LIFL Laboratoire d'Informatique Fondamentale de Lille

Numéro d'ordre : 1262 **Année** 1994

THESE

présentée à

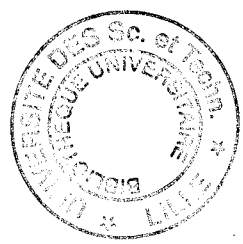
L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

pour obtenir le titre de

DOCTEUR EN INFORMATIQUE

par

Marc TOMMASI



AUTOMATES ET CONTRAINTES ENSEMBLISTES

Thèse soutenue le 1 Février 1994 , devant la commission d'examen composée de

Président	Max Dauchet
Rapporteurs	Bruno Courcelle
	Hélène Kirchner
Examinateurs	Philippe Devienne
	Rémi Gilleron
	Leszek Pacholski
	Michel Parigot
	Sophie Tison

UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE
U.F.R. d'I.E.E.A. Bât M3. 59655 Villeneuve d'Ascq CEDEX
Tél. 20.43.47.24 Fax. 20.43.65.66

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE

DOYENS HONORAIRES DE L'ANCIENNE FACULTE DES SCIENCES

M. H. LEFEBVRE, M. PARREAU

PROFESSEURS HONORAIRES DES ANCIENNES FACULTES DE DROIT ET SCIENCES ECONOMIQUES, DES SCIENCES ET DES LETTRES

MM. ARNOULT, BONTE, BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, DECUYPER, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, GERMAIN, GLACET, GONTIER, KOURGANOFF, LAMOTTE, LASSERRE, LELONG, LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, PEREZ, ROIG, ROSEAU, ROUELLE, SCHILTZ, SAVARD, ZAMANSKI, Mes BEAUJEU, LELONG.

PROFESSEUR EMERITE

M. A. LEBRUN

ANCIENS PRESIDENTS DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. M. PARREAU, J. LOMBARD, M. MIGEON, J. CORTOIS, A.DUBRULLE

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

M. P. LOUIS

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CHAMLEY Hervé	Géotechnique
M. CONSTANT Eugène	Electronique
M. ESCAIG Bertrand	Physique du solide
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. LABLACHE COMBIER Alain	Chimie
M. LOMBARD Jacques	Sociologie
M. MACKE Bruno	Physique moléculaire et rayonnements atmosphériques

M. MIGEON Michel	EUDIL
M. MONTREUIL Jean	Biochimie
M. PARREAU Michel	Analyse
M. TRIDOT Gabriel	Chimie appliquée

PROFESSEURS - 1ère CLASSE

M. BACCHUS Pierre	Astronomie
M. BIAYS Pierre	Géographie
M. BILLARD Jean	Physique du Solide
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean Pierre	Chimie-Physique
M. BOSCQ Denis	Probabilités
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BRASSELET Jean Paul	Géométrie et topologie
M. BREZINSKI Claude	Analyse numérique
M. BRIDOUX Michel	Chimie Physique
M. BRUYELLE Pierre	Géographie
M. CARREZ Christian	Informatique
M. CELET Paul	Géologie générale
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. CROSNIER Yves	Electronique
Mme DACHARRY Monique	Géographie
M. DAUCHET Max	Informatique
M. DEBOURSE Jean Pierre	Gestion des entreprises
M. DEBRABANT Pierre	Géologie appliquée
M. DECLERCQ Roger	Sciences de gestion
M. DEGAUQUE Pierre	Electronique
M. DESCHEPPER Joseph	Sciences de gestion
Mme DESSAUX Odile	Spectroscopie de la réactivité chimique
M. DHAINAUT André	Biologie animale
Mme DHAINAUT Nicole	Biologie animale
M. DJAFARI Rouhani	Physique
M. DORMARD Serge	Sciences Economiques
M. DOUKHAN Jean Claude	Physique du solide
M. DUBRULLE Alain	Spectroscopie hertzienne
M. DUPOUY Jean Paul	Biologie
M. DYMENT Arthur	Mécanique
M. FOCT Jacques Jacques	Métallurgie
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie structurale
M. FRONTIER Serge	Ecologie numérique
M. GLORIEUX Pierre	Physique moléculaire et rayonnements atmosphériques
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie-Physique
M. GRANELLE Jean Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILBAULT Pierre	Physiologie animale
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. HENRY Jean Pierre	Génie mécanique
M. HERMAN Maurice	Physique spatiale
M. LACOSTE Louis	Biologie Végétale
M. LANGRAND Claude	Probabilités et statistiques

M. LATTEUX Michel	Informatique
M. LAVEINE Jean Pierre	Paléontologie
Mme LECLERCQ Ginette	Catalyse
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique atomique et moléculaire
M. LEROY Jean Marie	Spectrochimie
M. LHENAFF René	Géographie
M. LHOMME Jean	Chimie organique biologique
M. LOUAGE Françis	Electronique
M. LOUCHEUX Claude	Chimie-Physique
M. LUCQUIN Michel	Chimie physique
M. MAILLET Pierre	Sciences Economiques
M. MAROUF Nadir	Sociologie
M. MICHEAU Pierre	Mécanique des fluides
M. PAQUET Jacques	Géologie générale
M. PASZKOWSKI Stéfan	Mathématiques
M. PETIT Francis	Chimie organique
M. PORCHET Maurice	Biologie animale
M. POUZET Pierre	Modélisation - calcul scientifique
M. POVY Lucien	Automatique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. RAMAN Jean Pierre	Sciences de gestion
M. SALMER Georges	Electronique
M. SCHAMPS Joël	Spectroscopie moléculaire
Mme SCHWARZBACH Yvette	Géométrie
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
M. SLIWA Henri	Chimie organique
M. SOMME Jean	Géographie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. THIEBAULT François	Sciences de la Terre
M. THOMAS Jean Claude	Géométrie - Topologie
M. THUMERELLE Pierre	Démographie - Géographie humaine
M. TILLIEU Jacques	Physique théorique
M. TOULOTTE Jean Marc	Automatique
M. TREANTON Jean René	Sociologie du travail
M. TURRELL Georges	Spectrochimie infrarouge et raman
M. VANEECLOO Nicolas	Sciences Economiques
M. VAST Pierre	Chimie inorganique
M. VERBERT André	Biochimie
M. VERNET Philippe	Génétique
M. VIDAL Pierre	Automatique
M. WALLART Françis	Spectrochimie infrarouge et raman
M. WEINSTEIN Olivier	Analyse économique de la recherche et développement
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ABRAHAM Francis	
M. ALLAMANDO Etienne	Composants électroniques
M. ANDRIES Jean Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BALL Steven	Génétique
M. BART André	Biologie animale
M. BASSERY Louis	Génie des procédés et réactions chimiques
Mme BATTIAU Yvonne	Géographie
M. BAUSIERE Robert	Systèmes électroniques
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique atomique et moléculaire
M. BERNAGE Pascal	Physique atomique, moléculaire et du rayonnement
M. BERTHOUD Arnaud	Sciences Economiques
M. BERTRAND Hugues	Sciences Economiques
M. BERZIN Robert	Analyse
M. BISKUPSKI Gérard	Physique de l'état condensé et cristallographie
M. BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie végétale
M. BOHIN Jean Pierre	Biochimie métabolique et cellulaire
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie civil
M. BOIVIN Jean Claude	Spectrochimie
M. BOUCHER Daniel	Physique
M. BOUQUELET Stéphane	Biologie appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BROCARD Jacques	Chimie
Mme BROUSMICHE Claudine	Paléontologie
M. BUISINE Daniel	Mécanique
M. CAPURON Alfred	Biologie animale
M. CARRE François	Géographie humaine
M. CATTEAU Jean Pierre	Chimie organique
M. CAYATTE Jean Louis	Sciences Economiques
M. CHAPOTON Alain	Electronique
M. CHARET Pierre	Biochimie structurale
M. CHIVE Maurice	Composants électroniques optiques
M. COMYN Gérard	Informatique théorique
Mme CONSTANT Monique	Composants électroniques et optiques
M. COQUERY Jean Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique nucléaire et corpusculaire
M. COUTURIER Daniel	Chimie organique
M. CRAMPON Norbert	Tectolique géodynamique
M. CURGY Jean Jacques	Biologie
M. DANGOISSE Didier	Physique théorique
M. DE PARIS Jean Claude	Analyse
M. DECOSTER Didier	Composants électroniques et optiques
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELAHAYE Jean Paul	Informatique
M. DELORME Pierre	Physiologie animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
Mme DEMUYNCK Claire	Physique atomique, moléculaire et du rayonnement
M. DENEL Jacques	Informatique
M. DEPREZ Gilbert	Physique du solide - christallographie

M. DERIEUX Jean Claude	Microbiologie
M. DERYCKE Alain	Informatique
M. DESCAMPS Marc	Physique de l'état condensé et cristallographie
M. DEVRAINNE Pierre	Chimie minérale
M. DEWAILLY Jean Michel	Géographie humaine
M. DHAMELINCOURT Paul	Chimie physique
M. DI PERSIO Jean	Physique de l'état condensé et cristallographie
M. DUBAR Claude	Sociologie démographique
M. DUBOIS Henri	Spectroscopie hertzienne
M. DUBOIS Jean Jacques	Géographie
M. DUBUS Jean Paul	Spectrométrie des solides
M. DUPONT Christophe	Vie de la firme
M. DUTHOIT Bruno	Génie civil
Mme DUVAL Anne	Algèbre
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FARVACQUE Jean Louis	Physique de l'état condensé et cristallographie
M. FAUQUEMBERGUE Renaud	Composants électroniques
M. FELIX Yves	Mathématiques
M. FERRIERE Jacky	Tectonique - Géodynamique
M. FISCHER Jean Claude	Chimie organique, minérale et analytique
M. FONTAINE Hubert	Dynamique des cristaux
M. FORSE Michel	Sociologie
M. GADREY Jean	Sciences économiques
M. GAMBLIN André	Géographie urbaine, industrielle et démographie
M. GOBLOT Rémi	Algèbre
M. GOURIEROUX Christian	Probabilités et statistiques
M. GREGORY Pierre	I.A.E.
M. GREMY Jean Paul	Sociologie
M. GREVET Patrice	Sciences Economiques
M. GRIMBLOT Jean	Chimie organique
M. GUELTON Michel	Chimie physique
M. GUICHAOUA André	Sociologie
M. HAIMAN Georges	Modélisation,calcul scientifique, statistiques
M. HOUDART René	Physique atomique
M. HUEBSCHMANN Johannes	Mathématiques
M. HUTTNER Marc	Algèbre
M. ISAERT Noël	Physique de l'état condensé et cristallographie
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et statistiques
M. JEAN Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KOENIG Gérard	Sciences de gestion
M. KOSTRUBIEC Benjamin	Géographie
M. KREMBEL Jean	Biochimie
Mme KRIFA Hadjila	Sciences Economiques
M. LANGEVIN Michel	Algèbre
M. LASSALLE Bernard	Embryologie et biologie de la différenciation
M. LE MEHAUTE Alain	Modélisation,calcul scientifique,statistiques
M. LEBFEVRE Yannic	Physique atomique,moléculaire et du rayonnement
M. LECLERCQ Lucien	Chimie physique
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Marc	Composants électroniques et optiques
M. LEFEVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
M. LEGRAND Michel	Astronomie - Météorologie
M. LEGRAND Pierre	Chimie
Mme LEGRAND Solange	Algèbre
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne

M. LE MAROIS Henri	Vie de la firme
M. LEMOINE Yves	Biologie et physiologie végétales
M. LESCURE François	Algèbre
M. LESENNE Jacques	Systèmes électroniques
M. LOCQUENEUX Robert	Physique théorique
Mme LOPES Maria	Mathématiques
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU François	Sciences économiques
M. MAHIEU Jean Marie	Optique - Physique atomique
M. MAIZIERES Christian	Automatique
M. MANSY Jean Louis	Géologie
M. MAURISSON Patrick	Sciences Economiques
M. MERIAUX Michel	EUDIL
M. MERLIN Jean Claude	Chimie
M. MESMACQUE Gérard	Génie mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MOCHE Raymond	Modélisation,calcul scientifique,statistiques
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie organique
M. MORE Marcel	Physique de l'état condensé et cristallographie
M. MORTREUX André	Chimie organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
M. NIAY Pierre	Physique atomique,moléculaire et du rayonnement
M. NICOLE Jacques	Spectrochimie
M. NOTELET Francis	Systèmes électroniques
M. PALAVIT Gérard	Génie chimique
M. PARSY Fernand	Mécanique
M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée
M. PERTUZON Emile	Physiologie animale
M. PETIT Daniel	Biologie des populations et écosystèmes
M. PLIHON Dominique	Sciences Economiques
M. PONSOLLE Louis	Chimie physique
M. POSTAIRE Jack	Informatique industrielle
M. RAMBOUR Serge	Biologie
M. RENARD Jean Pierre	Géographie humaine
M. RENARD Philippe	Sciences de gestion
M. RICHARD Alain	Biologie animale
M. RIETSCH François	Physique des polymères
M. ROBINET Jean Claude	EUDIL
M. ROGALSKI Marc	Analyse
M. ROLLAND Paul	Composants électroniques et optiques
M. ROLLET Philippe	Sciences Economiques
Mme ROUSSEL Isabelle	Géographie physique
M. ROUSSIGNOL Michel	Modélisation,calcul scientifique,statistiques
M. ROY Jean Claude	Psychophysiologie
M. SALERNO Françis	Sciences de gestion
M. SANCHOLLE Michel	Biologie et physiologie végétales
Mme SANDIG Anna Margarette	
M. SAWERYSYN Jean Pierre	Chimie physique
M. STAROSWIECKI Marcel	Informatique
M. STEEN Jean Pierre	Informatique
Mme STELLMACHER Irène	Astronomie - Météorologie
M. STERBOUL François	Informatique
M. TAILLIEZ Roger	Génie alimentaire
M. TANRE Daniel	Géométrie - Topologie
M. THERY Pierre	Systèmes électroniques
Mme TJOTTA Jacqueline	Mathématiques
M. TOURSEL Bernard	Informatique
M. TREANTON Jean René	Sociologie du travail

M. TURREL Georges	Spectrochimie infrarouge et raman
M. VANDIJK Hendrik	
Mme VAN ISEGHEM Jeanine	Modélisation,calcul scientifique,statistiques
M. VANDORPE Bernard	Chimie minérale
M. VASSEUR Christian	Automatique
M. VASSEUR Jacques	Biologie
Mme VIANO Marie Claude	
M. WACRENIER Jean Marie	Electronique
M. WARTEL Michel	Chimie inorganique
M. WATERLOT Michel	géologie générale
M. WEICHERT Dieter	Génie mécanique
M. WERNER Georges	Informatique théorique
M. WIGNACOURT Jean Pierre	
M. WOZNIAK Michel	Spectrochimie
Mme ZINN JUSTIN Nicole	Algèbre

Je remercie Max Dauchet qui me fait l'honneur de présider le jury de cette thèse. Ses reflexions passionnantes et passionnées m'ont souvent motivé.

J'exprime ma gratitude à Hélène Kirchner et Bruno Courcelle et pour l'interêt qu'ils ont porté à ces travaux et pour avoir accepté d'être rapporteurs.

Ce travail doit beaucoup à Sophie Tison et à Rémi Gilleron qui m'ont accordé leur confiance en DEA puis en thèse. Leurs conseils, leur soutien et leurs remarques m'ont beaucoup aidé. Je les en remercie.

Mes remerciements vont également à Philippe Devienne, Michel Parigot, Leszek Pacholski et qui me font l'honneur de participer au jury.

Je tiens également à remercier Yves, Francis et Eric pour leur sympathique compagnie dans le bureau 316, la chaleureuse équipe GRAAL et tous les membres et ex-membres du LIFL.

Je remercie surtout Anne parce que et voila.

Table des matières

Introduction

Ce mémoire est une contribution à l'étude de la résolution de systèmes de contraintes ensemblistes.

Ce travail s'intègre dans une démarche générale d'équipe, orientée vers l'utilisation des automates comme outils de décision. Par exemple, la théorie de la réécriture close est montrée décidable par M. Dauchet et S. Tison [DT90] en utilisant des automates d'arbres finis. Les problèmes liés à la non-linéarité sont abordés en définissant des automates adaptés. C'est le cas des automates avec tests d'égalités entre fils de B. Bogaert et S. Tison [BT92] et des automates de filtrage de A.C. Caron, J.L. Coquidé et M. Dauchet [CCD93a].

Dans cet esprit, nous avons introduit les Automates d'Ensembles d'Arbres. Ce sont des reconnaisseurs de n-uplets de langages d'arbres, capables de représenter de façon exploitable l'ensemble des solutions d'un système de contraintes ensemblistes. Outre leur caractère unificateur, les automates ont aussi l'avantage d'être proches des spécifications, ce qui les rend expressifs, et proches de l'algorithmique, ce qui les rend adéquats.

Grâce aux propriétés de ces automates, nous déduisons des résultats dans le cadre des contraintes ensemblistes et notamment le principal : une procédure de décision pour la satisfiabilité des contraintes ensemblistes positives ou négatives sans symbole de projection.

Cette introduction se compose de quatre parties. La première présente les contraintes ensemblistes par le biais de la problématique des types.

La deuxième expose quelques utilisations des systèmes de contraintes ensemblistes en programmation.

Une troisième partie propose un état de l'art succinct des méthodes de résolution de contraintes.

Enfin, dans la dernière partie nous présentons sur un exemple les automates d'ensembles d'arbres et leurs connexions avec les systèmes de contraintes.

"La notion de type formalise le fait qu'on ne peut appliquer n'importe quelle opération à n'importe quelle valeur. Calculer ou vérifier le type d'un programme est donc une preuve de correction partielle", comme le dit Marie-Claude Gaudel. "Le problème majeur dans ce domaine est d'être flexible tout en restant rigoureux, c'est-à-dire d'autoriser le polymorphisme (une valeur peut avoir plusieurs types) afin d'éviter les redites, et de pouvoir écrire des programmes très généraux dont la correction vis-à-vis des types reste décidable."

Le formalisme des contraintes ensemblistes est à cet égard un compromis entre expressivité et décidabilité qui fait depuis quelques années l'objet de recherches actives.

Il permet d'exprimer des relations entre ensembles de termes. Par exemple, si l'ensemble des entiers naturels est défini comme le plus petit ensemble contenant 0 et le successeur de tout entier (noté s), alors la contrainte

$$Nat = 0 \cup s(Nat) \tag{1}$$

formalise cette définition. Considérons le système suivant :

$$\begin{array}{rcl} Nat & = & 0 \cup s(Nat) \\ Liste & = & cons(Nat, Liste) \cup nil \\ Liste_+ & \subseteq & Liste \\ car(Liste_+) & \subseteq & s(Nat) \end{array} \tag{2}$$

Si la première contrainte désigne Nat comme l'ensemble des entiers naturels, la deuxième représente l'ensemble des listes d'entiers construites à la façon des listes LISP. La liste vide nil est une liste et la construction, à l'aide du "symbole" $cons$, d'un entier suivi d'une liste est une liste. Les deux dernières contraintes définissent l'ensemble $Liste_+$ des listes d'entiers dont le premier élément est non nul.

De façon plus générale, un système de *contraintes ensemblistes* est une conjonction de contraintes positives de la forme $exp \subseteq exp'$,[1] et négatives de la forme $exp \not\subseteq exp'$ dont les membres gauche et droit sont appelés *expressions ensemblistes*. Les expressions sont construites à l'aide

- de symboles de fonction ; dans l'exemple, 0, s, $cons$, nil sont des symboles de fonction.
- d'opérateurs ; l'union $\cup$, l'intersection $\cap$, le complémentaire $\sim$
- de symboles de projection ; par exemple car qui dans la dernière équation du système (2) désigne la valeur de la première composante de $cons$. Dans la syntaxe des contraintes, la notation utilisée est $cons^{-1}_{(1)}$.

[1] ou encore $exp = exp'$ pour abréger $exp \subseteq exp'$ et $exp' \subseteq exp$.

– de variables comme Nat et $Liste$.

Les variables d'un système sont interprétées comme des ensembles de termes construits uniquement à l'aide des symboles de fonction. Une interprétation qui satisfait le système est appelée une *solution*. Par exemple, $\{0, s(0), s(s(0)), \ldots\}$ est solution de l'équation (1).

L'inclusion ou l'union définissent un polymorphisme qui peut être paramétrique naturellement. Par exemple $Liste \subseteq Nil \cup cons(X, Liste)$.

Les premiers algorithmes de résolution ou de manipulation de contraintes ensemblistes, par exemple Reynolds [Rey69], Mishra [Mis84], sont apparus avec la volonté de réaliser des outils pour l'inférence et le contrôle de types.

En programmation fonctionnelle ou logique le contrôle de types est souvent dynamique. En d'autre termes, la procédure qui se charge de vérifier si une expression est bien typée intervient durant l'exécution. De grandes libertés sont donc laissées au programmeur aux dépens de la sécurité et de l'efficacité. Pour pallier à ces inconvénients, des procédures d'extraction et de vérification de types sont ajoutées à la phase de compilation. On obtient alors un système riche d'informations qui pourront être utilisées pour l'optimisation du programme.

Le texte du programme est analysé automatiquement et les informations extraites sont représentées dans un formalisme adéquat. Si les types sont considérés comme des ensembles de valeurs, alors les expressions ensemblistes sont bien adaptées pour représenter ces valeurs et les contraintes pour exprimer leurs relations. De nombreux algorithmes d'inférence de types en programmation fonctionnelle, logique ou impérative évoluent autour d'un noyau qui consiste en la résolution de contraintes ensemblistes.

Les algorithmes les plus anciens manipulent des contraintes au pouvoir d'expression assez pauvre. Le plus souvent, ces contraintes ont la propriété de toujours avoir une plus petite solution qui correspond à un ensemble régulier de termes. Ce sont alors des sortes usuelles définies par signature.[2] En conséquence, ces méthodes s'articulent autour d'algorithmes connus sur les automates finis d'arbres finis.

Afin d'obtenir des informations plus consistantes, il est nécessaire d'enrichir le vocabulaire des contraintes. Plus ce vocabulaire est riche, plus l'analyse aura les moyens d'être précise et pertinente, mais aussi, plus les solutions seront difficiles à trouver.

Néanmoins, pour que l'analyse garde un sens, une première propriété essentielle doit être préservée: la décidabilité de la satisfiabilité. Il doit exister une procédure qui détermine si un système de contraintes ensemblistes possède ou non des solutions. En d'autres termes, il faut que les informations extraites permettent de dire si les objets du programme étudié possèdent un type.

[2] Hubert Comon [Com90] a précisé les relations très étroites entre automates et sortes.

La seconde caractéristique importante est qu'il doit exister un moyen de représenter les solutions de façon exploitable. Savoir qu'un objet possède un type est aussi déterminant que de pouvoir calculer son type, ou encore, l'ensemble de ses types possibles. Le système original est déjà une représentation des solutions, mais non utilisable directement. Nous voulons le transformer en une forme résolue dont on peut dire si elle admet ou non des solutions, et dont on peut facilement "calculer" des solutions.

Le mémoire traite essentiellement de la classe des systèmes de contraintes ensemblistes positives et négatives sans symbole de projection, notée *PNSC*. L'apport principal de ce travail dans le domaine des contraintes ensemblistes est une procédure de décision pour la satisfiabilité des systèmes de la classe *PNSC*.

Contraintes Ensemblistes et Programmation

Nous distinguons deux techniques avancées parmi les utilisations des contraintes ensemblistes en programmation. La plus ancienne est l'inférence et le contrôle de types vus comme des ensembles de termes. Elle a des applications en programmation logique et fonctionnelle. La seconde a été introduite récemment par Heintze et Jaffar : l'analyse ensembliste (*Set Based Analysis*) calcule une approximation des valeurs prises par les variables lors de l'exécution d'un programme[3] logique, fonctionnel, ou impératif, et extrait des informations d'une autre nature[4].

Inférence et Contrôle de Types

Dans le développement de programmes, la vérification de types permet la détection de nombreuses erreurs et de ce fait, enrichit les contrôles du compilateur. De plus, grâce à l'utilisation de procédures d'optimisation, les informations de typage peuvent être utilisées pour augmenter l'efficacité lors de l'exécution.

L'avantage le plus immédiat est sans doute l'optimisation de la gestion de la mémoire, mais des opérations sur les structures de contrôle sont envisageables. Par exemple, savoir que y est un entier non nul évite de contrôler que la division x/y est bien définie. Un test équivalent en LISP peut aussi être évité lors de l'évaluation de $car(L)$ si L est une liste toujours non vide. En PROLOG, des clauses peuvent être écartées si le typage des paramètres d'un prédicat indique que l'unification ne sera pas possible lors de l'exécution.

Souvent, en programmation logique ou en programmation fonctionnelle, aucune indication de type n'est donnée par l'utilisateur. Des algorithmes doivent

[3]Ce qu'on peut donc voir aussi comme de l'inférence de types

[4]Il existe aussi des extensions vers l'inférence de modes et le partage de structures en programmation logique.

donc se charger de construire automatiquement les types, puis de contrôler la correction du programme. Ces sytèmes sont appelés *systèmes de typage descriptifs*, l'opération de construction est *l'inférence de types* (en anglais *type inference*) et le contrôle, la *vérification de types* (en anglais *type checking*)[CW85].

Les systèmes de typage interviennent dans des domaines très larges et de nombreux auteurs ont contribué à une évolution de ces algorithmes. On peut rencontrer plusieurs types de vérification: la vérification statique, pendant la phase de compilation, et la vérification dynamique, pendant la phase d'exécution. En cas de vérification statique (exemple ML), un programme est rejeté s'il n'est pas bien typé, au sens de la procédure de vérification. Le but est de vérifier qu'un programme bien typé ne peut produire d'erreur de type à l'exécution. La formule souvent utilisée est *"well typed programs do not go wrong"*[Mil78, MO84]. La vérification statique évite le contrôle de type durant l'exécution. Le coût de cette efficacité et de cette sécurité est la perte de flexibilité dans la programmation. Puisqu'il n'existe pas de procédure de typage correcte et complète, des programmes bien typés risquent d'être rejetés.

La vérification dynamique n'impose aucune contrainte sur la programmation mais cette facilité a sa contrepartie dans la perte d'efficacité et de sécurité. Le contrôle de type doit être effectué lors de l'exécution. Pour une meilleure performance, des algorithmes d'inférence de types et de contrôle de types ("typage souple") ont été conçus pour ce genre de langages [AWL94, CF91].

Les concepts de base du typage de programmes PROLOG ont été proposés par Mishra [Mis84].

> Le type d'un prédicat décrit les termes pour lesquels le prédicat **pourrait** réussir. Pour tout terme qui n'est pas du type, le prédicat **ne peut** réussir.

La fonction de typage doit être naturellement extraite du langage. Le travail de Mishra est inspiré de celui de Milner pour le langage ML. Le système de typage dans ML s'extrait de la structure du langage et se justifie par sa sémantique.

Le typage est, dans [Mis84] et dans de nombreux travaux qui l'ont succédé ([HJ90b, JM79, MR85, YO88]), une description symbolique d'ensembles de termes. Mishra représente un type par un ensemble. Les opérations + pour l'union de deux types et # pour le produit cartésien de deux types finis. Pour le programme,

```
edge(a,b).
edge(b,c).
edge(c,d).
edge(d,b).
```

le type du prédicat **edge**, $T(\mathbf{edge})$, est représenté par l'expression $(a+b+c+d)\#(b+c+d)$.

Le formalisme de Mishra permet aussi de décrire des ensembles infinis. Par exemple, le prédicat **sub** réussit pour les paires d'entiers n_1, n_2 avec $n_1 = 1 + n_2$:

```
sub(succ(zero),zero).
sub(succ(succ(x)),succ(y)):- sub(succ(x),y)
```

Le type de **sub** est interprété par le plus petit couple d'ensembles de termes qui satisfait les équations :

$$Z_1 = \mathbf{succ(zero)} + \mathbf{succ}(Z_1); Z_2 = \mathbf{zero} + \mathbf{succ}(Z_2).$$

Mishra obtient Z_1 et Z_2 de la façon suivante. Aux variables **x** et **y** du programme sont associées des ensembles de termes X et Y. La première clause, **sub(succ(zero),zero)**, est traduite par l'algorithme en une contrainte ensembliste $T(\mathbf{sub}) \supseteq \mathbf{succ(zero)\#zero}$. La seconde est vue comme une contrainte récursive sur $T(\mathbf{sub})$. En effet, si $T(\mathbf{sub}) = T_1\#T_2$, alors

- $T_1 \subseteq \mathbf{succ}(\mathbf{succ}(X))$ pourvu que $T_1 \subseteq \mathbf{succ}(X)$ et
- $T_2 \subseteq \mathbf{succ}(Y)$ pourvu que $T_2 \subseteq Y$.

L'ensemble $T(\mathbf{sub})$ contient alors toutes les interprétations qui peuvent réussir pour **sub**. C'est l'orientation de la preuve de correction de cette inférence de types : les programmes mal typés ne peuvent réussir. (*"Ill-typed programs cannot succeed"*).

Les types générés par cette inférence sont essentiellement des langages réguliers. La plupart des opérations de manipulation de types vont alors se transposer dans des opérations sur des grammaires ou encore des automates d'arbres.

Les expressions ensemblistes, avec une utilisation restreinte de symboles de complémentaire[5], ont été implantées par Aiken et Murphy [AM91]. En général, les algorithmes de complémentation et d'inclusion demandent un temps de calcul exponentiel. A l'aide d'heuristiques, les auteurs ont obtenu des algorithmes pratiquables et performants sur des cas concrets.

L'apparition de procédures de décision pour la résolution de contraintes ensemblistes a suscité d'autres travaux. Par exemple le système de typage du langage FOL ([PS93]) articulé autour des mêmes techniques prend en charge les opérations de surcharge, de liaison tardive et de sous-typage.

[5]Les opérateurs de complément sont utilisés de façon à obtenir une interprétation monotone des expressions ensembliste. Syntaxiquement, le complémentaire ne porte que sur des expressions sans variable ensembliste inconnue. En d'autres termes, la complémentation a une portée limitée à des expressions paramétrées.

Aiken et Wimmers ont parallèlement à leur algorithme de résolution de contraintes ensemblistes, développé des outils pour l'inférence de types en programmation fonctionnelle [AW93]. Les types prennent leur valeur dans un domaine différent, substantiellement plus difficile puisque des fonctions sur les ensembles (fonctions de type parfois partiellement définies) sont permises. Néanmoins, des techniques analogues à celles mises en œuvre dans le cadre des contraintes ensemblistes sont utilisées.

Les contraintes ensemblistes peuvent définir des classes de langages non réguliers lorsqu'aucune restriction n'est faite sur l'emploi des opérateurs. Un travail voisin d'Uribe [Uri92], à l'aide d'une interprétation différente des contraintes, défini des types non réguliers. Les contraintes sont alors très proches des automates avec tests d'égalités de Bogaert et Tison [BT92].

Analyse Ensembliste

L'analyse de programmes ensembliste est proposée par Heintze et Jaffar [Hei92, HJ90b]. C'est un type d'interprétation abstraite ou pseudo-évaluation. Brièvement, l'interprétation abstraite suppose des valeurs qui approximent les valeurs manipulées par un programme et des opérations qui approximent les opérations réalisées par un programme.

Dans l'analyse ensembliste, la seule approximation est une abstraction de toutes les dépendances entre variables. En d'autres termes, dans cette approximation, la valeur que peut prendre une variable n'affecte pas les valeurs que les autres variables peuvent prendre.

Cette approximation est obtenue en considérant les variables du programme comme des ensembles. Si par exemple ([Hei92]) nous considérons le programme

$$x := cons(y, x),$$

pendant la compilation, la phase d'analyse marque des étapes (1 et 2), juste avant et juste après, cette instruction.

$$\overset{1}{\underset{2}{x}} := cons(y, x) \longrightarrow \mathcal{S} = \begin{cases} cons(Y^1, X^1) \subseteq X^2 \\ Y^1 \subseteq Y^2 \end{cases}$$

Des variables ensemblistes X^1, Y^1 et X^2, Y^2 sont introduites dans le but de capturer l'ensemble des valeurs possibles de x et de y à ces étapes. Ceci est modélisé par le système d'inclusions $\mathcal{S}$ ci-dessus, que nous appelons un *système de contraintes ensemblistes*. L'objet de l'analyse se divise alors en deux parties: d'une part, (i) trouver la spécification du programme en terme de système de contraintes ensemblistes et d'autre part (ii) résoudre le système.

Dans l'optique de l'analyse ensembliste, la résolution des contraintes est la recherche de la plus petite solution. C'est-à-dire, les ensembles de termes les plus

petits qui, substitués aux variables ensemblistes satisfont le système dans une interprétation canonique des opérateurs. Par les propriétés du système généré, si une solution existe alors cette plus petite solution existe et de plus, elle est régulière. Donc l'analyse ensembliste est *décidable* et *régulière*. La plus petite solution sera alors représentée par des outils classiques, ici une grammaire régulière de termes, c'est-à-dire un automate fini d'arbres.

Sur l'exemple précédent, nous remarquons que les dépendances entre les variables x et y sont effacées. A partir du système, on ne peut déduire la valeur de x sachant celle de y à un instant donné, et vice-versa. Heintze et Jaffar montrent deux choses importantes. Premièrement, ignorer les dépendances entre variables est la seule approximation faite dans cette analyse. Ce qui veut dire que la plus petite solution du système $\mathcal{S}$ est le meilleur résultat que l'on peut obtenir compte tenu de cette approximation. Deuxièmement, toute solution de $\mathcal{S}$ est une approximation réelle du comportement du programme, dans le sens où elle contient les "vraies" valeurs.

Heintze ([Hei92]) attend de son analyse qu'elle soit :

- **déclarative** : une définition simple et intuitive de l'approximation, indépendante de considérations algorithmiques,
- **précise** : une approximation qui ait du sens, qui apporte de l'information,
- **décidable** : il doit exister des algorithmes pour calculer cette approximation.

Nous illustrons le premier et le second point par des exemples d'analyse en programmation impérative, logique et fonctionnelle. La dernière condition quant à elle sera évoquée page 14.

Programmation impérative

Le langage impératif dans lequel nous présentons cet exemple manipule avec `car`, `cdr` des structures de données construites avec les opérateurs `cons`, `nil` et des symboles comme a, b, c, ... Les valeurs que nous voulons approximer ne contiennent que des symboles de construction.

$$
\begin{array}{l}
l := cons(a, cons(b, nil)); \\
x := c; \\
\text{While } (l \neq nil) \text{ do} \\
\quad x := car(l); \\
\quad l := cdr(l);
\end{array}
$$

Le programme ([Hei92]) ci dessus décompose la valeur $cons(a, cons(b, nil))$ jusqu'à l'obtention de la liste vide. Le système de contraintes associé à ce programme est généré de la façon suivante. D'abord des étapes dans le programme, vues comme des points de contrôle, sont désignées.

$l := cons(a, cons(b, nil))$;
1 $x := c$;
While $(l \neq nil)$ do
2 $x := car(l)$;
3 $l := cdr(l)$;
5 4

A chaque étape i, pour chaque variable v du programme on associe une variable ensembliste V^i dans le but de capturer l'ensemble des valeurs que prendra v au point i durant l'exécution. Pour capturer les valeurs de l au point 4, la contrainte naturelle est de dire qu'elles doivent contenir celles de l au point 3, mais préfixées de l'opérateur cdr. Soit $cdr(L^3) \subseteq L^4$. On aboutit par ce raisonnement au système

$$\begin{array}{rcl} cons(a, cons(b, nil)) & \subseteq & L^1 \\ L^1 \cap \overline{nil} & \subseteq & L^2 \\ L^4 \cap \overline{nil} & \subseteq & L^2 \\ L^2 & \subseteq & L^3 \\ cdr(L^3) & \subseteq & L^4 \\ L^1 \cap nil & \subseteq & L^5 \\ L^4 \cap nil & \subseteq & L^5 \end{array} \qquad \begin{array}{rcl} c & \subseteq & X^1 \\ X^1 & \subseteq & X^2 \\ X^4 & \subseteq & X^2 \\ car(L^2) & \subseteq & X^3 \\ X^3 & \subseteq & X^4 \\ X^1 & \subseteq & X^5 \\ X^4 & \subseteq & X^5 \end{array}$$

Nous avons donné la sémantique d'un tel système. Précisons toutefois que les opérations car et cdr sont traduites par $cons^{-1}_{(1)}$ et $cons^{-1}_{(2)}$, puisqu'elles correspondent respectivement aux projections sur la première et la seconde composante du symbole $cons$. Par construction, toute solution à ce système va contenir l'ensemble des valeurs rencontrées lors de l'exécution. Il est raisonnable et naturel de déduire que la plus petite solution sera l'information la plus précise. Soit,

$$\begin{array}{rcl} L^1 & = & \{cons(a, cons(b, nil))\} \\ L^2 & = & \{cons(a, cons(b, nil)), cons(b, nil)\} \\ L^3 & = & \{cons(a, cons(b, nil)), cons(b, nil)\} \\ L^4 & = & \{cons(b, nil), nil\} \\ L^5 & = & \{nil\} \end{array} \qquad \begin{array}{rcl} X^1 & = & \{c\} \\ X^2 & = & \{a, b, c\} \\ X^3 & = & \{a, b\} \\ X^4 & = & \{a, b\} \\ X^5 & = & \{a, b, c\} \end{array}$$

Programmation Logique

Dans l'analyse ensembliste automatique de programmes logiques, des points de contrôle sont ajoutés avant et après chaque appel de prédicat. Ils apparaissent donc en partie droite des clause $P : -Q$. Pour une illustration plus simple, dans ce programme élémentaire, nous avons supprimé les points donnant lieu à une information non essentielle.

