr()
c=e
if c1==e
  c='?'
  c1='?'
elseif c1==%0
  c=%L
elseif c1==%L
  c=%0
end
op=Presence(c1,[%inte,%unio,%prod,%inve,%resg,%resd])
if c==e & (part(c1,length(c1))==%comp) then
 if part(c1,1)=='(' & part(c1,length(c1)-1)==')'
    c=part(c1,2:length(c1)-2)
    if Presence(c,['(',')']) then, c=e, end
 elseif ~op | (part(c1,1)=='[' & part(c1,length(c1)-1)==']')
    c=part(c1,1:length(c1)-1)
 end
end
if c==e then
  carre1=(part(c1,1)=='[' & part(c1,length(c1))==']')
  if op & ~carre1 then, c1='('+c1+')', end
  c=c1+%comp
end


function SRcompH()
// Fonction d'aide pour SRcomp
msg=['--------------------';
 'DESCRIPTION ';
 '      <''> ou <SRcomp>  donne le complement d''une matrice SRE ';
 ' ';
```

```
  'APPEL ';
  '      var = mat''  ou bien  var = SRcomp(mat) ';
  ' ';
  'PARAMETRES ';
  '      mat : la matrice symbolique a complementer';
  '      var : variable qui recoit le resultat ';
  ' ';
  'EXEMPLES ';
  '      A = B''          => A recoit le complement de B ';
  '      A = SRcomp(B)   => A recoit le complement de B ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRcreer(n,m,objet)
// Creation d'une matrice SRE
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~ValideArgType(n,1), end
if ~err then, err=~ValideArgType(m,1), end
if ~err then
  obj=emptystr()
  if rhs==3 then
    obj=string(objet(1))
  end
  srm=Creer(n,m,obj)
  sr=tlist('sre',srm)
  if rhs==2 then, sr=SRedit(sr), end
end


function SRcreerH()
// Fonction d'aide pour SRcreer
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRcreer> permet de creer une nouvelle matrice SRE ';
 ' ';
 'APPELS ';
 '      var = SRcreer(li,co,''texte'') ';
 '      var = SRcreer(li,co)           ';
 ' ';
 'PARAMETRES ';
 '      li : nb de lignes';
 '      co : nb de colonnes';
 '      texte : expression a inserer dans la matrice';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLES ';
 '      A = SRcreer(2,3,''w1'')  => ! w1 w1 w1 ! ';
 '                                  ! w1 w1 w1 ! ';
 ' ';
```

```
  '      A = SRcreer(2,3)  => appel de l''editeur de matrice ';
  '                           avec une matrice vide de 2 x 3 ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRdim(sr1,n,m,objet)
// Modifie la dimension d'une matrice SRE
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,3)
if ~err then, err=~ValideArgType(n,1), end
if ~err then, err=~ValideArgType(m,1), end
if ~err then
  [%L,%0,%I]=SRsign()
  if rhs==4 then
    objet=string(objet(1))
  else
    objet=emptystr()
  end
  if SRvalide(sr1) then
    srm1=sr1(2)
    [li1,co1]=size(srm1)
    srm1=Resize(srm1,n,m,objet)
    [li2,co2]=size(srm1)
    sr=tlist('sre',srm1)
    if objet == emptystr() then
        if li2>li1 | co2>co1 then, sr=SRedit(sr), end
    end
  end
end


function SRdimH()
// Fonction d'aide pour SRdim
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRdim> sert a modifier les dimensions d''une ';
 '              matrice SRE ';
 ' ';
 'APPELS ';
 '      var = SRdim(mat,li,co) ';
 '      var = SRdim(mat,li,co,''texte'')     ';
 ' ';
 'PARAMETRES ';
 '      mat : matrice symbolique a modifier ';
 '      li : nouveau nb de lignes';
 '      co : nouveau nb de colonnes';
 '      texte : expression a inserer dans la matrice';
 '              si on agrandit les dimensions ';
 '      var : variable qui recoit le resultat ';
 ' ';
```

```
  'EXEMPLES ';
  '      Soit  A <= ! a a a ! ';
  '                 ! a a a ! ';
  ' ';
  '      B = SRdim(A,2,5,''w1'')  => ! a a a w1 w1 ! ';
  '                                  ! a a a w1 w1 ! ';
  ' ';
  '      B = SRdim(A,2,5)  => appel de SRedit avec  ! a a a     ! ';
  '                                                 ! a a a     ! ';
  ' ';
  '      A = SRdim(A,1,2)  => ! a a !  ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRedit(sr1)
// Edition d'une matrice SRE
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,1)
if ~err then
  sr=sr1
  if SRvalide(sr1) then
    srm=sr1(2)
    srm=Xmatc('Matrice symbolique',srm)
    sr=tlist('sre',srm)
  end
end


function SReditH()
// Fonction d'aide pour SRedit
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRedit> sert a editer une matrice SRE ';
 ' ';
 'APPEL ';
 '      var  = SRedit(mat) ';
 ' ';
 'PARAMETRES ';
 '      mat : matrice symbolique a modifier ';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLES ';
 '      A = SRedit(A)  => modifier le contenu de A ';
 ' ';
 '      B = SRedit(A)  => creer B a partir de modifications sur A';
' ';'--------------------']
Affiche(msg)


function [b]=SRegale(sr1,sr2)
```

```
// Verifie si 2 matrices SRE sont identiques
b=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~SRvalide(sr1), end
if ~err then, err=~SRvalide(sr2), end
if ~err then
  srm1=sr1(2)
  srm2=sr2(2)
  b=Equal(srm1,srm2)
end


function SRegaleH()
// Fonction d'aide pour SRegale
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRegale> sert a comparer 2 matrices SRE ';
 ' ';
 'APPEL ';
 '      SRegale(m1,m2) ';
 ' ';
 'PARAMETRES ';
 '      m1   : matrice symbolique a comparer ';
 '      m2   : matrice symbolique a comparer ';
 ' ';
 'EXEMPLE ';
 '      SRegale(B,C)  => T  si les 2 matrices sont identiques ';
 '                       F  dans le cas contraire ';
' ';'--------------------']
Affiche(msg)


function SRH()
// Fonction d'aide pour SR
msg=['--------------------';
 'DESCRIPTION ';
 '      <SR> appelle le menu des operateurs relationnels ';
 ' ';
 'APPELS ';
 '      var = SR(m1) ';
 '      var = SR(m1,m2) ';
 '      var = SR(m1,m2,''tex1'',''tex2'') ';
 '      var = SR(m1,m2,''tex1'',''tex2'',''tex3'',''tex4'') ';
 ' ';
 'PARAMETRES ';
 '      m1 - m2 : matrices symboliques a utiliser';
 '      tex1 - tex4 : expressions a utiliser';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLES ';
 '      A = SR(B,C)  => une operation avec B et C comme arguments ';
```

```
  '      A = SR(B)     => une operation avec B seulement ';
  '      A = SR(B,C,''w1'',''w2'')  => pour une operation avec 4 arguments ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRI(n,objet)
// Genere une matrice carree SRE avec l'expression desiree dans la diagonale
[%L,%0,%I]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,1)
if ~err then, err=~ValideArgType(n,1), end
if ~err & rhs==2 then, err=~ValideArgType(objet,2), end
if ~err then
  obj=%I
  if rhs==2 then, obj=objet, end
  srm=Creer(n,n,%0)
  for i=1:n
    srm(i,i)=obj
  end
  sr=tlist('sre',srm)
end


function SRIH()
// Fonction d'aide pour SRI
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRI> sert a creer une matrice SRE diagonale ';
 ' ';
 'APPELS ';
 '      var = SRI(nb,''texte'') ';
 '      var = SRI(nb)           ';
 ' ';
 'PARAMETRES ';
 '      nb : dimension de la matrice carree';
 '      texte : expression a inserer dans la diagonale';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLES ';
 '      A = SRI(3,''w1'')  => ! w1 0  0  ! ';
 '                            ! 0  w1 0  ! ';
 '                            ! 0  0  w1 ! ';
 ' ';
 '      A = SRI(2)  => ! I 0 ! ';
 '                     ! 0 I ! ';
' ';'--------------------']
Affiche(msg)


function SRinfo(no)
```

```
// Aide sur les fonctions de la librairie
err=%f
[lhs,rhs]=argn(0)
msg=[
 '--------------------';
 'Sommaire des fonctions pour les matrices symboliques';
 ' ';
 ' 1. SRcreer     - Creation';
 ' 2. SRedit      - Edition';
 ' 3.    ^ n      - Exposant n';
 ' 4.    ^(-1)    - Inverse (ou bien SRinv)';
 ' 5.    +        - Union (ou bien SRunion)';
 ' 6.    -        - Intersection (ou bien SRinter)';
 ' 7.    *        - Produit (ou bien SRproduit)';
 ' 8.    ''        - Complement (ou bien SRcomp)';
 ' 9.    /        - Residu a gauche (ou bien SRresg)';
 '10.    \        - Residu a droite (ou bien SRresd)';
 '11. SRresidug   - Residu a gauche developpe ';
 '12. SRresidud   - Residu a droite developpe ';
 '13. SR          - Menu des operations';
 '14. SRsymb      - Modification des symboles utilises';
 '15. SRdim       - Modification des dimensions';
 '16. SRL         - Creation d''une matrice universelle';
 '17. SR0         - Creation d''une matrice vide';
 '18. SRI         - Creation d''une matrice diagonale';
 '19. SRegale     - Comparaison de 2 matrices';
 '20. SRmat       - Creation a partir d''une matrice non symbolique';
 '21. SRversion   - Version de la librairie';
 '22. SRremplace  - Substitution des expressions';
 '23. SRparal     - Produit parallele ';
 '24. SRparalS    - Produit parallele synchrone';
 '25. SRparalE    - Produit parallele entrelacant';
 '26. SRparalEP   - Produit parallele entrelacant particulier';
 '27. SRpi1       - Creation d''une matrice PI1';
 '28. SRpi2       - Creation d''une matrice PI2';
 '29. SRprodBool  - Produit booleen';
 '30. SRinvBool   - Inverse booleen';
 ' ';
 'Pour obtenir plus d''information sur une fonction ';
 'tapez SRinfo( no fonction ) ';
 'ex: SRinfo(2) pour de l''information sur SREdit ';
 '--------------------']
if rhs==0 then
  Affiche(msg)
else
  err=~ValideArgType(no,1)
  if ~err then
    select no
      case  1 then SRcreerH()
      case  2 then SReditH()
      case  3 then SRinvH1()
      case  4 then SRinvH2()
```

```
      case  5 then SRunionH()
      case  6 then SRinterH()
      case  7 then SRproduitH()
      case  8 then SRcompH()
      case  9 then SRresgH()
      case 10 then SRresdH()
      case 11 then SRresidugH()
      case 12 then SRresidudH()
      case 13 then SRH()
      case 14 then SRsymbH()
      case 15 then SRdimH()
      case 16 then SRLH()
      case 17 then SR0H()
      case 18 then SRIH()
      case 19 then SRegaleH()
      case 20 then SRmatH()
      case 21 then SRversionH()
      case 22 then SRremplaceH()
      case 23 then SRparalH()
      case 24 then SRparalSH()
      case 25 then SRparalEH()
      case 26 then SRparalEPH()
      case 27 then SRpi1H()
      case 28 then SRpi2H()
      case 29 then SRprodBoolH()
      case 30 then SRinvBoolH()
      else disp('No incorrect')
    end
  end
end


function [%SR]=SRinit()
// Valeurs par defaut pour les symboles
%SR=['1','0','I','+','.','*','^','''','/','\']


function [sr]=SRinter(sr1,sr2)
// Intersection de 2 matrices SRE
[%L,%0,%I,%unio,%inte,%prod,%inve,%comp,%resg,%resd]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~SRvalide(sr1), end
if ~err then, err=~SRvalide(sr2), end
if ~err then
  [srm1,l1,c1]=SRxtract(sr1)
  [srm2,l2,c2]=SRxtract(sr2)
  if l1<>l2 | c1<>c2 then
     disp('Dimensions incompatibles')
     err=%t
  end
```

```
end
if ~err
  srm1=Map(Trim,srm1)
  srm2=Map(Trim,srm2)
  srm=Map(SRinterC,srm1,srm2)
  sr=tlist('sre',srm)
end


function [c]=SRinterC(c1,c2)
// Intersection de 2 expressions
e=emptystr()
c=e
if c1==e | c2==e
  c='?'
elseif c1==c2
  c=c1
elseif c1==%0 | c2==%0
  c=%0
elseif c1==%L
  c=c2
elseif c2==%L
  c=c1
end
if c==e then
  p1=Presence(c1,%unio);p2=Presence(c2,%unio)
  carre1=(part(c1,1)=='[' & part(c1,length(c1))==']')
  carre2=(part(c2,1)=='[' & part(c2,length(c2))==']')
  if p1 & ~carre1 then, c1='('+c1+')', end
  if p2 & ~carre2 then, c2='('+c2+')', end
  c=c1+%inte+c2
end


function SRinterH()
// Fonction d'aide pour SRinter
msg=['----------------------';
 'DESCRIPTION ';
 '      <-> ou <SRinter> fait l''intersection de 2 matrices SRE ';
 ' ';
 'APPELS ';
 '      var = m1 - m2 ';
 '      var = SRinter(m1,m2)          ';
 ' ';
 'PARAMETRES ';
 '      m1   : premiere matrice symbolique';
 '      m2   : deuxieme matrice symbolique';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLES ';
 '      C = A-B           => C contient l''intersection de A et B ';
 '      C = SRinter(A,B)  => C contient l''intersection de A et B ';
```

```
' ';'--------------------']
Affiche(msg)


function [sr]=SRinv(sr1)
// Inverse (ou transposee) d'une matrice SRE
[%L,%0,%I,%unio,%inte,%prod,%inve,%comp,%resg,%resd]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,1)
if ~err then, err=~SRvalide(sr1), end
if ~err then
  [srm1,l1,c1]=SRxtract(sr1)
  srm1=Map(Trim,srm1)
  srm=srm1
  for i=1:l1
    for j=1:c1
      srm(i,j)=SRinvC(srm(i,j))
    end
  end
  srm=srm'
  sr=tlist('sre',srm)
end


function [sr]=SRinvBool(sr1)
// Inverse booleen d'une matrice SRE
[%L,%0,%I,%unio,%inte,%prod,%inve,%comp,%resg,%resd]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,1)
if ~err then, err=~SRvalide(sr1), end
if ~err then
  [srm1,l1,c1]=SRxtract(sr1)
  srm=srm1'
  sr=tlist('sre',srm)
end


function SRinvBoolH()
// Fonction d'aide pour SRinvBool
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRinvBool> donne l''inverse booleen d''une matrice SRE ';
 ' ';
 'APPEL ';
 '      var = SRinvBool(mat)          ';
 ' ';
 'PARAMETRES ';
 '      mat : la matrice symbolique a inverser';
 '      var : variable qui recoit le resultat ';
 ' ';
```

```
  'EXEMPLE ';
  '       C = SRinvBool(A)  => C contient l''inverse booleen de A ';
' ';'--------------------']
Affiche(msg)


function [c]=SRinvC(c1)
// Inverse (ou transposee) d'une expression
e=emptystr()
c=e
if c1==e
  c='?'
  c1='?'
elseif c1==%0 | c1==%L | c1==%I
  c=c1
end
op=Presence(c1,[%inte,%unio,%prod,%comp,%resg,%resd])
if c==e & (part(c1,length(c1))==%inve) then
  if part(c1,1)=='(' & part(c1,length(c1)-1)==')'
     c=part(c1,2:length(c1)-2)
     if Presence(c,['(',')']) then, c=e, end
  elseif ~op | (part(c1,1)=='[' & part(c1,length(c1)-1)==']')
    c=part(c1,1:length(c1)-1)
  end
end
if c==e then
  carre1=(part(c1,1)=='[' & part(c1,length(c1))==']')
  if op & ~carre1 then, c1='('+c1+')', end
  c=c1+%inve
end


function SRinvH1()
// Fonction d'aide pour exposant
msg=['--------------------';
 'DESCRIPTION ';
 '      <^>  sert a multiplier une matrice SRE ';
 '           par elle-meme. Si la valeur de l''exposant ';
 '           est negative alors c''est l''inverse de la ';
 '           matrice qui sera utilise ';
 ' ';
 'APPEL ';
 '      var = mat ^ n';
 ' ';
 'PARAMETRES ';
 '      n : l''exposant ';
 '      mat : la matrice symbolique ';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLES ';
 '      A = C^3      => A recoit C*C*C ';
 '      A = C^(-1)   => A recoit l''inverse de C ';
```