	$Ret_p = p(X)$
$p(x) : -q(x), r(x).$	$Ret_q = q(a) \cup q(f(Y))$
$q(a).$	$Ret_r = r(f(Z))$
$q(f(y)) : -q(y).$	$X = q^{-1}(Ret_q) \cap r^{-1}(Ret_r)$
$r(f(z)).$	$Y = q^{-1}(Ret_q)$
	$Z = \top$

Dans cet exemple, la construction correspond à une approximation du déroulement du programme dans une stratégie bottom-up. Les variables ensemblistes X et Y doivent capturer l'ensemble des termes clos que peuvent atteindre x et y. Notons que les variables Ret_p, Ret_q, Ret_r sont ajoutées et doivent contenir les atomes clos de l'ensemble des succès qui correspondent aux prédicats p, q, et r (en quelque sorte les valeurs "retournées" par p, q et r). La quatrième contrainte signifie que X contient les retours de p *et* de q.

La plus petite solution de ce système donne les ensembles :

$$
\begin{aligned}
Ret_p &= \{p(f(a)), p(f(f(a))), \ldots\} = \{p(f^n(a)) \mid n > 0\} \\
Ret_q &= \{q(f^n(a)) \mid n \geq 0\} \\
Ret_r &= \{r(f(t)) \mid t \text{ est un terme clos}\} \\
X &= \{f^n(a) \mid n > 0\} \\
Y &= \{f^n(a) \mid n \geq 0\} \\
Z &= \{t \mid t \text{ est un terme clos}\}
\end{aligned}
$$

Pour ce programme, le résultat de l'analyse donne exactement le résultat après exécution. Mais puisque dans l'analyse ensembliste toutes les dépendances entre variables sont effacées, cette propriété remarquable ici sera le plus souvent perdue. Par exemple, pour :

$$
\begin{aligned}
&p(x, y) : -q(x, y). \\
&q(a, a). \\
&q(b, b).
\end{aligned}
$$

l'approximation donne $Ret_p = \{p(a, a), p(a, b), p(b, a), p(b, b)\}$.

Programmation Fonctionnelle

Le cas de programmes fonctionnels suppose l'introduction de nouveaux opérateurs et symboles dont l'interprétation n'est pas tout à fait directe.

Pour mener à bien cette approximation, il est nécessaire de considérer :

- des symboles constructeurs de données.
- des symboles qui correspondent aux fonctions du programme,
- des variables qui capturent le domaine et le co-domaine des fonctions du programme, $Dom(x)$, $Ran(x)$, ...
- un opérateur ensembliste nouveau, *apply* qui modélise l'application d'une fonction à ses arguments. Elle exprime le fait que les arguments doivent faire partie du domaine et le résultat au co-domaine.

Soit le programme suivant :

```
Let fun id x = x
in
   id c
end
```

Les contraintes sont construites de la façon suivante. Comme toujours, l'ensemble X contient toutes les valeurs que x peut prendre à l'exécution. A la définition de la fonction id, correspondent deux contraintes. La première exprime le fait que X doit contenir toutes les valeurs possibles avec lesquelles la fonction peut être appelée : $dom(id)$. La seconde, que X est inclus dans l'ensemble des valeurs retournées par la fonction : $ran(id)$. Finalement, la dernière indique que id est appliquée à c.

$$\begin{aligned} dom(id) &\subseteq X \\ X &\subseteq ran(id) \\ apply(id, c) &\subseteq Res \end{aligned}$$

La variable Res capture le résultat du programme. Compte tenu de l'interprétation des opérateurs (voir [Hei92]), la plus petite solution du système donne dans cet exemple le comportement exact du programme.

Les algorithmes d'analyse ensembliste ont depuis fait l'objet d'optimisations en temps et précision. Les implantations ont montré que malgré une complexité théorique élevée, des techniques pouvaient dans des cas réels, rendre la résolution de contraintes praticable.

Techniquement, pour construire une procédure de typage ou d'analyse automatique où les types sont vus comme des ensembles de termes, il faut avant tout représenter ces ensembles par une expression. Ensuite il faut réaliser des opérations sur les ensembles de termes, donc sur ces expressions, par exemple l'union,

ou encore tester l'inclusion. C'est donc essentiellement, une résolution de contraintes ensemblistes. Avant de disposer d'une procédure suffisamment générale, beaucoup d'auteurs ont utilisé des contraintes de classes syntaxiques restreintes au pouvoir d'expression souvent limité. Nous voyons maintenant les différentes classes de contraintes et leurs algorithmes de résolution.

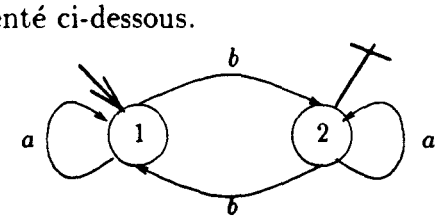
Méthodes de Résolution

Il existe aujourd'hui de nombreuses méthodes de résolution de contraintes ensemblistes. Nous présentons les plus connues ou les plus originales en tentant de souligner les liens avec l'algorithme proposé dans ce mémoire. Les méthodes se rapprochant fortement de la notre seront étudiées plus en détail dans la section 4.5.5.

Equations linéaires à gauche

Dans le cadre du monoïde libre, la volonté de caractériser de façon équationnelle les automates, entraîne la définition et la résolution de systèmes de contraintes sur des ensembles de mots. En effet, dans la preuve de l'inclusion des langages reconnaissables dans les langages rationnels, Kleene associe à un automate de mots sur l'alphabet $\Sigma = \{a_1, \ldots, a_k\}$, un système d'équations linéaires gauche, à coefficients dans Σ. Si les états de l'automate sont $q_1, \ldots, q_p$, les variables de ce système capturent les langages $L_{q_i} = \{w \in \Sigma^* \mid w \overset{*}{\mapsto} q_i\}$, c'est-à-dire l'ensemble des mots qui amènent l'automate dans un état donné. (voir par exemple [HU79]).

Exemple 0.1 Soit A l'automate d'états $\{q_1, q_2\}$ sur l'alphabet $\Sigma = \{a, b\}$ et dont le graphe est représenté ci-dessous.



Le système où ε représente le mot vide.

$$\begin{cases} X_1 = a.X_1 \cup \varepsilon \cup b.X_2 \\ X_2 = a.X_2 \cup b.X_1 \end{cases}$$

admet pour solution l'interprétation $\mathcal{I}(X_1) = L_{q_1} = (a \cup ba^*b)^*$ et $\mathcal{I}(X_2) = L_{q_2} = (a \cup ba^*b)^*ba^*$.

La résolution de ces contraintes est obtenue via la résolution, due à Arden [Ard61], de

$$X = AX \cup B \tag{3}$$

où A et B sont des langages de mots. L'équation (3) a pour unique solution en X, lorsque $\varepsilon \notin A$, le langage régulier A^*B.

En résumé, les systèmes d'équations de la forme

$$\begin{aligned} X_1 &= \bigcup_{a_i \in \Sigma} a_i X_{i_1} \cup \delta \\ X_2 &= \bigcup_{a_i \in \Sigma} a_i X_{i_2} \cup \delta \\ &\vdots \\ X_n &= \bigcup_{a_i \in \Sigma} a_i X_{i_n} \cup \delta \end{aligned}$$

où δ est soit le mot vide ε ou l'ensemble vide $\emptyset$, correspondent à des automates de mots sur l'alphabet Σ. De tels systèmes admettent une seule solution, qui est une solution régulière, c'est-à-dire que chacun des X_i est interprété par un langage régulier de mots.

Dans [BL80], Brzozowski et Leiss étendent ce résultat en permettant l'emploi d'opérateurs ensemblistes, union, intersection et complémentaire, entre les variables. Les systèmes d'équations sont maintenant de la forme

$$X_i = \bigcup_{j=1}^{k} a_j E_{i,j} \cup \delta_i \quad i = 1, \ldots, n \tag{4}$$

Chaque $E_{i,j}$ est une expression ensembliste construite sur l'ensemble des variables $X_1, \ldots, X_n$, les opérateurs $\cup$, $\cap$ et $\sim$ et $\delta_i \in \{\{\varepsilon\}, \emptyset\}$.

De tels systèmes sont en fait les caractérisations équationnelles des automates booléens (Boolean Automata) de Brzozowski et Leiss [BL80] ou encore des automates parallèles (Parallel Finite Automata) de Kozen [Koz76]. La classe des langages reconnus par ces deux types d'automates est exactement la classe des langages réguliers.

Les systèmes d'équations de la forme (4) ont une unique solution qui est régulière.

Comme nous le verrons dans la section 4.2, les contraintes sur les ensembles de mots sont résolues en utilisant des techniques habituelles autour d'automates d'arbres infinis de Rabin. Dans ce cas, l'utilisation non restreinte d'opérateurs ensemblistes entraîne évidemment la perte des propriétés remarquables, comme l'unicité de la solution. Par exemple, $X =\sim X$ n'a pas de solution, et $X = X$ a une infinité de solutions (régulières ou non).

Les Arbres

Le transfert, au cas des arbres, des résultats obtenus à l'aide d'automates de mots finis classiques, est direct (voir section 1.2).

Dans [Rey69], Reynolds utilise des contraintes sur des ensembles de termes dans le cadre de la programmation fonctionnelle. Il donne une méthode pour calculer une description ensembliste des résultats d'une fonction (en pur LISP). Les systèmes traités (*recursive set definition*) sont des ensembles d'équations de la forme

$$X_i = F_i(X_1, \ldots, X_n) \tag{5}$$

avec $F_i(X_1, \ldots, X_n)$ une expression faisant intervenir les variables $X_1, \ldots, X_n$, des constantes et des opérateurs monotones .

Un opérateur *op* k-aire est monotone si son interprétation $\boxed{\text{op}}$ vérifie la propriété suivante :

$$L_1 \subseteq L'_1, \ldots, L_k \subseteq L'_k \Rightarrow \boxed{\text{op}}(L_1, \ldots, L_k) \subseteq \boxed{\text{op}}(L'_1, \ldots, L'_k)$$

Grâce à cette propriété de monotonie des opérateurs, les systèmes (5) ont toujours une solution et une plus petite solution.

Reynolds veut donner une définition algébrique de l'ensemble des valeurs d'une fonction LISP. C'est donc une démarche d'inférence de types et les opérateurs utilisés correspondent aux primitives LISP standard comme **cons**, **car** et **cdr** avec l'opérateur ensembliste d'union. On vérifie bien que ces opérateurs sont monotones.

La méthode de Reynolds n'est pas une résolution mais plutôt une simplification de contraintes ensemblistes. En effet, l'existence d'une (plus petite) solution est garantie puisque le système obtenu après l'exécution du programme d'inférence de types est déjà de la forme (5).

L'objet de la simplification est de rendre le système de contraintes plus lisible en éliminant les opérateurs "analytiques". Il s'agit des opérateurs **car** et **cdr** généralisés aux ensembles de termes. Ces opérateurs sont en fait les opérateurs de projection $\mathbf{cons}^{-1}_{(1)}$ et $\mathbf{cons}^{-1}_{(2)}$.

Dans [Mis84], le but est aussi de produire une procédure de typage basée sur l'utilisation de contraintes ensemblistes mais cette fois dédiée à la programmation logique en PROLOG.

Le formalisme utilisé diffère dans sa présentation de celui utilisé dans ce mémoire. Les expressions ensemblistes sont appelées *Regular Tree*[6]. Les expressions utilisent l'union, des symboles de fonctions et des constantes, et un opérateur ! qui dans $X!exp(X)$ désigne la plus petite solution de l'équation

$$X = exp(X)$$

Là encore, les opérateurs utilisés étant monotones, ! est toujours bien défini et les expressions gardent leur sens. L'utilisation de ! permet de faire intervenir des langages (infinis) de termes dans les expressions.

[6]Plus justement appelées par la suite *Regular tree expressions* par Aiken et Murphy.

Le nœud du problème est de résoudre les questions d'inclusion entre deux expressions, donc de résoudre des contraintes ensemblistes. La procédure doit répondre par un échec si le système n'est pas solvable et sinon doit donner la plus grande solution. Les expressions de part et d'autre du symbole d'inclusion utilisent des variables différentes et la stratégie utilisée est de faire des suppositions sur les valeurs que peuvent prendre celles situées à gauche du symbole d'inclusion.

Contraintes Définies

L'article de Heintze et Jaffar [HJ90a] constitue une étape importante dans le domaine de la résolution de contraintes ensemblistes. Leur méthode traite la classe des contraintes définies. Heintze et Jaffar justifient l'emploi de ce nom par l'analogie avec les clauses définies :

> Un système de contraintes ensemblistes définies a une plus petite solution si il admet des solutions.

L'algorithme présenté, avec en entrée une conjonction de contraintes définies,

1. détermine si cette conjonction est satisfiable,
2. si tel est le cas, donne en sortie une grammaire de termes régulière (i.e. un automate d'arbres), qui représente la plus petite solution.

Syntaxiquement, une contrainte définie est de la forme,

$$exp \subseteq a.$$

a est un terme variable, c'est-à-dire que dans a n'apparaissent que des symboles de fonctions et des variables. exp est une expression sans symbole de complémentaire.

Dans une étape préliminaire, un système de contraintes définies C est aplati en utilisant des opérations de projection dans le but d'obtenir un système C_p. Les expressions de C_p s'écrivent

$$exp \subseteq t$$

ou

$$exp \subseteq X$$

avec t un terme construit uniquement avec les symboles de fonction, et X une variable.

A ce point, on peut considérer deux systèmes dont l'un, le système *actif* C_a, rassemble les contraintes de la forme $exp \subseteq X$, possède toujours une (plus petite) solution.

Les phases suivantes consistent en la simplification du système C_a, afin d'obtenir des inclusions directement transformables en des règles d'une grammaire de termes G.

Finalement, il est prouvé que si la plus petite solution de C_a, exprimée à l'aide de la grammaire G, est une solution pour le système complet C, alors, c'est la plus petite solution de C. Dans le cas contraire, C est insatisfiable.

Nous devons noter que les transformations, notamment la première qui aplatit le système C, ne préservent pas l'ensemble des solutions, mais seulement la plus petite.

Avec le complémentaire

Aiken et Wimmers ([AW92]), traitent pour la première fois, la classe PSC, c'est-à-dire la classe des contraintes positives (sans symbole $\not\subseteq$), et sans symbole de projection.

Pour la première fois, l'opérateur non monotone de complément est autorisé. De ce fait, PSC contient des systèmes non satisfiables

$$X = \sim X$$

et des systèmes sans plus petite solution

$$\begin{aligned} X \cup Y &= a \\ X \cap Y &= \bot \end{aligned}$$

qui admet deux solutions incomparables $\mathcal{I}$ et $\mathcal{I}'$ avec $\mathcal{I}(X) = a$, $\mathcal{I}(Y) = \emptyset$ et $\mathcal{I}'(X) = \emptyset$, $\mathcal{I}'(Y) = \{a\}$.

La technique utilisée consiste à transformer le système initial C, en préservant l'ensemble des solutions, (les transformations sont correctes et complètes), jusqu'à l'obtention d'un système en forme résolue ou jusqu'à l'obtention d'une incompatibilité montrant que C est insatisfiable.

Un système en forme résolue admet toujours des solutions. L'idée est de parvenir à un système sous la forme d'une conjonction de contraintes

$$X_i = exp_i$$

où les X_i sont les inconnues et les exp_i vérifient :

1. Les négations n'apparaissent que devant des variables libres, en quelque sorte, des paramètres différents syntaxiquement des X_i,

2. Les inconnues, les X_i, apparaissent sous un symbole de fonction.

De ce fait, toute interprétation des variables libres se dérive en une et une seule interprétation de toutes les variables. En résumé, quand le système est sous forme résolue, une solution est obtenue en donnant une valeur aux paramètres.

Le centre de l'algorithme est le résultat de la constatation que si $X \subseteq exp$ et $exp' \subseteq X$ alors, toute interprétation $\mathcal{I}$ pour X se situe, vis à vis de $\subseteq$, entre

$\mathcal{I}(exp)$ et $\mathcal{I}(exp')$. Ceci entraîne l'introduction d'un paramètre α, ou variable libre, exprimant cet écart : $X = exp' \cup (\alpha \cap exp)$.

Cette méthode de résolution donne aussi un encadrement de la complexité de la résolution des contraintes positives. Le problème est montré dans NEXPTIME et EXPTIME-dur.

Nous avons montré dans [GTT93a] le même résultat, i.e. la satisfiabilité des contraintes ensemblistes avec complémentaire, en utilisant des automates d'ensembles d'arbres.

Aspects logiques

Bachmair, Ganzinger et Waldmann dans [BGW92] étudient les relations entre les contraintes ensemblistes et la logique. Ce travail permet de donner une nouvelle procédure de décision pour la classe PSC des contraintes positives et la complexité du problème, à savoir la NEXPTIME complétude.

La classe monadique est la classe des formules logiques du premier ordre sans symbole de fonction, avec des prédicats unaires et une quantification arbitraire. De nombreux travaux, depuis ceux de Lövenheim en 1915 [Low15] et d'Ackermann [Ack54] ont été consacrés à l'étude de cette classe. Le papier de Bachmair et al montre l'équivalence, via quelques transformations, entre la classe monadique et la classe des contraintes ensemblistes positives avec projections pour les symboles de fonction unaires.

Les formules skolemisées de la classe monadique peuvent être caractérisées syntaxiquement. Les contraintes sont codées à l'aide de ces formules. Un ensemble de termes qui sera l'interprétation d'une expression ensembliste E correspondra à l'interprétation d'un prédicat P_E. Informellement, $P_E(t)$ signifie, t *appartient à* E.

Il reste ensuite à coder les interprétations des opérateurs. Par exemple $\top$ s'interprète comme l'ensemble de tous les termes, donc pour tout t, $P_\top(t)$ est vrai, ou encore $P_\top(x) \Leftrightarrow true$. Ce qui donne plus formellement :

$$\begin{aligned}
P_\top(x) &\Leftrightarrow 1 \\
P_\bot(x) &\Leftrightarrow 0 \\
P_{E\cup F}(x) &\Leftrightarrow P_E(x) \vee P_F(x) \\
P_{E\cap F}(x) &\Leftrightarrow P_E(t) \wedge P_F(x) \\
P_{\sim E}(x) &\Leftrightarrow \neg P_E(x) \\
P_{b(E_1,\ldots,E_p)}(b(x_1,\ldots,x_p)) &\Leftrightarrow P_{E_1}(x_1) \wedge \ldots \wedge P_{E_p}(x_p) \\
P_{b(E_1,\ldots,E_p)}(c(x_1,\ldots,x_q)) &\Leftrightarrow 0
\end{aligned}$$

Pour tous les symboles de fonction b et c, $c \neq b$.

Il suffit maintenant d'ajouter à cette définition la traduction des contraintes selon le même codage. Nous tirons l'exemple ci-dessous de [BGW92, AW92]:

$$C \equiv (X \subseteq Y) \wedge (b(Y) \subseteq\sim Y) \wedge (b(\sim Y) \subseteq Y).$$

donne:

$$\begin{aligned} P_X(x) &\Rightarrow P_Y(x) \\ P_{b(Y)}(x) &\Rightarrow P_{\sim Y}(x) \\ P_{b(\sim Y)}(x) &\Rightarrow P_Y(x) \end{aligned}$$

La conjonction des deux systèmes de formules logiques donne la traduction, dans la classe des skolémisés des formules de la classe monadique, du système C.

Ajoutons enfin que ce codage peut être étendu au cas où des symboles de projection apparaissent avec une polarité négative: à gauche du symbole d'inclusion sous un nombre pair de symboles de complémentarité ($\sim$) ou encore à droite du symbole d'inclusion sous un nombre impair de symboles $\sim$.

Le codage étant réalisé, les nombreux résultats autour de la classe monadique peuvent alors s'appliquer et en particulier, Bachmair et al déduisent une autre preuve de la décidabilité de la satisfiabilité des contraintes positives et que:

- La complexité de ce problème est NEXPTIME-complet,
- La construction s'étend au cas où des symboles de projection apparaissent avec une polarité négative, et à des opérateurs de diagonalisation.

Contraintes positives et négatives

Dans ce mémoire, nous proposons un algorithme de résolution de systèmes de contraintes ensemblistes positives (i.e. $exp \subseteq exp'$) et négatives (i.e. $exp \not\subseteq exp'$). Cette méthode implique la définition et l'utilisation d'un nouveau type d'automates, les automates d'ensembles d'arbres.

Aiken, Kozen et Wimmers, dans [AKW93], ont récemment proposé un algorithme de décision basé sur la réduction à un problème d'accessibilité dans une certaine classe d'hypergraphes.

En fait, ce papier a pour préliminaires un autre article de Aiken, Kozen, Vardi et Wimmers, [AKVW93], qui fait "l'inventaire" des complexités de la résolution de contraintes suivant leurs caractéristiques syntaxiques. Par les algorithmes mis en œuvre, ces deux articles sont très proches de nos travaux [GTT93a, GTT93b].

Avant tout, un système C de contraintes sur un alphabet Σ et sur les inconnues $X_1, \ldots, X_n$ est transformé en une forme normale qui met en évidence l'interprétation des symboles de fonction et aplatit les expressions.

Par cette opération, un certain nombre de variables auxiliaires sont introduites. On peut alors distinguer trois sous-systèmes dans cette forme normale. De façon informelle, nous avons :

- un système S_1 qui exprime les relations entre les variables auxiliaires et les symboles de fonction.
- un système S_2 constitué d'une seule contrainte $B = \top$ où dans B n'interviennent que des inconnues (variables X_i) et des opérateurs ensemblistes $\cup$, $\cap$, $\sim$. On peut dire que $B = \top$ est l'expression des contraintes sur les inconnues.
- un système S_3 de contraintes de la forme $C_b = \top$ pour tout b de Σ. Dans une contrainte $C_b = \top$ n'interviennent que des variables auxiliaires en relation avec le symbole b et les opérateurs $\cup$, $\cap$, $\sim$. On peut dire que $C_b = \top$ est l'expression des contraintes sur b.

Mis sous cette forme, un système C est la spécification d'un hypergraphe H de nœuds dans U et de relations dans $\{E_b \mid b \in \Sigma\}$. Une relation dans E_b est d'arité $arité(b) + 1$.

$$H = (U, E_b \mid b \in \Sigma).$$

Si nous interprétons chaque contrainte comme une formule propositionnelle[7] alors U est l'ensemble des valuations booléennes des variables $X_1, \ldots, X_n$, les inconnues, qui satisfont la formule $B = 1$.

De même, chaque relation $(u_0, \ldots, u_p) \in E_b$ entre les nœuds de H signifie que la formule $C_b = 1$ est vraie sous les assignations des variables auxiliaires induites par les valuations $u_0, \ldots, u_p$.

Aiken et al montrent que le système C est satisfiable si et seulement si il existe un sous-graphe clos induit par H. Un sous-graphe $H' = (U', E'_b \mid b \in \Sigma)$ induit par H est clos si pour tout symbole b p-aire, tout p-uplet de nœuds de U' est en relation par E'_b.

La table 0.1 résume les résultats de complexité trouvés par Aiken et al dans [AKVW93]. La colonne de droite de ce tableau est déduite du codage donné précédemment et des propriétés des hypergraphes.

Le cas, beaucoup plus "résistant" de la satisfiabilité de contraintes positives et négatives est obtenu via un codage similaire. Au système en forme normale, donc aux trois sous-systèmes S_1, S_2, et S_3, est ajouté un ensemble $\mathcal{D}$ de contraintes de la forme

$$D \neq \bot,$$

où D est construit avec les inconnues $X_1, \ldots, X_n$ et les opérateurs ensemblistes $\cap$, $\cup$, $\sim$.

[7] interprétation canonique des variables, des opérateurs ensemblistes $\sim$, $\cap$, $\cup$, $\bot$ et $\top$ en variables propostionnelles, $\neg$, $\wedge$, $\vee$, 0, et 1.

Nombre d'éléments dans Σ			Complexité du problème de la satisfiabilité
d'arité 0	d'arité 1	d'arité ≥ 2	
0	0 ou plus	0 ou plus	trivial
1 ou plus	0	0	NP-complet
1 ou plus	1	0	PSPACE-complet
1 ou plus	2 ou plus	0	EXPTIME-complet
1 ou plus	0 ou plus	1 ou plus	NEXPTIME-complet

TAB. 0.1 - : Complexité des contraintes

Le codage dans un hypergraphe $H = (U, E_b \mid b \in \Sigma)$ est identique mais la satisfiabilité correspond alors à une propriété plus complexe.

A son tour, ce problème est résolu en le ramenant à l'existence d'une solution au problème d'accessibilité non linéaire (NRP). Nous verrons plus en détail les relations entre les travaux d'Aiken et al et les notres dans la section 4.5.5.

Finalement, Stefansson montre, dans [Ste93], en affinant la résolution de NRP (qui est en fait un système d'inéquations diophantiennes), que la satisfiabilité des contraintes positives et négatives est NEXPTIME-complet.

Automates d'Ensembles d'Arbres

Les *Automates d'Ensembles d'Arbres* ou *TSA* sont à la fois proches des contraintes, proches des automates d'arbres finis et proches des automates d'arbres infinis. Les propriétés que vérifient les automates d'ensembles d'arbres sont immédiatement transposées dans le cadre des contraintes ensemblistes. Par exemple, nous montrons directement que pour les contraintes de la classe *PNSC* :

- la satisfiabilité est décidable,
- il existe une solution régulière pour tout système satisfiable,
- l'appartenance d'un "langage régulier" à l'ensemble des solutions est décidable.

Un Automate d'Ensemble d'Arbres reconnaît un ensemble de n-uplets de langages d'arbres écrits avec les symboles d'un alphabet donné. Appelons Σ un alphabet composé d'un seule constante a et d'une lettre binaire b et soit $\mathcal{A}$ un *TSA*.

On peut voir $\mathcal{A}$ comme une machine qui examine, pendant un de ses *calculs*, suivant un ensemble de *règles* $\mathcal{S}$, tous les arbres construits avec les symboles de Σ. Les arbres sont examinés dans l'ordre "sous-arbre", c'est-à-dire que a est

examiné avant $b(a,a)$ alors que $b(a,b(a,a))$ et $b(b(a,a),a)$ peuvent être examinés indépendamment l'un après l'autre ou en même temps.

Un calcul doit associer à chaque arbre, un *état* de la machine, de façon unique. L'état d'un arbre est déterminé par les états de ses sous-arbres directs et par les règles du *TSA*. Par exemple, l'état de $b(a,b(a,a))$ dépend de l'état de a et de l'état de $b(a,a)$ et des règles de $\mathcal{A}$. Un calcul est donc une *application* de l'ensemble de tous les arbres, noté T_Σ dans l'ensemble des états compatible avec les règles $\mathcal{S}$.

Enfin, un calcul r accepte un n-uplet de langages d'arbres. Ce n-uplet est composé en contrôlant l'appartenance ou la non appartenance des états atteints pendant le calcul r aux éléments d'un n-uplet d'ensembles d'états *finaux* $(F_1,\ldots,F_n)$. Si l'état associé à un arbre t, noté $r(t)$, fait partie de F_i, alors t sera dans la i^{eme} composante du n-uplet de langages accepté.

Détaillons un *TSA* et son fonctionnement à l'aide d'un exemple. Soit $\mathcal{A}$ un *TSA* sur l'alphabet $\Sigma = \{a,b\}$, un ensemble d'états $\mathcal{Q} = \{\mathbf{q},\mathbf{q}'\}$, un couple d'ensembles d'états finaux $(F_1,F_2) = (\{\mathbf{q},\mathbf{q}'\},\{\mathbf{q}'\})$, un ensemble de règles :

$$\begin{aligned} a &\rightarrow \mathbf{q} \\ b(\mathbf{q},\mathbf{q}) &\rightarrow \mathbf{q} \\ b(\mathbf{q},\mathbf{q}) &\rightarrow \mathbf{q}' \\ b(\mathbf{q}',\mathbf{q}) &\rightarrow \mathbf{q} \\ b(\mathbf{q},\mathbf{q}') &\rightarrow \mathbf{q} \\ b(\mathbf{q}',\mathbf{q}') &\rightarrow \mathbf{q} \end{aligned}$$

Un calcul associe à chaque arbre un état unique de $\mathcal{Q}$ selon les règles ci-dessus. Définissons un calcul appelé r_0. Pour l'arbre a, la seule règle applicable est $a \rightarrow \mathbf{q}$:

$$r_0(a) = \mathbf{q} \ .$$

Pour calculer l'état associé à l'arbre $b(a,a)$ dans le calcul r_0, il faut utiliser une règle de membre gauche $b(r_0(a),r_0(a))$, c'est-à-dire, de membre gauche $b(\mathbf{q},\mathbf{q})$. Choisissons $b(\mathbf{q},\mathbf{q}) \rightarrow \mathbf{q}$, alors :

$$r_0(b(a,a)) = \mathbf{q} \ .$$

Nous pouvons répéter ce choix pour tous les autres arbres. Sachant que pour un arbre $b(t,t')$ la règle à utiliser est de membre gauche $b(r_0(t),r_0(t'))$, nous pouvons toujours choisir la même règle $b(\mathbf{q},\mathbf{q}) \rightarrow \mathbf{q}$. Nous avons alors :

$$\forall t \in T_\Sigma \quad r_0(t) = \mathbf{q} \ .$$

Le couple de langages accepté par le calcul r_0 est alors (L_1^0, L_2^0) où $L_1^0 = T_\Sigma$, puisque $\mathbf{q} \in F_1$, et L_2^0 est l'ensemble vide, puisque $\mathbf{q} \notin F_2$.

Un autre calcul, r_1, consiste par exemple à faire les choix suivants :

$$\begin{array}{rcl}
r_1(a) = \mathbf{q} & \text{avec} & a \to \mathbf{q} \\
r_1(b(a,a)) = \mathbf{q}' & \text{avec} & b(\mathbf{q},\mathbf{q}) \to \mathbf{q}' \\
r_1(b(b(a,a),a)) = \mathbf{q} & \text{avec} & b(\mathbf{q}',\mathbf{q}) \to \mathbf{q} \\
r_1(b(a,b(a,a))) = \mathbf{q} & \text{avec} & b(\mathbf{q},\mathbf{q}') \to \mathbf{q} \\
r_1(b(b(a,a),b(a,a))) = \mathbf{q} & \text{avec} & b(\mathbf{q}',\mathbf{q}') \to \mathbf{q} \\
r_1(b(b(a,a),a),a)) = \mathbf{q}' & \text{avec} & b(\mathbf{q},\mathbf{q}) \to \mathbf{q}' \\
\vdots & &
\end{array}$$

La figure 0.1 représente les évolutions de $\mathcal{A}$ pendant le calcul r_1. Chaque noeud du graphe représente un arbre (indiqué dans la partie droite). Les arcs vont d'un arbre vers ses sous-arbres directs et la machine progresse du bas vers le haut. Les étiquettes $\mathbf{q}$ et $\mathbf{q}'$ sont déposées dans l'ordre "sous-arbre".

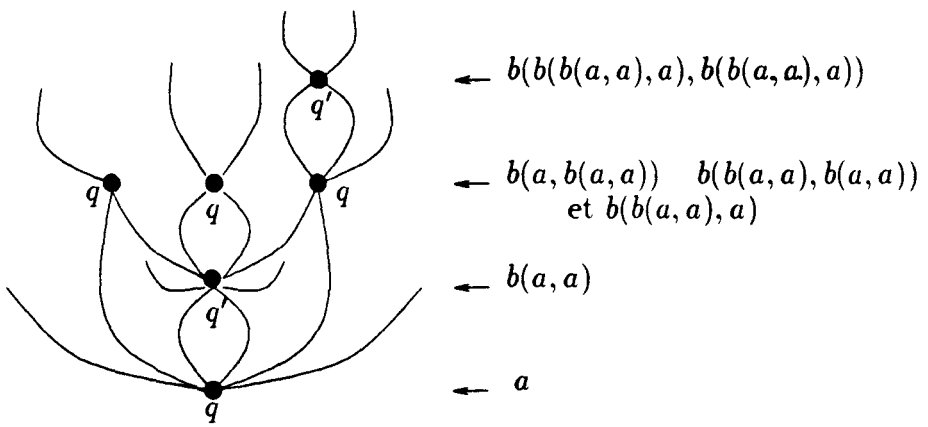


FIG. 0.1 - : Le calcul r_1

En fait, la règle $b(\mathbf{q},\mathbf{q}) \to \mathbf{q}'$ est choisie à chaque fois que c'est possible. En conséquence, dans le calcul r_1, un arbre $b(t,t')$ a pour image $\mathbf{q}'$ si et seulement si t et t' ont pour image $\mathbf{q}$. Le calcul r_1 accepte le couple (L_1^1, L_2^1) avec :

- L_1^1 est T_Σ.
- L_2^1 est l'ensemble de tous les arbres de la forme $b(t,t')$ avec $t \notin L_2^1$ et $t' \notin L_2^1$.

Il est facile ici de donner une caractérisation des calculs qu'il est possible de réaliser dans ce *TSA* et des couples de langages qu'ils acceptent. Par la forme des règles de $\mathcal{S}$, (L,L') est dans l'ensemble $\mathcal{L}(\mathcal{A})$ des couples de langages d'arbres reconnus par $\mathcal{A}$ si et seulement si :

- $L = T_\Sigma$,

- L' vérifie : $b(t,t') \in L' \Rightarrow (t \notin L' \wedge t' \notin L')$.

En fait, tous les calculs qu'il est possible d'effectuer dans un *TSA* ne sont pas des calculs *réussis*. Une condition d'acceptation complète la définition d'un automate d'ensembles d'arbres. Nous ne considérons dans le langage reconnu par un *TSA*, que les n-uplets acceptés par les calculs réussis.

Les conditions sont exprimées par une collection (finie) Ω d'ensembles d'états. Elles sont satisfaites par un calcul r si l'ensemble des états associés aux termes de T_Σ par r est dans Ω.

Par exemple, dans l'automate $\mathcal{A}$, si nous ajoutons la condition $\Omega = \{\{\mathbf{q}, \mathbf{q}'\}, \{\mathbf{q}'\}\}$, le calcul r_1 sera réussi alors que r_0 ne le sera pas. Avec cette condition, (L, L') est dans $\mathcal{L}(\mathcal{A})$ si :

- $L = T_\Sigma$,

- L' vérifie : $b(t,t') \in L' \Rightarrow (t \notin L' \wedge t' \notin L')$.

- L' est non vide.

Les *TSA* sont capables de représenter par les langages qu'ils reconnaissent, l'ensemble des solutions de tout système de contraintes ensemblistes dans *PNSC*.

L'automate d'ensembles d'arbres $\mathcal{A}$ reconnaît les solutions du système de contraintes :

$$\begin{array}{lll} \top & \subseteq & X \\ Y & \subseteq & b(\sim Y, \sim Y) \\ Y & \not\subseteq & \bot \end{array}$$

La construction de l'automate à partir du système est simple. L'idée est de conserver dans l'état associé à un arbre t les informations de la forme $t \in e_i$ pour toutes les sous-expressions e_i qui apparaissent dans le système. Si *nexp* est le nombre de ces sous-expressions alors il existe au plus 2^{nexp} états dans l'automate. Les transitions donneront alors les règles d'intégrité et de propagation de ces informations.

- *intégrité* : pour la contrainte $Y \subseteq b(\sim Y, \sim Y)$, aucun état ne peut signifier *t n'est pas dans* $b(\sim Y, \sim Y)$ et *t est dans* Y.

- *propagation* : Par exemple, dans l'expression $b(\sim Y, \sim Y)$, si il existe deux états $\mathbf{q}$ et $\mathbf{q}'$ signifiant $t \notin Y$ et $t' \notin Y$ alors nécessairement dans une règle $b(\mathbf{q}, \mathbf{q}') \rightarrow \mathbf{q}''$, l'état $\mathbf{q}''$ doit contenir l'information $b(t,t') \in b(\sim Y, \sim Y)$.

Enfin, les conditions d'acceptation contrôlent qu'un calcul "satisfait" toutes les contraintes négatives. Par exemple, pour vérifier que Y n'est pas vide, il suffit de contrôler qu'au moins un état qui signifie $t \in Y$ est atteint. Il faut donc rassembler dans Ω, tous les sous-ensembles d'états qui contiennent au moins un tel état.