```
  '      A = C^(-2)  => A recoit (inverse C)*(inverse C) ';
' ';'--------------------']
Affiche(msg)


function SRinvH2()
// Fonction d'aide pour SRinv
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRinv>  donne l''inverse d''une matrice SRE, ';
 '               c''est l''equivalent de  ^(-1) ';
 ' ';
 'APPELS ';
 '      var = SRinv(mat)  ou bien  var = mat ^ (-1) ';
 ' ';
 'PARAMETRES ';
 '      mat : la matrice symbolique a inverser';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLES ';
 '      A = SRinv(C)  => A recoit l''inverse de C ';
 '      A = C^(-1)    => A recoit l''inverse de C ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRL(n,m)
// Genere une matrice SRE remplie avec la relation universelle
[%L,%0,%I]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,1)
if ~err & rhs==1 then, m=n, end
if ~err then, err=~ValideArgType(n,1), end
if ~err then, err=~ValideArgType(m,1), end
if ~err then
  sr=SRcreer(n,m,%L)
end


function SRLH()
// Fonction d'aide pour SRL
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRL> sert a creer une matrice SRE universelle ';
 ' ';
 'APPEL ';
 '      var = SRL(li,co)           ';
 ' ';
 'PARAMETRES ';
 '      li : nb de lignes';
 '      co : nb de colonnes';
```

```
  '      var : variable qui recoit le resultat ';
  ' ';
  'EXEMPLE ';
  '      A = SRL(2,3)  => ! 1 1 1 ! ';
  '                       ! 1 1 1 ! ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRmat(srm)
// Genere une matrice SRE a partir d'une matrice quelconque
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,1)
if ~err then
  srm=string(srm)
  sr=tlist('sre',srm)
end


function SRmatH()
// Fonction d'aide pour SRmat
msg=['--------------------';
  'DESCRIPTION ';
  '      <SRmat> sert a transformer une matrice non symbolique';
  '              en une matrice SRE ';
  ' ';
  'APPEL ';
  '      var = SRmat(mat)    ';
  ' ';
  'PARAMETRES ';
  '      mat : matrice non symbolique';
  '      var : variable qui recoit le resultat ';
  ' ';
  'EXEMPLE ';
  '      A = SRmat(m1)  => une matrice SRE identique a m1';
' ';'--------------------']
Affiche(msg)


function [sr]=SRparal(p1,p2,pi1,pi2)
// Produit parallele ACFAS de 2 matrices SRE
[%L,%0,%I]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,4)
if ~err then, err=~SRvalide(p1), end
if ~err then, err=~SRvalide(p2), end
if ~err then, err=~ValideArgType(pi1,2), end
if ~err then, err=~ValideArgType(pi2,2), end
if ~err then
  [srm1,li1,co1]=SRxtract(p1)
```

```
  [srm2,li2,co2]=SRxtract(p2)
  nb=co1*li2
  pi1=SRpi1(nb,li1,pi1)
  pi2=SRpi2(nb,li2,pi2)
  n=-1
  x1=(pi1*p1*pi1^n)-(pi2*p2*pi2^n)
  x2=(pi1*p1*pi1^n)-(pi2*pi2^n)
  x3=(pi1*pi1^n)-(pi2*p2*pi2^n)
  sr=x1+x2+x3
end


function SRparalA(x1,x2,x3,x4)
// Ancienne fonction
disp('La fonction SRparalA a ete renommee SRparal')


function SRparalAH()
// Ancienne fonction
disp('La fonction SRparalAH a ete renommee SRparalH')


function SRparalH()
// Fonction d'aide pour SRparal
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRparal> calcul le produit parallele ';
 '                pour 2 matrices SRE ';
 ' ';
 'APPEL ';
 '      var = SRparal(m1,m2,''textePI1'',''textePI2'') ';
 ' ';
 'PARAMETRES ';
 '      m1    : premiere matrice symbolique';
 '      m2    : deuxieme matrice symbolique';
 '      textePI1   : expression a inserer dans la matrice symbolique PI1 ';
 '      textePI2   : expression a inserer dans la matrice symbolique PI2 ';
 '      var   : variable qui recoit le resultat ';
 ' ';
 'EXEMPLE ';
 '      E = SRparal(A,B,''w1'',''w2'')  => E contient le resultat ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRparalE(p1,p2,pi1,pi2,ppi1,ppi2)
// Produit parallele entrelacant de 2 matrices SRE
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,6)
if ~err then, err=~SRvalide(p1), end
if ~err then, err=~SRvalide(p2), end
```

```
if ~err then, err=~ValideArgType(pi1,2), end
if ~err then, err=~ValideArgType(pi2,2), end
if ~err then, err=~ValideArgType(ppi1,2), end
if ~err then, err=~ValideArgType(ppi2,2), end
if ~err then
  [srm1,li1,co1]=SRxtract(p1)
  [srm2,li2,co2]=SRxtract(p2)
  nb=co1*li2
  pi1=SRpi1(nb,li1,pi1)
  pi2=SRpi2(nb,li2,pi2)
  ppi1=SRpi1(nb,li1,ppi1)
  ppi2=SRpi2(nb,li2,ppi2)
  n=-1
  x1=(pi1*p1*pi1^n)-(ppi2*ppi2^n)
  x2=(ppi1*ppi1^n)-(pi2*p2*pi2^n)
  sr=x1+x2
end


function SRparalEH()
// Fonction d'aide pour SRparalE
msg=['---------------------';
 'DESCRIPTION ';
 '      <SRparalE> calcul le produit parallele entrelacant ';
 '                 pour 2 matrices SRE ';
 ' ';
 'APPEL ';
 '      var = SRparalE(m1,m2,''tp1'',''tp2'',''tp1p'',''tp2p'')';
 ' ';
 'PARAMETRES ';
 '      m1    : premiere matrice symbolique';
 '      m2    : deuxieme matrice symbolique';
 '      tp1   : expression a inserer dans la matrice symbolique PI1 ';
 '      tp2   : expression a inserer dans la matrice symbolique PI2 ';
 '      tp1p  : expression a inserer dans la matrice symbolique PI1 propre ';
 '      tp2p  : expression a inserer dans la matrice symbolique PI2 propre ';
 '      var   : variable qui recoit le resultat ';
 ' ';
 'EXEMPLE ';
 '      G = SRparalE(A,B,''w1'',''w2'',''wp1'',''wp2'') ';
 '                => G contient le resultat ';
' ';'---------------------']
Affiche(msg)


function [sr]=SRparalEP(p1,p2,pi1,pi2)
// Produit parallele entrelacant particulier de 2 matrices SRE
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,4)
if ~err then, err=~SRvalide(p1), end
if ~err then, err=~SRvalide(p2), end
```

```
if ~err then, err=~ValideArgType(pi1,2), end
if ~err then, err=~ValideArgType(pi2,2), end
if ~err then
  [srm1,li1,co1]=SRxtract(p1)
  [srm2,li2,co2]=SRxtract(p2)
  nb=co1*li2
  pi1=SRpi1(nb,li1,pi1)
  pi2=SRpi2(nb,li2,pi2)
  n=-1
  x1=(pi1*p1*pi1^n)-(pi2*pi2^n)
  x2=(pi1*pi1^n)-(pi2*p2*pi2^n)
  sr=x1+x2
end


function SRparalEPH()
// Fonction d'aide pour SRparalEP
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRparalEP> calcul un cas particulier du produit parallele ';
 '                  entrelacant pour 2 matrices SRE ';
 ' ';
 'APPEL ';
 '      var = SRparalEP(m1,m2,''textePI1'',''textePI2'') ';
 ' ';
 'PARAMETRES ';
 '      m1    : premiere matrice symbolique';
 '      m2    : deuxieme matrice symbolique';
 '      textePI1   : expression a inserer dans la matrice symbolique PI1 ';
 '      textePI2   : expression a inserer dans la matrice symbolique PI2 ';
 '      var   : variable qui recoit le resultat ';
 ' ';
 'EXEMPLE ';
 '      E = SRparalEP(A,B,''w1'',''w2'')  => E contient le resultat ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRparalS(p1,p2,pi1,pi2)
// Produit parallele synchrone de 2 matrices SRE
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,4)
if ~err then, err=~SRvalide(p1), end
if ~err then, err=~SRvalide(p2), end
if ~err then, err=~ValideArgType(pi1,2), end
if ~err then, err=~ValideArgType(pi2,2), end
if ~err then
  [srm1,li1,co1]=SRxtract(p1)
  [srm2,li2,co2]=SRxtract(p2)
  nb=co1*li2
  pi1=SRpi1(nb,li1,pi1)
```

```
  pi2=SRpi2(nb,li2,pi2)
  n=-1
  sr=(pi1*p1*pi1^n)-(pi2*p2*pi2^n)
end


function SRparalSH()
// Fonction d'aide pour SRparalS
msg=['---------------------';
 'DESCRIPTION ';
 '      <SRparalS> calcul le produit parallele synchrone ';
 '                 pour 2 matrices SRE ';
 ' ';
 'APPEL ';
 '      var = SRparalS(m1,m2,''textePI1'',''textePI2'') ';
 ' ';
 'PARAMETRES ';
 '      m1    : premiere matrice symbolique';
 '      m2    : deuxieme matrice symbolique';
 '      textePI1   : expression a inserer dans la matrice symbolique PI1 ';
 '      textePI2   : expression a inserer dans la matrice symbolique PI2 ';
 '      var   : variable qui recoit le resultat ';
 ' ';
 'EXEMPLE ';
 '      E = SRparalS(A,B,''w1'',''w2'')  => E contient le resultat ';
' ';'---------------------']
Affiche(msg)


function [sr]=SRpi1(n,m,objet)
// Genere une matrice SRE de format pi1
[%L,%0,%I]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~ValideArgType(n,1), end
if ~err then, err=~ValideArgType(m,1), end
if ~err & rhs==3 then, err=~ValideArgType(objet,2), end
nb=n/m
if int(nb) <> nb then
  err=%t
  disp('Dimensions incorrectes')
end
if ~err then
  if rhs==2 then, objet=%I, end
  srm=Creer(n,m,%0)
  li=0
  for i=1:m
    for j=1:nb
      li=li+1
      srm(li,i)=objet
    end
```

```
  end
  sr=tlist('sre',srm)
end


function SRpi1H()
// Fonction d'aide pour SRpi1
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRpi1> sert a creer une matrice SRE de forme PI1 ';
 ' ';
 'APPEL ';
 '      var = SRpi1(li,co,''texte'') ';
 ' ';
 'PARAMETRES ';
 '      li : nb de lignes ';
 '      co : nb de colonnes ';
 '      texte : expression a inserer dans la matrice';
 '              aux endroits appropries ';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLE ';
 '      A = SRpi1(4,2,''w1'')  => ! w1  0  ! ';
 '                                ! w1  0  ! ';
 '                                ! 0   w1 ! ';
 '                                ! 0   w1 ! ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRpi2(n,m,objet)
// Genere une matrice SRE de format pi2
[%L,%0,%I]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~ValideArgType(n,1), end
if ~err then, err=~ValideArgType(m,1), end
if ~err & rhs==3 then, err=~ValideArgType(objet,2), end
nb=n/m
if int(nb) <> nb then
  err=%t
  disp('Dimensions incorrectes')
end
if ~err then
  if rhs==2 then, objet=%I, end
  srm=Creer(n,m,%0)
  li=0
  for i=1:nb
    for j=1:m
      li=li+1
      srm(li,j)=objet
```

```
    end
  end
  sr=tlist('sre',srm)
end


function SRpi2H()
// Fonction d'aide pour SRpi2
msg=['--------------------';
 'DESCRIPTION ';
 '      <SRpi2> sert a creer une matrice SRE de forme PI2 ';
 ' ';
 'APPEL ';
 '      var = SRpi2(li,co,''texte'') ';
 ' ';
 'PARAMETRES ';
 '      li : nb de lignes ';
 '      co : nb de colonnes ';
 '      texte : expression a inserer dans la matrice';
 '              aux endroits appropries ';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLE ';
 '      A = SRpi2(4,2,''w2'')  => ! w2  0  ! ';
 '                                ! 0   w2 ! ';
 '                                ! w2  0  ! ';
 '                                ! 0   w2 ! ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRprodBool(sr1,sr2)
// Produit booleen de 2 matrices SRE
[%L,%0,%I,%unio,%inte,%prod,%inve,%comp,%resg,%resd]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~(SRvalide(sr1) & SRvalide(sr2)), end
if ~err then
  [srm1,l1,c1]=SRxtract(sr1)
  [srm2,l2,c2]=SRxtract(sr2)
  if c1<>l2 then
     disp('Dimensions incorrectes?')
     err=%t
  end
end
if ~err then
  srm1=Map(Trim,srm1)
  srm2=Map(Trim,srm2)
  srm=Creer(l1,c2,%0)
  for i=1:l1
    for j=1:c2
```

```
      for k=1:c1
         srm(i,j)=SRunionC(srm(i,j),SRinterC(srm1(i,k),srm2(k,j)))
      end
    end
  end
  sr=tlist('sre',srm)
end


function SRprodBoolH()
// Fonction d'aide pour SRprodBool
msg=['---------------------';
 'DESCRIPTION ';
 '      <SRprodBool> fait le produit booleen de 2 matrices SRE ';
 ' ';
 'APPEL ';
 '      var = SRprodBool(m1,m2)          ';
 ' ';
 'PARAMETRES ';
 '      m1   : premiere matrice symbolique';
 '      m2   : deuxieme matrice symbolique';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLE ';
 '       C = SRprodBool(A,B)  => C contient le produit booleen de A et B ';
' ';'---------------------']
Affiche(msg)


function [sr]=SRproduit(sr1,sr2)
// Produit relationnel de 2 matrices SRE
[%L,%0,%I,%unio,%inte,%prod,%inve,%comp,%resg,%resd]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~(SRvalide(sr1) & SRvalide(sr2)), end
if ~err then
  [srm1,l1,c1]=SRxtract(sr1)
  [srm2,l2,c2]=SRxtract(sr2)
  if c1<>l2 then
     disp('Dimensions incorrectes?')
     err=%t
  end
end
if ~err then
  srm1=Map(Trim,srm1)
  srm2=Map(Trim,srm2)
  srm=Creer(l1,c2,%0)
  for i=1:l1
    for j=1:c2
      for k=1:c1
         srm(i,j)=SRunionC(srm(i,j),SRproduitC(srm1(i,k),srm2(k,j)))
```

```
      end
    end
  end
  sr=tlist('sre',srm)
end


function [c]=SRproduitC(c1,c2)
// Produit de 2 expressions
e=emptystr()
c=e
if c1==e | c2==e
   c='?'
elseif c1==%0 | c2==%0
  c=%0
elseif c1==%L & c2==%L
  c=%L
elseif c1==%I
  c=c2
elseif c2==%I
  c=c1
end
if c==e then
  p1=Presence(c1,[%unio,%inte,%resg,%resd])
  p2=Presence(c2,[%unio,%inte,%resg,%resd])
  carre1=(part(c1,1)=='[' & part(c1,length(c1))==']')
  carre2=(part(c2,1)=='[' & part(c2,length(c2))==']')
  if p1 & ~carre1 then, c1='('+c1+')', end
  if p2 & ~carre2 then, c2='('+c2+')', end
  c=c1+%prod+c2
end


function SRproduitH()
// Fonction d'aide pour SRproduit
msg=['--------------------';
 'DESCRIPTION ';
 '      <*> ou <SRproduit> fait le produit de 2 matrices SRE ';
 ' ';
 'APPELS ';
 '      var = m1 * m2 ';
 '      var = SRproduit(m1,m2)          ';
 ' ';
 'PARAMETRES ';
 '      m1   : premiere matrice symbolique';
 '      m2   : deuxieme matrice symbolique';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLES ';
 '       C = A*B             => C contient le produit de A et B ';
 '       C = SRproduit(A,B)  => C contient le produit de A et B ';
' ';'--------------------']
```