Ceci constitue l'idée de base de la construction du *TSA* associé à un système, et cette construction constitue un algorithme pour la satisfiabilité parce que les *TSA* possèdent les "bonnes" propriétés des automates, et particulièrement la décision du vide.

Un système n'est pas satisfiable si et seulement si il n'existe pas de calcul réussi dans l'automate d'ensemble d'arbres qui lui est associé. Le langage reconnu par le *TSA* est vide.

Nous terminons en insistant sur le caractère expressif des contraintes négatives. Les systèmes de contraintes positives et négatives ont un pouvoir d'expression strictement supérieur aux systèmes contenant seulement des contraintes positives (voir section 4.5 et [AKW93]). Comme nous l'avons mentionné plus haut, plus le vocabulaire des contraintes est riche, plus une analyse ensembliste aura les moyens d'être précise.

Supposons un système de contrainte SC qui approxime par X les valeurs d'une variable x. Il est par exemple possible, en ajoutant une contrainte négative C

$$X \neq \bot \ ,$$

de contrôler si x peut prendre une valeur. Si la conjonction SC et C n'est pas satisfiable, il n'existe pas d'interprétation non vide pour X. Le type de la variable x ne peut donc être non vide. En d'autres termes, x est inutile ou sa manipulation est incohérente.

La résolution des contraintes négatives est aussi une étape importante vers la résolution des contraintes avec symboles de projection, dont la satisfiabilité reste un problème ouvert à ce jour. Par exemple, $b_i^{(-1)}(b(X, Y)) = X$ est satisfiable si Y est non vide. Nous remarquons aussi sur cet exemple que la projection (ou la non-inclusion) exprime des dépendances entre les variables : X n'a de valeurs que si Y en possède.

Le premier chapitre introduit les notions usuelles d'alphabet gradué, d'arbres, finis et infinis. Dans la section 1.2, nous présentons les automates d'arbres, les automates de Büchi et de Rabin. Leur propriétés sont résumées et nous rappelons aussi les connexions entre logique et automates.

Le second chapitre est une introduction aux Automates d'Ensembles d'Arbres (*TSA*). Nous étudions les automates d'arbres infinis, vus comme reconnaisseurs d'ensembles de mots. Nous définissons de nouveaux outils : les automates d'ensembles de mots ou *WSA*. Les *WSA* sont en fait des reconnaisseurs d'arbres filiformes. Leur pouvoir d'expression est plus faible que celui des automates de Büchi, mais l'intérêt est le caractère évolutif de leur définition : les *TSA* sont une extension des *WSA*.

Le troisième chapitre présente les automates d'ensembles d'arbres ou *TSA*. La section 3.2 est dédiée à la démonstration de la décidabilité du problème : "l'ensemble reconnu par un *TSA* est-il vide?". Ce résultat est fondamental puisqu'il est le noyau de la méthode de résolution de contraintes que nous proposons. Une présentation informelle de cette démonstration apparaît en section 3.2.3. Des propriétés générales sur les ensembles reconnus par les *TSA* sont détaillés dans les sections 3.3 et 3.4.

Enfin, dans le dernier chapitre, nous proposons notre algorithme de résolution de systèmes de contraintes ensemblistes. Les propriétés des automates d'ensembles d'arbres y sont transposées ce qui permet de déduire des résulats sur les contraintes et l'ensemble de leurs solutions.

Chapitre 1

Préliminaires

1.1 Généralités

Les notations que nous utilisons sont celles de W. Thomas [Tho90] et de Jouannaud et Dershowitz [DJ90].

1.1.1 Arbres, Termes

Un *alphabet gradué* fini est un couple $(\Sigma, arité)$ où Σ est un ensemble fini de symboles (ou lettres) et *arité* une application de Σ dans $\mathbb{N}$. Pour toute lettre a de Σ, $arité(a)$ désigne l'*arité* de a. Les symboles d'arité 0 sont des *constantes*, ceux d'arité 1,2,...,n sont dits unaires, binaires,..., n-aires. La valeur maximale de *arité* sur Σ est $amax$. Pour tout entier positif p, Σ_p désigne l'ensemble des lettres d'arité p. Par la suite, nous écrirons Σ pour le couple $(\Sigma, arité)$ et nous supposerons que Σ contient toujours au moins une constante.

Soit X un ensemble dénombrable de variables. L'ensemble $T_\Sigma(X)$ des *termes* sur Σ est défini comme le plus petit ensemble qui vérifie :

- $\Sigma_0 \subseteq T_\Sigma$; $X \subseteq T_\Sigma(X)$;
- $b(t_1, \ldots, t_p) \in T_\Sigma(X)$ pour tout b de Σ_p avec $amax \geq p > 0$ et pour tout p-uplet $t_1, \ldots, t_p \in T_\Sigma(X)$.

L'ensemble $T_\Sigma(\emptyset)$ est l'ensemble des termes *clos* sur Σ et est noté T_Σ. Les lettres de Σ sont représentées par les symboles a, b, c, ... ; les termes par les symboles t, u, v, ..., parfois indicés ; les variables par les lettres x, y ...

Un *contexte* de T_Σ est un terme de $T_\Sigma(X)$ avec une seule occurrence d'une seule variable. Par exemple, $b(b(x,a),a)$. Pour tout contexte u, nous notons $u(t)$ le terme obtenu en substituant à la variable de u, le terme t. Dans l'exemple, $u(t)$ avec $u = b(b(x,a),a)$ et $t = c(a)$ représente le terme $b(b(c(a),a),a)$.

La *hauteur*, d'un arbre t, notée $h(t)$, est la longueur de sa plus grande branche. Plus formellement, $h(t) = 0$ si $t \in \Sigma_0 \cup X$, i.e. t est une constante ou une variable, et $h(t) = 1 + \max\{h(t_1), \ldots, h(t_p)\}$ si $t = b(t_1, \ldots, t_p)$.

L'ensemble des termes de hauteur k est $T_\Sigma^{=k}(X)$, celui des termes de hauteur inférieure à k est $T_\Sigma^{\leq k}(X)$.

Une forêt est un ensemble fini ou infini d'arbres finis. Si F est une forêt finie alors $Card(F)$, la *cardinalité* de F, est le nombre d'arbres qui la composent.

Les symboles F, G, H, ..., représentent des forêts.

1.1.2 Ensemble de Positions

Sur un ensemble de mots, l'opération $\cdot$ est la concaténation et ε le mot vide. La relation "*est préfixe de*" est notée $<$.

Un ensemble de positions **Pos** est un ensemble de mots d'entiers naturels supérieurs ou égaux à 1 qui vérifie les deux propriétés suivantes :

- $\forall u \cdot v \in \mathbf{Pos} \Rightarrow u \in \mathbf{Pos}$. (L'ensemble **Pos** est clos par préfixe.)
- $\forall u \cdot i \in \mathbf{Pos}, \forall j, 1 \leq j \leq i \Rightarrow u \cdot j \in \mathbf{Pos}$.

Un arbre t sur Σ peut alors être vu comme une application t d'un ensemble de positions dans un ensemble d'étiquettes Σ. On parle alors d'un arbre étiqueté t, de domaine $Dom(t)$ ou encore d'arbre Σ-valué. Dans ce mémoire, nous confondons arbre et terme.

Soit p une position dans $Dom(t)$, $t_{|p}$ désigne le sous-terme de t à la position p et $t[u]_p$ le terme obtenu en remplacant dans t le sous-terme $t_{|p}$ par u.

Un *chemin* π à travers l'arbre t est un sous-ensemble de $Dom(t)$ totalement ordonné par $<$, clos par préfixe. $t \mid \pi$ est la restriction de t à l'ensemble π.

Les *sous-termes directs* de $t = b(t_1, \ldots, t_p)$ sont $t_1, \ldots, t_p$. L'ensemble des sous-termes de t est récursivement défini comme l'ensemble de ses sous-termes directs augmenté des sous-termes de ses sous-termes directs. La relation $\trianglelefteq$ est l'ordre "est sous terme de" :

$$t \trianglelefteq t' \Leftrightarrow \exists p \in Dom(t) \quad t_{|p} = t'.$$

Une forêt F est dite *close* si elle est close selon l'ordre $\trianglelefteq$. C'est-à-dire :

$$\forall t \in F \; \forall u \in T_\Sigma \quad u \trianglelefteq t \Rightarrow u \in F$$

Les positions sont aussi appelées *nœuds*. La *racine* est la position ε et les *feuilles* les positions maximales selon l'ordre préfixe.

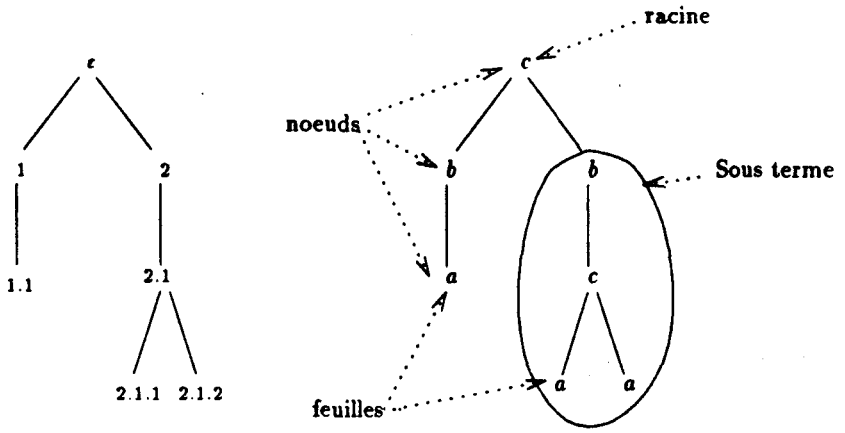


FIG. 1.1 - : Un arbre et son domaine

1.1.3 Arbres Infinis

Dans cette présentation des arbres infinis, nous nous restreignons aux arbres binaires infinis. Cette restriction n'est pas réductrice puisqu'un codage élémentaire permet de représenter tout arbre (infini) dans un arbre (infini) binaire.

Un arbre infini binaire Σ-valué de domaine $\{0,1\}^*$ (c'est-à-dire d'étiquettes dans Σ), est une application de $\{0,1\}^*$ dans Σ. Nous noterons T_Σ^∞, l'ensemble des arbres infinis et T_Σ^ω, l'ensemble des arbres finis ou infinis Σ-valués.

Soit t un arbre infini. Un chemin π dans t est un ensemble de positions infini, totalement ordonné et clos par préfixe. La restriction d'un arbre infini t à l'ensemble des positions de π est notée $t \mid \pi$. L'ensemble $In(t)$ représente les étiquettes rencontrées infiniment souvent dans l'arbre t :

$$In(t) = \{b \mid \exists^\infty p \quad t(p) = b\} \ , \ In(t \mid \pi) = \{b \mid \exists^\infty p \in \pi \quad t(p) = b\}.$$

Par la suite, nous écrirons souvent les positions dans un arbre infini avec des mots sur un alphabet (fini) Λ, plutôt qu'avec des mots d'entiers naturels. On parlera alors d'arbres infinis (binaires) Σ-valués de domaine Λ^*.

1.2 Automates d'Arbres

1.2.1 Automates Finis d'Arbres Finis

Les automates d'arbres finis sont une généralisation des automates de mots finis. En résumé, nous pouvons annoncer que les résultats classiques (Théorème de Kleene, déterminisation, minimalisation, propriétés de clôture par les opérations booléennes) sont facilement transposés au cas des arbres avec toutefois quelques

précisions. Il existe deux types d'automates d'arbres. Les automates *ascendants*[1] (*aaa*) qui parcourent les arbres des feuilles jusqu'à la racine, et les automates *descendants* (*ada*) qui commencent par la racine et terminent par les feuilles. Ces deux types d'automates reconnaissent la même famille de langages. La différence essentielle réside dans le fait qu'un automate descendant ne peut pas toujours être déterminisé, mais en revanche, sa définition est étendue pour la reconnaissance d'arbres infinis. Nous donnons brièvement les deux définitions.

Les résultats sont donnés sans preuve. Le lecteur pourra retrouver ces informations complètes dans [Dau53], [GS84],[HU79] et [Tho90].

Automates descendants

Définition 1.1 Un automate descendant d'arbres (*ada*) est un quadruplet $A = (\Sigma, Q, Q_i, \Delta)$. Σ est un alphabet gradué fini, Q est un ensemble fini d'états, $Q_i \subseteq Q$ est un ensemble d'états initiaux et $\Delta \subseteq \bigcup_{i=0}^{amax} \Sigma_i \times Q^{i+1}$est un ensemble de règles de transition.

Les règles sont notées $q \longrightarrow b(q_1, \ldots, q_p)$, où $b \in \Sigma_p$, $q_1, \ldots q_p, q \in Q$. Les états sont considérés comme des lettres d'arité 1. Un mouvement élémentaire dans A de t vers t' de $T_{\Sigma \cup Q}$ est noté

$$t \underset{\Delta}{\mapsto} t'$$

et vérifie:

$$t = u(q(b(t_1, \ldots t_p))), t' = u(b(q_1(t_1), \ldots q_p(t_p))) \ ,$$
$$q \longrightarrow b(q_1, \ldots q_p) \in \Delta \ ;$$
$$u \text{ est un contexte de } T_\Sigma \ .$$

La clôture réflexive et transitive de $\underset{\Delta}{\mapsto}$ est $\underset{\Delta}{\overset{*}{\mapsto}}$. Le langage *reconnu* par A est l'ensemble:

$$L(A) = \left\{ t \in T_\Sigma \mid \exists q \in Q_i \quad q(t) \underset{\Delta}{\overset{*}{\mapsto}} t \right\}.$$

Automates ascendants

On passe des automates ascendants aux automates descendants en renversant le sens des flèches dans les règles de transition.

Définition 1.2 Un automate ascendant d'arbres (*aaa*) est un quadruplet $A = (\Sigma, Q, Q_f, \Delta)$. Σ est un alphabet gradué fini, Q est un ensemble fini d'états, $Q_f \subseteq Q$ est un ensemble d'états finaux et $\Delta \subseteq \bigcup_{i=0}^{amax} \Sigma_i \times Q^{i+1}$ est un ensemble de règles de transition.

[1] Cette dénomination fait référence à la représentation graphique des arbres : racine en haut et feuilles en bas.

Les règles sont notées $b(q_1,\ldots,q_p) \rightarrow q$, où $b \in \Sigma_p$, $q_1,\ldots q_p, q \in Q$. Un mouvement élémentaire dans A de t vers t' est noté $t \underset{A}{\mapsto} t'$ et vérifie :

$$t = u(b(q_1(t_1),\ldots q_p(t_p))), t' = u(q(b(t_1,\ldots t_p)))$$
$$b(q_1,\ldots q_p) \rightarrow q \in \Delta$$

La clôture reflexive et transitive de $\underset{A}{\mapsto}$ est $\underset{A}{\overset{*}{\mapsto}}$. Le langage *reconnu* par A est l'ensemble :

$$L(A) = \left\{ t \in T_\Sigma \mid t \overset{*}{\mapsto} q(t) \wedge q \in Q_f \right\}.$$

Dans ce mémoire, nous utiliserons toujours des automates ascendants pour reconnaître des arbres finis. Nous simplifions souvent les notations en oubliant les termes sous les états dans l'écriture des transitions. Par exemple, nous écrirons $t \underset{\Delta}{\overset{*}{\mapsto}} q$ pour $t \underset{\Delta}{\overset{*}{\mapsto}} q(t)$.

Propriétés

L'automate A est *déterministe* si l'ensemble Δ ne contient pas deux règles de même membre gauche. Il est *complètement spécifié* si pour toute lettre $b \in \Sigma_p$, pour tout $q_1,\ldots,q_p \in Q$, il existe une règle de Δ de membre gauche $b(q_1,\ldots,q_p)$. Il est *minimal* s'il n'existe pas d'automate reconnaissant $L(A)$ et comportant moins d'états.

> Pour tout automate ascendant d'arbres, il existe un et un seul (au nom des états près) automate déterministe, complètement spécifié et minimal reconnaissant le même langage.

Cet automate peut être construit et cette construction est semblable à celle mise en œuvre dans le cas d'automates de mots [HU79].

La réduction du non déterminisme est perdue dans le cas d'automates descendants. Par exemple, la forêt $F = \{c(a,a), c(b,b)\}$ est reconnue par un automate d'arbres ascendant déterministe $A_a = (\{a,b,c\}, \{q,q',q_f\}, \{q_f\}, \Delta_a)$. L'ensemble des règles Δ_a est

$$a \rightarrow q\ , \qquad b \rightarrow q'\ ,$$
$$c(q,q) \rightarrow q_f\ , \quad c(q',q') \rightarrow q_f\ .$$

Il n'est pas possible de construire un automate d'arbres descendant déterministe reconnaissant F.

Mais pour les deux types d'automates, la propriété dite du vide est décidable :

> Il existe une procédure qui pour tout automate d'arbres A, termine et donne une valeur de vérité à la question "$L(A) = \emptyset$?"

Deux automates sont *équivalents* si ils reconnaissent le même langage.

La famille *REC*

Rappelons que la famille des langages reconnus par les automates d'arbres ascendants coïncide avec celle des langages reconnus par les automates d'arbres descendants. Cet ensemble est la famille *REC*.

Il est aussi possible de définir des opérations d'union, de concaténation et d'étoile pour des ensembles d'arbres [GS84]. On retrouve alors dans le cas des arbres, un théorème équivalent au théorème de Kleene. La famille des langages réguliers est la plus petite famille contenant les parties finies de T_Σ, close par union, concaténation et étoile, et est égale à *REC*.

Une caractérisation algébrique des forêts reconnaissables s'apparente au théorème de Myhill-Nerode [HU79]. Une forêt F est reconnaissable si et seulement si il existe une congruence d'index fini qui sature F. Soit $A = (\Sigma, Q, Q_f, \Delta)$, l'automate ascendant d'arbres minimal reconnaissant F. La relation $\approx$ définie par $\forall t, t' \in T_\Sigma \quad t \approx t' \Leftrightarrow \left(t \xrightarrow[\Delta]{*} q \wedge t' \xrightarrow[\Delta]{*} q\right)$ est une congruence d'index fini. C'est la congruence de Nerode.

La dernière propriété que nous donnons ici est la clôture de *REC* par les opérations booléennes, d'union, intersection et complémentaire. Cette clôture est effective, c'est-à-dire que pour tous langages réguliers d'arbres L_1 et L_2, on peut construire les automates reconnaissant $L_1 \cup L_2$, $L_1 \cup L_2$ et $\overline{L_1}$.

Les forêts reconnaissables seront aussi appelées forêts régulières ou ensembles réguliers d'arbres ou langages réguliers d'arbres. Par la suite, nous parlerons souvent de n-uplets de langages réguliers. Ce sont naturellement des n-uplets dont chaque composante est une forêt régulière.

1.2.2 Automates Finis d'Arbres Infinis

Parmi toutes les extensions des automates finis, les automates d'arbres infinis sont certainement l'une des plus motivantes. Nous verrons plus loin les applications en logique, aux problèmes de décision et en particulier, nous montrerons comment les automates d'arbres infinis permettent de résoudre une classe non triviale de problèmes liés aux contraintes ensemblistes.

Cette présentation est succincte. Pour une étude plus détaillée, le lecteur est invité à consulter l'article de W. Thomas [Tho90].

Définition 1.3 (Automate de Büchi) Un automate d'arbres infinis de Büchi sur l'alphabet Σ est un quadruplet $A = (Q, Q_0, \Delta, F)$ où Q est un ensemble fini d'états, $Q_0 \subseteq Q$, Δ est une relation de transition $\Delta \subseteq Q \times \Sigma \times Q \times Q$, et $F \subseteq Q$ est un ensemble d'états finaux.

Un calcul de A sur un arbre $t \in T_\Sigma^\infty$ est un arbre r de T_Q^∞, $r : \{0,1\}^* \to Q$ avec $r(\varepsilon) \in Q_0$, et $(r(w), t(w), r(w0), r(w1)) \in \Delta$ pour toute position w de $\{0,1\}^*$.

Le calcul r est réussi selon le critère de Büchi si pour tout chemin dans r, l'ensemble des états rencontrés un nombre infini de fois traverse F :

$$\text{Pour tout chemin } \pi \text{ de } r \quad In(r \mid \pi) \cap F \neq \emptyset \tag{1.1}$$

Définition 1.4 (Automate de Rabin) Un automate d'arbres infinis de Rabin sur l'alphabet Σ est un quadruplet $A = (Q, Q_0, \Delta, \Omega)$. Les ensembles Q, Q_0, Δ sont identiques à ceux de la définition précédente. Ω est une collection finie de couples d'ensembles d'états $\{(L_1, U_1), \ldots, (L_n, U_n)\}$.

Un calcul r est réussi selon le critère de Rabin si

$$\begin{array}{c}\text{Pour tout chemin } \pi \text{ de } r \text{ il existe } i \text{ dans } \{1, \ldots, n\} \text{ tel que} \\ In(r \mid \pi) \cap L_i = \emptyset \quad \text{et} \quad In(r \mid \pi) \cap U_i \neq \emptyset\end{array} \tag{1.2}$$

L'ensemble des arbres pour lesquels il existe un calcul réussi dans un automate de Büchi (respectivement Rabin) est le langage reconnu par cet automate. Un tel langage est dit Büchi-reconnaissable (respectivement Rabin-reconnaissable).

Propriétés

La puissance d'expression de ces deux types d'automates est différente. La classe des langages Büchi-reconnaissables, *BREC*, est strictement incluse dans la classe des langages Rabin-reconnaissables, *RREC*. La classe *RREC* est close par les opérations booléennes alors que *BREC* n'est pas close par complémentarité.

Propriété 1.1 - *La classe des langages Rabin-reconnaissables est close par les opérations d'union, intersection et complémentarité.*

- *Le problème du vide est décidable dans les classes des automates de Büchi et de Rabin.*

En ce qui concerne les automates de Rabin, la décision du vide est obtenue en tentant d'exhiber un calcul sur un arbre régulier, c'est-à-dire, un arbre infini dont le nombre de ses sous-arbres distincts est fini [Cou83].

Propriété 1.2 *Il existe un arbre infini régulier dans toute forêt non vide et Rabin-reconnaissable.*

1.2.3 Automates et Décision

Mots infinis

Une motivation pour l'étude d'automates d'objets infinis est leurs connexions avec la logique. Dans le cas de mots infinis, Büchi [Büc60] a montré que toute propriété exprimée dans le calcul séquentiel était équivalente à une condition d'acceptation dans un automate de mots infinis.

Les formules du calcul séquentiel permettent de formaliser des propriétés sur les mots. Par exemple [Pin93] :

$$\exists x\ \exists y\ (x < y) \wedge (x \in Q_a) \wedge (y \in Q_b).$$

Cette formule est interprétée sur le mot u par il existe deux entiers $x < y$ tels que dans u, la lettre à la position x est un a et la lettre à la position y est un b. La formule *définit* les mots infinis de $\{a,b\}^*a\{a,b\}^*b\{a,b\}^\infty$.

Les variables (du premier ordre) x et y portent sur des positions, mais le calcul séquentiel, permet aussi l'utilisation de variables (du second ordre) qui seront interprétées comme des *ensembles* de positions. Par exemple, dans

$$\exists X\ (0 \in X) \wedge (\forall x\ (x \in X) \Leftrightarrow (x+1 \notin X)) \wedge (\forall y\ y \in Q_a \Rightarrow y \in X),$$

X est l'ensemble des positions paires et toute lettre a d'un mot u satisfaisant cette formule apparaît à une position paire.

Les mots infinis u sur un alphabet Σ sont représentés par une structure de la forme $(\omega, 0, +1, <, (Q_a)_{a\in\Sigma})$, où $(\omega, 0, +1, <)$ est la structure des entiers naturels, avec 0, une fonction de successeur et l'ordre habituel, et les prédicats $(Q_a)_{a\in\Sigma}$ sont les ensembles de positions pour chaque lettre : $Q_a = \{x \in \omega \mid u(x) = a\}$.

Les formules sont définies par

Atomes 0, x, $t+1$ avec x une variable du premier ordre et t un atome ;

Formules atomiques $t \in Q_a$, $t \in X$, $t = t'$, $t < t'$ où t et t' sont des atomes et X une variable du second ordre ;

L'ensemble des formules est obtenu par clôture inductive des formules atomiques par les opérations booléennes et les quantificateurs. La théorie bâtie sur ce langage est la *théorie monadique du second ordre à 1 successeur* : $S1S_\Sigma$.

Souvent, les prédicats Q_a sont remplacés par des variables libres du second ordre. Dans ce cas les formules de $S1S_\Sigma$ sont remplacées par des formules du langage $S1S$, de la forme $\varphi(X_1, \ldots, X_n)$ où les X_k sont des variables libres. Toute formule $x \in Q_a$ est remplacée par $x \in X_k$. Pour un mot infini u de $(\{0,1\}^n)^\omega$, $x \in X_k$ est vraie si la $k$me composante de u est 1 (voir page 39). Maintenant, en codant les lettres de Σ sur $\{0,1\}^n$, pour un n approprié, toute formule de $S1S_\Sigma$ peut s'écrire dans $S1S$ et réciproquement.

Un mot $u = (\omega, 0, +1, <, Q_1, \ldots, Q_n)$ valide une formule $\varphi(X_1, \ldots, X_n)$ si φ est vraie avec Q_i pour interprétation pour X_i. La formule φ *définit* le langage $L(\varphi)$ des mots qui la valident.

Théorème 1.1 (Büchi [Büc60])

Un langage de mots infinis est définissable dans S1S si et seulement si il est régulier.

La théorie S1S est décidable.

Les langages de mots infinis réguliers sont les reconnaissables par les automates de mots de Büchi[2]. Nous ne présenterons pas les automates de mots infinis mais nous voulons simplement montrer la puissance d'expression de ces outils et comment elle est utilisée.

La preuve suit un schéma que l'on retrouve pour d'autres problèmes de décision (voir figure 1.2), d'autres théories [Rab69, DT90, CCD93b]. D'abord les automates ont de bonnes propriétés :

- Décision du vide,

- Clôture booléenne.

Ensuite, l'équivalence *formule de S1S—automate de Büchi* est montrée de façon constructive, le plus souvent en utilisant les propriétés de clôture booléenne. A partir d'une formule de *S1S*, on construit un automate de mots de Büchi et vice versa. A une formule ϕ de *S1S* correspond alors un automate A dont le langage reconnu $L(A)$ est le langage défini par ϕ. Grâce à la décision du vide, on peut alors dire si $L(A)$ est vide ou non, on peut donc dire si la formule est satisfiable ou non.

Une instance d'un problème $\mathcal{P}$ $\longleftrightarrow$ Un automate A
Une solution de $\mathcal{P}$ $\longleftrightarrow$ Un élément de $L(A)$
Satisfiabilité de $\mathcal{P}$ $\longleftrightarrow$ Vide dans A.

FIG. 1.2 - : Automate et décision

Arbres infinis

Pour étudier les aspects logiques des automates d'arbres infinis, un arbre infini binaire étiqueté sur l'alphabet $\{0,1\}^n$ est donné par :

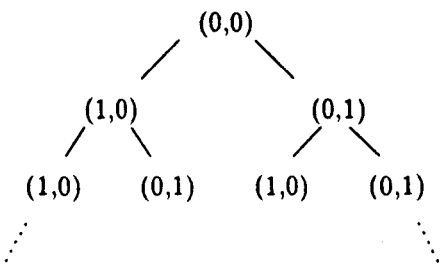
$$(\{a,b\}^*, \varepsilon, succ_a, succ_b, <, P_1, \ldots, P_n)$$

Les deux opérations $succ_a$ et $succ_b$ sont les fonctions de successeurs sur $\{a,b\}^*$. Pour un mot w de $\{a,b\}^*$, $succ_a(w)$ est noté wa et $succ_b(w)$ est noté wb. L'ordre $<$ est l'ordre préfixe sur les mots, et un mot w est dans P_i si et seulement si la

[2]A ne pas confondre avec les automates d'arbres de Büchi...

$i^{\text{ème}}$ composante de $t(w)$ est 1.

Exemple 1.1 L'arbre



est donné par $(\{a,b\}^*, \varepsilon, succ_a, succ_b, <, P_1, P_2)$ avec les ensembles $P_1 = \{a,b\}^*a$ et $P_2 = \{a,b\}^*b$.

Etant donnés un alphabet $\{a,b\}^*$, un ensemble de variables $x, y, \ldots$ du premier ordre, un ensemble de variables X, Y, ...du second ordre, les formules sont définies par :

Atomes ε, x, ta, tb où t est un atome et x une variable ;

Formules atomiques $t \in X$, $t < t'$, $t = t'$ où t, t' sont des atomes et X une variable ;

Formules Elles ont obtenues par la clôture inductive des formules atomiques par les connecteurs logiques (et, ou, non) et quantificateurs (universel et existentiel).

Un langage d'arbres F est définissable dans ce langage si il existe une formule ϕ à n variables $X_1, \ldots, X_n$ telle que pour tout arbre $t = (\{a,b\}^*, \varepsilon, succ_a, succ_b, <, P_1, \ldots, P_n)$ de F, ϕ est vraie, chaque X_i ayant pour interprétation P_i.

La théorie monadique du second ordre à 2 successeurs *S2S* est l'ensemble des formules satisfiables. Par abus de langage, on appelle souvent *S2S* l'ensemble des formules bien formées.

Théorème 1.2 (Théorème de Rabin [Rab69])
La théorie monadique du second ordre à 2 successeurs est décidable.

Comme dans le cas des mots, la preuve consiste à associer à une formule ϕ de *S2S* un automate de Rabin A tel que ϕ soit vraie si et seulement si il existe un calcul dans A.

Techniquement, cette preuve est similaire à celle du cas des mots, toute la difficulté repose sur le problème du passage au complémentaire et de la détermination du vide dans la classe des automates de Rabin.

Les résultats énoncés ici l'ont été dans le cadre de l'arbre infini binaire. Ils restent valides pour les arbres d'arité quelconque. Par exemple, la théorie monadique du second ordre à k (arbres k-aires) successeurs est décidable (*SkS*).

Lorsque les variables du second ordre sont interprétées par des ensembles finis, on parle de la théorie monadique *faible WS2S*. Le théorème suivant dû aussi à Rabin donne une caractérisation de *WS2S* et des langages d'arbres Büchi reconnaissables.

Théorème 1.3 *Une forêt F est Büchi-reconnaissable si et seulement si F est définissable par une formule de S2S $\exists Y_1, \ldots, Y_m \varphi(Y_1, \ldots, Y_m, X_1, \ldots, X_n)$ où φ est une formule de WS2S.*

Arbres Caractéristiques

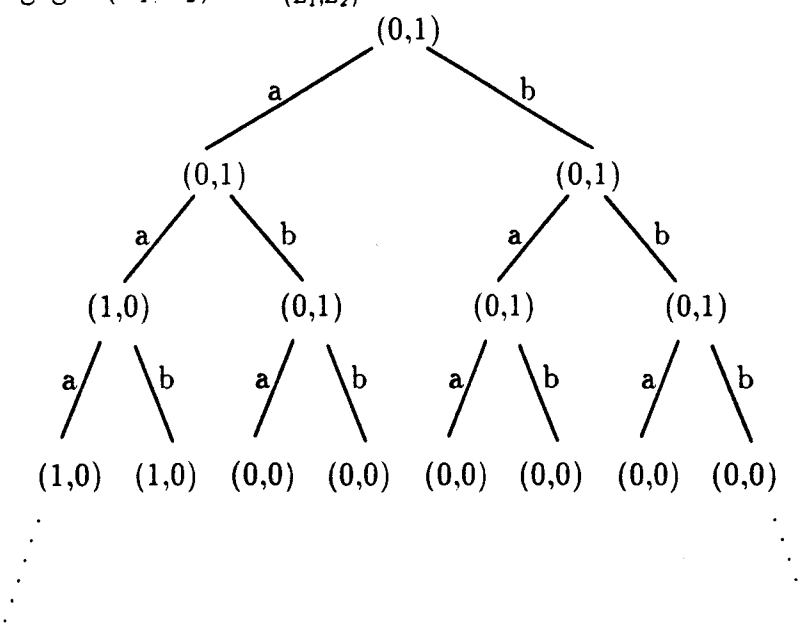
Nous appelons *arbre caractéristique* de n-uplets de langages de mots sur un alphabet à k lettres tout arbre infini k-aire $\{0,1\}^n$ valué. C'est-à-dire, tout arbre infini dont tous les noeuds ont k successeurs qui sont étiquetés par des n-uplets de valeurs binaires 0 ou 1.

L'arbre caractéristique d'un n-uplet $(L_1, \ldots, L_n)$ de langages (de mots) sur $\Sigma = \{a_1, \ldots, a_k\}$ est l'arbre $\{0,1\}^n$-valué de domaine Σ^* défini par

$$\text{si } t_{(L_1,\ldots,L_n)}(m) = (d_0, \ldots, d_n) \text{ alors } \quad m \in L_i \Leftrightarrow d_i = 1$$

Exemple 1.2 Soit Σ l'alphabet composé des deux lettres a et b. Soient $L_1 = aa(a+b)^*$; $L_2 = \{\varepsilon, a, b, ab, ba, bb\}$ deux langages sur Σ. L'arbre caractéristique du couple de langages (L_1, L_2) est $t_{(L_1,L_2)}$:

$t_{(L_1,L_2)} =$



La racine de cet arbre correspond au mot vide ε. L'étiquette $(0,1)$ signifie donc $\varepsilon \notin L_1$ et $\varepsilon \in L_2$. Sous le nœud qui correspond au mot aa n'apparaissent plus que des étiquettes $(1,0)$ et sous les autres nœuds $(aba, abb, baa, bab, bba, bbb)$ n'apparaissent plus que des étiquettes $(0,0)$.

Chapitre 2

Une Approche par les Mots

Ce chapitre est une introduction aux Automates d'Ensembles d'Arbres par la théorie des automates d'arbres infinis.

Nous avons vu dans la section 1.2.3 les liens entre automates et logique et le codage d'ensembles de mots dans un automate d'arbres (ou de mots) infinis. Nous étudions maintenant les automates de Rabin en tant que reconnaisseurs de mots ou d'arbres caractéristiques de n-uplets de langages de mots. Nous définissons ensuite un nouvelle classe d'automates : les Automates d'Ensembles de Mots (*WSA* pour *Word Set Automata*) qui sont la restriction aux arbres filiformes des *TSA*.

En identifiant les arbres filiformes et les mots, nous comparons les *WSA* et les automates d'arbres infinis et nous montrons que la classe des langages acceptés par les *WSA* est une sous-classe des langages Büchi reconnaissables.

Pour marquer les similitudes et les différences entre les *WSA* et les automates d'arbres infinis, nous donnons d'abord la définition des automates de Rabin désynchronisés. La synchronisation dans l'évolution d'un automate d'arbres infinis, d'un nœud vers ses sous-nœuds, s'exprime par la syntaxe des règles de transition, mais n'est pas nécessaire : elle ne conditionne pas la puissance d'expression des automates. Cette remarque permet de définir les automates désynchronisés dont les calculs sont menés sur une branche, indépendamment des résultats sur les autres branches.

L' introduction des automates à entrée libre, nous permet de nous détacher des étiquettes des arbres lus.

Le fonctionnement des automates désynchronisés, à entrée libre est alors très proche de celui des *WSA*. La différence essentielle se situe dans la condition de réussite des calculs, moins puissante pour les *WSA* (voir la proposition 2.2 et le fait 2.2).

Ce chapitre n'est pas essentiel pour la compréhension des *TSA* et de l'algorithme de résolution des contraintes ensemblistes.

2.1 Désynchronisation

Rappel Soit Λ un alphabet à k lettres, un arbre infini k-aire $\{0,1\}^n$-valué de domaine Λ^*. Chaque nœud de l'arbre correspond à une position qui est un mot de Λ^*. Un chemin est un ensemble infini de mots clos par préfixe, totalement ordonné.

Par souci de clarté, les résultats suivants sont présentés avec un alphabet Λ réduit à deux lettres, donc sur des arbres binaires de domaine $\{a,b\}^*$.

Tout n-uplet de langages sur l'alphabet Λ possède un arbre caractéristique sur $\{0,1\}^n$ de domaine Λ^*. Un automate de Büchi ou de Rabin de mots infinis sur $\{0,1\}^n$ peut donc être vu comme un reconnaisseur de n-uplets d'ensembles de mots sur Λ.

Intéressons nous aux automates de Rabin et particulièrement à leur fonctionnement. Rappelons la définition donnée page 35.

Définition 2.1 (Automate de Rabin)
Un automate d'arbres infinis de Rabin sur l'alphabet Σ de domaine $\{a,b\}^*$ est un quadruplet $A = (Q, Q_0, \Delta, \Omega)$, où Q est un ensemble fini d'états, $Q_0 \subseteq Q$, Δ est une relation de transition $\Delta \subseteq Q \times \Sigma \times Q \times Q$, et Ω est une collection finie de couples d'ensembles d'états $\{(L_1, U_1), \ldots, (L_n, U_n)\}$.