```
Affiche(msg)


function [sr]=SRremplace(sr1,table)
// Substitution de chaines de caracteres dans une matrice SRE
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
//if ~err then, err=~SRvalide(sr1), end
if ~err then
  [li,co]=size(table)
  if co<>2 & typeof(table)<>'character' then
       err=%t
       disp('Table de conversion incorrecte')
    end
end
if ~err then
  [srm1,l1,c1]=SRxtract(sr1)
  srm=srm1
  for i=1:l1
    for j=1:c1
      srm(i,j)=SRremplaceC(srm(i,j),table)
    end
  end
  sr=tlist('sre',srm)
end


function [c]=SRremplaceC(c1,table)
// Substitution de chaines de caracteres dans une expression
c=c1
[li,co]=size(table)
for i=1:li
   c=SRremplaceCC(c,table(i,1),table(i,2))
end


function [c]=SRremplaceCC(c1,car1,car2)
// Substitution d'une chaine de caracteres dans une expression
c=emptystr()
n=length(car1)
l=length(c1);j=1
while (j<=l)
  if part(c1,[j:j-1+n])==car1 then
    c=c+car2
    j=j+n
  else
    c=c+part(c1,j)
    j=j+1
  end
end
```

```
function SRremplaceH()
// Fonction d'aide pour SRremplace
msg=['---------------------';
 'DESCRIPTION ';
 '      <SRremplace> permet de substituer des expressions';
 ' ';
 'APPEL ';
 '      var = SRremplace(mat,table) ';
 ' ';
 'PARAMETRES ';
 '      mat : matrice SRE a modifier ';
 '      table : table de conversion (une matrice caractere de ';
 '              dimension n x 2)';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLES ';
 '      Soit  A <= ! a b c ! ';
 ' ';
 '      B = SRremplace(A,[''a'',''w1''])  => ! w1 b c ! ';
 ' ';
 '      Soit  tab <= [''a'',''w1'';''c'',''{Un}''] ';
 ' ';
 '      A = SRremplace(A,tab)  => ! w1 b {Un} ! ';
' ';'---------------------']
Affiche(msg)


function [sr]=SRresd(sr1,sr2)
// Residu a droite de 2 matrices SRE
[%L,%0,%I,%unio,%inte,%prod,%inve,%comp,%resg,%resd]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~(SRvalide(sr1) & SRvalide(sr2)), end
if ~err then
  [srm1,l1,c1]=SRxtract(sr1)
  [srm2,l2,c2]=SRxtract(sr2)
  if l1<>l2 then
     disp('Dimensions incompatibles')
     err=%t
  end
end
if ~err then
  srm1=Map(Trim,srm1)
  srm2=Map(Trim,srm2)
  srm=Creer(c1,c2,%L)
  for i=1:c1
    for j=1:c2
      for k=1:l1
         srm(i,j)=SRinterC(srm(i,j),SRresdC(srm1(k,i),srm2(k,j)))
      end
```

```
    end
  end
  sr=tlist('sre',srm)
end


function [c]=SRresdC(c1,c2)
// Residu a droite de 2 expressions
e=emptystr()
c=e
if c1==e | c2==e
   c='?'
elseif c2==%L
  c=%L
elseif c1==%L & c2==%0
  c=%0
elseif c1==%0
  c=%L
elseif c1==%I
  c=c2
end
if c==e then
  p1=Presence(c1,[%unio,%inte,%prod,%resg,%resd])
  p2=Presence(c2,[%unio,%inte,%prod,%resg,%resd])
  carre1=(part(c1,1)=='[' & part(c1,length(c1))==']')
  carre2=(part(c2,1)=='[' & part(c2,length(c2))==']')
  if p1 & ~carre1 then, c1='('+c1+')', end
  if p2 & ~carre2 then, c2='('+c2+')', end
  c=c1+%resd+c2
end


function SRresdH()
// Fonction d'aide pour SRresd
msg=['---------------------';
 'DESCRIPTION ';
 '      <\> ou <SRresd> calcul le residu a droite de 2 matrices SRE ';
 ' ';
 'APPELS ';
 '      var = m1 \ m2 ';
 '      var = SRresd(m1,m2)          ';
 ' ';
 'PARAMETRES ';
 '      m1   : premiere matrice symbolique';
 '      m2   : deuxieme matrice symbolique';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLES ';
 '      C = A\B          => C contient le residu a droite de A et B';
 '      C = SRresd(A,B)  => C contient le residu a droite de A et B';
' ';'---------------------']
Affiche(msg)
```

```
function [sr]=SRresg(sr1,sr2)
// Residu a gauche de 2 matrices SRE
[%L,%0,%I,%unio,%inte,%prod,%inve,%comp,%resg,%resd]=SRsign()
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~(SRvalide(sr1) & SRvalide(sr2)), end
if ~err then
  [srm1,l1,c1]=SRxtract(sr1)
  [srm2,l2,c2]=SRxtract(sr2)
  if c1<>c2 then
     disp('Dimensions incorrectes?')
     err=%t
  end
end
if ~err then
  srm1=Map(Trim,srm1)
  srm2=Map(Trim,srm2)
  srm=Creer(l1,l2,%L)
  for i=1:l1
    for j=1:l2
      for k=1:c1
         srm(i,j)=SRinterC(srm(i,j),SRresgC(srm1(i,k),srm2(j,k)))
      end
    end
  end
  sr=tlist('sre',srm)
end


function [c]=SRresgC(c1,c2)
// Residu a gauche de 2 expressions
e=emptystr()
c=e
if c1==e | c2==e
   c='?'
elseif c1==%L
  c=%L
elseif c2==%I
  c=c1
elseif c2==%0
  c=%L
elseif c1==%0 & c2==%L
  c=%0
end
if c==e then
  p1=Presence(c1,[%unio,%inte,%prod,%resg,%resd])
  p2=Presence(c2,[%unio,%inte,%prod,%resg,%resd])
  carre1=(part(c1,1)=='[' & part(c1,length(c1))==']')
  carre2=(part(c2,1)=='[' & part(c2,length(c2))==']')
```

```
  if p1 & ~carre1 then, c1='('+c1+')', end
  if p2 & ~carre2 then, c2='('+c2+')', end
  c=c1+%resg+c2
end


function SRresgH()
// Fonction d'aide pour SRresg
msg=['---------------------';
 'DESCRIPTION ';
 '      </> ou <SRresg> calcul le residu a gauche de 2 matrices SRE ';
 ' ';
 'APPELS ';
 '      var = m1 / m2 ';
 '      var = SRresg(m1,m2)           ';
 ' ';
 'PARAMETRES ';
 '      m1   : premiere matrice symbolique';
 '      m2   : deuxieme matrice symbolique';
 '      var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLES ';
 '      C = A/B          => C contient le residu a gauche de A et B';
 '      C = SRresg(A,B)  => C contient le residu a gauche de A et B';
' ';'---------------------']
Affiche(msg)


function [sr]=SRresidud(sr1,sr2)
// Residu a droite developpe de 2 matrices SRE
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~SRvalide(sr1), end
if ~err then, err=~SRvalide(sr2), end
if ~err then
  n=-1
  sr=(sr1^n*sr2')'
end


function SRresidudH()
// Fonction d'aide pour SRresidud
msg=['---------------------';
 'DESCRIPTION ';
 '      <SRresidud> calcul le residu a droite developpe ';
 '                  de 2 matrices SRE  ';
 ' ';
 'APPEL ';
 '      var = SRresidud(m1,m2)           ';
 ' ';
 'PARAMETRES ';
```

```
  '       m1   : premiere matrice symbolique';
  '       m2   : deuxieme matrice symbolique';
  '       var : variable qui recoit le resultat ';
  ' ';
  'EXEMPLE ';
  '       C = SRresidud(A,B)  => C contient le residu a droite developpe ';
  '                              de A et B ';
' ';'--------------------']
Affiche(msg)


function [sr]=SRresidug(sr1,sr2)
// Residu a gauche developpe de 2 matrices SRE
sr=[];err=%f
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~SRvalide(sr1), end
if ~err then, err=~SRvalide(sr2), end
if ~err then
  n=-1
  sr=(sr1'*sr2^n)'
end


function SRresidugH()
// Fonction d'aide pour SRresidug
msg=['--------------------';
 'DESCRIPTION ';
 '       <SRresidug> calcul le residu a gauche developpe ';
 '                   de 2 matrices SRE  ';
 ' ';
 'APPEL ';
 '       var = SRresidug(m1,m2)           ';
 ' ';
 'PARAMETRES ';
 '       m1   : premiere matrice symbolique ';
 '       m2   : deuxieme matrice symbolique ';
 '       var : variable qui recoit le resultat ';
 ' ';
 'EXEMPLE ';
 '       C = SRresidug(A,B)  => C contient le residu a gauche developpe ';
 '                              de A et B ';
' ';'--------------------']
Affiche(msg)


function [%L,%0,%I,%unio,%inte,%prod,%inve,%comp,%resg,%resd]=SRsign()
// Lecture des symboles a utiliser
%L=%SR(1)
%0=%SR(2)
%I=%SR(3)
%unio=%SR(4)
```

```
%inte=%SR(5)
%prod=%SR(6)
%inve=%SR(7)
%comp=%SR(8)
%resg=%SR(9)
%resd=%SR(10)


function [%SR]=SRsymb(v1)
// Modifications de la table des symboles
[lhs,rhs]=argn(0)
symb=SRinit()
if rhs==0 then
  v=SRxSymb()
else
  v=string(v1)
end
v=Map(Trim,v)
if SRvalidSymb(v) then
  %SR=v
else
  %SR=symb
end


function SRsymbH()
// Fonction d'aide pour SRsymb
msg=['---------------------';
 'DESCRIPTION ';
 '      <SRsymb> sert a modifier les symboles (signes) utilises ';
 ' ';
 'APPELS ';
 '      %SR = SRsymb() ';
 '      %SR = SRsymb(vecteur)     ';
 ' ';
 'PARAMETRES ';
 '      vecteur : vecteur des nouveaux symboles';
 '      %SR : variable de systeme qui contient les symboles ';
 ' ';
 'EXEMPLES ';
 '      %SR = SRsymb()    => redefinition des symboles en utilisant ';
 '                           l''editeur de symboles ';
 '      %SR = SRsymb(x)   => redefinition des symboles en utilisant ';
 '                           le contenu du vecteur x';
' ';'---------------------']
Affiche(msg)


function [sr]=SRunion(sr1,sr2)
// Union de 2 matrices SRE
[%L,%0,%I,%unio,%inte,%prod,%inve,%comp,%resg,%resd]=SRsign()
sr=[];err=%f
```

```
[lhs,rhs]=argn(0)
err=~ValideArgNb(rhs,2)
if ~err then, err=~(SRvalide(sr1) & SRvalide(sr2)), end
if ~err then
  [srm1,l1,c1]=SRxtract(sr1)
  [srm2,l2,c2]=SRxtract(sr2)
  if l1<>l2 | c1<>c2 then
     disp('Dimensions incompatibles')
     err=%t
  end
end
if ~err
  srm1=Map(Trim,srm1)
  srm2=Map(Trim,srm2)
  srm=Map(SRunionC,srm1,srm2)
  sr=tlist('sre',srm)
end


function [c]=SRunionC(c1,c2)
// Union de 2 expressions
e=emptystr()
c=e
if c1==e | c2==e
  c='?'
elseif c1==c2
  c=c1
elseif c1==%L | c2==%L
  c=%L
elseif c1==%0
  c=c2
elseif c2==%0
  c=c1
end
if c==e then
  p1=Presence(c1,%inte);p2=Presence(c2,%inte)
  carre1=(part(c1,1)=='[' & part(c1,length(c1))==']')
  carre2=(part(c2,1)=='[' & part(c2,length(c2))==']')
  if p1 & ~carre1 then, c1='('+c1+')', end
  if p2 & ~carre2 then, c2='('+c2+')', end
  c=c1+%unio+c2
end


function SRunionH()
// Fonction d'aide pour SRunion
msg=['--------------------';
 'DESCRIPTION ';
 '      <+> ou <SRunion> fait l''union de 2 matrices SRE ';
 ' ';
 'APPELS ';
 '      var = m1 + m2 ';
```

```
  '      var = SRunion(m1,m2)           ';
  ' ';
  'PARAMETRES ';
  '      m1  : premiere matrice symbolique';
  '      m2  : deuxieme matrice symbolique';
  '      var : variable qui recoit le resultat ';
  ' ';
  'EXEMPLES ';
  '      C = A+B           => C contient l''union de A et B ';
  '      C = SRunion(A,B)  => C contient l''union de A et B ';
' ';'--------------------']
Affiche(msg)


function [b]=SRvalidSymb(v)
// Validation du vecteur pour la table des symboles
b=%t
[li,co]=size(v)
if li<>1 | co<>10 then
  disp('Dimensions incorrectes')
  b=%f
else
  e=emptystr()
  for i=1:co
    if v(i)==e | v(i)==' ' then, b=%f, end
  end
  if ~b then, disp('Contenu incorrect'), end
end
if b then
   i=1
   while i<=co & b
      j=i+1
      while j<=co & b
        if i<>j then, b=~(v(i)==v(j)), end
        j=j+1
      end
      i=i+1
   end
   if ~b then, disp('Elements non unique'), end
end
if ~b then, disp('Operateurs refuses'), end


function [b]=SRvalide(sr)
// Valide si c'est une matrice SRE
b=%t
if ~(typeof(sr)=='tlist' | typeof(sr)=='list')
   b=%f
elseif sr(1)<>'sre'
   b=%f
elseif typeof(sr(2))<>'character'
   b=%f
```

```
end
if ~b then
  disp('L''argument n''est pas une matrice symbolique')
end


function SRversion()
// Affiche la version de la librairie
disp('Version du 97-03-01 (scilab 2.2)')


function SRversionH()
// Fonction d'aide pour SRversion
msg=['---------------------';
 'DESCRIPTION ';
 '       <SRversion> affiche la version de la librairie';
 ' ';
 'APPEL ';
 '       SRversion() ';
 ' ';
 'PARAMETRE ';
 '       aucun ';
 ' ';
 'EXEMPLE ';
 '       SRversion()  => Version du 97-03-01 (scilab 2.2)';
' ';'---------------------']
Affiche(msg)


function [v]=SRxSymb()
// Edition de la table des symboles
v=%SR
labelv=['Relation Universelle';'Relation Vide';'Relation Identite';
        'Signe union';'Signe intersection';'Signe produit';
        'Signe inverse';'Signe complement';
        'Signe residu a gauche';'Signe residu a droite']
labelh=['Symboles']
msg=['Entrez vos preferences']
v=x_mdialog(msg,labelv,labelh,v')'
if v==[] then, v=%SR, end


function [srm,li,co]=SRxtract(sr)
// Extrait les informations d'une matrice SRE
srm=sr(2)
[li,co]=size(srm)


function [c]=Trim(chaine)
// Enleve les blancs au debut et a la fin d'une chaine de caracteres
// mais conserve au moins un blanc si une chaine de blancs
c=chaine
```

```
l=length(chaine)
if l<>0 then
  if part(chaine,1)==' ' | part(chaine,1)==' ' then
    i=1
    while part(chaine,i)==' ' & i<l do
      i=i+1
    end
    j=l
    while part(chaine,j)==' ' & j>i do
      j=j-1
    end
    c=part(chaine,[i:j])
  end
end


function [b]=ValideArgNb(rh,n)
// Valide le nombre d'arguments
b=%t
if rh<n then
  disp('Nombre d''arguments incorrect')
  b=%f
end


function [b]=ValideArgType(x,no)
// Valide le type d'un argument
b=%t
typ=typeof(x)
[li,co]=size(x)
nb=li*co
select no
  case 1 then b=(typ=='usual' & nb==1)
  case 2 then b=(typ='character' & nb==1)
end
if ~b then
   disp('Type d''argument incorrect')
end


function [mat]=Xmatc(msg,mat1)
// Edition d'une matrice caractere
[n,m]=size(mat1)
row='lig';labelv=row(ones(1,n))+string(1:n)'
col='col';labelh=col(ones(1,m))+string(1:m)'
z=x_mdialog(msg,labelv,labelh,mat1)
if z==[] then
  mat=mat1
else
  mat=z
end
```