Un calcul de A sur un arbre $t \in T_\Sigma^\infty$ est une application $r : \{a,b\}^* \to Q$ qui vérifie une relation de *compatibilité avec les règles de* Δ :

- $r(\epsilon) \in Q_0$,
- $(r(m), t(m), r(ma), r(mb)) \in \Delta$ pour tout mot m de $\{a,b\}^*$.

Un calcul est réussi selon le critère de Rabin si

$$\text{Pour tout chemin } \pi \text{ de } r \text{ il existe } i \text{ dans } \{1,\ldots,n\} \text{ tel que } In(r \mid \pi) \cap L_i = \emptyset \quad \text{et} \quad In(r \mid \pi) \cap U_i \neq \emptyset. \tag{2.1}$$

Le langage reconnu par A est l'ensemble $L(A)$ des arbres pour lesquels il existe un calcul réussi.

Par la forme des règles de Δ, les évolutions de l'automate dans la "direction" a sont synchronisées avec celles dans la "direction" b. Nous allons montrer que cette synchronisation est opérationnelle, c'est-à-dire qu'elle n'influence pas sur la puissance d'expression des automates. Intuitivement, nous pouvons remplacer les règles de la forme (q, l, q', q'') en deux règles de la forme (s, l, a, s') et (s, l, b, s''). Les définition et fonctionnement des automates ne sont que légèrement modifiés.

Définition 2.2 (Automate désynchronisé d'arbres — ADA)
Un automate d'arbres infinis désynchronisé sur l'alphabet Σ est un quadruplet $A = (Q, Q_0, \Delta, \Omega)$, où Q est un ensemble fini d'états, $Q_0 \subseteq Q$, Δ est une relation

de transition $\Delta \subseteq Q \times \Sigma \times \{a,b\} \times Q$, et Ω est une collection finie de couples d'ensembles d'états $\{(L_1,U_1),\ldots,(L_n,U_n)\}$.

Un calcul de A sur un arbre $t \in T_\Sigma^\infty$ est un arbre $r : \{a,b\}^* \to Q$ qui vérifie une relation de *compatibilité avec les règles de* Δ :

- $r(\epsilon) \in Q_0$,
- $(r(m),t(m),a,r(ma)) \in \Delta$ et $(r(m),t(m),b,r(mb)) \in \Delta$ pour tout mot m de $\{a,b\}^*$.

Un calcul est réussi si

$$\begin{array}{c}\text{Pour tout chemin } \pi \text{ de } t \text{ il existe } i \text{ dans } \{1,\ldots,n\} \text{ tel que} \\ In(r \mid \pi) \cap L_i = \emptyset \quad \text{et} \quad In(r \mid \pi) \cap U_i \neq \emptyset.\end{array} \tag{2.2}$$

Le langage reconnu par A est l'ensemble $L(A)$ des arbres pour lesquels il existe un calcul réussi.

Cette définition apparaît dans les travaux de N. Klarlund sur les conditions d'acceptation des automates d'arbres infinis [Kla90].

Exemple 2.1 Soit $A=(Q,Q_0,\Delta,\Omega)$ un ADA sur $\Sigma = \{0,1\}$. Soit $Q = \{q_f,q_i\}$, $Q_0 = \{q_i\}$, $\Omega = \{(\emptyset,Q)\}$, et Δ est l'ensemble des règles suivantes :

$$\begin{array}{ll}(q_i,a,0,q_i) & (q_i,b,0,q_i) \\ (q_i,a,1,q_f) & (q_i,b,1,q_f) \\ (q_f,a,1,q_f) & (q_f,b,1,q_f)\end{array}$$

Sur chaque branche d'un arbre reconnu par A, les successeurs (selon l'ordre préfixe des valeurs du domaine) d'un nœud étiqueté par 1 sont des nœuds étiquetés par 1.

A reconnaît les arbres infinis qui sont les arbres caractéristiques de langages sur $\{a,b\}^*$ clos par suffixe.

Proposition 2.1 *La classe des langages reconnus par les automates désynchronisés DREC est la classe des langages Rabin-reconnaissables RREC.*

Preuve.

• *DREC* $\subseteq$ *RREC*. Soit $A^d = \left(Q^d,Q_0^d,\Delta^d,\Omega^d\right)$ un ADA. Donnons la construction d'un automate de Rabin A^r équivalent à A^d.

Construction : $A^r = (Q^r,Q_0^r,\Delta^r,\Omega^r)$ avec

- $Q^r = Q^d$,
- $Q_0^r = Q_0^d$.
- $\Delta^r = \{(q,l,q',q'') \mid (q,l,a,q') \in \Delta^d \wedge (q,l,b,q'') \in \Delta^d\}$,

- $\Omega^r = \Omega^d$.

Correction de la construction.

— $L(A^d) \subseteq L(A^r)$

Soit t un arbre infini reconnu par A^d. Il existe alors un calcul réussi r^d sur l'arbre t dans A^d. Montrons qu'il existe aussi un calcul réussi sur t dans A^r.

Nous prouvons que r^d est aussi un calcul réussi dans A^r. Pour cela, il suffit de remarquer que r^d est compatible avec les règles de Δ^r :

$$\begin{aligned} & \forall m \in \{a,b\}^* \begin{cases} (r^d(m), t(m), a, r^d(ma)) \in \Delta^d \\ (r^d(m), t(m), b, r^d(mb)) \in \Delta^d \end{cases} \\ \Rightarrow \quad & \forall m \in \{a,b\}^* \ (r^d(m), t(m), r^d(ma), r^d(mb)) \in \Delta^r \end{aligned}$$

— $L(A^r) \subseteq L(A^d)$

Si t est un arbre reconnu par A^r, alors il est reconnu par A^d. En effet, si r^r un calcul réussi sur t dans A^r, alors r^r est encore compatible avec les règles de Δ^d, ceci pour les même raisons que précédemment.

• $RREC \subseteq DREC$. Dans ce sens, la construction est moins immédiate. L'idée principale est d'anticiper le choix des règles de l'automate de Rabin.

Construction Soit $A^r = (Q^r, Q_0^r, \Delta^r, \Omega^r)$ un automate de Rabin. Soit $A^d = \left(Q^d, Q_0^d, \Delta^d, \Omega^d\right)$ avec :

- $Q^d = \Delta^r$;
- $Q_0^d = \{(q,l,q',q'') \in Q^d \mid q \in Q_0^r\}$;
- Δ^d est défini par

$$\begin{cases} ((q,l,q',q''),l,a,(q',l',q'_1,q'_2)) \in \Delta^d \\ ((q,l,q',q''),l,b,(q'',l'',q''_1,q''_2)) \in \Delta^d \end{cases}$$

  pour tout (q,l,q',q''), (q',l',q'_1,q'_2), (q'',l'',q''_1,q''_2) de Q^d.

- Ω^d contient les paires (L^d, U^d) dont la projection sur la première composante (projection sur Q^r) donne une paire (L^r, U^r) de Ω^r.

Correction de la construction. Montrons que $L(A^d) = L(A^r)$.

— $L(A^d) \subseteq L(A^r)$. Soit r^d un calcul réussi dans A^d sur un arbre t. Le calcul r^r qui associe à un mot m de $\{a,b\}^*$ la projection sur la première composante de $r^d(m)$ est un calcul réussi de A^r. Formellement, soit p la projection :

$$\begin{aligned} p : Q^d &\rightarrow Q^r \\ (q,l,q',q'') &\mapsto q \end{aligned}$$

Alors $r^r = p(r^d)$ est un calcul réussi de A^r. En effet r^r est compatible avec les règles de Δ^r, car :

$$\forall m \in \{a,b\}^* \begin{cases} (r^d(m), t(m), a, r^d(ma)) \in \Delta^d \\ (r^d(m), t(m), b, r^d(mb)) \in \Delta^d \end{cases}$$

Par construction, $r^d(m) = (q, t(m), q', q'')$, $r^d(ma) = (q', l', q'_1, q'_2)$, $r^d(mb) = (q'', l'', q''_1, q''_2)$. Donc,

$$\forall m \in \{a,b\}^* \ (r^r(m), t(m), r^r(ma), r^r(mb)) \in \Delta^r.$$

La vérification de la condition de réussite est immédiate.

— $L(A^r) \subseteq L(A^d)$. Soit r^r un calcul réussi dans A^r sur un arbre t. Le calcul r^d qui associe à un mot m de $\{a,b\}^*$ la règle utilisée au nœud m pour calculer $r^r(ma)$ et $r^r(mb)$ est un calcul réussi de A^d. Formellement, r^d est défini par :

$$\forall m \in \{a,b\}^* \ r^d(m) = (r^r(m), t(m), r^r(ma), r^r(mb)).$$

Le calcul r^d est évidemment compatible avec les règles de Δ^d et la construction de Ω^d implique directement la condition de réussite. □

2.2 Automates à Entrée Libre

Les étiquettes des arbres lus (les lettres de Σ ou de $\{0,1\}^n$ dans le cas précédent) ne sont pas essentielles pour étudier le déroulement d'un calcul. Dans un automate de Rabin à entrée libre (*input free* [Tho90]), les étiquettes sont "rangées" ou "intégrées" dans les états.

On passe d'un automate de Rabin $A^r = (Q^r, Q^r_0, \Delta^r, \Omega^r)$ sur Σ à un automate à entrée libre $A = (Q, Q_0, \Delta, \Omega)$ en posant :

- $Q = Q^r \times \Sigma$,
- $Q_0 = Q^r_0 \times \Sigma$,
- $\Delta \subseteq Q \times Q \times Q$ et $((q,a),(q',a'),(q'',a'')) \in \Delta$ si et seulement si $(q,a,q',q'') \in \Delta^r$,
- Ω contient les paires (U, L) dont la projection sur la première composante (projection sur Q^r) donne une paire (U^r, L^r) de Ω^r.

Clairement, les calculs réussis dans A sont des paires (r,t) telles que r soit un calcul réussi sur t dans l'automate A^r.

Le passage inverse, peut être réalisé sans difficulté en ajoutant par exemple une valuation des états, i.e. une application de Q dans Σ.

Cette présentation s'étend aussi facilement aux automates désynchronisés. Nous pouvons alors donner une définition équivalente des automates désynchronisés :

Définition 2.3 Un automate désynchronisé d'arbres Σ-valués est un quadruplet $A = (Q, Q_0, \Delta, \Omega, Val)$. Q est un ensemble fini d'états, Q_0 sont les états initiaux, Ω est une collection finie de couples d'ensembles d'états $\{(L_1, U_1), \ldots, (L_n, U_n)\}$, Val est une application de Q dans Σ, et $\Delta \subseteq Q \times \{a, b\} \times Q$ est un ensemble de règles.

- Un calcul est une application r de Σ^* dans Q compatible avec les règles de $\mathcal{S}$. C'est-à-dire qu'elle vérifie :

$$\begin{array}{l} r(\varepsilon) \in Q_0 \\ \forall a_i \in \Sigma \ \forall m \in \Sigma^* \quad (r(m), a_i, r(m.a_i)) \in \Delta \ . \end{array}$$

- Un calcul est réussi si

$$\begin{array}{c} \text{Pour tout chemin } \pi \text{ de } t \text{ il existe } i \text{ dans } \{1, \ldots, n\} \text{ tel que} \\ In(r \mid \pi) \cap L_i = \emptyset \quad \text{et} \quad In(r \mid \pi) \cap U_i \neq \emptyset \end{array} \tag{2.3}$$

- Le langage reconnu par A est l'ensemble

$$L(A) = \{Val \circ r \mid r \text{ est un calcul réussi}\}.$$

où $Val \circ r$ est la composée de Val et de r.

Exemple 2.2 Soit $A' = (Q', Q'_0, \Delta', \Omega', Val)$ un ADA à entrée libre. Soit $Q' = \{q_f, q_i\}$, $Q'_0 = \{q_i\}$, $\Omega' = \{(\emptyset, Q')\}$, et Δ' est l'ensemble des règles suivantes :

$$\begin{array}{ll} (q_i, a, q_i) & (q_i, b, q_i) \\ (q_i, a, q_f) & (q_i, b, q_f) \\ (q_f, a, q_f) & (q_f, b, q_f) \end{array}$$

et Val vérifie : $Val(q_i) = 0$, $Val(q_f) = 1$.

A' reconnaît le même langage que l'automate A de l'exemple 2.1.

Dans le cas où $\Sigma = \{0, 1\}^n$, un automate désynchronisé A d'arbres $\{0, 1\}^n$-valués reconnaît l'ensemble $L(A)$ des arbres caractéristiques de n-uplets de langages de mots. Alors, par extension,

Un automate désynchronisé A d'arbres $\{0, 1\}^n$-valués reconnaît l'ensemble $\mathcal{L}$ des n-uplets de langages de mots défini par :

$$\begin{array}{c} \mathcal{L} = \{((L_1, r), \ldots, (L_n, r)) \mid r \text{ est un calcul réussi}\} \\ \text{avec } (L_i, r) = \{m \in \{a, b\}^* \mid r(m)(i) = 1\} \ . \end{array}$$

La notation $r(m)(i)$ désigne la ime *valeur du n-uplet de $\{0, 1\}^n$, $r(m)$.*

2.3 Automates d'Ensembles de Mots : *WSA*

Les automates d'ensembles de mots - *WSA* pour *Word Set Automata* - sont des objets très proches des automates désynchronisés d'arbres à entrée libre.

La présentation des règles de transition est légèrement différente. Nous utilisons les lettres de $\{a, b\}$ comme des symboles de fonction unaires et le symbole ϵ comme une constante. Cette notation se justifiera par l'extension des *WSA* en des reconnaisseurs de n-uplets de langages d'arbres. Nous identifions en fait, les mots avec des arbres filiformes.

Si Λ est un alphabet de mots alors on considère l'alphabet gradué $\Sigma = (\Lambda \cup \{\#\}, \mathit{arité})$ où $\mathit{arité}(a) = 1$ si $a \in \Lambda$ et $\mathit{arité}(\#) = 0$. A un mot de Λ^*, correspond par la bijection f, un terme de T_Σ inductivement défini par :

$$f(\epsilon) = \#$$
$$f(u.a) = a(f(u)) \ .$$

Par la suite, nous identifierons $\#$ avec le symbole ϵ, l'alphabet Σ avec l'alphabet Λ et tout mot u de Λ^* avec le terme $f(u)$.

La bijection f s'étend sur les langages et les n-uplets de langages : $f(\mathcal{L}) = \{(f(L_1), \ldots, f(L_n)) \mid (L_1, \ldots, L_n) \in \mathcal{L}\}$ et $L_i = \{f(m) \mid m \in L_i\}$.

Cette présentation implique que l'identification avec les automates classiques est obtenue via la bijection f.

Enfin, les conditions de réussite des calculs sont plus faibles que dans le cas des automates de Rabin.

Définition 2.4 (WSA) Un Automate d'Ensembles de Mots, ou *WSA*, est un quintuplet $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{S}, \Omega, \mathit{Val})$. $\mathcal{Q}$ est un ensemble fini d'états, $\Omega \subseteq 2^{\mathcal{Q}}$ et Val est une application de $\mathcal{Q}$ dans $\{0,1\}^n$, et $\mathcal{S}$ est un ensemble de règles de la forme :

$$\epsilon \rightarrow \mathbf{q}$$
$$a_i(\mathbf{q}) \rightarrow \mathbf{q}'$$

où $\mathbf{q}$ et $\mathbf{q}'$ sont des états de $\mathcal{Q}$ et a_i une lettre de Σ.

• Un calcul est une application r de T_Σ dans $\mathcal{Q}$ compatible avec les règles de $\mathcal{S}$. C'est-à-dire qu'elle vérifie :

$$\forall a_i \in \Sigma \ \forall m \in T_\Sigma \quad a_i(r(m)) \rightarrow r(a_i.m) \in \mathcal{S} \ .$$

Un calcul est réussi si et seulement si $r(T_\Sigma) \in \Omega$.

• Le langage $\mathcal{L}(\mathcal{A})$ reconnu par $\mathcal{A}$ est défini par les équations suivantes :

$$\mathcal{L}(\mathcal{A}) = \{\mathcal{L}(\mathcal{A}, r) \mid r \text{ est un calcul réussi}\}$$
$$\mathcal{L}(\mathcal{A}, r) = (L_1, \ldots, L_n)$$
$$\text{où } L_i = \{m \in T_\Sigma \mid \mathit{Val}(r(m))(i) = 1\}$$

($\mathit{Val}(r(m))(i)$ représente la $i$me valeur du n-uplet $\mathit{Val}(r(m))$)

Lorsque la valuation Val est une application de $\mathcal{Q}$ dans $\{0,1\}^n$, alors le *WSA* est dit de rang n.

2.4 *WSA* et Automates de Büchi

Nous comparons maintenant la puissance d'expression des *WSA* et des automates d'arbres infinis de Büchi. L'arbre caractéristique d'un n-uplet de langages de mots $\mathcal{L} = (L_1, \ldots, L_n)$ est noté $t_{\mathcal{L}}$.

Rappelons que nous identifions mots et arbres filiformes.

Proposition 2.2 *Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{S}, \Omega, Val)$ un WSA de rang n. Alors, il existe un automate de Büchi B d'arbres infinis de domaine $\Sigma^* \{0,1\}^n$-valués tel que*

$$\mathcal{L} \in \mathcal{L}(\mathcal{A}) \Leftrightarrow t_{\mathcal{L}} \in L(B).$$

Preuve. Dans la suite, nous identifions donc les éléments du domaine des arbres reconnus par B (i.e. Σ^*) et les termes traités par $\mathcal{A}$ (dans T_Σ). Par conséquent, nous identifions Σ^* et T_Σ.

Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{S}, \Omega, Val)$ un *WSA* de rang n. Nous voulons construire un automate de Büchi B qui reconnaît les arbres caractéristiques des n-uplets de langages reconnus par $\mathcal{A}$. Soit r^w un calcul dans $\mathcal{A}$. r^w est une application qui à tout élément m de Σ^* associe un état $r^w(m)$ de $\mathcal{Q}$.

Il est facile de construire un automate de Büchi et un calcul r^b qui simulent r^w, c'est-à-dire tel que $r^b(m) = r^w(m)$ pour tout m. Mais la difficulté repose essentiellement sur la définition des conditions d'acceptation. Il faut transformer une condition globale ($r^w(T_\Sigma) \in \Omega$) en une condition sur chaque chemin.

Le comportement d'un calcul de B sur t le long d'un chemin π, i.e. $r^b(t \mid \pi)$, est mis en parallèle avec le comportement d'un calcul r^w dans $\mathcal{A}$ sur l'ensemble des mots correspondants de π ($r^w(f(\pi))$.

L'idée ici est de deviner dès le début du calcul r^b de B, l'ensemble ω des états atteints par r^w, ainsi que l'ensemble des états rencontrés sur chaque chemin. Pour un mot m donné, l'état $r^b(m)$ doit donc contenir cet ensemble ω, mais aussi l'ensemble des états de ω qui doivent encore être atteints à partir de m (ce que nous notons ω_+) et l'état $r^w(m)$.

Construction.

Soit $B = (Q^b, Q_0^b, \Delta^b, F^b)$ un automate de Büchi sur $\{0,1\}^n$ avec :

- $Q^b = \{(\omega_+, \omega, \mathbf{q}) \mid \omega \in \Omega,\ \omega_+ \subseteq \omega,\ \mathbf{q} \in \omega\}$;
- $Q_0^b = \{(\omega, \omega, \mathbf{q}) \in Q^b \mid \exists \varepsilon \rightarrow \mathbf{q} \in \mathcal{S}\}$;
- $\Delta^b \subseteq Q^b \times \{0,1\}^n \times Q^b \times Q^b$ est un ensemble des règles qui vérifie :

$$((\omega_+, \omega, \mathbf{q}), l, (\omega_1, \omega, \mathbf{q_1}), (\omega_2, \omega, \mathbf{q_2})) \in \Delta^b$$
$$\Leftrightarrow$$
$$\begin{cases} \omega_+ \setminus \{\mathbf{q}\} = \omega_1 \cup \omega_2; \\ \exists a(\mathbf{q}) \rightarrow \mathbf{q_1} \in \mathcal{S}; \\ \exists b(\mathbf{q}) \rightarrow \mathbf{q_2} \in \mathcal{S}; \\ l = Val(\mathbf{q}); \end{cases}$$

- $F^b = \{(\emptyset, \omega, \mathbf{q}) \in Q^b\}$.

Correction de la construction.

Soit p la projection de Q^b sur $\mathcal{Q}$.

$$\begin{aligned} p : Q^b &\rightarrow \mathcal{Q} \\ (\omega_0, \omega, \mathbf{q}) &\mapsto \mathbf{q} \end{aligned}$$

— Montrons que $\mathcal{L} \in \mathcal{L}(\mathcal{A}) \Leftarrow t_{\mathcal{L}} \in L(B)$.

Soit r^b un calcul réussi dans l'automate B sur l'arbre t. Soit r^w tel que:

$$\forall m \in \Sigma^* \; r^w(m) = p(r^b(m)).$$

Par construction, r^w est une application de Σ^* dans $\mathcal{Q}$ compatible avec les règles de $\mathcal{S}$. En effet, r^b est compatible avec les règles de Δ^b, donc,

$$\begin{aligned} & r^b(\varepsilon) \in Q_0^b \ , \\ & \forall m \in \Sigma^* \; (r^b(m), t(m), r^b(ma), r^b(mb)) \in \Delta^b \ , \end{aligned}$$

et, par construction,

$$\begin{aligned} & \exists \, \varepsilon \rightarrow p(r^b(\varepsilon)) \in \mathcal{S} \ , \\ & \forall m \in \Sigma^* \begin{cases} \exists \, a(p(r^b(m))) \rightarrow p(r^b(ma)) \in \mathcal{S} \\ \exists \, b(p(r^b(m))) \rightarrow p(r^b(mb)) \in \mathcal{S} \ . \end{cases} \end{aligned}$$

Soit,

$$\begin{aligned} & \exists \, \varepsilon \rightarrow p(r^b(\varepsilon)) \in \mathcal{S} \ , \\ & \forall m \in \Sigma^* \begin{cases} \exists \, a(r^w(m)) \rightarrow r^w(a.m) \in \mathcal{S} \\ \exists \, b(r^w(m)) \rightarrow r^w(b.m) \in \mathcal{S} \ . \end{cases} \end{aligned}$$

Par définition de Δ^b, chaque état de $r^b(\Sigma^*)$ est un triplet dont la deuxième composante est un ensemble unique ω. Prouvons que r^w est un calcul réussi et que $r^w(\Sigma^*) = \omega$.

D'une part, $r(\Sigma^*) \subseteq \omega$ car

$$\begin{aligned} & ((\omega_+, \omega, \mathbf{q}), l, (\omega_1, \omega, \mathbf{q_1}), (\omega_2, \omega, \mathbf{q_2})) \in \Delta^b \\ & \qquad \Rightarrow \mathbf{q}, \mathbf{q_1}, \mathbf{q_2} \in \omega \ . \end{aligned}$$

Donc

$$\forall m \in \Sigma^* \; r^w(m) \in \omega.$$

D'autre part, d'après les conditions d'acceptation de Büchi, une infinité d'états de la forme $(\emptyset, \omega, \mathbf{q})$ sont rencontrés sur chaque chemin. Mais si une transition $((\omega_+, \omega, \mathbf{q}), l, (\omega_1, \omega, \mathbf{q_1}), (\omega_2, \omega, \mathbf{q_2}))$ est dans Δ^b , alors $\omega_+ \setminus \{\mathbf{q}\} = \omega_1 \cup \omega_2$. En d'autres termes, un état n'est écarté que lorsqu'il est effectivement rencontré. Pour

que sur chaque branche existe un (une infinité d') état $(\emptyset, \omega, \mathbf{q})$, il est nécessaire que chaque état de ω soit rencontré au moins une fois.

Enfin, t est l'arbre caractéristique de $(L_1, \ldots, L_n)$. Puisque

$$((\omega_+, \omega, \mathbf{q}), l, (\omega_1, \omega, \mathbf{q_1}), (\omega_2, \omega, \mathbf{q_2})) \in \Delta^b \\ \Rightarrow l = Val(\mathbf{q}) \ ,$$

alors,

$$\forall m \in \Sigma^* \ r^b(m) = (\omega_+, \omega, \mathbf{q}) \Rightarrow Val(\mathbf{q}) = t(m) \ .$$

Donc,

$$\forall m \in \Sigma^* \ Val(r^w(m)) = Val(p(r^b(m))) = t(m).$$

Par la définition des langages reconnus par un *WSA*, nous déduisons

$$t_{(L_1,\ldots,L_n)} \in L(B) \Rightarrow (L_1, \ldots, L_n) \in \mathcal{L}(\mathcal{A}).$$

— Montrons que $\mathcal{L} \in \mathcal{L}(\mathcal{A}) \Rightarrow t_{\mathcal{L}} \in L(B)$.

Soit r^w un calcul réussi dans $\mathcal{A}$. Soit $\mathcal{L}$ le langage accepté par r^w, et $\omega = r^w(\Sigma^*)$. Nous allons construire un calcul réussi r^b dans B sur l'arbre $t_{\mathcal{L}}$ de la façon suivante :

> Soit $\omega_{<m} = \{r^w(v) \mid v \lhd m\}$ et $\omega_{\geq m} = \{r^w(v) \mid m \trianglelefteq v\}$ pour tout nœud m de Σ^*.
>
> Alors, soit r^b défini par
>
> $$\forall m \in \Sigma^* \ r^b(m) = (\omega_{\geq m} \setminus \omega_{<m}, \omega, r^w(m)).$$

1. Montrons que r^b est un calcul de B, c'est-à-dire que r^b est compatible avec les règles de Δ^b. Premièrement, $r^b(\varepsilon) = (\omega, \omega, r^w(\varepsilon))$ est un état de Q'_0 car, il existe une règle $\varepsilon \longrightarrow r^w(\varepsilon)$ dans $\mathcal{S}$ et l'ensemble ω est dans Ω.

   Deuxièmement, pour tout mot m de Σ^*, $(r^b(m), t_{\mathcal{L}}(m), r^b(m.a), r^b(m.b))$ est une règle de Δ^b. En effet, soit $m \in \Sigma^*$. Alors,

   $$r^b(m) = (\omega_{\geq m} \setminus \omega_{<m}, \omega, r^w(m)).$$

   Les deux nœuds $m.a$ et $m.b$ sont étiquetés dans le calcul r^b par

   $$r^b(m.a) = (\omega_{\geq m.a} \setminus \omega_{<m.a}, \omega, r^w(m.a)) \\ r^b(m.b) = (\omega_{\geq m.b} \setminus \omega_{<m.b}, \omega, r^w(m.b)).$$

   Maintenant, puisque

   - $[\omega_{\geq m} \setminus \omega_{<m}] \setminus \{r^b(m)\} = (\omega_{\geq m.a} \setminus \omega_{<m.a}) \cup (\omega_{\geq m.b} \setminus \omega_{<m.b})$;

- $\exists a(r^w(m)) \rightarrow r^w(m.a) \in \mathcal{S}$ et $\exists b(r^w(m)) \rightarrow r^w(m.b)) \in \mathcal{S}$;
- $\{r^w(m.a), r^w(m.b)\} \in \omega_{\geq m} \setminus \omega_{<m}$;
- $Val(r^w(m)) = t_{\mathcal{L}}(m)$.

alors, il existe une règle $(r^b(m), t_{\mathcal{L}}(m), r^b(m.a), r^b(m.b))$ dans Δ^b.

2. Il reste à montrer que r^b est un calcul réussi. Ceci se déduit directement de la finitude de l'ensemble ω. Il existe donc un arbre fini de domaine inclus dans Σ^* tel que la restriction de r^w à cet arbre soit ω. Par conséquent, à chaque nœud m n'appartenant pas à cet arbre fini, correspond un état $r^b(m)$ dont la première composante est nulle. En effet, pour de tels mots m, l'ensemble $\omega_{<m}$ contient l'ensemble $\omega_{\geq m}$. Donc, la différence $\omega_{\geq m} \setminus \omega_{<m}$ est $\emptyset$.

   Un nombre infini d'états de la forme $(\emptyset, \omega, \mathbf{q})$ apparaît donc le long de chaque chemin, et le calcul est réussi. □

Comme nous l'avons annoncé, la condition de réussite des calculs dans les automates d'ensembles de mots est moins puissante qu'une condition "à la Büchi".

Proposition 2.3

- *La classe des langages WSA-reconnaissables est une sous-classe stricte de la classe des langages Büchi-reconnaissables.*
- *La classe des langages WSA-reconnaissables n'est pas close par complémentarité.*

Preuve. Soit $\mathcal{L}$, l'ensemble des langages L sur l'alphabet $\Sigma = \{a\}$ qui vérifient : $\exists k$ tel que $\forall n > k$, $a^n \in L$. Ces langages L sont aussi définis par la formule monadique :

$$\exists X \ (\forall x \ (x \in X) \Rightarrow (a(x) \in X \wedge x \in L)) \wedge (\exists x \ x \in X). \tag{2.4}$$

L'ensemble $\mathcal{L}$ est *WSA*-reconnaissable et (donc) Büchi-reconnaissable. Par contre, l'ensemble des langages L' qui ne sont pas dans $\mathcal{L}$ est Büchi-reconnaissable mais non *WSA*-reconnaissable.

Fait 2.1 *$\mathcal{L}$ est WSA-reconnaissable et Büchi-reconnaissable.*

Preuve. Le *WSA* $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ avec

- $\mathcal{Q} = \{\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_f\}$,
- $\mathcal{F} = \{\mathbf{q}_1, \mathbf{q}_f\}$,

- $\Omega = \{\omega \mid \mathbf{q}_f \in \omega\}$,
- $\mathcal{S}$ l'ensemble

$$\begin{array}{rcl} \varepsilon & \rightarrow & \mathbf{q}_0 \mid \mathbf{q}_1 \mid \mathbf{q}_f \; ; \quad a(\mathbf{q}_0) \rightarrow \mathbf{q}_0 \mid \mathbf{q}_1 \mid \mathbf{q}_f \\ a(\mathbf{q}_1) & \rightarrow & \mathbf{q}_0 \mid \mathbf{q}_1 \mid \mathbf{q}_f \; ; \quad a(\mathbf{q}_f) \rightarrow \mathbf{q}_f \; . \end{array}$$

reconnaît $\mathcal{L}$.

Le langage $\mathcal{L}(\mathcal{A})$ est non vide car le calcul r_0 défini par $r_0(m) = q_f$ pour tout m est réussi, et $L(\mathcal{A}, r_0) = \Sigma^*$.

$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}$. Pour tout calcul réussi r, il existe un mot m tel que $r(m) = q_f$. D'après les règles, r applique tous les mots u tels que $m \trianglelefteq u$ sur q_f. Il existe alors $X = \{u \mid m \trianglelefteq u\}$ tel que

$$\forall x \; (x \in X) \Rightarrow (a(x) \in X \wedge x \in L(\mathcal{A}, r)) \wedge \exists x \; x \in X$$

De plus, $\mathcal{L} \subseteq \mathcal{L}(\mathcal{A})$. Pour tout L de $\mathcal{L}$, nous construisons le calcul r_L avec

$$\begin{aligned} r_L(t) = q_0 &\Leftrightarrow t \notin L \\ r_L(t) = q_1 &\Leftrightarrow (t \notin X \wedge t \in L) \\ r_L(t) = q_f &\Leftrightarrow (t \in X \wedge t \in X) \end{aligned}$$

□

Le langage $\overline{\mathcal{L}} = \{L' \mid L' \notin \mathcal{L}\}$, c'est-à-dire, le langage $\overline{\mathcal{L}} = \{L \mid \forall k \; \exists n \; n > k \wedge a^n \notin L\}$ est reconnaissable par l'automate de Büchi $\bar{B} = (\{0,1\}, Q, q_0, S, F)$ avec

- $Q = \{q_0, q_1\}$
- $F = \{q_0\}$
- $S = \{(q_0, 0, q_0), (q_0, 1, q_1), (q_1, 0, q_0), (q_1, 1, q_1)\}$.

Mais,

Fait 2.2 *$\overline{\mathcal{L}}$ n'est pas WSA-reconnaissable.*

Preuve. Prouvons le fait par l'absurde. Soit $\overline{\mathcal{A}}$ le *WSA* tel que $\mathcal{L}(\overline{\mathcal{A}}) = \overline{\mathcal{L}}$. Supposons que le nombre d'états de $\mathcal{Q}$ soit $k-1$.

Considérons le langage L de mot caractéristique w_L défini par

$$\begin{aligned} w_L(a^n) &= 0 \text{ si } n \text{ est multiple de } k \\ w_L(a^n) &= 1 \text{ sinon} \end{aligned}$$

Ce langage est dans $\overline{\mathcal{L}}$.

Montrons que L n'est accepté par aucun calcul de $\overline{\mathcal{A}}$. Pour cela, supposons qu'il existe r, un calcul réussi et que $L(\mathcal{A}, r) = L$. Soit $\omega = r(\Sigma^*)$. Il existe m tel que $r(\Sigma^{\leq m}) = \omega$ et tel que m soit un multiple de k.

Maintenant, puisqu'il existe $k-1$ états dans Q, alors obligatoirement, il existe deux mots différents u et v de $\{a^{m+1}, \ldots, a^{m+k-1}\}$ tels que $r(u) = r(v)$.

Il est donc possible de répéter ultimement les applications de règles effectuées entre u et v, et de décrire de la sorte, un calcul réussi. Soit r' tel que $r'(a^n) = r(a^n)$ pour $n \leq m$ et $r'(a^n)$ est un état final pour $n > m$. Ce calcul r' reconnaît le langage L' de mot caractéristique $w_{L'}(a^n) = w_L(a^n)$ si $n \leq m$ et $w_{L'}(a^n) = 1$ si $n > m$.

w_L est le mot caractéristique d'un langage de $\mathcal{L}$, ce qui contredit les hypothèses. □

2.5 Introduction au Cas Général

Le cas général est le cas où les contraintes portent sur des ensembles de termes. C'est par exemple répondre à la question : existe-t-il des couples de langages d'arbres (X, Y) qui satisfont à la contrainte $b(X, \sim Y) \subseteq X \cap Y$?

La première constatation est que le même procédé de résolution est mis en œuvre pour de tels systèmes. Les automates d'ensembles de mots sont étendus de façon à reconnaître des ensembles d'arbres.

Pour cette classe d'automates, on peut résoudre le problème du vide qui consiste à décider si il existe un langage d'arbres accepté par l'automate, et donc décider de la satisfiabilité de la contrainte. La satisfiabilité des contraintes étant réduite au problème de décision du vide dans les automates, on s'aperçoit que ce résultat n'est simple que lorsque la contrainte ne comporte pas de symboles de non inclusion $\not\subseteq$.

Enfin, la procédure pourra facilement être décomposée en traitant indépendamment les différentes contraintes du système. La correction de cette modularité est assurée par des propriétés de clôture des automates (voir la section 3.4).

Dans le chapitre 3, nous étudions plus précisément ces nouveaux automates. Les résultats obtenus pourront le plus souvent être compris comme des résultats sur les systèmes de contraintes ensemblistes.

Chapitre 3

Automates d'Ensembles d'Arbres

3.1 Définitions

Les automates d'ensembles de mots sont une sous-classe des automates d'ensembles d'arbres. Les *WSA* sont des reconnaisseurs d'arbres filiformes. L'alphabet avec lequel ils évoluent est composé d'une seule constante ε, et de lettres d'arité 1. Dans cette extension, les automates manipulent des termes sur un alphabet gradué quelconque.

Nous avons défini un *WSA* $\mathcal{A}$ comme un quintuplet $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{S}, \Omega, Val)$, avec *Val* une application de $\mathcal{Q}$ dans $\{0,1\}^n$, pour faire le lien avec les automates d'arbres infinis de Rabin. La définition des *TSA* diffère légèrement puisque *Val* est remplacée par un n-uplet d'ensembles d'états finaux. Nous pensons que cette présentation est plus parlante.

Définition 3.1 Un Automate d'Ensembles d'Arbres est la donnée d'un alphabet gradué Σ fini, d'un ensemble fini d'états $\mathcal{Q}$, d'un n-uplet d'ensembles d'états finaux $\mathcal{F} = (F_1, \ldots, F_n)$ où chaque $F_i \subseteq \mathcal{Q}$, d'un ensemble fini d'ensembles acceptant $\Omega \subseteq 2^{\mathcal{Q}}$, et d'un ensemble fini de règles de transition $\mathcal{S} \subseteq \bigcup_{j=0}^{amax} \mathcal{Q}^j \times \Sigma_j \times \mathcal{Q}$.

Les règles de $\mathcal{S}$ sont écrites sous la forme :

$$b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \rightarrow \mathbf{q} \qquad \text{pour } \mathbf{b} \in \Sigma_p \text{ et } \mathbf{q}_1, \ldots, \mathbf{q}_p, \mathbf{q} \in \mathcal{Q}.$$

Définition 3.2 Un automate d'ensembles d'arbres est dit :

- de *Rang n* si $\mathcal{F} = (F_1, \ldots, F_n)$.
- *Déterministe* si il n'existe pas deux règles de $\mathcal{S}$ de même membre gauche.