```
function [mat]=Xmatn(msg,mat1)
// Edition d'une matrice numerique
[n,m]=size(mat1)
row='lig';labelv=row(ones(1,n))+string(1:n)'
col='col';labelh=col(ones(1,m))+string(1:m)'
z=evstr(x_mdialog(msg,labelv,labelh,string(mat1)))
if z==[] then
  mat=mat1
else
  mat=z
end


function [sr]=%sreasre(sr1,sr2)
// Definition du signe plus <+> comme etant l'union
// de 2 matrices symboliques
sr=SRunion(sr1,sr2)


function [sr]=%srelsre(sr1,sr2)
// Definition du signe barre oblique inverse <\> comme etant
// le residu a droite de 2 matrices symboliques
sr=SRresd(sr1,sr2)


function [sr]=%sremsre(sr1,sr2)
// Definition du signe multiplication <*> comme etant le produit
// de 2 matrices symboliques
sr=SRproduit(sr1,sr2)


function [sr]=%sreps(sr1,n)
// Definition du signe exposant <^> comme etant l'exposant
// d'une matrice symbolique (-1 etant l'inverse)
if n<0 then
  sr1=SRinv(sr1)
  n=abs(n)
end
sr=sr1
i=1
while i<n & length(sr)<>0
  sr=SRproduit(sr,sr1)
  i=i+1
end
if n==0 then
   [li,co]=size(sr1(2))
   if li<>co then
     disp('Dimensions incorrectes?')
     sr=[]
   else
     sr=SRI(li)
```

```
   end
end


function [sr]=%srersre(sr1,sr2)
// Definition du signe barre oblique </> comme etant le residu a gauche
// de 2 matrices symboliques
sr=SRresg(sr1,sr2)


function [sr]=%sressre(sr1,sr2)
// Definition du signe moins <-> comme etant l'intersection
// de 2 matrices symboliques
sr=SRinter(sr1,sr2)


function [sr]=%sret(sr1)
// Definition du signe transpose <'> comme etant le complement
// d'une matrice symbolique
sr=SRcomp(sr1)
```

Bibliographie

[1] E. Armando, A. Giovini, et G. Niesi. CoCoA User's Manual. Italie, 1991.

[2] J.M. Auteberg. *Langages Algébriques*. Masson, Paris, 1987.

[3] J. Backus. Can programming be liberated from the Von Neumann style? A functional style and its algebra of programs. *Comm. ACM*, 21(8):613–641, août 1978.

[4] C. Batut, D. Bernadi, H. Cohen, et M. Olivier. User's Guide to PARI-GP. ftp: //megrez.math.u-bordeaux.fr/pub/pari/, 1993.

[5] R. Berghammer. Relational Specification of Data Types and Programs. Rapport technique 9109, Fakultät für Informatik, Universität der Bundeswehr München, Allemagne, septembre 1991.

[6] C. Brink, W. Kahl, et G. Schmidt (éditeurs). *Relational Methods in Computer Science*. Springer, 1997.

[7] A. Caron. Spécifications exécutables. Rapport de projet en informatique mathématique, Dép. d'informatique, Univ. Laval, Québec, Canada, avril 1991.

[8] B. Chaib-draa, J. Desharnais, R. Khédri, I. Jarras, S. Sayadi, et F. Tchier. Une approche relationnelle à la décomposition parallèle. *Congrès ACFAS 94 « Méthodes mathématiques pour la synthèse des systèmes informatiques »*, p. 89–103, UQAM, Montréal, Québec, Canada, 1994.

[9] B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, et S. Watt. *Maple V Language Reference Manual*. Springer-Verlag, 1991.

[10] A. Ben Cherifa. Preuves de terminaison des systèmes de réécriture - Un outil fondé sur les interprétations polynomiales. Thèse de doctorat, Univ. de Nancy I, Centre de recherche en informatique de Nancy, 1986.

[11] L.H. Chin et A. Tarski. Distributive and modular laws in the arithmetic of relation algebras. *Univ. California Publ. Math.*, 1:341–384, 1951.

[12] J. Desharnais. Abstract relational semantics. Thèse de doctorat, School of Computer Science, Univ. McGill, Montréal, juillet 1989.

[13] J. Desharnais, N. Belkhiter, B.M. Sghaier, S. Tchier, F. Tchier, A. Jaoua, A. Mili, et N. Zaguia. Embedding a Demonic Semilattice in a Relation Algebra. *Theoretical Computer Science*, 149(2):333–360, octobre 1995.

[14] J. Desharnais, A. Jaoua, N. Belkhiter, et F. Tchier. Data Refinement in a Relation Algebra. Fondation Nationale de la Recherche Scientifique (édit.), *2e Conf. Maghrébine en génie logiciel et intelligence artificielle*, p. 222–236, Tunis, Tunisie, avril 1992.

[15] J. Desharnais, A. Jaoua, F. Mili, N. Boudrigua, et A. Mili. A relational division operator: The conjugate kernel. *Theoretical Computer Science*, 114:247–272, 1993.

[16] J. Desharnais et N.H. Madhavji. Abstract relational specifications. *IFIP TC 2 Working Conf. on Programming Concepts and Methods*, p. 267–284, Israël, avril 1990.

[17] J. Desharnais, A. Mili, et F. Mili. On the mathematics of sequential decompositions. *Sci. Comput. Program*, 20:253–289, 1993.

[18] R. Dewar et J. VandeKopple. The SETLS Programming Language. ftp://cs.nyu.edu/pub /languages/setls/, 1994.

[19] K. Dickey. The Scheme programming language. ftp://nexus.yorku.ca/pub/scheme/.

[20] E.W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[21] J.W. Eaton. Octave FAQ. ftp://ftp.che.utexas.edu/pub/octave/, 1994.

[22] H. Ehrig, B. Mahr, I. Classen, et F. Orejas. Introduction to algebraic specification. Part 1: Formal methods for software development. *The Computer Journal*, 35(5):460–470, 1992.

[23] H. Ehrig, B. Mahr, I. Classen, et F. Orejas. Introduction to algebraic specification. Part 2: From classical view to foundations of system specifications. *The Computer Journal*, 35(5):471–480, 1992.

[24] W. Ellis, E. Johnson, E. Lodi, et D. Schwalbe. *Maple V Flight Manual*. Symbolic Computation Series. Brooks/Cole Publishing Company, Pacific Grove, California, 1992.

[25] W.M. Farmer, J.D. Guttman, et F.J. Thayer. IMPS Version 1.2. ftp://math.harvard.edu /imps/, 1994.

[26] S.J. Garland et J.V. Guttag. LP, the Larch Prover: User and Reference Manual. ftp: //larch.lcs.mit.edu/pub/Larch/lp/, 1994.

[27] M. Gerberg et E.J. Moore. Matcalc User Manual. Kensington, Australie.

[28] Scilab Group. Introduction to Ψlab. INRIA Meta2 Project/ENPC Cergrene, France, 1996.

[29] M. Hermann, C. Kirchner, et H. Kirchner. Implementation of term rewriting. *The Computer Journal*, 34:20–23, février 1991.

[30] G. Huet et D. Oppen. Equations and rewrite rules: a survey. R. Book (édit.), *Formal Languages: Perspective and Open Problems*, Academic Press. 1980.

[31] The Math Works inc. *The Student Edition of Matlab*. The Matlab Curriculum Series. Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[32] A. Jaffer. JACAL Symbolic Mathematics System. ftp://prep.ai.mit.edu/pub/gnu/jacal/, 1994.

[33] S. Kalvala. A Gentle Introduction to Isabelle. Computer Laboratory, Univ. of Cambridge, Royaume-Uni, 1994.

[34] M. Kantrowitz et B. Margolin. Scheme FAQ. Newsgroups:comp.lang.scheme, 1994.

[35] R. Khédri. Résolution de l'équation relationnelle $A \parallel X =_{\mathcal{A}} B$. Document de travail, Dép. d'informatique, Univ. Laval, 1994.

[36] R. Khédri. La composition entrelaçante. Document de travail, Dép. d'informatique, Univ. Laval, Québec, Canada, 1995.

[37] R. Lalement. *Logique réduction résolution*. Masson, Paris, 1990.

[38] C. Lévesque. Systèmes de réécriture. Rapport de projet en informatique mathématique, Dép. d'informatique, Univ. Laval, Québec, Canada, août 1991.

[39] S. Lipschutz. *SET Theory and Related Topics*. Schaum's outline series. McGraw-Hill, 1964.

[40] Z. Manna et R. Waldinger. *The Logical Basis for Computer Programming, Volume 1: Deductive Reasoning*. Addison-Wesley, 1985.

[41] Z. Manna et R. Waldinger. *The Logical Basis for Computer Programming, Volume 2: Deductive Systems*. Addison-Wesley, 1990.

[42] W.W. McCune. Otter 2.0 Users Guide. Argonne, Illinois, 1990.

[43] H. Melenk et W. Neun. General Information on REDUCE/PSL. ftp://bath.ac.uk/pub /jpff/reduce/, 1993.

[44] A. Mili. *An Introduction to Program Fault Tolerance*. Prentice Hall, 1990.

[45] A. Mili, J. Desharnais, et J.R. Gagné. Formal models of stepwise refinement of programs. *ACM Computing Surveys*, 18(3):231–276, septembre 1986.

[46] A. Mili, J. Desharnais, et F. Mili. *Computer Program Construction*. Oxford Univ. Press, New York NY, 1994.

[47] M. Newborn. *The Great Theorem Prover v1.0*. Newborn Software, 1989.

[48] M.J. O'Donnell. *Equational Logic as a Programming Language*. The MIT Press, Cambridge, MA, 1985.

[49] L.C. Paulson. Introduction to Isabelle. Computer Laboratory, Univ. of Cambridge, Angleterre, 1995.

[50] A.D. Robison. Illinois FP User's Manual. Department of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana, Illinois, 1987.

[51] M. Rusinowitch. *Démonstration automatique - Techniques de réécriture*. InterEditions, Paris, 1989.

[52] G. Schmidt et T. Ströhlein. *Relations and Graphs*. EATCS Monographs in Computer Science. Springer-Verlag, Berlin, 1993.

[53] P.H. Schmitt. A survey or rewrite systems. *Lecture Notes in Computer Science*, 329, 1987.

[54] I.R. Searle et P. Musumeci. Rlab Primer. ftp://csi.jpl.nasa.gov/pub/matlab/RLab/, 1994.

[55] S.B.M. Sghaier. Spécifications relationnelles sans variables. Mémoire de maîtrise, Univ. Laval, 1993.

[56] V. Sperschneider et G. Antoniou. *LOGIC: A Foundation for Computer Science.* International Computer Science Series. Addison-Wesley, 1991.

[57] A. Strotmann. The REDUCE Computer Algebra System. ftp://bath.ac.uk/pub/jpff /reduce/, 1995.

[58] A. Tarski. On the calculus of relations. *J. Symb. Log.*, 6(3):73–89, 1941.

[59] D.J. Velleman. *How To Prove It - A Structured Approach.* Cambridge University Press, 1994.

[60] J. Voas, G. McGraw, L. Kassab, et L. Voas. A 'Crystal Ball' for Software Liability. *IEEE Computer*, 30(6):29–36, 1997.

[61] J. von Holten et R. Seifert. ASpecT 2.0 User Manual. ftp://gatekeeper.dec.com/.e /language/aspect/, 1993.

[62] S. Wolfram. *Mathematica, A System for Doing Mathematics by Computer.* Addison-Wesley, 1991.

[63] R. Wood. SyMan 1.2 a SYmbolic MANipulator. ftp://archives.math.utk.edu/software /mac/collegeAlgebra/sysman/, 1994.

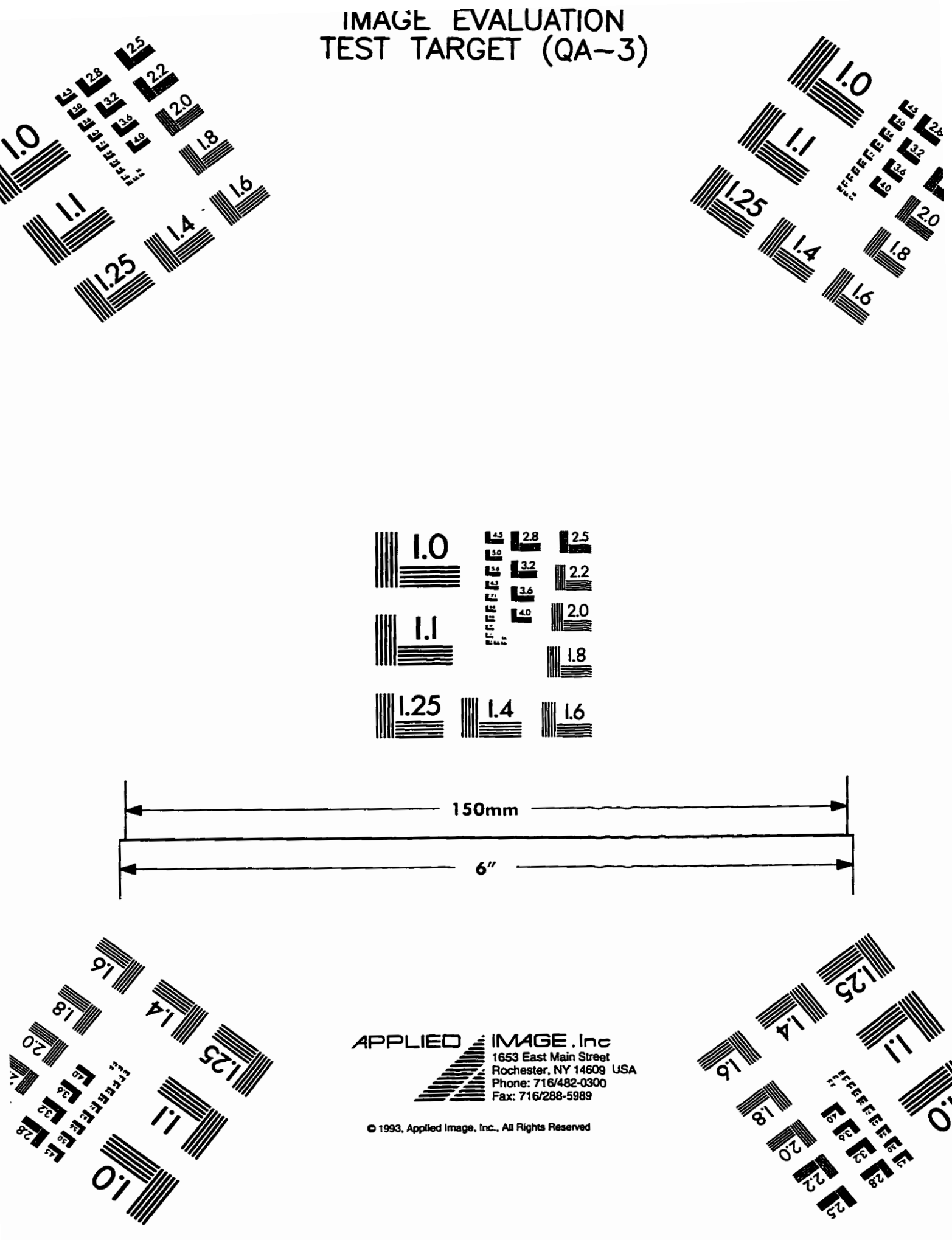


IMAGE EVALUATION
TEST TARGET (QA-3)
1.0
1.1
1.25
1.4
1.6
2.0
2.2
2.5
2.8
3.2
3.6
4.0
150mm
6"
APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989
© 1993, Applied Image, Inc., All Rights Reserved