- *Complet* si :

$$\forall p \in \{0, \ldots, \text{amax}\} \ \forall b \in \Sigma_p \ \forall \mathbf{q}_1, \ldots, \mathbf{q}_p \in \mathcal{Q} \quad \exists b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \rightarrow \mathbf{q} \in \mathcal{S}.$$

- *Sans condition d'acceptation* si $\Omega = 2^{\mathcal{Q}}$.

Deux règles de même membre gauche $b(\mathbf{q}) \rightarrow \mathbf{q}'$ et $b(\mathbf{q}) \rightarrow \mathbf{q}''$ sont notées en abrégé $b(\mathbf{q}) \rightarrow \mathbf{q}' \mid \mathbf{q}''$.

Exemple 3.1 Soit Σ un alphabet gradué composé d'une constante a et d'une lettre b d'arité 2. Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ avec :

- $\mathcal{Q} = \{\mathbf{q}, \mathbf{q}', \mathbf{q}''\}$;
- $\mathcal{F} = (F_1, F_2)$ et $F_1 = \{\mathbf{q}, \mathbf{q}', \mathbf{q}''\}$, $F_2 = \{\mathbf{q}'\}$;
- $\Omega = 2^{\mathcal{Q}}$;
- $\mathcal{S} = \left\{ \begin{array}{rcl} a & \rightarrow & \mathbf{q} \mid \mathbf{q}'' \\ b(\mathbf{q}, \mathbf{q}) & \rightarrow & \mathbf{q} \mid \mathbf{q}' \\ b(\mathbf{q}', \mathbf{q}) & \rightarrow & \mathbf{q} \\ b(\mathbf{q}', \mathbf{q}') & \rightarrow & \mathbf{q} \\ b(\mathbf{q}, \mathbf{q}') & \rightarrow & \mathbf{q} \end{array} \right.$

$\mathcal{A}$ est un *TSA* de rang 2, non complet (car par exemple il n'existe pas de règle de membre gauche $b(\mathbf{q}, \mathbf{q}'')$), non déterministe (car il existe deux règles de membre gauche $b(\mathbf{q}, \mathbf{q})$). Finalement, $\mathcal{A}$ est sans conditions d'acceptation car $\Omega = 2^{\mathcal{Q}}$.

Remarquons qu'un calcul est défini de telle sorte que l'image d'un terme dépend de celle de tous ses sous-termes. Les calculs sont les évolutions d'une machine, sur T_Σ, qui opère suivant l'ordre sous-terme.

Définition 3.3 Un *calcul* dans un *TSA* $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ est une application r de T_Σ dans $\mathcal{Q}$ *compatible* avec les règles de $\mathcal{S}$, c'est-à-dire qu'elle vérifie la propriété suivante :

$$\forall b(t_1, \ldots, t_p) \in T_\Sigma \qquad b(r(t_1), \ldots, r(t_p)) \rightarrow r(b(t_1, \ldots, t_p)) \in \mathcal{S}. \qquad (3.1)$$

Un calcul r est dit *réussi* si $r(T_\Sigma) \in \Omega$.

L'ensemble des calculs dans $\mathcal{A}$ est noté $\mathcal{R}_\mathcal{A}$. L'ensemble des calculs réussis est noté $\mathcal{SR}_\mathcal{A}$. Si $\mathcal{A}$ est déterministe, alors il existe un calcul au plus dans $\mathcal{R}_\mathcal{A}$. Si il est déterministe et complet alors il existe exactement un calcul dans $\mathcal{R}_\mathcal{A}$.

Définition 3.4 Le *langage reconnu* par $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ de rang n est défini par :

$$\mathcal{L}(\mathcal{A}) = \{\mathcal{L}(\mathcal{A}, r) \mid r \in \mathcal{SR}_\mathcal{A}\}$$
$$\text{avec } \mathcal{L}(\mathcal{A}, r) = (L_1, \ldots L_n)$$
$$\text{où } \forall i \in \{1, \ldots n\} \quad L_i = \{t \in T_\Sigma \mid r(t) \in F_i\}.$$

Si le rang d'un *TSA* $\mathcal{A}$ est 1, alors la donnée de $\mathcal{A}$ est exactement la donnée d'un automate fini d'arbre A. La différence se situe dans la définition des calculs. Si $\mathcal{A}$ est déterministe et complet alors $\mathcal{L}(\mathcal{A})$ est le langage reconnu par A, i.e. $L(A)$. Si $\mathcal{A}$ n'est pas déterministe, alors pour tout calcul r, $\mathcal{L}(\mathcal{A}, r)$ est inclus dans $L(A)$.

On peut remarquer que les *WSA* peuvent être considérés comme des reconnaisseurs d'arbres infinis $\{0,1\}^n$ valués de domaine Σ^*, c'est-à-dire d'applications de Σ^* dans $\{0,1\}^n$. De la même façon, les *TSA* sont aussi des reconnaisseurs d'applications de T_Σ dans $\{0,1\}^n$.

Exemple 3.2

1. Soit $\mathcal{A}_0 = (\Sigma, \mathcal{Q}_0, \mathcal{F}_0, \mathcal{S}_0, \Omega_0)$ avec :
   - $\mathcal{Q}_0 = \{\mathbf{q}, \mathbf{q}', \mathbf{q}''\}$;
   - $\mathcal{F}_0 = (F_1^0, F_2^0)$ et $F_1^0 = \{\mathbf{q}, \mathbf{q}', \mathbf{q}''\}$, $F_2^0 = \{\mathbf{q}'\}$;
   - $\Omega_0 = 2^{\mathcal{Q}_0}$;
   - $\mathcal{S}_0 = \begin{cases} a & \rightarrow \mathbf{q} \\ b(\mathbf{q}, \mathbf{q}) & \rightarrow \mathbf{q} \end{cases}$

   $\mathcal{A}_0$ est un *TSA* déterministe. Il n'existe qu'un seul moyen d'effectuer un calcul dans $\mathcal{A}_0$: ce calcul r_1 applique tout terme de T_Σ sur $\mathbf{q}$.

   $$r_1(a) = \mathbf{q} \ ; \ r_1(b(a,a)) = \mathbf{q} \ ; \ r_1(b((a, b(a,a))) = \mathbf{q} \ \ldots$$

   Puisque $\mathbf{q}$ est dans F_1 mais n'est pas dans F_2, le langage reconnu par $\mathcal{A}_0$ est le singleton $\mathcal{L}(\mathcal{A}_0) = (T_\Sigma, \emptyset)$.

2. Soit $\mathcal{A}_1 = (\Sigma, \mathcal{Q}_1, \mathcal{F}_1, \mathcal{S}_1, \Omega_1)$ avec :
   - $\mathcal{Q}_1 = \mathcal{Q}_0$;
   - $\mathcal{F}_1 = \mathcal{F}_0$;
   - $\Omega_1 = \Omega_0$;
   - $\mathcal{S}_1 = \begin{cases} a & \rightarrow \mathbf{q} \mid \mathbf{q}'' \\ b(\mathbf{q}, \mathbf{q}) & \rightarrow \mathbf{q} \end{cases}$

   Aucune application qui associe l'état $\mathbf{q}''$ à la constante a n'est un calcul. Il n'est pas possible de calculer les images des autres termes de T_Σ car il n'existe pas de règle de membre gauche comportant l'état $\mathbf{q}''$. Par conséquent $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_0)$.

3. Soit $\mathcal{A}_2 = (\Sigma, \mathcal{Q}_2, \mathcal{F}_2, \mathcal{S}_2, \Omega_2)$ avec :
   - $\mathcal{Q}_2 = \mathcal{Q}_0$;

- $\mathcal{F}_2 = \mathcal{F}_0$;
- $\Omega_2 = \Omega_0$;
- $\mathcal{S}_2 = \left\{ \begin{array}{rcl} a & \rightarrow & \mathbf{q} \mid \mathbf{q}'' \\ b(\mathbf{q},\mathbf{q}) & \rightarrow & \mathbf{q} \mid \mathbf{q}' \\ b(\mathbf{q}',\mathbf{q}) & \rightarrow & \mathbf{q} \\ b(\mathbf{q}',\mathbf{q}') & \rightarrow & \mathbf{q} \\ b(\mathbf{q},\mathbf{q}') & \rightarrow & \mathbf{q} \end{array} \right.$

Mis à part le choix posé dès la règle *initiale*, c'est-à-dire la règle dont le membre gauche est une constante, le seul non déterminisme apparaît lorsque l'on veut calculer l'image d'un terme à l'aide de l'une des deux règles $b(\mathbf{q},\mathbf{q}) \rightarrow \mathbf{q} \mid \mathbf{q}'$.

La stratégie qui consiste à toujours sélectionner la règle $b(\mathbf{q},\mathbf{q}) \rightarrow \mathbf{q}$ définit un calcul r_1 qui applique chaque terme de T_Σ sur $\mathbf{q}$.

$$\mathcal{L}(\mathcal{A}_2, r_1) = (T_\Sigma, \emptyset).$$

Maintenant, choisir d'utiliser $b(\mathbf{q},\mathbf{q}) \rightarrow \mathbf{q}'$ dès que cela est possible définit un calcul r_2

$$r_2(a) = \mathbf{q} \ ; \ r_2(b(a,a)) = \mathbf{q}' \ ; \ r_2(b(a,b(a,a))) = \mathbf{q} \ \ldots$$

qui applique un terme $b(t,t')$ sur $\mathbf{q}'$ si t et t' sont d'image $\mathbf{q}$ et sur $\mathbf{q}'$ sinon. Du point de vue des langages acceptés $(L_1(\mathcal{A}_2, r_2), L_2(\mathcal{A}_2, r_2))$, un terme $b(t,t')$ est à la fois dans $L_1(\mathcal{A}_2, r_2)$ et $L_2(\mathcal{A}_2, r_2)$ si et seulement si ses deux sous-termes directs t et t' ne sont pas dans $L_2(\mathcal{A}_2, r_2)$. Donc,

$$\mathcal{L}(\mathcal{A}_2, r_2) = (T_\Sigma, L)$$
$$\text{avec } (b(t,t') \in L) \Leftrightarrow (t \notin L \wedge t' \notin L) \ .$$

Le langage reconnu par $\mathcal{A}_2$ est un ensemble de couples de forêts.

$$\mathcal{L}(\mathcal{A}_2) = \{(T_\Sigma, L) \mid b(t,t') \in L \Rightarrow (t \notin L \wedge t' \notin L)\} \ .$$

4. En ajoutant des conditions d'acceptation, l'ensemble des langages reconnu par $\mathcal{A}$ est réduit car tous les calculs ne sont plus des calculs réussis. Par exemple :

   Soit $\mathcal{A}_3 = (\Sigma, \mathcal{Q}_3, \mathcal{F}_3, \mathcal{S}_3, \Omega_3)$ avec :

   - $\mathcal{Q}_3 = \mathcal{Q}_0$;
   - $\mathcal{F}_3 = \mathcal{F}_0$;

- $\Omega_3 = \{\{\mathbf{q}, \mathbf{q}'\}\}$;
- $\mathcal{S}_3 = \mathcal{S}_2$.

Le premier calcul r_1 n'est pas réussi. En effet, $r_1(T_\Sigma) = \{\mathbf{q}\} \notin \Omega_3$. Le langage reconnu devient :

$$\mathcal{L}(\mathcal{A}_3) = \{(T_\Sigma, L) \mid (L \neq \emptyset) \wedge (b(t,t') \in L \Rightarrow (t \notin L \wedge t' \notin L))\} \ .$$

Tout au long de ce mémoire nous adopterons les notations et le vocabulaire suivants :

- Les Automates d'Ensembles d'Arbres sont notés en abrégé *TSA* pour *Tree Set Automata*.
- Un *TSA* *reconnaît* un ensemble de n-uplets de langages d'arbres qui est l'ensemble des n-uplets *acceptés* par ses calculs réussis.
- Les lettres calligraphiées désignent des objets qui se rapportent à des *TSA* ou des n-uplets de langages d'arbres. Exemples : $\mathcal{A}$ est un *TSA* d'états dans $\mathcal{Q}$ ... ; $\mathcal{L}(\mathcal{A},r)$ est le n-uplet de langages d'arbres accepté par le calcul r ...
- Les états des *TSA* sont représentés en général par les lettres $\mathbf{q}$, $\mathbf{q}_1$, $\mathbf{q}_n$, $\mathbf{s}$, $\mathbf{s}_1$, $\mathbf{s}_n$,... en caractères **gras**.
- Nous dirons d'une application de T_Σ dans $\mathcal{Q}$ qu'elle est *compatible avec les règles de* $\mathcal{S}$ si elle vérifie la propriété 3.1.
- La classe des ensembles reconnus par les *TSA* est la classe $\mathcal{TSA}$.

3.2 Décision du Vide

Dans toute la suite, $\mathcal{A}$ désigne un Automate d'Ensembles d'Arbres $(\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$. Rappelons que décider du vide pour l'automate $\mathcal{A}$, c'est décider si le langage reconnu par $\mathcal{A}$ est vide ou non. Cela revient encore à décider de l'existence d'un calcul réussi dans $\mathcal{A}$. Comme nous allons le voir, c'est la condition de succès qui rend cette preuve difficile. En résumé, pour vérifier la condition de succès d'un calcul r de $\mathcal{A}$, il faut vérifier que l'image de T_Σ par r est un ensemble d'états désignés de l'ensemble Ω. Puisque T_Σ est un ensemble infini, et à cause du non déterminisme des règles de $\mathcal{S}$, l'accessibilité d'un état dans un calcul peut être repoussée arbitrairement loin. Par exemple, dans l'automate dont les règles sont les suivantes :

$$\begin{aligned} a &\rightarrow \mathbf{q} \\ b(\mathbf{q}) &\rightarrow \mathbf{q} \mid \mathbf{q}' \\ b(\mathbf{q}') &\rightarrow \mathbf{q} \mid \mathbf{q}' \end{aligned}$$

l'état $\mathbf{q}'$ peut être atteint la première fois pour des termes de hauteur non bornée. Le résultat est obtenu par l'intermédiaire d'un lemme clé qui montre qu'il suffit d'examiner tous les calculs possibles sur des forêts de taille bornée pour s'assurer de l'existence d'un calcul réussi.

Cette section est dédiée à la preuve du théorème:

Théorème 3.1 *Le problème du vide est décidable dans la classe des Automates d'Ensembles d'Arbres.*

En introduction, nous prouvons facilement que lorsque les conditions d'acceptation sont effacées, c'est-à-dire lorsque Ω est l'ensemble de toutes les parties de $\mathcal{Q}$, ce problème de décision est rapidement résolu.

Remarque: Nous avons choisi de présenter les exemples qui illustrent cette preuve dans le cas des arbres unaires i.e. le cas des *WSA*. Dans le cas de lettres d'arité supérieure, l'explosion du nombre de règles rend l'illustration par l'exemple trop confuse. Nous devons préciser que la preuve du vide dans la classe des *WSA* est plus facile à comprendre et moins complexe, parce que la notion de partage des termes ou encore de concurrence dans l'application des règles n'existe pas.

3.2.1 *TSA* sans Conditions d'Acceptation

Lorsque l'ensemble Ω est $2^{\mathcal{Q}}$, l'ensemble de toutes les parties de $\mathcal{Q}$, la condition de succès d'un calcul disparaît. La question de la décision du vide se réduit alors à celle de l'existence d'un calcul, puisque tout calcul est un calcul réussi. Il faut donc contrôler l'existence ou la non existence d'une application de T_Σ dans $\mathcal{Q}$ compatible avec les règles de $\mathcal{S}$.

Connaissant le début *dc* d'un calcul sur une forêt finie close par sous-terme, savoir si *dc* ne pourra être poursuivi c'est détecter les impasses, détecter les termes auxquels on ne peut associer d'image: *dc* atteint-il des états $\mathbf{q}_1, \ldots, \mathbf{q}_p$ pour lesquels il existe un symbole b d'arité p tel que $b(\mathbf{q}_1, \ldots, \mathbf{q}_p)$ ne soit le membre gauche d'aucune règle?

La condition COND(ω) formalise l'idée qu'en atteignant uniquement les états de ω, on évite les impasses.

Plus formellement, si ω est un sous-ensemble de $\mathcal{Q}$, appelons COND(ω) la condition suivante:

$$\text{COND}(\omega) \equiv \forall b \in \Sigma_p \quad \forall \mathbf{q}_1, \ldots, \mathbf{q}_p \in \omega \quad \exists \mathbf{q} \in \omega \quad b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \rightarrow \mathbf{q} \in \mathcal{S}$$

Nous allons maintenant prouver qu'il existe un sous-ensemble d'états ω pour lequel COND(ω) est vrai si et seulement si il existe un calcul dans $\mathcal{A}$. Nous pourrons alors facilement en déduire que l'existence d'un calcul est décidable. En

effet, l'ensemble $\mathcal{Q}$ est fini. Il existe donc un nombre fini de sous-ensembles ω et la condition COND(ω) est calculable.

Lemme 3.1 *Il existe $\omega \subseteq \mathcal{Q}$ tel que* COND(ω) *si et seulement si il existe un calcul dans $\mathcal{A}$.*

Preuve. La preuve est directe dans le sens $\Leftarrow$. En effet, Soit r un calcul dans $\mathcal{A}$, soit $\omega = r(T_\Sigma)$. Montrons que COND(ω) est vrai. Puisque r est un calcul d'image ω, nous avons :

$$\forall \mathbf{q}_1, \ldots, \mathbf{q}_p \in \omega, \exists t_1, \ldots, t_p \text{ tels que } r(t_1) = \mathbf{q}_1, \ldots, r(t_p) = \mathbf{q}_p \ .$$

Soient b une lettre d'arité p, $t = b(t_1, \ldots, t_p)$ et $\mathbf{q} = r(t)$, r est compatible avec les règles de $\mathcal{S}$:

$$\forall b \in \Sigma_p, b(r(t_1), \ldots, r(t_p)) \rightarrow r(t) \in \mathcal{S} \ .$$

Donc,

$$\forall b \in \Sigma_p \quad \forall \mathbf{q}_1, \ldots, \mathbf{q}_p \in \omega \quad \exists \mathbf{q} \in \omega \quad b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \rightarrow \mathbf{q} \in \mathcal{S} \ .$$

Pour montrer la réciproque, nous prouvons par induction sur la structure des termes de T_Σ qu'il existe un calcul r d'image incluse dans ω si COND(ω) est vrai.

Base Vérifions qu'il existe une application des constantes de Σ sur $\mathcal{Q}$, compatible avec $\mathcal{S}$. D'après COND(ω), $\forall a \in \Sigma_0$, $\exists \mathbf{q} \in \omega \quad a \rightarrow \mathbf{q} \in \mathcal{S}$. On peut donc construire une telle application r en posant $r(a) = \mathbf{q}$.

Induction Soit $t = b(t_1, \ldots, t_p)$ et supposons le lemme vérifié pour le p-uplet de termes $t_1, \ldots, t_p$.

Par les hypothèses d'induction, il existe r une application et $\mathbf{q}_1, \ldots, \mathbf{q}_p \in \omega$ tels que, pour tout i de $\{1, \ldots, p\}$, $r(t_i) = \mathbf{q}_i$. Puisque COND(ω) est satisfaite, nous avons :

$$\forall b \in \Sigma_p \quad \forall \mathbf{q}_1, \ldots, \mathbf{q}_p \in \omega \quad \exists \mathbf{q} \in \omega \quad b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \rightarrow \mathbf{q} \in \mathcal{S} \ .$$

Nous posons $r(b(t_1, \ldots, t_p)) = \mathbf{q}$ et le lemme est vérifié.

Nous concluons alors qu'il existe un calcul r d'image $r(T_\Sigma) \subseteq \omega$. □

Proposition 3.1 *L'existence d'un calcul dans un TSA est décidable, ou encore, le problème du vide est décidable pour la classe des TSA sans condition d'acceptation.*

Preuve. La preuve est directe par le lemme précédent puisque la condition COND(ω) est calculable et le nombre de sous-ensembles de $\mathcal{Q}$ est fini. □

3.2.2 Notations et Définitions

Soit E un ensemble fini. Un multi-ensemble fini A est une application de E dans $\mathbb{N}$. La représentation souvent choisie est celle d'ensemble avec répétition des éléments. Par exemple, on note $A = \{a, a, b, b, b\}$ ou encore $A = \{a : 2, b : 3\}$ le multi-ensemble sur $\{a, b\}$ tel que $A(a) = 2$ et $A(b) = 3$.

Soient A et A' deux multi-ensembles d'éléments de E.

- La somme, notée $A + A'$ est le multi-ensemble défini par :

$$\forall e \in E \quad (A + A')(e) = A(e) + A'(e).$$

- La différence, notée $A - A'$ est le multi-ensemble défini par :

$$\forall e \in E \quad (A - A')(e) = \max(0, A(e) - A'(e)).$$

- L'union, notée $A \cup A'$ est l'ensemble défini par :

$$\forall e \in E \quad e \in A \cup A' \Leftrightarrow (A + A')(e) \neq 0.$$

- L'intersection, notée $A \cap A'$ est l'ensemble défini par :

$$\forall e \in E \quad e \in A \cap A' \Leftrightarrow (A(e) \neq 0 \text{ et } A'(e) \neq 0).$$

- Les relations d'inclusion et d'égalité sont de type "ensembliste" ou "multi-ensembliste" :

$$\left.\begin{array}{l} A \subseteq_m A' \Leftrightarrow \forall e \in E \quad A(e) \leq A'(e) \\ A =_m A' \Leftrightarrow (A \subseteq_m A') \wedge (A' \subseteq_m A) \end{array}\right\} \text{relations multi-ensemblistes}$$

$$\left.\begin{array}{l} A \subseteq_s A' \Leftrightarrow \forall e \in E \quad A(e) > 0 \Rightarrow A'(e) > 0 \\ A =_s A' \Leftrightarrow (A \subseteq_s A') \wedge (A' \subseteq_s A) \end{array}\right\} \text{relations ensemblistes}$$

- La cardinalité d'un multi-ensemble est :

$$Card(A) = \sum_{e \in E} A(e)$$

Par la suite, nous dirons aussi $e \in A$, si $A(e) \neq 0$.

Rappelons que t est un sous-terme de u est noté $t \trianglelefteq u$. Une forêt F sera dite *close* si elle est close pour l'ordre $\trianglelefteq$. C'est-à-dire, F est close si et seulement si $\forall s \in T_\Sigma \forall t \in F \ (s \trianglelefteq t) \Rightarrow (s \in F)$. La *cardinalité* d'une forêt F est le nombre de termes de F.

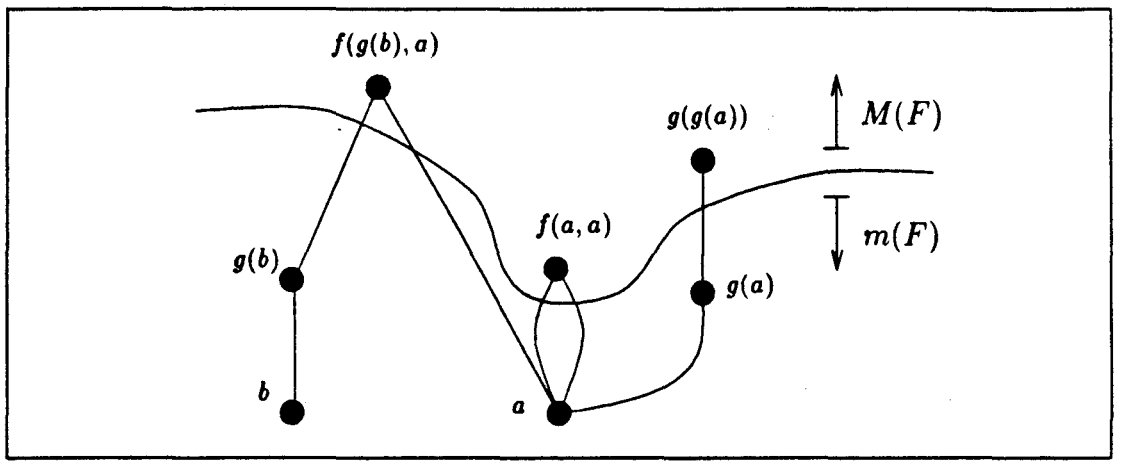


FIG. 3.1 - : Représentation d'un ensemble F clos par sous-terme et des ensembles $m(F)$ et $M(F)$

L'ensemble $M(F)$ est l'ensemble des termes maximaux selon l'ordre $\trianglelefteq$ de la forêt F.

$$M(F) = \{t \in F \mid \forall u \in F \quad \neg(t \trianglelefteq u)\}.$$

L'ensemble des termes non maximaux de F est $m(F)$:

$$m(F) = F \setminus M(F) = \{t \in F \mid t \notin M(F)\}.$$

Soit F une forêt close et t un terme de F. Alors

$$Sup_F(t) = \{u \in F \mid t \trianglelefteq u \wedge u \neq t\}.$$

Nous définissons aussi la fonction *Inf* par

$$Inf(b(t_1, \ldots, t_p)) = \{t_1, \ldots, t_p\}.$$

Exemple 3.3 Soit $F = \{a, b, g(b), f(a,a), g(a), f(g(b),a), g(g(a))\}$. Alors F est une forêt close, et

$$\begin{aligned} M(F) &= \{f(a,a), f(g(b),a), g(g(a))\} \\ m(F) &= \{a, b, g(b), g(a)\} \\ Sup_F(b) &= \{g(b), f(g(b),a)\} \\ Inf(f(g(b),a)) &= \{a, g(b)\} \ . \end{aligned}$$

Un terme t est *obtenu* dans un calcul r, par une règle s d'un automate d'ensembles d'arbres, si s est la dernière règle utilisée pour calculer $r(t)$. Plus formellement, $t = b(t_1, \ldots, t_p)$ est obtenu par $b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \rightarrow \mathbf{q}$ si $r(t) = \mathbf{q}$ et pour tout i de $\{1, \ldots, p\}$, $r(t_i) = \mathbf{q}_i$.

Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ un automate d'ensembles d'arbres. nous appellerons un *candidat* , toute application γ d'une forêt close F finie dans $\mathcal{Q}$ compatible avec les règles de $\mathcal{S}$, c'est-à-dire :

$$\forall t = b(t_1, \ldots, t_p) \in F \quad b(\gamma(t_1), \ldots, \gamma(t_p)) \rightarrow \gamma(b(t_1, \ldots, t_p)) \in \mathcal{S}.$$

La restriction à une forêt close d'un calcul est un candidat, mais en revanche tout candidat n'est pas la restriction d'un calcul.

Exemple 3.4 Soit un *TSA* dont les règles sont $a \rightarrow \mathbf{q} \mid \mathbf{q}'$ et $b(\mathbf{q}') \rightarrow \mathbf{q}'$. Soient deux candidats γ_1 et γ_2 définis sur $F = \{a\}$:

$$\gamma_1(a) = \mathbf{q} \text{ et } \gamma_2(a) = \mathbf{q}'.$$

Alors, γ_2 est la restriction d'un calcul à l'inverse de γ_1.

Un candidat γ défini sur une forêt F est *prolongeable* sur une forêt F' si

- $F \subseteq F'$
- il existe un candidat γ' sur F' tel que $\gamma'(F) = \gamma(F)$

Nous emploierons aussi *extension* pour prolongement.

3.2.3 Idée de la preuve

La preuve de décision du vide est longue et technique. Nous allons d'abord la présenter informellement, c'est-à-dire en donner les grandes lignes et les idées principales.

Nous insistons premièrement sur le fait que la preuve de l'existence d'un calcul ne s'étend pas directement en une preuve de l'existence d'un calcul réussi. Effectivement, vérifier COND(ω) ne suffit pas pour affirmer qu'il existe un calcul r atteignant exactement ω, c'est-à-dire dont l'image $r(T_\Sigma)$ est exactement ω. Nous pouvons illustrer ce fait par l'exemple suivant :

Exemple 3.5 Soit le *TSA* $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$. L'ensemble $\mathcal{S}$ est :

$$\begin{gathered} a \rightarrow \mathbf{q}_1 \\ b(\mathbf{q}_1) \rightarrow \mathbf{q}_2 \mid \mathbf{q}_3 \\ b(\mathbf{q}_2) \rightarrow \mathbf{q}_2 \\ b(\mathbf{q}_3) \rightarrow \mathbf{q}_3 \end{gathered}$$

La condition COND(ω) avec $\omega = \{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3\}$ est vérifiée mais il n'existe pas de calcul atteignant exactement l'ensemble ω. Dans tout calcul de $\mathcal{A}$, seul a a pour image $\mathbf{q}_1$. Donc, une seule des deux règles de membre gauche $b(\mathbf{q}_1)$ peut être choisie. Enfin, puisque dans les deux cas l'automate boucle dans le même état, $\mathbf{q}_2$ et $\mathbf{q}_3$ ne peuvent être recontrés dans un même calcul. Donc sur cet exemple, $r(T_\Sigma) = \{\mathbf{q}_1, \mathbf{q}_3\}$, ou $r(T_\Sigma) = \{\mathbf{q}_1, \mathbf{q}_2\}$.

Il existe en général dans un *TSA* un nombre infini de calculs. Néanmoins, le nombre d'images accessibles, c'est-à-dire l'ensemble $\{\omega \mid \exists r \in \mathcal{R}_{\mathcal{A}}\ r(T_\Sigma) = \omega\}$ est lui fini, puisque le nombre d'états est fini.

Nous pouvons décomposer la méthode que nous utilisons ici en deux parties. Dans un premier temps (i), l'examen d'un nombre borné de débuts de calculs, appelés *candidats* (voir leur définition page 64) donne l'ensemble de toutes les images accessibles, ensuite (ii), un critère d'extension permet de vérifier qu'un candidat peut devenir un calcul complet.

Même si la vérification de la condition COND(ω) n'est pas suffisante, elle reste importante puisqu'elle correspond à l'étape (ii).

Comme nous l'avons mentionné plus haut, toutes les images possibles de calculs peuvent être calculées en un temps fini. La partie (i) consiste à montrer que ce temps est borné par une constante qui ne dépend que de l'automate. En d'autres termes, nous devons montrer que toute image d'un calcul, i.e. un ensemble d'états ω, peut être atteinte par un candidat sur une forêt de taille inférieure à une borne K.

Lemme 3.2 *Soit $\mathcal{A}$ un Automate d'Ensembles d'Arbres. On peut calculer un entier K, fonction du nombre d'états de $\mathcal{A}$, qui vérifie l'assertion suivante :*

> *Pour tout calcul r de $\mathcal{A}$, il existe un calcul r' de $\mathcal{A}$ et une forêt close F de cardinalité Card(F) inférieure à K telle que $r(T_\Sigma) = r'(F)$.*

Pour démontrer ce lemme, nous considérons un *TSA* $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ et un calcul r de $\mathcal{A}$.

Les définitions des automates d'ensembles d'arbres et des calculs dans ces automates, nous autorisent à considérer toute forêt close par sous-terme comme une *étape* dans un calcul. Cette appellation est principalement motivée par le fait que l'image d'un terme ne peut être calculée que lorsque l'image de tous ses sous-termes l'ont été. De façon informelle, la "machine" $\mathcal{A}$ ne peut être interrompue qu'à une étape où un ensemble de termes, clos par sous-terme, a été traité.

Par la finitude du nombre d'états de $\mathcal{Q}$, nous savons que l'image de r, $r(T_\Sigma)$, est atteinte pour une forêt close finie. Il existe donc F_0, une forêt close et finie telle que $r(F_0) = r(T_\Sigma)$.

L'objet du lemme est de montrer qu'il existe une forêt F de cardinalité inférieure à K et un calcul r' tels que $r'(F) = r(F_0)$. On peut donc voir cette preuve comme une *réduction* ou un *pompage* dans le calcul r. La difficulté est de mettre en évidence les mouvements dans r qui sont indispensables pour l'obtention de l'image $r(F_0)$. Les mouvements sont des applications des règles de $\mathcal{S}$. Le procédé que nous employons est une décomposition, à partir de F_0, du calcul r.

A partir de la forêt F_0, nous allons construire une suite finie $(F_i)_{i\geq 0}$ de forêts closes finies, dont chaque élément F_i est strictement inclus dans son prédécesseur

F_{i-1}, et reliées par la propriété suivante :

Pour tout i il existe un candidat sur F_i prolongeable sur F_{i-1}.

La plus petite forêt de cette suite $(F_i)_{i\geq 0}$ ne contient que des constantes de Σ.

Nous appelons ce processus, la *décomposition en étapes successives* du calcul r, à partir de l'étape F_0, jusqu'à son démarrage. Pendant la décomposition, il est nécessaire de collecter des informations qui permettront de recomposer le calcul de façon économique, c'est-à-dire, en appliquant le moins de règles possible. Nous verrons plus loin le type des informations qui doivent être collectées.

La recomposition est la construction à partir des informations collectées, d'une nouvelle suite finie d'étapes successives, i.e. une suite finie $(F'_i)_{i\geq 0}$ de forêts closes finies, ayant les mêmes caractéristiques que $(F_i)_{i\geq 0}$ et telle que, d'une part :

- il existe un candidat γ d'image $r(F_0) = r(T_\Sigma)$ sur F'_0, c'est-à-dire $\gamma(F'_0) = r(F_0) = r(T_\Sigma)$; et γ est prolongeable en un calcul de $\mathcal{A}$;

et d'autre part :

- Pour tout i, le nombre de termes ajoutés à F'_i pour obtenir F'_{i-1} est inférieur à une certaine constante J.

La constante J ne dépendra que du nombre d'états dans l'automate $\mathcal{A}$.

Les informations collectées doivent permettre de dire :

(a) "La dernière étape est atteinte".

C'est-à-dire, l'étape G atteinte est telle qu'il existe un candidat γ qui rencontre $r(T_\Sigma)$ sur G. Soit $\exists\gamma\ \gamma(G) = r(T_\Sigma)$.

Sans cette condition, il n'est pas possible de vérifier que la suite $(F'_i)_{i\geq 0}$ assure la propriété selon laquelle il existerait un candidat γ sur F'_0 d'image $\gamma(F'_0) = r(F_0) = r(T_\Sigma)$ (voir ci-dessus).

(b) "L'information suffit à la recomposition."

Ou encore, après avoir collecté l'information, on peut oublier les étapes du calcul.

La qualité des informations collectées est déterminante dans la recomposition. Il est nécessaire que l'information soit suffisamment complète pour recomposer, mais aussi suffisamment vague pour permettre une économie.

Soit γ un candidat à une étape G et I l'information mémorisée. De la première condition, nous pouvons en déduire que I doit contenir l'ensemble des états atteints, soit $\gamma(G)$, mais ce n'est pas suffisant. Supposons que γ doive être prolongé pour rencontrer les états $\mathbf{q}'$ et $\mathbf{q}''$ par les règles $c(\mathbf{q},\mathbf{q}) \rightarrow \mathbf{q}' \mid \mathbf{q}''$. Alors, nous devons savoir que :

1. il existe deux termes t_1 et t_2 dans G tels que $\gamma(t_1) = \gamma(t_2) = \mathbf{q}$.

2. deux termes parmi $c(t_1,t_1)$, $c(t_1,t_2)$, $c(t_2,t_1)$ et $c(t_2,t_2)$ ne sont pas dans G.

Exemple 3.6 Soit un automate d'ensembles d'arbres dont les règles sont :

$$\begin{array}{rcl} a & \rightarrow & \mathbf{q} \\ b(\mathbf{q}) & \rightarrow & \mathbf{q} \mid \mathbf{q}_1 \\ b(\mathbf{q}_1) & \rightarrow & \mathbf{q}_1 \\ c(\mathbf{q},\mathbf{q}) & \rightarrow & \mathbf{q}' \mid \mathbf{q}'' \end{array}$$

Pour tous les autres membres gauche g, il existe une règle $g \rightarrow \mathbf{q}_1$.

Considérons un calcul r, d'image $r(T_\Sigma) = \{\mathbf{q}, \mathbf{q}_1, \mathbf{q}', \mathbf{q}''\}$.

$$r(a) = \mathbf{q} \ ; \ r(b(a)) = \mathbf{q} \ ; \ r(b(b(a))) = \mathbf{q} \ ; \ r(b^3(a)) = \mathbf{q}_1$$
$$r(c(a,a)) = \mathbf{q}' \ ; \ r(c(a,b(a))) = \mathbf{q}' \ ; \ r(c(b(a),b(a))) = \mathbf{q}'' \ \ldots$$

Le candidat γ_1 qui applique a sur $\mathbf{q}$ et $b(a)$ sur $\mathbf{q}_1$ ne peut être prolongé en un calcul de même image que r, à l'inverse de γ_2 qui envoie a sur $\mathbf{q}$, $b(a)$ sur $\mathbf{q}$ et $b(b(a))$ sur $\mathbf{q}_1$. Il faut donc se souvenir que $\mathbf{q}$ est atteint deux fois.

Il apparaît donc nécessaire de conserver d'une part, les multiplicités des états rencontrés comme le montre l'exemple précédent, et d'autre part les multiplicités des états images des termes maximaux de l'étape pour satisfaire 2.

Mais, conserver dans l'information I le multi-ensemble exact des états atteints par un r à l'étape G est trop restrictif. Deux candidats qui rencontrent à chaque étape le même multi-ensemble d'états sont égaux.

Dans l'exemple 3.6, bien que γ_2 puisse être prolongé en un calcul d'image $\{\mathbf{q}, \mathbf{q}_1, \mathbf{q}', \mathbf{q}''\}$, il n'atteindra jamais trois occurrences de $\mathbf{q}$ comme r.

L'information qui sera associée à un candidat γ et une étape G sera donc de la forme (A, B), où A est le multi-ensemble des états atteints par les termes *maximaux* de G et B est *contenu* dans le multi-ensemble des états atteints par les termes *non maximaux* de G. De plus, dans la somme $A + B$ doivent apparaître chacun des états de $\gamma(G)$.

Il existe donc plusieurs informations A, B associées à un candidat γ sur une forêt G.

Exemple 3.7 (3.6 suite) Pour r à l'étape $G = \{a, b(a), b^2(a), b^3(a)\}$, les multi-ensembles A et B possibles sont :

$$\begin{array}{ll} A = \{\mathbf{q}_1 : 1\} & B = \{\mathbf{q} : 1\} \\ A = \{\mathbf{q}_1 : 1\} & B = \{\mathbf{q} : 2\} \\ A = \{\mathbf{q}_1 : 1\} & B = \{\mathbf{q} : 3\} \end{array}$$

Nous avons vu qu'il était nécessaire de conserver au moins deux états $\mathbf{q}$, mais en conserver plus est inutile. Intuitivement, il est inutile de "boucler" sur la règle $b(\mathbf{q}) \rightarrow \mathbf{q}$.

Nous conservons donc l'information $I = (A, B)$ avec $A = \{\mathbf{q}_1 : 1\}$ et $B = \{\mathbf{q} : 2\}$, pour le calcul r à l'étape G

Le candidat γ_2 peut être prolongé à partir de l'étape $G' = \{a, b(a), b^2(a)\}$ de la même façon que r à partir de l'étape $G = \{a, b(a), b^2(a), b^3(a)\}$ car il est possible de trouver des informations identiques à r, c'est-à-dire I.

L'information (A, B) est formellement définie par l'intermédiaire du prédicat $\mathbf{P}$ (voir la définition 3.5).

3.2.4 Preuve de la Décision du Vide

La preuve est organisée selon le plan suivant.

Dans un premier temps nous montrons le lemme 3.3 (de la page 68 à la page 86). C'est le lemme le plus technique, décomposé lui-même en trois parties (Parties I, II et III).

Ensuite, nous montrons le lemme 3.2 de la page 86 à la page 89.

Finalement, le théorème 3.1 est montré en page 89.

Rappelons que $Sup_F(t) = \{u \in F \mid t \trianglelefteq u \wedge u \neq t\}$ et si E est une forêt quelconque, alors $Inf(E) = \bigcup_{t \in E} Inf(t)$ où $Inf(b(t_1, \ldots, t_p) = \{t_1, \ldots, t_p\}$.

Définition 3.5 Soit $\mathcal{A} = (\Sigma, Q, \mathcal{F}, \mathcal{S}, \Omega)$ un automate d'ensembles d'arbres. Soit F une forêt close et soient A et B deux multi-ensembles d'états. Le prédicat $\mathbf{P}(F, A, B)$ vaut VRAI si et seulement si il existe un candidat γ tel que :

$$\begin{array}{lll} A & =_m & \gamma(M(F)); \\ B & =_s & \gamma(m(F)); \\ B & \subseteq_m & \gamma(m(F)). \end{array}$$

L'égalité ensembliste $A \cup B =_s \gamma(F)$ nous permet de tester l'égalité entre $\gamma(F)$ et $r(T_\Sigma)$ (i.e., la condition (a)). Les états de A donnent, en quelque sorte, quelles règles pourront être appliquées depuis cette étape, et le nombre d'occurrences dans A et B déterminent combien de règles de même membre gauche pourront être appliquées en même temps.

Nous commençons par prouver le lemme :

Lemme 3.3 *Soit $\mathcal{A} = (\Sigma, Q, \mathcal{F}, \mathcal{S}, \Omega)$ un automate d'ensembles d'arbres, soit une forêt close F et deux multi-ensembles A et B tels que $\mathbf{P}(F, A, B)$.*

Si B est non vide, il existe A' et B' vérifiant les trois propriétés suivantes :

1. *$Card(A' + B') \leq Card(A + B)$;*

2. *Il existe une forêt close F' qui vérifie $\mathbf{P}(F', A', B')$ et telle que $F' \subsetneq F$;*

3. *Pour toute forêt close H' telle que $\mathbf{P}(H', A', B')$, il existe une forêt close H qui vérifie $\mathbf{P}(H, A, B)$ avec $Card(H) \leq Card(H') + Card(A)$.*

La preuve est composée de trois parties.

1. *Construction.* Nous donnons un algorithme pour la construction de F', A', B' (de la page 69 à la page 75). Pour faciliter la preuve de correction de cet algorithme, nous distinguons une étape où quatre objets intermédiaires sont créés : une forêt close F_{int}, un ensemble de termes T, deux multi-ensembles d'états A_{int} et B_{int}.

2. *Propriétés.* Les objets A, B, F, A_{int}, B_{int}, F_{int} et A', B', F' sont liés par des propriétés (fait 3.1, lemmes 3.4 et 3.5) que nous mettons en évidence dans cette partie. Ces résultats serviront essentiellement à montrer les deux premiers points du lemme 3.3.

3. *Preuve du lemme* (page 82). La dernière partie montre que la construction donne bien les multi-ensembles A' et B' qui vérifient les trois points du lemme 3.3.

PARTIE I : Construction de F', A', B'

Introduction de la construction

Si nous considérons un terme t tel que $Sup_F(t)$ soit un sous-ensemble de $M(F)$, alors l'ensemble $Sup_F(t)$ peut être vu comme une couche supérieure de la forêt F. C'est-à-dire un ensemble de termes incomparables par l'ordre sous-terme, dont on pourra calculer les images simultanément, à l'aide d'un ensemble $\mathcal{S}_{F,t}$ de règles dont le membre gauche contient toujours au moins une occurrence de l'état $r(t)$. C'est cette couche que nous allons supprimer de F pour obtenir F' (figure 3.2).

Mais, la condition de recomposition (la condition 3 du lemme 3.3), nous oblige à calculer la bonne information (A', B') à l'étape F'. C'est l'information qui permettra de dire que tout candidat qui rencontre (A', B') à une étape G' peut être prolongé de façon à rencontrer (A, B). L'état $r(t)$ joue alors un rôle de repère dans la recomposition. C'est parce que cet état fera partie de l'ensemble A' que nous assurerons l'application des règles nécessaires, celles de $\mathcal{S}_{F,t}$.

Premièrement, nous constatons que le multi-ensemble $r(Sup_F(t))$ inclus dans A doit disparaître de A' car $Sup_F(t)$ disparaît de F. Puis, une occurrence de $r(t)$ doit être ajoutée à A', puisque t devient maximal. Plus généralement, un ensemble de termes non maximaux dans F deviennent maximaux dans F'. Nous décomposons cet ensemble en l'union disjointe $T \cup \{t\}$ (figure 3.3).

Pour s'assurer de la recomposition (condition 3 du lemme 3.3), il serait suffisant d'ajouter simplement $r(T)$ à A. Mais dans ce cas, la condition 1 du lemme 3.3 ($Card(A' + B') \leq Card(A + B)$) risque de ne plus être satisfaite. Certaines occurrences d'états doivent donc aussi disparaître de B'.

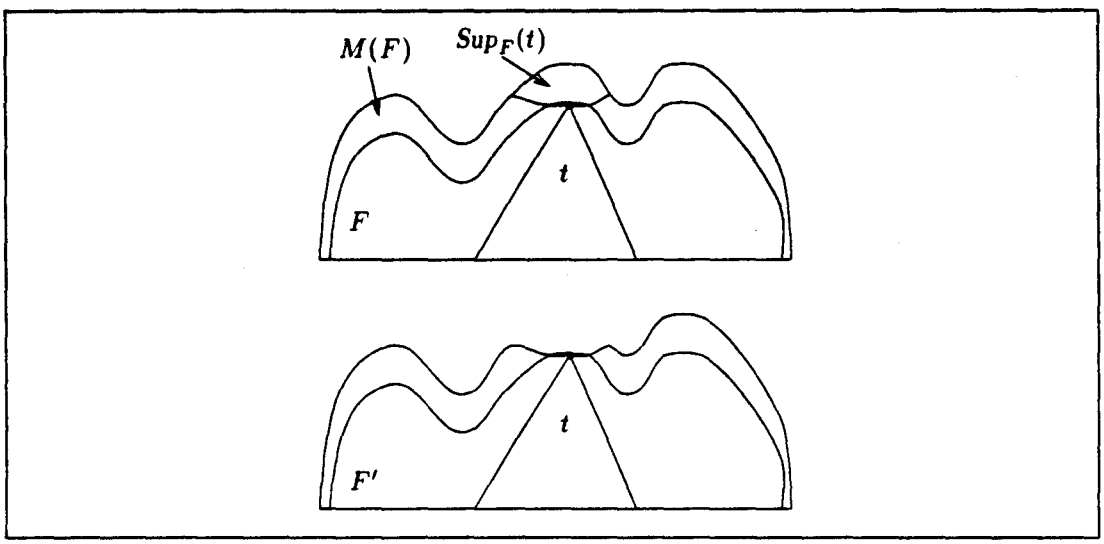


FIG. 3.2 - : Le terme t et l'ensemble $Sup_F(t)$ dans la forêt close F. La forêt F' obtenue.

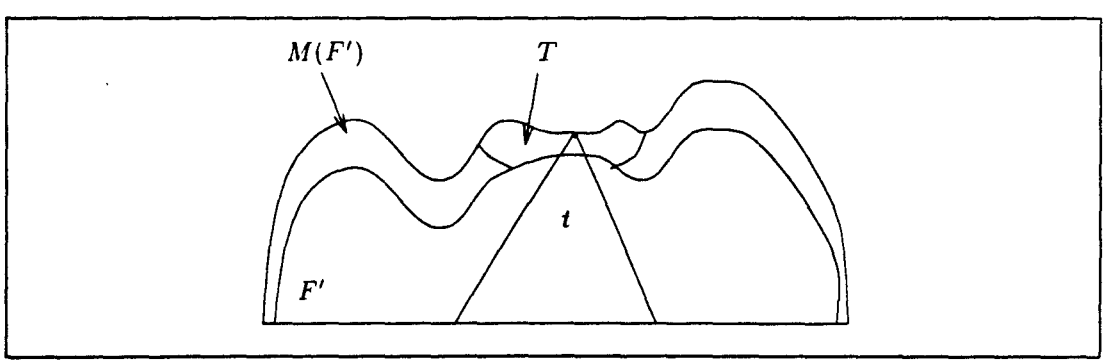


FIG. 3.3 - : Le terme t, la forêt close F et l'ensemble T.

Soient $\mathcal{A}$ un *TSA* et F, A, B tels que $\mathbf{P}(F, A, B)$ et B non vide. Puisque F, A, B vérifient $\mathbf{P}(F, A, B)$, il existe un candidat qui rencontre l'information (A, B) à l'étape F. Soit donc r le candidat tel que:

$$\begin{aligned} A &=_m r(M(F)) \\ B &\subseteq_m r(m(F)) \\ B &=_s r(m(F)) \end{aligned}$$

Nous considérons un terme t tel que tous les termes de $Sup_F(t)$ soient maximaux dans F. Soit $\mathcal{S}_{F,t}$, le multi-ensemble des règles utilisées pour obtenir les termes de $Sup_F(t)$. Une même règle peut-être utilisée plusieurs fois pour obtenir les termes de $Sup_F(t)$.

$$\mathcal{S}_{F,t} =_m \{b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \rightarrow \mathbf{q} \in \mathcal{S} \mid \begin{array}{l} \exists b(t_1, \ldots, t_p) \in Sup_F(t)\ r(b(t_1, \ldots, t_p)) = \mathbf{q} \\ \wedge \forall i \quad r(t_i) = \mathbf{q}_i \} \end{array}$$

Nous allons construire F_{int} en supprimant $Sup_F(t)$ de F et calculer la nouvelle information (A_{int}, B_{int}) obtenue. L'ensemble T sera alors l'ensemble des nouveaux termes maximaux (hormis t). A ce point de la construction, nous n'avons PAS $\mathbf{P}(F_{int}, A_{int}, B_{int})$. C'est la raison pour laquelle nous n'obtenons pas immédiatement A', B' et F'. En effet, les états de $r(T)$ sont nécessaires à la recomposition mais ne doivent pas obligatoirement figurer dans A'. La présence seule de $r(t)$ suffit. Nous allons donc dans une deuxième étape, nettoyer F_{int} des termes inutiles parce que l'état auquel ils sont associés par r est déjà en nombre suffisant. Ce nettoyage s'accompagne d'un rajustement de A_{int} et B_{int}.

La difficulté réside essentiellement dans le calcul du multi-ensemble B_{int}. Il faut effectivement calculer le nombre d'états nécessaires (i.e. à conserver dans B_{int}) pour appliquer simultanément les règles de $\mathcal{S}_{F,t}$.

On s'aperçoit que ce nombre est directement lié au nombre de règles de même membre gauche de $\mathcal{S}_{F,t}$. Par exemple, pour appliquer les deux règles $b(\mathbf{q}) \rightarrow \mathbf{q}_1 \mid \mathbf{q}_2$ il est nécessaire de disposer de deux occurrences de $\mathbf{q}$.

Nous décomposons donc la construction en deux cas:

(i) il existe t tel que $Sup_F(t) \subseteq M(F)$ et tel que $\mathcal{S}_{F,t}$ ne contienne pas deux règles de même membre gauche.

(ii) Pour tout terme t tel que $Sup_F(t) \subseteq M(F)$, $\mathcal{S}_{F,t}$ contient au moins deux règles de même membre gauche.

Construction

Premier cas: *il existe t tel que $Sup_F(t) \subseteq M(F)$ et tel que $\mathcal{S}_{F,t}$ ne contienne pas deux règles de même membre gauche.*

- *On retire de F les termes maximaux ayant t comme sous-terme direct.*
$F_{int} = F \setminus Sup_F(t)$
- *T contient les termes devenus maximaux à l'exception de t.*
$T = M(F_{int}) \setminus [M(F) \cup \{t\}]$
$A_{int} =_m [A - r(Sup_F(t))] + \{r(t)\}$
Si $r(m(F_{int}))(r(t)) > 1$ et $B(r(t)) = 1$ **Alors**
  - *L'égalité ensembliste $r(m(F_{int})) =_s B_{int}$ doit être préservée. Donc B_{int} est inchangé*
  $B_{int} =_m B$
Sinon
  $B_{int} =_m B - \{r(t)\}$
Fin Si

FIG. 3.4 - : Première étape. Construction des objets intermédiaires dans le premier cas : il n'existe pas deux règles de même membre gauche dans $\mathcal{S}_{F,t}$.

Nous pouvons dire qu'il n'y a pas de *concurrence* dans l'application des règles de $\mathcal{S}_{F,t}$. L'état $r(t)$ ne peut toujours être ôté de B pour préserver l'égalité ensembliste $r(m(F_{int})) =_s B_{int}$.

La construction apparaît en figure 3.4.

Second cas : *Pour tout terme t tel que $Sup_F(t) \subseteq M(F)$, $\mathcal{S}_{F,t}$ contient au moins deux règles de même membre gauche.*

Contrairement au cas précédent, plusieurs occurrences d'états sont nécessaires à l'application des règles de $\mathcal{S}_{F,t}$. Ceci justifie la construction suivante.

Soit Ψ la partition dont les classes regroupent tous les termes de l'ensemble $Sup_F(t)$ obtenus par des règles de même membre gauche.

$$\psi_{b(\mathbf{q}_1,\ldots,\mathbf{q}_p)} \in \Psi$$
$$\Leftrightarrow$$
$$\psi_{b(\mathbf{q}_1,\ldots,\mathbf{q}_p)} = \{b(t_1,\ldots,t_p) \mid \forall i \in \{1,\ldots,p\} r(t_i) = \mathbf{q}_i\}.$$

Pour une classe ψ correspondant à un membre gauche $b(\mathbf{q}_1,\ldots,\mathbf{q}_p)$, l'ensemble $r(Inf(\psi))$ représente le multi-ensemble d'états utilisés par le candidat r pour construire $r(\psi)$ à l'aide des règles de $\mathcal{S}_{F,t}$ de membre gauche $b(\mathbf{q}_1,\ldots,\mathbf{q}_p)$.

Les états non nécessaires peuvent être écartés. Certains états sont obligatoirement présents dans le multi-ensemble B, sinon la condition $r(m(F)) =_s B$ ne serait pas remplie. Soit $r(D^\psi)$ cet ensemble : D^ψ est un sous-ensemble de $Inf(\psi)$ sur lequel r atteint tous les éléments de $r(\psi)$ au plus une fois. D^ψ vérifie :

$$\forall t \in D^\psi \; r(D^\psi)(t) = 1 \text{ et } r(D^\psi) =_s r(Inf(\psi)).$$

Les autres états $r(Inf(\psi)) - r(D^\psi)$ sont des "copies" dont un certain nombre est utile à la construction de $r(\psi)$. Nous posons :

$$\text{si } Card(Inf(\psi)) < Card(\psi) \text{ alors } \Delta^\psi = Inf(\psi) \setminus D^\psi$$
$$\text{sinon } \Delta^\psi \subseteq Inf(\psi) \setminus D^\psi \text{ de sorte que } Card(\Delta^\psi) = Card(\psi) - 1$$

Finalement, l'algorithme de construction est donné en figure 3.5.

$F_{int} = F \setminus Sup_F(t)$
$T = M(F_{int}) \setminus [M(F) \cup \{t\}]$
$A_{int} =_m [A - r(Sup_F(t))] + \{r(t)\}$
- *On ajoute à B toutes occurrences d'états nécessaires pour l'application simultanée des règles de $S_{F,t}$. Chaque $r(\Delta^\psi)$ représente les états ajoutés pour appliquer des règles de même membre gauche.*

$B_{int} =_m B + r\left(\bigcup_{\psi \in \Psi} \Delta^\psi\right)$
Si $r(m(F_{int}))(r(t)) > 1$ et $B(r(t)) = 1$ **Alors**
- *L'égalité ensembliste $r(m(F_{int})) =_s B_{int}$ doit être préservée. Donc B_{int} est inchangé*

$B_{int} =_m B$
Sinon
$B_{int} =_m B - \{r(t)\}$
Fin Si

FIG. 3.5 - : Première étape : Construction des objets intermédiaires dans le second cas : des règles de même membre gauche sont utilisées.

Les objets intermédiaires A_{int}, B_{int}, F_{int} et T étant désignés, nous donnons la construction de F', A' et B' en figure 3.6.

$A_R = A_{int}$, $B_R = B_{int}$, $F_R = F_{int}$, $T_R = T$
Tant Que $T_R \neq \emptyset$
Soit $u \in T_R$
Si $B_R(r(u)) < r(m(F_R) \cup T_R)(u)$ **Alors**
- *Dans ce premier cas, conserver u dans F_R n'est pas nécessaire car il reste suffisamment de copies de $r(u)$. Donc u est supprimé, A_R et B_R ne sont pas modifiés mais le même nettoyage reste à faire pour les termes "juste sous" u.*

$F_R = F_R \setminus \{u\}$
$T_R = T_R \cup [Inf(u) \cap M(F_R)] \setminus \{u\}$
$A_R =_m A_R$
$B_R =_m B_R$
Sinon $B_R(r(u)) = r(m(F_R) \cup T_R)(u)$ **et**
- *Dans ce second cas, u est maximal dans F_R et $r(u)$ est nécessaire. Donc F_R est inchangé, A_R et B_R sont ajustés et u est traité.*

$F_R = F_R$
$T_R = T_R \setminus \{u\}$
$A_R =_m A_R + r(\{u\})$
$B_R =_m B_R - r(\{u\})$
Fin Si
Fin Tant Que
$A' = A_R$, $B' = B_R$, $F' = F_R$

FIG. 3.6 - : Seconde étape. Construction de la forêt F' et des multi-ensembles A' et B' par "nettoyage" de l'ensemble T et "rajustement" de A et B. Les états de $r(T)$ sont nécessaires à la recomposition mais ne doivent pas obligatoirement être présents dans A.

PARTIE II : Propriétés

Avant tout, quelques propriétés ensemblistes qui lient F, F_{int} et T.

Fait 3.1 *Soit F une forêt close et $t \in F$ tel que $Sup_F(t) \subseteq M(F)$. Si $F_{int} = F \setminus Sup_F(t)$ et $T = M(F_{int}) \setminus [M(F) \cup \{t\}]$ alors*

F1 : $M(F_{int}) \setminus T = [M(F) \setminus Sup_F(t)] \cup \{t\}$.

F2 : $m(F_{int}) = m(F) \setminus [T \cup \{t\}]$.

F3 : $m(F_{int}) \cup T = m(F) \setminus \{t\}$.

Preuve.

F1 : $M(F_{int}) \setminus T = [M(F) \setminus Sup_F(t)] \cup \{t\}$.

L'ensemble T est l'ensemble des nouveaux termes maximaux différents de t obtenus lorsqu'on supprime $Sup_F(t)$ de F. En fait, dans $M(F_{int})$ on retrouve une partie de $M(F)$, i.e. $M(F) \setminus Sup_F(t)$, mais aussi T et $\{t\}$. Soit

$$\begin{array}{rrcl} & M(F_{int}) & = & M(F \setminus Sup_F(t)) \\ & & = & [M(F) \setminus Sup_F(t)] \cup T \cup \{t\} \\ \Rightarrow & M(F_{int}) \setminus T & = & [M(F) \setminus Sup_F(t)] \cup \{t\}. \end{array}$$

F2 : $m(F_{int}) = m(F) \setminus [T \cup \{t\}]$.

Cette deuxième égalité signifie simplement que les termes devenus maximaux, $T \cup \{t\}$, ne sont plus non maximaux, i.e. ne sont pas dans $m(F_{int})$.

Enfin, puisque T est inclus dans $m(F)$, nous obtenons,

F3 : $m(F_{int}) \cup T = m(F) \setminus \{t\}$. □

Lemme 3.4 *Considérons les objets obtenus après application de l'algorithme de la première étape (figure 3.4 et 3.5). Alors,*

1. $Card(A_{int} + B_{int}) \leq Card(A + B)$;
2. *F_{int} est une forêt close strictement incluse dans F. $F_{int} \subsetneq F$;*
3. *Le candidat r vérifie :*
   - $r(M(F_{int}) \setminus T) =_m A_{int}$
   - $r(m(F_{int})) \subseteq_s B_{int} \subseteq_s r(m(F_{int}) \cup T)$
   - $B_{int} \subseteq_m r(m(F_{int}) \cup T)$

Preuve.

A l'image de la construction de F_{int}, A_{int}, B_{int} et T, la preuve de ce lemme est divisée en deux cas.

Premier cas : *il existe t tel que $Sup_F(t) \subseteq M(F)$ et tel que $S_{F,t}$ ne contienne pas deux règles de même membre gauche.*

- *Preuve de 1 :*

  Nous montrons que $Card(A_{int} + B_{int}) \leq Card(A + B)$. Rappelons que nous avons posé $A_{int} =_m [A - r(Sup_F(t))] + \{r(t)\}$. Puisque t n'est pas dans $Sup_F(t)$ et que $Sup_F(t)$ contient au moins un élément, alors nous avons

  $$Card(A_{int}) \leq Card(A).$$

  Maintenant, d'après la définition de B_{int}

  $$Card(B_{int}) \leq Card(B).$$

  Nous concluons :

  $$Card(A_{int} + B_{int}) \leq Card(A + B).$$

- *Preuve de 2 :*

  Nous montrons que F_{int} est une forêt close strictement incluse dans F. La preuve est directe car $Sup_F(t)$ est inclus dans $M(F)$ et F est close. Les termes ôtés de la forêt close F sont tous maximaux.

- *Preuve de 3 :*

  Vérifions le dernier point du lemme : le candidat r est tel que :

  - $r(M(F_{int}) \setminus T) =_m A_{int}$,
  - $r(m(F_{int})) \subseteq_s B_{int} \subseteq_s r(m(F_{int}) \cup T)$,
  - $B_{int} \subseteq_m r(m(F_{int}) \cup T)$.

  En effet,

  - $r(M(F_{int}) \setminus T) =_m A_{int}$ car pour construire A_{int} nous avons ôté de A les images des termes de F immédiatement plus grands que ceux de T.

    Plus formellement, d'après le fait 3.1, $M(F_{int}) \setminus T = [M(F) \setminus Sup_F(t)] \cup \{t\}$. Donc,

    $$r(M(F_{int}) \setminus T) =_m [r(M(F)) - r(Sup_F(t))] + r(\{t\}),$$

    Soit

    $$r(M(F_{int}) \setminus T) =_m [A - r(Sup_F(t))] + r(\{t\}) =_m A_{int}.$$

- Montrons les deux dernières relations multi-ensemblistes sur B.

  Par le fait 3.1, $m(F_{int}) = m(F) \backslash [T \cup \{t\}]$; et par la propriété $\mathbf{P}(F, A, B)$, $r(m(F)) \subseteq_s B$. La définition de B_{int} est soit $B_{int} =_m B$ soit $B_{int} =_m B - \{r(t)\}$. Nous déduisons directement $r(m(F_{int})) \subseteq_s B_{int}$.

  Toujours par le fait 3.1, nous avons $m(F_{int}) \cup T = m(F) \setminus \{t\}$; et par la propriété $\mathbf{P}(F, A, B)$, $B \subseteq_m r(m(F))$. Donc, $r(m(F)) \subseteq_m r(m(F_{int}) \cup T)$, soit $B \subseteq_m r(m(F_{int}) \cup T)$. Maintenant, puisque $B_{int} \subseteq_m B$, nous déduisons $B_{int} \subseteq_m r(m(F_{int}) \cup T)$.

Second cas : *Pour tout terme t tel que $Sup_F(t) \subseteq M(F)$, $S_{F,t}$ contient au moins deux règles de même membre gauche.*

- *Preuve de 1 :* Nous montrons que $Card(A_{int} + B_{int}) \leq Card(A + B)$.

  Remarquons que pour toute classe ψ de Ψ, $Card(\Delta^{\psi}) \leq Card(\psi) - 1$. Donc, $r(\bigcup_{\psi \in \Psi} \Delta^{\psi})$ est un multi-ensemble de cardinalité inférieure à $Card(Sup_F(t)) - 1$.

  $$Card(B_{int}) \leq Card(B) + Card(Sup_F(t)) - 1.$$

  Puisque par définition, $A =_m [A - r(Sup_F(t))] + \{r(t)\}$, nous avons :

  $$Card(A_{int}) \leq Card(A) - Card(Sup_F(t)) + 1.$$

  Soit,

  $$Card(A_{int}) + Card(B_{int}) \leq Card(A) + Card(B).$$

- *Preuve de 2 :*

  Elle est identique au cas précédent.

- *Preuve de 3 :* Le candidat r satisfait à la condition 3 du lemme. C'est-à-dire :

  - $r(M(F_{int}) \setminus T) =_m A_{int}$,
  - $r(m(F_{int})) \subseteq_s B_{int} \subseteq_s r(m(F_{int}) \cup T)$,
  - $B_{int} \subseteq_m r(m(F_{int}) \cup T)$.

  - En ce qui concerne la première égalité, le multi-ensemble A_{int} ainsi que les ensembles F_{int} et T sont calculés de la même façon que dans le cas précédent. Donc,

    $$r(M(F_{int}) \setminus T) =_m A_{int}.$$

- Par construction, nous avons $B \subseteq_m B_{int}$, donc $B \subseteq_s B_{int}$. Par le fait 3.1, $m(F_{int}) = m(F) \setminus [T \cup \{t\}]$. Soit

$$\begin{array}{rl} & r\,(m(F_{int})) =_m r\,(m(F) \setminus [T \cup \{t\}]) \\ \Rightarrow & r\,(m(F_{int})) \subseteq_s B - r\,([T \cup \{t\}]) \\ \Rightarrow & r\,(m(F_{int})) \subseteq_s B \subseteq_s B_{int} \end{array}$$

- Pour prouver que $B_{int} \subseteq_m r(m(F_{int}) \cup T)$, nous remarquons d'abord que $B_{int} =_m \left[B + r\left(\bigcup_{\psi\in\Psi} \Delta^{\psi}\right)\right] - \{r(t)\}$ ou $B_{int} =_m B + r\left(\bigcup_{\psi\in\Psi} \Delta^{\psi}\right)$. Dans les deux cas,

$$B_{int} \subseteq_m B + r\left(\bigcup_{\psi\in\Psi} \Delta^{\psi}\right).$$

Puisque $r\left(\bigcup_{\psi\in\Psi} \Delta^{\psi}\right) \subseteq_m r(T)$, nous avons

$$B_{int} \subseteq_m B + r(T).$$

Maintenant, d'après les hypothèses du lemme, $B \subseteq_m r(m(F))$, donc

$$B_{int} \subseteq_m r(m(F)) + r(T).$$

Les trois relations multi- ensemblistes sont alors vérifiées.

Ceci termine la preuve du lemme 3.4. □

Cette deuxième partie de l'algorithme réduit au vide l'ensemble T afin d'obtenir F', A' et B' tels que $\mathbf{P}(F', A', B')$. Pendant cette réduction, le nombre d'états mémorisés $(A + B)$ doit rester globalement inchangé pour satisfaire la contrainte sur la taille de l'information (premier point du lemme 3.3).

Lemme 3.5 *Après application de l'algorithme de la seconde étape (figure 3.6) :*

1. $F' \subseteq F_{int}$,
2. $\mathbf{P}(F', A', B')$,
3. $A_{int} \subseteq_m A'$,
4. $A' + B' =_m A_{int} + B_{int}$.

Preuve.

Le lemme est montré par induction sur la cardinalité de F_{int}, $Card(F_{int})$.

Base Si $Card(F_{int}) = 1$, alors $M(F_{int}) = F_{int}$ et $T \subsetneq F_{int}$. Donc $T = \emptyset$, l'algorithme donne $A' =_m A_{int}$. $B' =_m B_{int}$ et $F' = F_{int}$ et le lemme est trivialement vérifié.

Induction Supposons le lemme vrai pour toute forêt de cardinalité inférieure à k et soit F_{int}, $Card(F_{int}) = k$. Nous prouvons par induction sur $Card(T)$ que le lemme est vrai pour F_{int}.

Base $Card(T) = 0$. De nouveau, la construction donne l'affectation $A' =_m A_{int}$, $B' =_m B_{int}$ et $F' = F_{int}$, ce qui implique le lemme.

Induction Supposons que pour toute forêt F_{int} de cardinalité $Card(F_{int}) = k$ et pour tout T de cardinalité $Card(T) < k'$ le lemme soit vrai. Soit T tel que $Card(T) = k'$ et soit u un terme de T.

Par le lemme 3.4, $B_{int} \subseteq_m r(m(F_{int}) \cup T)$, donc deux cas sont étudiés : soit (i) $B_{int}(r(u)) < r(m(F) \cup T)(u)$, soit (ii) $B_{int}(r(u)) = r(m(F_{int}) \cup T)(u)$.

- *Premier Cas :* $B_{int}(r(u)) < r(m(F_{int}) \cup T)(u)$. L'algorithme en figure 3.6 donne :
  - $F_R = F_{int} \setminus \{u\}$,
  - $T_R = T \cup [Inf(u) \cap M(F_R)] \setminus \{u\}$,
  - $A_R =_m A_{int}$,
  - $B_R =_m B_{int}$.

  Puisque F_R est maintenant de cardinalité inférieure à k, nous pourrons conclure si F_R, T_R, A_R, B_R vérifient les hypothèses du lemme. Nous devons donc montrer que :
  F_R est une forêt close, $T_R \subsetneq M(F_R)$, il existe un candidat r_R qui satisfait les trois relations : $r_R(M(F_R) \setminus T_R) =_m A_R$, $r_R(m(F_R)) \subseteq_s B_R \subseteq_s r_R(m(F_R) \cup T_R)$ et $B_R \subseteq_m r_R(m(F_R) \cup T_R)$.

  Remarquons que les termes devenus maximaux par la suppression de u sont les termes de $Inf(u) \cap M(F_R)$.
  - Premièrement, F_{int} est close, $T \subsetneq M(F_{int})$ et $u \in T$. Donc F_R est close.
  - T_R est strictement inclus dans $M(F_{int})$. En effet, $M(F_{int}) \setminus \{u\} \subseteq M(F_R)$ et $T \subsetneq M(F_{int})$, donc

    $$T \setminus \{t\} \subsetneq M(F_R).$$

    Maintenant puisque $Inf(t) \cap M(F_R)$ est inclus dans $M(F_R)$, nous avons

    $$T_R = T \cup [Inf(t) \cap M(F_R)] \setminus \{t\} \subsetneq M(F_R)$$

  - Nous montrons que le candidat r satisfait les trois relations multi-ensemblistes $r(M(F_R) \setminus T_R) =_m A_R$, $r(m(F_R)) \subseteq_s B_R \subseteq_s r(m(F_R) \cup T_R)$ et $B_R \subseteq_m r(m(F_R) \cup T_R)$.

• $r(M(F_R)\backslash T_R) =_m A_R$. Montrons que $M(F_R)\backslash T_R = M(F_{int})\backslash T$. Puisque $u \in M(F_{int})$ et $M(F_R) = M(F_{int} \setminus T)$, alors

$$M(F_R) = [M(F_{int}) \setminus T] \cup [Inf(u) \cap M(F_R)]$$

Le membre droit se traduit par l'ensemble des termes qui restent maximaux union l'ensemble des termes qui deviennent maximaux en ôtant T de F_{int}.
Par les hypothèses du lemme, $r(M(F_{int}) \setminus T) =_m A_{int}$. Avec l'égalité $M(F_R)\backslash T_R = M(F_{int})\backslash T$, nous concluons $r(M(F_R)\backslash T_R) =_m A_R$.

• $r(m(F_R)) \subseteq_s B_R$. Par construction $F_R \subseteq F_{int}$, donc

$$r(F_R) \subseteq_s r(F_{int}).$$

Mais $B_{int} =_m B_R$ et par hypothèse d'induction $r(m(F_{int})) \subseteq_s B_{int}$. Soit :

$$r(m(F_R)) \subseteq_s B_R.$$

• $B_R \subseteq_m r(m(F_R) \cup T_R)$. Montrons que $[m(F_{int}) \cup T] \setminus \{u\} = m(F_R) \cup T_R$. D'abord, puisque $F_R = F_{int} \setminus \{u\}$, $m(F_R) = m(F_{int} \setminus \{u\})$. Or,

$$m(F_{int} \setminus \{u\}) = [m(F_{int}) \setminus \{u\}] \setminus [Inf(u) \cap M(F_R)]$$

C'est-à-dire que les termes minimaux de F_R, sont les termes minimaux de F_{int} hormis ceux qui sont devenus maximaux par la suppression de u, i.e. $Inf(u) \cap M(F_R)$. De cette dernière égalité, et par la définition de T_R, nous obtenons directement :

$$[m(F_{int}) \cup T] \setminus \{u\} = m(F_R) \cup T_R.$$

Nous avons $B_R =_m B_{int}$ et par les hypothèse du lemme $B_{int} \subseteq_m r(m(F_{int}) \cup T)$. Mais puisque nous avons supposé que $B_{int}(r(u)) < r(m(F_{int}) \cup T)(u)$, alors $B_{int} \subseteq_m r(m(F_{int}) \cup T \setminus \{u\})$. Comme $B_R =_m B_{int}$, nous déduisons

$$B_R \subseteq_m r(m(F_R) \cup T_R).$$

Finalement, la cardinalité de F_R est strictement inférieure à k car $F_R = F_{int} \setminus \{t\}$. Nous avons donc montré que les hypothèses d'application du lemme et les hypothèses d'induction sont bien vérifiées et donc il existe F', A' et B', tels que $\mathbf{P}(F', A', B')$, $F' \subseteq F_R \subseteq F_{int}$, $A_{int} \subseteq_m A_R \subseteq_m A'$ et $A' + B' =_m A_R + B_R =_m A_{int} + B_{int}$.

Ceci termine la preuve du lemme 3.5 pour le premier cas.

- *Second Cas:* $B_{int}(r(u)) = r(m(F_{int}) \cup T)(u)$. L'algorithme en figure 3.6 donne:
  - $F_R = F_{int}$,
  - $T_R = T \setminus \{u\}$,
  - $A_R =_m A_{int} + r(\{u\})$,
  - $B_R =_m B_{int} - r(\{u\})$.

  Nous allons montrer que F_R, T_R, A_R et B_R vérifient les hypothèses d'induction et d'application du lemme.
  - F_R est évidemment close et clairement T_R est strictement inclus dans $M(F_R)$.
  - Montrons que A_R et B_R avec le candidat r vérifient les trois relations multi-ensemblistes.

    • $r(M(F_R) \setminus T_R) =_m A_R$. Nous avons directement, grâce aux hypothèses du lemme:

$$\begin{array}{rcl} r(M(F_R) \setminus T_R) & =_m & r([M(F_{int}) \setminus T] \cup \{u\}) \\ & =_m & A_{int} + r(\{u\}) \\ & =_m & A_R \end{array}$$

    • $r(m(F_R)) \subseteq_s B_R$. Par construction $F_R = F_{int}$, et par les hypothèses du lemme, $r(m(F_{int})) \subseteq_s B_{int}$, donc

$$r(m(F_R)) \subseteq_s B_{int}.$$

    Ce second cas suppose que $B_{int}(r(u)) = r(m(F_{int}) \cup T)(u)$, mais puisque $u \in T$, nous avons directement $B_{int}(r(u)) < r(m(F_{int}))$, soit :

$$B_{int}(r(u)) < r(m(F_R)).$$

    • $B_R \subseteq_m r(m(F_R) \cup T_R)$. Là encore, la preuve est directe car $r(m(F_R) \cup T_R) = r(m(F_{int}) \cup (T \setminus \{u\}))$, soit

$$r(m(F_R) \cup T_R) = r(m(F_{int}) \cup T) - r(\{u\}).$$

    Nous avons posé $B_R =_m B_{int} - r(\{u\})$, et par les hypothèses du lemme $B_{int} \subseteq_m r(M(F_{int}) \cup T)$. Donc $B_R \subseteq_m r(M(F_{int}) \cup T) - r(\{u\})$ et

$$B_R \subseteq_m r(m(F_R) \cup T_R).$$

Nous remarquons que la cardinalité de F_R égale à k mais par contre la cardinalité de T_R est strictement inférieure à k'. Nous avons donc montré que les hypothèses d'induction et d'application du lemme sont bien vérifiées et donc il existe F', A' et B', tels que $\mathbf{P}(F', A', B')$, $F' \subseteq F_R \subseteq F_{int}$, $A_{int} \subseteq_m A_R \subseteq_m A'$ et $A' + B' =_m A_R + B_R =_m A_{int} + B_{int}$.
Ceci termine la preuve du second et dernier cas et par conséquent le lemme 3.5 est prouvé. □

PARTIE III: Preuve du lemme 3.3

Grâce aux constructions précédentes (Partie I), nous sommes en mesure de prouver le lemme 3.3. Rappelons son énoncé.

Lemme 3.3 *Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ un automate d'ensembles d'arbres, soit une forêt close F et deux multi-ensembles A et B tels que $\mathbf{P}(F, A, B)$.*
Si B est non vide, il existe A' et B' vérifiant les trois propriétés suivantes :

1. $Card(A' + B') \leq Card(A + B)$;
2. *Il existe une forêt close F' qui vérifie $\mathbf{P}(F', A', B')$ et telle que $F' \subsetneq F$;*
3. *Pour toute forêt close G' telle que $\mathbf{P}(G', A', B')$, il existe une forêt close G qui vérifie $\mathbf{P}(G, A, B)$ avec $Card(G) \leq Card(G') + Card(A)$.*

Preuve. Soient $\mathcal{A}$ un TSA et F, A, B tels que $\mathbf{P}(F, A, B)$ et B non vide. Puisque F, A, B vérifient $\mathbf{P}(F, A, B)$, il existe un candidat qui rencontre l'information (A, B) à l'étape F. Soit donc r_F le candidat tel que :

$$A =_m r_F(M(F))$$
$$B \subseteq_m r_F(m(F))$$
$$B =_s r_F(m(F))$$

Premier et deuxième point: Les ensembles et multi-ensembles F, A, et B remplissent les conditions d'application de la construction de la partie I. Nous obtenons alors les forêts F_{int} et T et les multi-ensembles A_{int} et B_{int}.

Grâce aux lemmes 3.4 et 3.5, nous prouvons que F', A', B' vérifient les trois points du lemme 3.3.

Par le lemme 3.4, nous savons que $Card(A_{int} + B_{int}) \leq Card(A + B)$. Par le second, $Card(A_{int} + B_{int}) = Card(A' + B')$. Alors,

$$Card(A' + B') \leq Card(A + B).$$

La forêt F' obtenue après les deux applications successives des lemmes 3.4 et 3.5 satisfait bien au deuxième point de ce lemme. En effet, F' est close, $\mathbf{P}(F', A', B')$ est vrai, et puisque $F_{int} \subsetneq F$ et $F' \subseteq F_{int}$, alors $F' \subsetneq F$.

Troisième point : Le dernier point est plus technique. Nous devons montrer que si un candidat quelconque rencontre l'information (A', B') à une étape G', alors il existe un candidat avec l'information (A, B) à l'étape G où G est une forêt de taille ne dépassant pas $Card(G') + Card(A)$.

En fait, nous allons montrer que les termes ajoutés à G' sont dans un ensemble E qui vérifie :

- $G' \cup E$ est close et $G' \cap E = \emptyset$. Cela signifie que $G' \cup E$ est une étape et que pour prolonger un candidat de G' à $G' \cup E$, on calcule les images des termes de E.
- $E \subseteq M(G' \cup E)$. C'est-à-dire, les termes de E sont tous plus grands (par l'ordre sous-terme) que les termes de G', mais aussi incomparables par $\trianglelefteq$. Le prolongement d'un candidat de G' à $G' \cup E$ peut être réalisé en calculant simultanément les images des termes de E.
- $\mathbf{P}(G' \cup E, A, B)$.
- $Card(E) \leq Card(A)$. C'est la condition importante qui exprime le fait que la recomposition sera "économique".

Selon les hypothèses, $\mathbf{P}(G', A', B')$ est vrai. Soit $r_{G'}$ le calcul qui vérifie

$$r_{G'}(M(G')) =_m A \ ,$$
$$r_{G'}(m(G')) =_s B \ ,$$
$$B \subseteq_m r_{G'}(m(G')) \ .$$

Rappelons aussi que r_F est le candidat tel que $A =_m r_F(M(F))$, $B \subseteq_m r_F(m(F))$, $B =_s r_F(m(F))$.

Nous allons montrer l'existence de E en suivant la construction de la partie I. La forêt F' et les multi-ensembles A' et B' ont été obtenus de façon différente, suivant deux cas.

• *il existe t tel que $Sup_F(t) \subseteq M(F)$ et tel que $\mathcal{S}_{F,t}$ ne contienne pas deux règles de même membre gauche.*

Nous avons désigné un terme t satisfaisant cette propriété et nous avons construit F_{int} en ôtant $Sup_F(t)$ de F. L'état $r_F(t)$ a été ajouté à A_{int}. Par le lemme 3.5, $A_{int} \subseteq_m A'$. Donc

$$r_F(t) \in A'.$$

Puisque $r_{G'}(M(G')) =_m A'$, nous déduisons qu'il existe un terme $t_{G'}$ dans $M(G')$ tel que $r(t_{G'}) = r_F(t)$.

Par définition, tous les termes de $Inf(Sup_F(t))$ ne sont pas maximaux dans F et donc $Inf(Sup_F(t)) \subseteq m(F)$. De plus, par le lemme 3.4, nous avons

$$r_F(Inf(Sup_F(t))) \subseteq_s B_{int} \ ,$$

et par le lemme 3.5,

$$A_{int} + B_{int} = A' + B' \ .$$

Donc, les états nécessaires à la construction des images des termes de $Sup_F(t)$ sont dans $A' + B'$, i.e. :

$$r_F(Inf(Sup_F(t))) \subseteq_s A' + B'.$$

Par cette constatation, et parce que $\mathbf{P}(G', A', B')$, il existe un sous-ensemble de G', noté D de termes d'images par $r_{G'}$ exactement égales à $r_F(Inf(Sup_F(t)))$. Plus formellement, D vérifie :

$$\forall u = b(t_1, \ldots, t_p) \in Sup_F(t) \quad \exists v_1, \ldots, v_p \in D$$
$$\text{avec } r_{G'}(u_i) = r_F(v_i) \text{ et tel que } \exists j \ v_j = t_G.$$

Soit E l'ensemble des termes $b(v_1, \ldots, v_p)$ ainsi définis. Par construction $G' \cup E$ est une forêt close et $E \subseteq G' \cup E$. La cardinalité de E est exactement celle de $Sup_F(t)$. Puisque $Sup_F(t) \subseteq M(F)$, nous déduisons que $Card(E) \leq Card(A)$.

Le dernier point à vérifier est direct puisqu'il n'existe pas deux règles de même membre gauche dans $\mathcal{S}_{F,t}$. Il est donc possible de prolonger $r_{G'}$ sur $G' \cup E$ en appliquant les règles de $\mathcal{S}_{F,t}$, ce qui implique $\mathbf{P}(G' \cup E, A, B)$.

• *Pour tout terme t tel que $Sup_F(t) \subseteq M(F)$, $\mathcal{S}_{F,t}$ contient au moins deux règles de même membre gauche.*

Montrons que les constructions données dans ce second cas entraînent l'existence d'un ensemble E tel que

- $G' \cup E$ est close et $G' \cap E = \emptyset$.
- $E \subseteq M(G' \cup E)$.
- $\mathbf{P}(G' \cup E, A, B)$.
- $Card(E) \leq Card(A)$.

Comme dans le cas précédent, nous avons nous avons désigné un terme t et nous avons construit F_{int} en ôtant $Sup_F(t)$ de F. L'état $r_F(t)$ a été ajouté à A_{int}. Par le lemme 3.5, $A_{int} \subseteq_m A'$. Donc

$$r_F(t) \in A'.$$

Il existe donc un terme $t_{G'}$ dans $M(G')$ tel que $r(t_{G'}) = r_F(t)$.

Par contre, vérifier la relation $r(Inf(Sup_F(t))) \subseteq_s A' + B'$ n'est pas suffisant pour assurer l'application des règles de $\mathcal{S}_{F,t}$. Par exemple, pour appliquer les deux règles $b(\mathbf{q}_1, \ldots \mathbf{q}_p) \rightarrow \mathbf{q} \mid \mathbf{q}'$, il doit exister au moins deux termes de la forme $b(t_1, \ldots t_p)$ et $b(t'_1, \ldots t'_p)$ dans G' avec $r(t_i) = r(t'_i) = \mathbf{q}_i$. Donc, il est nécessaire que parmi $\mathbf{q}_1, \ldots, \mathbf{q}_p$, au moins un état apparaît deux fois ou plus dans $A' + B'$.

Rappelons enfin que la condition $r(Inf(Sup_F(t))) \subseteq_m A' + B'$ n'est en général pas remplie pour permettre $Card(A' + B') \leq Card(A + B)$.

Les "copies" d'états nécessaires à l'application de règles de même membre gauche ont été ajoutées par l'intermédiaire de l'ensemble $\bigcup_{\psi \in \Psi} \Delta^\psi$. Nous allons montrer que ces copies ont été ajoutées en nombre suffisant.

La partition Ψ de l'ensemble $Sup_F(t)$ est telle que chaque classe contienne les termes obtenus par des règles de même membre gauche. D^ψ est un sous-ensemble de $Inf(\psi)$ sur lequel r_F atteint tous les éléments de $r_F(\psi)$ au plus une fois. D^ψ vérifie :

$$\forall t \in D^\psi \; r_F(D^\psi)(t) = 1 \text{ et } r_F(D^\psi) =_s r_F(Inf(\psi)).$$

$r(\Delta^\psi)$ contient les copies pour obtenir $r(\psi)$:

$$\begin{array}{c} \text{si } Card(Inf(\psi)) < Card(\psi) \text{ alors } \Delta^\psi = Inf(\psi) \setminus D^\psi \\ \text{sinon } \Delta^\psi \subseteq Inf(\psi) \setminus D^\psi \text{ de sorte que } Card(\Delta^\psi) = Card(\psi) - 1 \end{array}$$

Par définition de A_{int}, B_{int} et de A' et B', nous montrons comme dans le cas précédent que

$$r_F(Inf(Sup_F(t))) \subseteq_s A' + B'.$$

L'ensemble $r\left(\bigcup_{\psi \in \Psi} \Delta^\psi\right)$ est ajouté à B_{int}, par conséquent,

$$r\left(\bigcup_{\psi \in \Psi} \Delta^\psi\right) \subseteq_m A' + B'.$$

L'ensemble E est donné par une union d'ensembles disjoints E^ψ qui correspondent dans G' aux ensembles ψ dans F'. $E\psi$ sont les ensembles minimaux (en nombre de termes) qui vérifient :

$$\begin{array}{c} \forall u = b(t_1, \ldots, t_p) \in \psi \quad \exists b(v_1, \ldots, v_p) \in E^\psi \\ \text{avec } r_{G'}(u_i) = r_F(v_i) \text{ et tel que } \exists j \; v_j = t_G. \end{array}$$

Par construction $G' \cup E$ est une forêt close et $E \subseteq G' \cup E$. De plus, il est clair que $Card(E^\psi) = Card(\psi)$ pour tout ψ. Donc $Card(E) = Card(\sup t)$, ce qui implique que $Card(E) \leq Card(A)$.

Le dernier point à montrer est $\mathbf{P}(G' \cup E, A, B)$. En fait, il faut montrer que le nombre de copies ajoutées est suffisant. Dans tous les cas, ce nombre ne dépasse pas $Card(\psi) - 1$.

Or, dans un calcul pour appliquer simultanément k règles de même membre gauche $b(\mathbf{q}_1, \ldots, \mathbf{q}_p)$, il suffit de p termes d'images $\{\mathbf{q}_1, \ldots, \mathbf{q}_p\}$ et de $k-1$ copies.

Fait 3.2 *Soit r une application de $\{t_1, \ldots, t_{p+k}\}$ dans $\{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$ avec $k \geq 0$, $m \leq p$ et $\forall i \in \{1, \ldots, m\}$, $r(t_i) = \mathbf{q}_i$. Soient $j_1, \ldots j_p \in \{1, \ldots, m\}$ et M_k*

l'ensemble des p-uplets $(t_{i_1}, \ldots, t_{i_p})$ *tels que*

$$(r(t_{i_1}), \ldots, r(t_{i_p})) = (\mathbf{q}_{j_1}, \ldots, \mathbf{q}_{j_p}) \ .$$

Alors,

$$Card(M_k) \geq k + 1 \ .$$

Preuve. Le fait est facilement montré par induction sur k.

- $k = 0$. Par hypothèse, r est une application de $\{t_1, \ldots, t_p\}$ dans $\{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$ avec $m \leq p$. De plus, $\forall i \in \{1, \ldots, m\}$, $r(t_i) = \mathbf{q}_i$. Donc

  $$(r(t_{j_1}), \ldots, r(t_{j_m})) = (\mathbf{q}_{j_1}, \ldots, \mathbf{q}_{j_p}) \ ,$$

  et $Card(M_0) \geq 1$.

- Supposons $Card(M_{k-1}) \geq k$, alors $Card(M_k) \geq k$ puisque $M_{k-1} \subseteq M_k$.

  Soit $t \notin \{t_1, \ldots, t_{p+k-1}\}$ tel que $r(t) \in \{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$. Puisque $m \leq p$, il existe x tel que $r(t) = \mathbf{q}_{j_x}$. Il existe donc un p-uplet $(t_{i_1}, \ldots, t_{i_p})$ avec $t_{i_x} = t$. Clairement, ce p-uplet n'est pas dans M_{k-1} et vérifie

  $$(r(t_{i_1}), \ldots, r(t_{i_p})) = (\mathbf{q}_{j_1}, \ldots, \mathbf{q}_{j_p}) \ .$$

  Donc $Card(M_k) \geq k + 1$ et le fait est vérifié. □

Il est donc possible d'appliquer simultanément $Card(\psi)$ règles de même membre gauche. Cette remarque implique qu'il existe un prolongement de $r_{G'}$ sur $G \cup E$ par les règles de $\mathcal{S}_{F,t}$. Donc, $\mathbf{P}(G' \cup E, A, B)$ est vérifié.

Ceci termine la preuve du second cas et par conséquent la preuve du lemme 3.3. □

Preuve du théorème 3.1

Le lemme 3.3 étant prouvé, nous pouvons montrer le lemme 3.2.

Pour cela, nous formalisons l'idée de décomposition d'un calcul en étapes successives donnée en page 66. Le prédicat CHAINE est introduit pour désigner suite finie d'informations $(A_i, B_i)_{i \leq k}$, de la plus grande (A_0, B_0) à la plus petite (A_k, B_k).

Si il existe un calcul, il est donc possible de construire une telle suite et nous pouvons considérer l'information rencontrée par la restriction de ce calcul à chaque étape.

Définition 3.6 Soit $(A_i, B_i)_{i \leq k}$ une suite de couples de multi-ensembles d'états. Le prédicat CHAINE$((A_i, B_i)_{i \leq k})$ est inductivement défini par :

Si $k = 0$ alors CHAINE$((A_i, B_i)_{i \leq 0})$ si il existe F_0 une forêt close telle que $\mathbf{P}(F_0, A_0, B_0)$;

Si $k > 0$ alors CHAINE$((A_i, B_i)_{i \leq k})$ si il existe F_k une forêt close telle que :

- $\mathbf{P}(F_k, A_k, B_k)$

- CHAINE$((A_i, B_i)_{i \leq k-1})$ et il existe F_{k-1} tel que :
  - $Card(A_k + B_k) \leq Card(A_{k-1} + B_{k-1})$;
  - $F_k \subsetneq F_{k-1}$ et $\mathbf{P}(F_{k-1}, A_{k-1}, B_{k-1})$;
  - Pour tout H' tel que $\mathbf{P}(H', A_k, B_k)$, il existe H tel que $\mathbf{P}(H, A_{k-1}, B_{k-1})$ avec $Card(H) \leq Card(H') + Card(A_{k-1})$.

Supposons qu'il existe un calcul r dans $\mathcal{A}$. Comme annoncé dans la section 3.2.3, il existe une forêt F telle que $r(F) = r(T_\Sigma)$. En fait, cette forêt peut être choisie de sorte qu'à cette étape, la taille de l'information rencontrée par r soit inférieure au nombre d'états dans $\mathcal{A}$.

Lemme 3.6 *Soit $\mathcal{A}$ un TSA avec n états. Si r est un calcul dans $\mathcal{A}$, alors il existe F, A, B tels que*

- $r(F) = r(T_\Sigma)$,
- $A =_m r(M(F))$,
- $B \subseteq_m r(m(F))$,
- $B =_s r(m(F))$,
- $Card(A + B) \leq n$.

Preuve.

Supposons donc qu'il existe un calcul r dans $\mathcal{A}$. Alors, par la finitude du nombre d'états de $\mathcal{Q}$, il existe F une forêt close minimale selon l'ordre induit par la relation d'inclusion ($\subseteq$), qui vérifie $r(F) = r(T_\Sigma)$.

Nous construisons A et B :

- $A =_m r(M(F))$,
- $B = \{\mathbf{q} : 1 \mid \mathbf{q} \in r(m(F))\}$.

Nous avons alors $B \subseteq_m r(m(F))$, $B =_s r(m(F))$, $\forall \mathbf{q} \in \mathcal{Q}\ B(\mathbf{q}) \leq 1$. Avant de prouver que $Card(A + B) \leq n$, montrons que $A \cap B = \emptyset$. Si il existe un terme t dans $A \cap B$, alors la forêt $F \setminus \{t\}$ est une forêt close qui vérifie $r(F \setminus \{t\}) = r(T_\Sigma)$ et $F \subseteq F \setminus \{t\}$. Ceci contredit le fait que F est minimale.

De la même façon, nous montrons que tout état apparaît dans A au plus une fois. En effet, dans le cas contraire il existerait deux termes t_1 et t_2 de même image et $F \setminus \{t_1\}$ vérifierait $r(F \setminus \{t_1\}) = r(T_\Sigma)$ et $F \subseteq F \setminus \{t_1\}$.

Maintenant, puisque chaque état apparaît dans B au plus une fois, alors

$$Card(A + B) \leq n \ .$$

□

Lemme 3.7 *Soit $\mathcal{A}$ un TSA avec n états. Si il existe un calcul r dans $\mathcal{A}$, alors il existe $(A_i, B_i)_{i\leq k}$ tel que* CHAINE$((A_i, B_i)_{i\leq k})$ *avec :*

- $B_k = \emptyset$,
- $A_0 + B_0 =_s r(T_\Sigma)$ *et* $Card(A_0 + B_0) \leq n$.

Preuve.

L'existence $(A_i, B_i)_{i\leq k}$ est prouvée par induction sur k à l'aide des lemmes 3.6 et 3.3.

Par le lemme 3.6, nous prouvons directement qu'il existe A_0, B_0 tels que $A_0 + B_0 =_s r(T_\Sigma)$ et $Card(A_0+B_0)) \leq n$. Clairement, nous avons CHAINE$((A_i, B_i)_{i\leq 0})$.

Supposons maintenant le prédicat CHAINE$((A_i, B_i)_{i\leq j})$ vrai. Si B_j est vide alors le lemme est montré. Dans le cas contraire, il existe F_j tel que $\mathbf{P}(F_j, A_j, B_j)$ et par le lemme 3.3, il existe A_{j+1} et B_{j+1} tels que

(i) $Card(A_{j+1} + B_{j+1}) \leq Card(A_j + B_j)$,

(ii) il existe F_{j+1} qui vérifie $F_{j+1} \subsetneq F_j$ et $\mathbf{P}(F_{j+1}, A_{j+1}, B_{j+1})$, et

(iii) pour tout H' tel que $\mathbf{P}(H', A_{j+1}, B_{j+1})$, il existe H tel que $\mathbf{P}(H, A_j, B_j)$ avec $Card(H) \leq Card(H') + Card(A_j)$.

D'après la définition 3.6, nous avons alors CHAINE$((A_i, B_i)_{i\leq j+1})$ vrai.

L'induction termine bien car à chaque pas la taille de la forêt décroit et donc il existe une étape k pour laquelle B_k est vide. □

Lemme 3.8 *Soit* $(A_i, B_i)_{i\leq k}$ *tel que* CHAINE$((A_i, B_i)_{0\leq i\leq k})$.

Alors il existe $(A'_i, B'_i)_{i\leq k}$ *tel que* CHAINE$((A'_i, B'_i)_{0\leq i\leq k'})$, $(A_0, B_0) = (A'_0, B'_0)$ *et pour tout l et l', si* $l \neq l'$ *alors* $(A'_l, B'_l) \neq (A'_{l'}, B'_{l'})$.

Preuve.

Soit $(A_i, B_i)_{i\leq k}$ tel que CHAINE$((A_i, B_i)_{i\leq k})$. Soient l et l' tels que $l \neq l'$ et $(A_l, B_l) = (A_l, B_l)$. Alors, par définition de CHAINE, il existe F_l et $F_{l'}$ tels que $\mathbf{P}(F_l, A_l, B_l)$ et $\mathbf{P}(F_{l'}, A_{l'}, B_{l'})$.

De plus, toujours par définition du prédicat CHAINE, pour tout H' vérifiant $\mathbf{P}(H', A_l, B_l)$, il existe H tel que $\mathbf{P}(H, A_{l-1}, B_{l-1})$. C'est donc le cas pour F_l.

Il existe donc $(A'_i, B'_i)_{i\leq k'}$ tel que CHAINE$((A'_i, B'_i)_{i\leq k'})$ avec $k' = k-(l'-l)$, $(A'_i, B'_i) = (A_i, B_i)$ pour $i \in 0, \ldots, l$ et $(A'_i, B'_i) = (A_{i+l'-l}, B_{i+l'-l})$ pour $i \in \{l, \ldots, k'\}$. □

Lemme 3.2 *Soit $\mathcal{A}$ un Automate d'Ensembles d'Arbres. On peut calculer un entier K, fonction du nombre d'états de $\mathcal{A}$, qui vérifie l'assertion suivante :*

> *Pour tout calcul r de $\mathcal{A}$, il existe un calcul r' de $\mathcal{A}$ et une forêt close F de cardinalité Card(F) inférieure à K telle que $r(T_\Sigma) = r'(F)$.*

Preuve.

Supposons qu'il existe un calcul r dans $\mathcal{A}$. Alors, d'après le lemme 3.7, il existe $(A_i, B_i)_{i \leq k}$ tel que CHAINE$((A_i, B_i)_{i \leq k})$ avec $B_k = \emptyset$, $A_0 + B_0 =_s r(T_\Sigma)$ et $Card(A_0 + B_0) \leq n$.

Par le lemme 3.8, la suite $(A_i, B_i)_{i \leq k}$ peut être trouvée telle que tous ses éléments soient différents.

Selon le lemme 3.3, pour tout $i \in \{2, \ldots, k\}$, et tout H' tel que $\mathbf{P}(H', A_i, B_i)$, il existe H de taille $Card(H) \leq Card(H') + Card(A_{i-1})$ telle que $\mathbf{P}(H, A_{i-1}, B_{i-1})$. Nous en déduisons qu'il existe F_0 telle que $\mathbf{P}(F_0, A_0, B_0)$ et

$$Card(F_0) \leq Card(F_k) + \sum_{i=1}^{k} Card(A_i).$$

D'après la définition 3.6, chaque couple (A_i, B_i) est de taille inférieure à la taille de (A_{i-1}, B_{i-1}). Puisque $Card(A_0 + B_0) \leq n$, alors :

$$\forall i \in \{1, \ldots, k\} \; Card(A_i + B_i) \leq n.$$

En conséquence, k est inférieur au nombre de couples de multi-ensembles d'états de taille inférieure à n.

Le nombre de couples de multi-ensembles (A, B) tel que $Card(A) = p$ et $Card(B) = q$ est majoré par $n^p \times n^q$. Nous obtenons une majoration pour k :

$$\begin{array}{rcl} k & \leq & \sum_{p=0}^{n} \sum_{q=0}^{p} n^{p+q} \\ & \leq & \sum_{p=0}^{n} \sum_{q=0}^{p} n^{n} \\ & \leq & n^{n+2} \end{array}$$

Il existe alors une forêt F de taille $Card(F_0) \leq n^{n+2} \times n + Card(F_k)$. Mais puisque F_k ne contient que des constantes de Σ, alors $Card(F_k) \leq n$, donc $Card(F_0) = \mathcal{O}(n^{n+3})$.

Finalement,

$$K = \mathcal{O}(n^{n+3}).$$

□

Nous sommes maintenant en mesure de prouver

Théorème 3.1 *Le problème du vide est décidable dans la classe des automates d'ensembles d'arbres.*

Preuve.

Soit $\mathcal{A}$ un automate d'ensembles d'arbres à n états. D'après le lemme 3.2, l'examen de toutes les candidats sur toutes les forêts closes de taille inférieure à K, donne toutes les images possibles de calculs sur $\mathcal{A}$.

Il suffit maintenant de vérifier qu'un candidat γ sur une forêt close F atteignant $\omega = \gamma(F)$ est prolongeable en un calcul d'image ω. Pour cela, nous vérifions la propriété $slCOND_\omega$ suivante :

$$\forall b \in \Sigma_p \; \forall \mathbf{q}_1, \ldots, \mathbf{q}_p \in \omega \; \exists b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \rightarrow \mathbf{q} \in \mathcal{S} \text{ avec } \mathbf{q} \in \omega.$$

□

3.3 TSA et n-uplets de Langages Réguliers

3.3.1 Calculs Réguliers

Cette section est dédiée à la définition et l'étude de calculs réguliers. Intuitivement un calcul ρ est régulier s'il peut être finiment défini. Dans ce cas le langage accepté $\mathcal{L}(\mathcal{A}, \rho)$ est toujours un n-uplet de langages réguliers. Nous prouvons qu'un ensemble non vide reconnaissable par un TSA contient toujours un n-uplet de langages réguliers. Cette propriété se rapporte à celle de l'existence d'un langage régulier dans une forêt Rabin reconnaissable non vide.

Définition 3.7 Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ un TSA. Un calcul ρ dans $\mathcal{A}$ est *régulier* si il existe un ensemble fini F, une application f de T_Σ dans F et une application g de F dans $\mathcal{Q}$ telles que $\rho = g \circ f$ et :

$$f(t_1) = f(t_2) \Rightarrow f(u(t_1)) = f(u(t_2)) \text{ pour tout contexte } u \qquad (3.2)$$

L'ensemble des calculs réguliers de $\mathcal{A}$ est noté $\mathcal{RR}_\mathcal{A}$. L'ensemble des calculs réguliers réussis est $\mathcal{RSR}_\mathcal{A}$.

Exemple 3.8 Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ un TSA avec $\Sigma = \{a, b\}$, $\mathcal{Q} = \{\mathbf{q}_1, \mathbf{q}_2\}$, $\mathcal{F} = F = \{\mathbf{q}_1\}$ et $\mathcal{S}$, l'ensemble des règles suivantes :

$$\begin{array}{rcl} a & \rightarrow & \mathbf{q}_1 \\ b(\mathbf{q}_1) & \rightarrow & \mathbf{q}_1 \mid \mathbf{q}_2 \\ b(\mathbf{q}_2) & \rightarrow & \mathbf{q}_1 \mid \mathbf{q}_2 \end{array}$$

Soit r le calcul défini par $r(t) = \mathbf{q}_1$ si $t \in \{b^i(a) \mid i \leq 1\}$ et $r(t) = \mathbf{q}_2$ sinon.

$$\begin{array}{l} r(a) = \mathbf{q}_1, \quad r(b(a)) = \mathbf{q}_1, \\ r(b^2(a)) = \mathbf{q}_2, \quad r(b^3(a)) = \mathbf{q}_2 \quad \ldots \end{array}$$

Le langage accepté par r est régulier. C'est le langage $\{a, b(a)\}$. Le calcul r est régulièrement défini par l'ensemble F et les applications g et f tels que :

$$F = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3\} \; ,$$

$$\begin{array}{rcll} f : T_\Sigma & \rightarrow & F & \\ a & \mapsto & \mathbf{s_1} & \\ b(a) & \mapsto & \mathbf{s_2} & \\ t & \mapsto & \mathbf{s_3} & \forall t \in T_\Sigma \setminus \{a, b(a)\} \end{array} \qquad \begin{array}{rcl} g : F & \rightarrow & \mathcal{Q} \\ \mathbf{s_1} & \mapsto & \mathbf{q_1} \\ \mathbf{s_2} & \mapsto & \mathbf{q_1} \\ \mathbf{s_3} & \mapsto & \mathbf{q_2} \end{array}$$

La proposition suivante donne une caractérisation des calculs réguliers en termes d'automates d'arbres ascendants.

Proposition 3.2 *Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ un TSA. Soit ρ un calcul dans $\mathcal{A}$. Alors ρ est un calcul régulier si et seulement si il existe un automate d'arbres ascendant complet et déterministe $A = (\Sigma, Q_A, Q_f, S_A)$ et une application h de Q_A dans $\mathcal{Q}$ vérifiant :*

$$\forall b \in \Sigma_p \; b(q_1, \ldots, q_p) \rightarrow q \in S_A \Rightarrow b(h(q_1), \ldots, h(q_p)) \rightarrow h(q) \in \mathcal{S}. \qquad (3.3)$$

et tels que $\forall t \in T_\Sigma \quad \rho(t) = h(q_t)$ où q_t est l'état défini par $t \overset{*}{\underset{S}{\mapsto}} q_t$.

Preuve.

$\Leftarrow$ Nous prouvons que si $A = (\Sigma, Q_A, Q_f, S_A)$ est un automate d'arbres ascendant complet et déterministe et si h est une application de Q_A dans $\mathcal{Q}$ vérifiant la condition 3.3 alors l'application ρ de T_Σ dans $\mathcal{Q}$ qui associe à tout terme t l'état $h(q_t)$ où q_t est défini par $t \overset{*}{\underset{S}{\mapsto}} q_t$ est un calcul régulier.

Montrons que ρ est un calcul dans $\mathcal{A}$. En effet, puisque A est complet et déterministe, l'état q_t est défini pour tout t de T_Σ. Nous en déduisons que ρ est bien une application de T_Σ dans $\mathcal{Q}$. Cette application est compatible avec les règles de $\mathcal{S}$ car la propriété 3.3 (équation 3.4) et la complétude de A (équation 3.5) impliquent la propriété 3.1 :

$$\begin{array}{rll} & \forall t_1, \ldots, t_p \in T_\Sigma \quad b(q_{t_1}, \ldots, q_{t_p}) \rightarrow q_{b(t_1,\ldots,t_p)} \in S_A & (3.4) \\ \Rightarrow & \forall t_1, \ldots, t_p \in T_\Sigma \quad b(h(q_{t_1}), \ldots, h(q_{t_p})) \rightarrow h(q_{b(t_1,\ldots,t_p)}) \in \mathcal{S} & (3.5) \\ \Rightarrow & \forall t_1, \ldots, t_p \in T_\Sigma \quad b(\rho(t_1), \ldots, \rho(t_p))) \rightarrow \rho(b(t_1, \ldots, t_p))) \in \mathcal{S} & \end{array}$$

Le calcul ρ est régulier. Il est défini par l'ensemble fini $F = Q_A$, l'application f de T_Σ dans Q_A qui a tout terme t associe q_t et l'application g égale à h. Clairement $\rho = g \circ f$, et la condition 3.2 est vérifiée car l'automate d'arbres A est déterministe: pour tout membre gauche de règle, $b(q_1, \ldots, q_p)$ il n'existe qu'un seul état q tel que $b(q_1, \ldots, q_p) \rightarrow q$ soit une règle de S_A, ceci implique que pour tout contexte u, $q_{t_1} = q_{t_2} \Rightarrow q_{u(t_1)} = q_{u(t_2)}$. Nous avons donc :

$$\begin{array}{rrcl} & f(t_1) & = & f(t_2) \\ \Leftrightarrow & q_{t_1} & = & q_{t_2} \\ \Rightarrow & q_{u(t_1)} & = & q_{u(t_2)} \\ \Rightarrow & f(u(t_1)) & = & f(u(t_2)) \end{array}$$

Le calcul ρ est donc un calcul régulier.

$\Rightarrow$ Pour prouver l'implication inverse, nous devons montrer que si ρ est un calcul régulier caractérisé par l'ensemble F et les applications f et g de la définition 3.7 , alors on peut construire un automate d'arbres complet et déterministe et une application h qui vérifient la propriété 3.3.

Soit $A = (\Sigma, Q_A, Q_A, S_A)$ un automate ascendant d'arbres où :

- $Q_A = f(T_\Sigma)$;
- S_A est l'ensemble des règles satisfaisant :

$$b(q_1,\ldots,q_p) \rightarrow q \in S_A \Leftrightarrow \exists t_1,\ldots,t_p \in T_\Sigma \begin{cases} f(b(t_1,\ldots,t_p)) = q \\ f(t_1) = q_1,\ldots,f(t_p) = q_p \end{cases}$$

Montrons par l'absurde que A est déterministe. Supposons qu'il existe deux règles $b(q_1,\ldots,q_p) \rightarrow q \mid q'$ dans l'ensemble S_A. Il existe donc $u_1,\ldots,u_p$ et $t_1,\ldots,t_p$ tels que pour tout i, $f(t_i) = f(u_i)$ et $f(b(t_1,\ldots,t_p)) \neq f(b(u_1,\ldots,u_p))$. Or ceci est impossible puisque d'après la condition 3.2 de la définition 3.7 nous avons,

$$\begin{array}{rcll} f(b(t_1,\ldots,t_p)) & = & f(b(u_1,t_2\ldots,t_p)) & \text{car } f(t_1) = f(u_1) \\ & = & f(b(u_1,u_2,t_3\ldots,t_p)) & \text{car } f(t_2) = f(u_2) \\ & \vdots & \vdots & \vdots \\ & = & f(b(u_1,\ldots,u_p)) & \text{car } f(t_p) = f(u_p) \end{array}$$

A est donc déterministe. Pour montrer que A est complet, il suffit de remarquer que f est une application de T_Σ dans Q_A. Pour tout terme, il existe donc une image par f et par conséquent, pour tout membre gauche de règle $b(q_1,\ldots,q_p)$ il existe un état q tel que $b(q_1,\ldots,q_p) \rightarrow q$ soit une règle de S_A.

Remarque: la définition de S_A implique que $f(t) = q_t$. Cette égalité est facilement prouvée par induction sur la taille de t . Si t est une constante a, alors il existe une règle $a \rightarrow q$ si et seulement si $f(a) = q$. Donc $q_a = f(a)$. Supposons que $f(t) = q_t$ pour tout t de hauteur inférieure à k et soit t de hauteur k. Alors $t = b(t_1,\ldots,t_p)$ et par les hypothèse d'induction chaque t_i vérifie $f(t_i) = q_{t_i}$. Maintenant, il existe une seule règle de membre gauche $b\left(q_{t_1},\ldots,q_{t_p}\right)$ puisque A est déterministe et c'est $b\left(q_{t_1},\ldots,q_{t_p}\right) \rightarrow f(t)$. Donc $q_t = f(t)$.

Soit h définie par $h(q_t) = \rho(t)$ pour tout t de T_Σ. Pour terminer la preuve de la proposition 3.2, nous montrons que h est une application de Q_A dans Q qui vérifie la condition 3.3. Par la remarque précédente $h(q_t) = h(f(t))$, donc $\rho(t) = h(f(t))$. ρ et f sont des applications donc h est une application de Q_A dans Q. Par définition de S_A,

$$\begin{aligned} & \forall b \in \Sigma_p \quad b(q_1, \ldots, q_p) \rightarrow q \in S_A \\ \Leftrightarrow \quad & \exists t_1, \ldots, t_p \quad f(t_i) = q_i \; f(b(t_1, \ldots, t_p)) = q \end{aligned}$$

Donc $b\left(q_{t_1}, \ldots q_{t_p}\right) \rightarrow q_{b(t_1,\ldots t_p)} \in S_A$. Or l'application $\rho = h \circ f$ est un calcul. Elle est donc compatible avec les règles de $\mathcal{S}$ et $\forall b \in \Sigma_p$ et $\forall b(t_1, \ldots, t_p) \in T_\Sigma$,

$$\begin{aligned} & b(h(f(t_1)), \ldots, h(f(t_p))) \rightarrow h(f(b(t_1, \ldots, t_p))) \in \mathcal{S} \\ \Rightarrow \quad & b\left(h(q_{t_1}), \ldots, h\left(q_{t_p}\right)\right) \rightarrow h\left(q_{b(t_1,\ldots,t_p)}\right) \in \mathcal{S} \end{aligned}$$

□

Calculs et Langages Réguliers

Nous donnons maintenant trois propositions qui relient les calculs réguliers et les n-uples de langages réguliers.

Proposition 3.3 *Si ρ est un calcul régulier réussi dans un TSA $\mathcal{A}$, alors $\mathcal{L}(\mathcal{A}, \rho)$ est un n-uplet de langages réguliers.*

Preuve. Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S})$ un *TSA* et ρ un calcul régulier réussi dans $\mathcal{A}$. Alors, $\mathcal{L}(\mathcal{A}, \rho) = (L_1(\mathcal{A}, \rho), \ldots, L_n(\mathcal{A}, \rho))$ où chaque $L_i(\mathcal{A}, \rho)$ est défini par $L_i(\mathcal{A}, \rho) = \{t \mid \rho(t) \in F_i\}$. Selon la propriété 3.2, il existe un automate ascendant d'arbres déterministe et complet $A = (\Sigma, Q_A, Q_f, S_A)$ et une application $h : Q_A \rightarrow \mathcal{Q}$ vérifiant la propriété (3.3) tels que $L_i(\mathcal{A}, \rho) = \{t \mid h(q_t) \in F_i\} = \{t \mid q_t \in h^{-1}(F_i)\}$. Donc, pour tout i, $L_i(\mathcal{A}, \rho)$ est le langage d'arbres régulier reconnu par l'automate $(\Sigma, Q_A, h^{-1}(F_i), S_A)$. □

La réciproque est fausse : un n-uple de langages réguliers accepté peut être accepté par un calcul non régulier. Nous pouvons donner un exemple basé sur l'idée que deux états différents peuvent jouer un rôle identique. Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ avec $\Sigma = \{a, b()\}$, $\mathcal{Q} = \{\mathbf{q}_1, \mathbf{q}_2\}$, $\mathcal{F} = \mathcal{Q}$, $\Omega = 2^{\mathcal{Q}}$ et $\mathcal{S}$ est $a \rightarrow \mathbf{q}_1 \mid \mathbf{q}_2$; $b(\mathbf{q}_1) \rightarrow \mathbf{q}_1 \mid \mathbf{q}_2$; $b(\mathbf{q}_2) \rightarrow \mathbf{q}_1 \mid \mathbf{q}_2$. Quel que soit le calcul r dans $\mathcal{A}$, le langage accepté par r est régulier et $\mathcal{L}(\mathcal{A}, r) = T_\Sigma$. Or, le calcul qui associe l'état $\mathbf{q}_1$ au terme t si et seulement si t commence par un nombre premier de b n'est pas régulier. Néanmoins,

Proposition 3.4 *Si r est un calcul réussi dans un TSA $\mathcal{A}$ et $\mathcal{L}(\mathcal{A}, r)$ est un n-uplet de langages réguliers, alors il existe un calcul régulier réussi ρ tel que $\mathcal{L}(\mathcal{A}, r) = \mathcal{L}(\mathcal{A}, \rho)$.*

Preuve. Soient r un calcul réussi dans le *TSA* $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, (F_1, \ldots, F_n))$ et $(L_1, \ldots, L_n)$ le n-uplet de langages réguliers accepté par r. Dans le but de montrer l'existence d'un calcul régulier réussi ρ acceptant le même n-uplet $(L_1, \ldots, L_n)$, nous construisons un automate ascendant d'arbres A et une application h correspondants à la caractérisation donnée dans la proposition 3.2. Dans un premier temps, nous donnons la définition de A et nous prouvons qu'il est déterministe et complet. Ensuite, nous décrivons la fonction h et nous montrons qu'elle vérifie bien la condition (3.3) de la propriété 3.2 (fait 3.3). Le fait technique 3.4 permet de conclure à l'égalité des ensembles acceptés par ρ et r (Fait 3.5). Finalement, le fait 3.6 indique que le calcul ρ ainsi défini est un calcul réussi.

Soit F une forêt finie et close telle que $r(T_\Sigma) = r(F)$. Pour tout i, $1 \leq i \leq n$, le langage L_i est régulier donc reconnaissable. Considérons pour chaque i un automate déterministe et complet $A_i = (\Sigma, V_i, V_{i_f}, S_i)$ reconnaissant L_i. Appelons $A_{n+1} = (\Sigma, V_{n+1}, V_{n+1_f}, S_{n+1})$ un automate complet et déterministe reconnaissant F tel que

$$\forall t, t' \in F \quad (t \overset{*}{\mapsto} q \wedge t' \overset{*}{\mapsto} q') \Rightarrow q \neq q'$$

$$\begin{array}{rccl} \text{Soit } \phi : & T_\Sigma & \rightarrow & \mathcal{Q} \times V_1 \times \ldots \times V_n \times V_{n+1} \\ & t & \mapsto & (r(t), v_1, \ldots, v_{n+1}) \quad \text{où} \quad \forall i \in \{1, \ldots, n+1\}\ t \underset{S_i}{\overset{*}{\mapsto}} v_i. \end{array}$$

Soit $A = (\Sigma, Q_A, Q_A, S_A)$ avec :

- $Q_A = \phi(T_\Sigma)$. Pour tout état q de Q_A, soit t_q un terme tel que $\phi(t_q) = q$.
- L'ensemble S_A des règles de A est

$$S_A = \{b(q_1, \ldots, q_p) \rightarrow q \mid q = \phi(b(t_{q_1}, \ldots, t_{q_p})) \text{ avec } q_i \in Q_A,\ 1 \leq i \leq p, q \in Q_A, b \in \Sigma_p\}$$

Puisque un seul terme t_q est associé à un état q, l'automate ainsi obtenu est déterministe et complet.

$$\begin{array}{rccl} \text{Soit } h : & Q_A & \rightarrow & \mathcal{Q} \\ & (\mathbf{q}, v_1, \ldots, v_{n+1}) & \mapsto & \mathbf{q}. \end{array}$$

Remarquons que $q = \phi(t) \Rightarrow h(q) = r(t)$. En particulier, $h(q) = r(t_q)$

Fait 3.3 *h vérifie la condition (3.3) de la proposition 3.2.*

Preuve. D'après la définition de S_A, pour tout $b \in \Sigma_p$ si $b(q_1, \ldots, q_p) \rightarrow q$ est une règle de l'automate A, alors $q = \phi(b(t_{q_1}, \ldots, t_{q_p}))$ et $h(q) = r\left(b(t_{q_1}, \ldots, t_{q_p})\right)$. Le calcul r est compatible avec les règles de $\mathcal{S}$ donc, $b\left(r(t_{q_1}), \ldots, r\left(t_{q_p}\right)\right) \rightarrow r\left(b(t_{q_1}, \ldots, t_{q_p})\right)$ est une règle de $\mathcal{S}$. De plus, $r(t_{q_i}) = h(q_i)$ pour tout i, $1 \leq i \leq p$. Nous en déduisons, $\forall b \in \Sigma_p$

$$\begin{array}{rl} & b\left(q_1,\ldots,q_p\right) \to q \qquad \in S_A \\ \Rightarrow & b\left(h\left(q_1\right),\ldots,h\left(q_p\right)\right) \to h\left(q\right) \quad \in \mathcal{S} \end{array}$$

□

Nous définissons maintenant le calcul régulier ρ par $\rho(t) = h(q_t)$, $\forall t \in T_\Sigma$. Prouvons que $\mathcal{L}(\mathcal{A},\rho) = \mathcal{L}(\mathcal{A},r)$. Pour tout i, $1 \leq i \leq n+1$, soit $\approx_i$ la relation d'équivalence associée à chaque A_i et définie par $(t \approx_i t') \Leftrightarrow \left(t \overset{*}{\underset{S_i}{\mapsto}} v \text{ et } t' \overset{*}{\underset{S_i}{\mapsto}} v\right)$. Alors,

Fait 3.4 $(t \overset{*}{\underset{S_A}{\mapsto}} q) \Rightarrow (\forall i \quad t \approx_i t_q)$.

Preuve. Procédons par induction sur la hauteur de t.

- $h(t) = 1$. Le terme t est une constante que nous nommons a. Supposons $a \underset{S_A}{\mapsto} q$. Alors, la règle de S_A utilisée pour cette dérivation est $a \to q$ avec $q = \phi(a)$. Puisque $\phi(t_q) = q$, nous avons $\phi(a) = \phi(t_q)$.
  Le fait est maintenant évident car, si $\phi(a) = (r(a), v_1, \ldots, v_{n+1})$, alors $a \overset{*}{\underset{S_i}{\mapsto}} v_i$ et $t_q \overset{*}{\underset{S_i}{\mapsto}} v_i$. Donc, $\forall i \quad t \approx_i t_q$.

- Supposons la propriété vraie pour tout terme de hauteur inférieure à k et soit $t = b(t_1,\ldots,t_p)$ de hauteur k. L'automate A est complet et déterministe donc il existe une dérivation $t \overset{*}{\underset{S_A}{\mapsto}} b(q_1,\ldots,q_p) \underset{S}{\mapsto} q$. Si, pour tout i, $t_i \overset{*}{\underset{S}{\mapsto}} q_i$, alors, selon les hypothèses d'induction $\forall j \quad t_i \approx_j t_{q_i}$. Par la dernière règle $b(q_1,\ldots,q_p) \to q$, utilisée, nous avons

  $$(b(q_1,\ldots,q_p) \to q \in S_A) \Rightarrow q = \phi(b(t_{q_1},\ldots,t_{q_p}))$$

  Or, t_q est tel que $\phi(t_q) = q$. Donc, $\phi(t_q) = \phi(b(t_{q_1},\ldots,t_{q_p}))$. D'après les hypothèses d'induction $\forall j \quad t_i \approx_j t_{q_i}$ donc

  $$\forall j \quad b(t_{q_1},\ldots,t_{q_p}) \approx_j b(t_1,\ldots,t_p)$$

  Finalement, $\forall i \quad t_q \approx_i t$. □

Fait 3.5 $\mathcal{L}(\mathcal{A},\rho) = \mathcal{L}(\mathcal{A},r)$.

Preuve. Nous avons $\mathcal{L}(\mathcal{A},r) = (L_1,\ldots,L_n)$. Il faut donc montrer :

$$\forall i \in \{1,\ldots,n\} \quad t \in L_i \Leftrightarrow \left(t \overset{*}{\underset{S}{\mapsto}} q \quad h(q) \in F_i\right).$$

Soit $t \in L_i$. Supposons $t \overset{*}{\underset{S}{\mapsto}} q$ alors d'après le fait précédent, $\forall i \in \{1,\ldots,n\}$ $t_q \approx_i t$. Nous avons alors,

$$\forall i \in \{1,\ldots,n\} \quad t \in L_i \Leftrightarrow t_q \in L_i.$$

De plus, selon la définition 3.4 du langage reconnu par un calcul

$$\forall i \in \{1,\ldots,n\} \quad t_q \in L_i \Leftrightarrow r(t_q) \in F_i.$$

Nous pouvons maintenant conclure puisque $r(t_q) = h(q)$ et

$$r(t_q) \in F_i \Leftrightarrow h(q) \in F_i.$$

□

Fait 3.6 *ρ est un calcul réussi de $\mathcal{A}$.*

Preuve. Nous allons montrer que $\rho(T_\Sigma)$ est exactement $r(T_\Sigma)$.

Remarquons premièrement que:

$$\forall t \in F, \forall t' \in T_\Sigma \quad q_t = q_{t'} \Leftrightarrow t = t'.$$

En effet, l'automate A_{i+1} est défini de telle sorte que deux termes différents de F font aboutir A_{i+1} dans deux états différents. Nous en déduisons donc:

$$\forall t \in F \quad t_{q_t} = t.$$

Maintenant,

$$\forall t \in F, \rho(t) = h(q_t) = r(t_{q_t}) = r(t).$$

Nous avons alors $r(T_\Sigma) \subseteq \rho(T_\Sigma)$. L'inclusion inverse est facilement vérifiée puisque que $h(Q_A) = r(T_\Sigma)$. □

Proposition 3.5 *Soit $(L_1,\ldots,L_n)$ un n-uplet de langages réguliers et $\mathcal{A}$ un TSA. L'appartenance à $\mathcal{L}(\mathcal{A})$ de $(L_1,\ldots,L_n)$ est décidable.*

Preuve.

Si le n-uplet de langages réguliers $(L_1,\ldots,L_n)$ est reconnu par le *TSA* $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$, il existe un calcul régulier ρ qui accepte $(L_1,\ldots,L_n)$. La proposition précédente donne une construction pour un automate A et une application h qui, selon la propriété 3.2, caractérisent le calcul ρ.

Rappelons la définition de l'ensemble Q_A des états de l'automate A. Soit F une forêt finie et close telle que $r(T_\Sigma) = r(F)$. Soit pour chaque i un automate déterministe et complet $A_i = (\Sigma, V_i, V_{i_f}, S_i)$ reconnaissant L_i.

$A_{n+1} = (\Sigma, V_{n+1}, V_{n+1_f}, S_{n+1})$ est un automate complet et déterministe reconnaissant F tel que:

$$\forall t, t' \in F \quad (t \overset{*}{\mapsto} q \wedge t' \overset{*}{\mapsto} q') \Rightarrow q \neq q'$$

Alors, $Q_A = \phi(T_\Sigma)$ où ϕ est une application de T_Σ dans $Q \times V_1 \times \ldots \times V_{n+1}$

D'après la preuve de décision du vide dans la classe des TSA (voir page 59), toute image de calcul peut être exactement atteinte sur un domaine borné: une forêt close de taille inférieure à une constante K. Dans la preuve de la proposition précédente, étant donné un calcul r, une forêt close et finie vérifiant $r(F) = r(T_\Sigma)$ intervient dans la construction de l'automate qui définit le calcul régulier ρ équivalent à r. Cette forêt peut donc être choisie de taille inférieure à K.

En conséquence, l'automate A peut être construit avec un nombre d'états borné, n'excédant pas $Card(Q) \times Card(V_1) \times \cdots \times Card(V_n) \times K$. Le nombre de ces automates est alors fini ainsi que le nombre d'applications h de Q_A dans Q.

Finalement, nous pouvons construire ces automates et ces applications et tester si les calculs réguliers correspondants acceptent le n-uplet de langages $(L_1, \ldots, L_n)$. □

Nous résumons les trois propositions précédentes dans le

Théorème 3.2 *Soit $\mathcal{A}$ un TSA.*

1. *Si ρ est un calcul régulier réussi alors $\mathcal{L}(\mathcal{A}, \rho)$ est un n-uplet de langages réguliers.*

2. *Si r est un calcul réussi dans un TSA $\mathcal{A}$ et $\mathcal{L}(\mathcal{A}, r)$ est un n-uplet de langages réguliers, alors il existe un calcul régulier réussi ρ tel que $\mathcal{L}(\mathcal{A}, r) = \mathcal{L}(\mathcal{A}, \rho)$.*

3. *Soit $(L_1, \ldots, L_n)$ un n-uplet de langages réguliers. L'appartenance à $\mathcal{L}(\mathcal{A})$ de $(L_1, \ldots, L_n)$ est décidable.*

Un Calcul Régulier dans tout TSA

Dans toute forêt Rabin reconnaissable non vide, il existe un arbre infini régulier. Dans le cas des Automates d'Ensembles d'Arbres nous obtenons un résultat apparenté: toute famille non vide d'ensembles d'arbres reconnue par un *TSA* contient un ensemble régulier d'arbres.

Définition 3.8 Soient $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S})$ un TSA, r un calcul dans $\mathcal{A}$ et L une forêt close finie. Un *prolongement régulier* de r selon L est un calcul régulier ρ dans $\mathcal{A}$ qui vérifie $r(L) = \rho(L)$.

Nous donnons maintenant une construction d'un prolongement régulier dans le cas où la forêt L est telle que $r(L) = r(T_\Sigma)$. Nous utilisons la proposition 3.2, i.e., la caractérisation de calculs réguliers à l'aide d'automates d'arbres classiques.

Soit $A = (\Sigma, Q_A, Q_f, S_A)$ un automate ascendant d'arbres complet et déterministe avec $Q_A = \{s_t \mid t \in L\}$; $Q_f = Q_A$; pour tout membre gauche de règle possible, c'est-à-dire $\forall b(s_{t_1}, \ldots, s_{t_p})$ tel que $b \in \Sigma_p$ et $s_{t_1}, \ldots, s_{t_p} \in Q_A$:

- si $t = b(t_1, \ldots, t_p) \in L$ alors

$$(b(s_{t_1}, \ldots, s_{t_p}) \rightarrow s_t) \in S_A$$

- sinon, il existe au moins $u \in L$ tel que $r(u) = r(t)$,car $r(L) = r(T_\Sigma)$ et on pose :

$$(b(s_{t_1}, \ldots, s_{t_p}) \rightarrow s_u) \in S_A.$$

Par construction, cet automate est complet et déterministe. Soit h une application de Q_A dans $\mathcal{Q}$ qui vérifie $h(s_t) = r(t)$. Alors, la propriété (3.3) de la proposition 3.2 est satisfaite. En effet, si

$$(b(s_{t_1}, \ldots, s_{t_p}) \rightarrow s) \in S_A$$

alors sachant que $b(h(s_{t_1}), \ldots, h(s_{t_p})) = b(r(t_1), \ldots, r(t_p))$, $h(s) = r(b(t_1, \ldots, t_p))$, et sachant que r est un calcul, nous avons :

$$(b(r(t_1), \ldots, r(t_p) \rightarrow r(b(t_1, \ldots, t_p)) \in \mathcal{S}$$

Notons q_t l'état tel que $t \overset{*}{\underset{S}{\mapsto}} q_t$. D'après la proposition 3.2, le calcul ρ défini par $\rho(t) = h(q_t)$ pour tout t de T_Σ est un calcul régulier. Il vérifie $\rho(L) = r(L)$, c'est donc un prolongement régulier de r selon L.

Proposition 3.6 *Il existe un n-uplet de langages réguliers dans un ensemble non vide reconnu par un TSA.*

Preuve. Soient $\mathcal{A}$ un *TSA* et r un calcul réussi dans $\mathcal{A}$. Soit L une forêt close et finie telle que $r(L) = r(T_\Sigma)$, alors un prolongement régulier de r selon L par la construction décrite ci-dessus est un calcul régulier réussi dans $\mathcal{A}$. □

3.3.2 Propriétés Topologiques

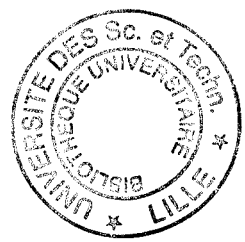
Dans cette section, nous munissons l'ensemble des n-uplets de langages d'arbres d'une métrique d. Les propriétés que nous montrons ne sont pas valides pour tous les types d'automates d'ensembles d'arbres. Dans le cas d'un *TSA* sans condition d'acceptation ($\Omega = 2^{\mathcal{Q}}$), nous prouvons que, dans l'espace métrique $(\mathcal{L}(\mathcal{A}), d)$, (i)

les rationnels sont denses et (ii) $\mathcal{L}(\mathcal{A})$ est compact. Dans le cas général, seule la première propriété reste vraie.

Nous déduisons ensuite des résultats sur la puissance d'expression des types de contraintes et sur l'équivalence de TSA.

Soient $\mathcal{L} = (L_1, \ldots, L_n)$ et $\mathcal{L}' = (L'_1, \ldots, L'_n)$ deux n-uplets de langages d'arbres. La différence symétrique, soit, $(L \setminus L') \cup (L' \setminus L)$ est notée $L \triangle L'$. Nous définissons la métrique d : par

$$\begin{gathered} d : (2^{T_\Sigma})^n \times (2^{T_\Sigma})^n \to \mathbb{N} \\ \text{si } \mathcal{L} = \mathcal{L}' \quad d(\mathcal{L}, \mathcal{L}') = 0 \\ \text{sinon} \quad d(\mathcal{L}, \mathcal{L}') = 2^{-n} \\ \text{avec } n = \min \{h(t) \mid t \in \mathrm{L}_i \triangle L'_i \; 1 \leq i \leq n\} \end{gathered}$$



Cette application d est bien une métrique car elle est symétrique et elle vérifie l'inégalité triangulaire : $\forall \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3 \in (2^{T_\Sigma})^n$, $d(\mathcal{L}_1, \mathcal{L}_2) \leq d(\mathcal{L}_1, \mathcal{L}_3) + d(\mathcal{L}_3, \mathcal{L}_2)$.

Soit $(\mathcal{L}_i)_{i>0}$ une suite de n-uplets de langages d'arbres. La *limite* $\lim \mathcal{L}_i$, si elle existe, est définie par le n-uplet de langages d'arbres $\mathcal{L}$ tel que $\lim_{i \to \infty} d(\mathcal{L}, \mathcal{L}_i) = 0$.

Rappelons qu'un automate d'ensembles d'arbres $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ est dit *sans condition d'acceptation* si $\Omega = 2^{\mathcal{Q}}$.

Proposition 3.7 *Soit $\mathcal{A}$ un TSA sans condition d'acceptation. L'ensemble $\mathcal{L}(\mathcal{A})$ est compact.*

Preuve. Soit $(\mathcal{L}_i)_{i \geq 0}$ une suite infinie de n-uplets de langages d'arbres reconnus par $\mathcal{A}$. Nous notons r_i le calcul dans $\mathcal{A}$ qui accepte le n-uplet $\mathcal{L}_i$. De cette suite $(r_i)_{i \geq 0}$, nous extrayons une autre suite suite infinie $(r_i^0)_{i \geq 0}$ de calculs ayant le même comportement sur T_Σ^0. C'est à dire, $\forall i \geq 0$ et $\forall t \in T_\Sigma^0$, $r_i^0(t) = r_0^0(t)$. Une telle suite existe puisque tout calcul ne peut prendre qu'un nombre fini de valeurs différentes sur $r(T_\Sigma^0)$.

Maintenant, de la suite infinie $(r_i^0)_{i \geq 0}$, nous extrayons une nouvelle suite de calculs $(r_i^1)_{i \geq 0}$ ayant le même comportement sur T_Σ^1, et ainsi de suite. Nous obtenons alors :

$$\begin{array}{ll} (r_i^0) & : r_0^0, r_0^1, \ldots, r_0^i, \ldots \\ (r_i^1) & : r_1^0, r_1^1, \ldots, r_1^i, \ldots \\ (r_i^j) & : r_j^0, r_j^1, \ldots, r_j^i, \ldots \\ \vdots & \quad \vdots \end{array}$$

Soit r l'application qui pour tout j associe à tout terme de T_Σ^j la valeur $r_j^j(t)$.

Remarquons que r est bien un calcul dans $\mathcal{A}$ puisque tout r_j^i est aussi un calcul. De plus, la suite $(\mathcal{L}(\mathcal{A}, r_i^i))_{i \geq 0}$ de n-uplets de langages d'arbres a pour limite

$\mathcal{L}(\mathcal{A}, r)$. Nous avons donc montré que de toute suite infinie de n-uplets de langages reconnus par un TSA, il est possible d'extraire une suite infinie convergente. □

L'ensemble des n-uplets de langages réguliers d'arbres sur l'alphabet Σ est noté REG_n.

Proposition 3.8 *Soit $\mathcal{A}$ un TSA de rang n. $\mathcal{L}(\mathcal{A}) \cap REG_n$ est dense dans $\mathcal{L}(\mathcal{A})$.*

Preuve. Soit $\mathcal{A}$ un *TSA* et soit $\mathcal{L}$ un langage reconnu par $\mathcal{A}$. Nous montrons alors qu'il existe une suite de n-uplets de langages réguliers $(\mathcal{L}_i)_{i>0}$ appartenant à $\mathcal{L}(\mathcal{A})$ telle que $\lim \mathcal{L}_i = \mathcal{L}$.

Soit r le calcul qui accepte $\mathcal{L}$ et notons $T_\Sigma^{\leq k}$ l'ensemble $\{t \in T_\Sigma \mid h(t) \leq k\}$.

L'ensemble des états de $\mathcal{A}$ est fini. Il existe donc une forêt close finie F telle que $r(F) = r(T_\Sigma)$. Soit x le plus petit entier tel que $T_\Sigma^{\leq k} \cap F = \emptyset$.

Soit $(\rho_j)_{j \geq k}$ la suite de calculs de $\mathcal{A}$ où chaque ρ_j est une extension régulière du calcul r selon $T_\Sigma^{\leq j}$ (voir en page 98). Puisque par définition, chaque extension régulière ρ_j vérifie $\rho(F) = r(F) = r(T_\Sigma)$, les éléments de $(\rho_j)_{j \geq k}$ sont des calculs réussis.

Appelons $\mathcal{L}_j$ le n-uplet $\mathcal{L}(\mathcal{A}, \rho_j)$. Nous considérons alors la suite de n-uplets de langages réguliers $(\mathcal{L}(\mathcal{A}, \rho_j))_{j \geq k}$. Pour tout j supérieur à k nous avons :

$$d(\mathcal{L}_j, \mathcal{L}) \leq 2^{-(j+1)}.$$

Donc, $\lim_{i \to \infty} d(\mathcal{L}, \mathcal{L}_i) = 0$ soit $\lim \mathcal{L}_j = \mathcal{L}$. □

Proposition 3.9 *Soit $\mathcal{A}$ et $\mathcal{A}'$ deux TSA de rang n sans condition d'acceptation.*

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}') \Leftrightarrow \mathcal{L}(\mathcal{A}) \cap REG_n = \mathcal{L}(\mathcal{A}') \cap REG_n.$$

Preuve. La partie de la preuve $\mathcal{L}(\mathcal{A}) \cap REG_n = \mathcal{L}(\mathcal{A}') \cap REG_n \Leftarrow \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ est directe. La seconde partie est conséquence directe des deux propositions précédentes. □

3.4 Propriétés de Clôture

Nous montrons que la classe $\mathcal{TSA}$ des langages reconnus par les automates d'ensembles d'arbres est close par union, intersection, cylindrification et projection. Toutes ces propriétés de clôture sont effectives. En d'autres termes, si $\mathcal{D}_1$ et $\mathcal{D}_2$ sont les deux n-uplets de langages reconnus par les *TSA* $\mathcal{A}_1$ et $\mathcal{A}_2$, alors on peut par exemple construire un *TSA* reconnaissant $\mathcal{D}_1 \cap \mathcal{D}_2$. Les opérations de cylindrification et de projection sont le plus souvent utilisées pour étendre l'union et l'intersection à des *TSA* de rangs différents.

Enfin, nous remarquons que $\mathcal{TSA}$ n'est pas close par complémentarité.

Avant tout, nous définissons l'union, l'intersection, la cylindrification et la projection de n-uplets d'ensembles d'arbres. Soient $\mathcal{D}_1$ et $\mathcal{D}_2$ deux n-uplets de langages et x un entier naturel compris entre 1 et n.

Intersection

$$\mathcal{D}_1 \cap \mathcal{D}_2 = \{(L_1, \ldots, L_n) \mid (L_1, \ldots, L_n) \in \mathcal{D}_1 \wedge (L_1, \ldots, L_n) \in \mathcal{D}_2\}.$$

Union

$$\mathcal{D}_1 \cup \mathcal{D}_2 = \{(L_1, \ldots, L_n) \mid (L_1, \ldots, L_n) \in \mathcal{D}_1 \vee (L_1, \ldots, L_n) \in \mathcal{D}_2\}.$$

Cylindrification

$$(L_1, \ldots, L_{n+1}) \in C_x(\mathcal{D}) \Leftrightarrow (L_1, \ldots, L_{x-1}, L_{x+1}, \ldots, L_{n+1}) \in \mathcal{D} \text{ et } L_x \subseteq T(\Sigma)$$

Projection

$$(L_1, \ldots, L_{n-1}) \in \Pi_x(\mathcal{D}) \Leftrightarrow \begin{array}{c} \exists L \subseteq T_\Sigma \quad \text{tel que} \\ (L_1, \ldots, L_{x-1}, L, L_x, \ldots, L_{n-1}) \in \mathcal{D}. \end{array}$$

Proposition 3.10 *La classe $\mathcal{TSA}$ est effectivement close par intersection.*

Preuve. Soient $\mathcal{A}_1 = (\Sigma, \mathcal{Q}_1, \mathcal{F}_1, \mathcal{S}_1), \Omega_1$ et $\mathcal{A}_2 = (\Sigma, \mathcal{Q}_2, \mathcal{F}_2, \mathcal{S}_2, \Omega_2)$ deux *TSA* de rang n. Nous allons construire $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$. Nous notons $\mathcal{F}_1 = (F_1^1, \ldots, F_n^1)$ et $\mathcal{F}_2 = (F_1^2, \ldots, F_n^2)$.

Soit le produit cartésien $\mathcal{Q}_1 \times \mathcal{Q}_2 = \{(\mathbf{q}_1, \mathbf{q}_2) \mid \mathbf{q}_1 \in \mathcal{Q}_1 \text{ et } \mathbf{q}_2 \in \mathcal{Q}_2\}$. Soit π_1 et π_2 les projections de $\mathcal{Q}_1 \times \mathcal{Q}_2$ respectivement sur $\mathcal{Q}_1$ et $\mathcal{Q}_2$.

Construisons le *TSA* $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ avec :

- $\mathcal{Q} = \{(\mathbf{q}_1, \mathbf{q}_2) \mid (\mathbf{q}_1 \in F_i^1) \Leftrightarrow (\mathbf{q}_2 \in F_i^2)\ \forall i \quad 1 \leq i \leq n\}$;
- $\mathcal{F} = (F_1, \ldots, F_n)$ où chaque $F_i = F_i^1 \times F_i^2$;
- $\Omega = \left\{\omega \in 2^{\mathcal{Q}} \mid \pi_1(\omega) \in \Omega_1 \text{ et } \pi_2(\omega) \in \Omega_2\right\}$;
- $\mathcal{S}$ est l'ensemble des règles qui vérifient:

$$b(\mathbf{s}_1, \ldots, \mathbf{s}_p) \to \mathbf{s} \in \mathcal{S} \Leftrightarrow \left\{ \begin{array}{l} b(\pi_1(\mathbf{s}_1), \ldots, \pi_1(\mathbf{s}_p)) \to \pi_1(\mathbf{s}) \in \mathcal{S}_1 \\ b(\pi_2(\mathbf{s}_1), \ldots, \pi_2(\mathbf{s}_p)) \to \pi_2(\mathbf{s}) \in \mathcal{S}_2 \end{array} \right. .$$

Correction de la construction. Prouvons que $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$. Supposons qu'il existe un calcul réussi r dans $\mathcal{A}$. Soit r_1 l'application définie par

$$r_1(t) = \pi_1(r(t)) \ .$$

Alors, r_1 est un calcul dans $\mathcal{A}_1$. En effet, r_1 est compatible avec les règles de $\mathcal{S}_1$ puisque pour tout terme $t = b(t_1, \ldots, t_p)$ de T_Σ il existe une règle

$$b(r(t_1), \ldots, r(t_p)) \rightarrow r(t) \in \mathcal{S} \ ,$$

et par la construction de $\mathcal{S}$, il existe une règle

$$b(\pi_1(r(t_1)), \ldots, \pi_1(r(t_p))) \rightarrow \pi_1(r(t)) \in \mathcal{S}_1 \ .$$

Le calcul r_1 est bien un calcul réussi car $r(T_\Sigma) = \omega \in \Omega$ et par définition de Ω, $\pi_1(\omega) \in \Omega_1$.

Maintenant, montrons que $\mathcal{L}(\mathcal{A}, r) = \mathcal{L}(\mathcal{A}_1, r_1)$. Par définition de $\mathcal{F}$ et de $\mathcal{Q}$, $(\mathbf{q}_1, \mathbf{q}_2) \in F_i$ si et seulement si $\mathbf{q}_1 \in F_i^1$ donc

$$\begin{aligned} t \in L_i(\mathcal{A}, r) &\Leftrightarrow \quad r(t) \in F_i \\ &\Leftrightarrow \quad \pi_1(r(t)) \in F_i^1 \\ &\Leftrightarrow \quad t \in L_i(\mathcal{A}_1, r_1). \end{aligned}$$

De la même façon, nous montrons que $r_2 = \pi_2(r)$ est un calcul réussi dans $\mathcal{A}_2$ et que $\mathcal{L}(\mathcal{A}, r) = \mathcal{L}(\mathcal{A}_2, r_2)$. Donc $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

Prouvons que $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) \subseteq \mathcal{L}(\mathcal{A})$. Si r_1 et r_2 sont respectivement des calculs réussis dans $\mathcal{A}_1$ et $\mathcal{A}_2$ et si $\mathcal{L}(\mathcal{A}_1, r_1) = \mathcal{L}(\mathcal{A}_2, r_2)$ alors, l'application de T_Σ dans $\mathcal{Q}_1 \times \mathcal{Q}_2$ définie par $\forall t \in T_\Sigma \quad r(t) = (r_1(t), r_2(t))$ est un calcul réussi dans $\mathcal{A}$. L'égalité $\mathcal{L}(\mathcal{A}, r) = \mathcal{L}(\mathcal{A}_1, r_1) = \mathcal{L}(\mathcal{A}_2, r_2)$ est alors directe. □

Proposition 3.11 *La classe TSA est effectivement close par union.*

Preuve. Soient $\mathcal{A}_1 = (\Sigma, \mathcal{Q}_1, \mathcal{F}_1, \mathcal{S}_1, \Omega_1)$ et $\mathcal{A}_2 = (\Sigma, \mathcal{Q}_2, \mathcal{F}_2, \mathcal{S}_2, \Omega_2)$ deux TSA de rang n. Nous allons construire $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$. Nous notons $\mathcal{F}_1 = (F_1^1, \ldots, F_n^1)$ et $\mathcal{F}_2 = (F_1^2, \ldots, F_n^2)$. Nous renommons les états des automates de façon à obtenir $\mathcal{Q}_1$ et $\mathcal{Q}_2$ disjoints.

Construisons l'automate d'ensembles d'arbres $\mathcal{A} = (\Sigma, \mathcal{Q}, (F_1, \ldots, F_n), \mathcal{S}, \Omega)$ avec

- $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$;
- $F_i = F_i^1 \cup F_i^2$;
- $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$;
- $\Omega = \Omega_1 \cup \Omega_2$.

Correction de la construction.

Elle est directe. Clairement, $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2) \subseteq \mathcal{L}(\mathcal{A})$. Tout calcul dans $\mathcal{A}_1$ ou $\mathcal{A}_2$ peut être fait dans $\mathcal{A}$ puisque $\mathcal{S}$ contient les règles de $\mathcal{S}_1$ et $\mathcal{S}_2$. L'inclusion inverse est assurée par la disjonction de $\mathcal{Q}_1$ et $\mathcal{Q}_2$. □

Proposition 3.12 *La classe TSA est close par cylindrification.*

Preuve. Nous allons montrer que, étant donnés un TSA $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ de rang n et un entier x compris entre 1 et n, nous pouvons construire un TSA $\mathcal{A}'$ de rang $n+1$ tel que $\mathcal{L}(\mathcal{A}') = \mathcal{C}_x(\mathcal{L}(CA))$.

Construisons $\mathcal{A}' = (\Sigma, \mathcal{Q}', (F'_1, \ldots, F'_{n+1}), \mathcal{S}', \Omega')$ où :

- $\mathcal{Q}' = \mathcal{Q} \times \{0,1\}$
- $(F'_1, \ldots, F'_{n+1})$ est le n-uple d'ensembles d'états finaux défini par :
  - $F'_i = F_i \times \{0,1\}$ si $i \in \{1, \ldots, x-1\}$,
  - $F'_x = \mathcal{Q} \times \{1\}$,
  - $F'_i = F_{i-1} \times \{0,1\}$ si $i \in \{x+1, \ldots, n+1\}$;
- $\mathcal{S}'$ est l'ensemble des règles telles que :

$$b((\mathbf{q}_1, \delta_1), \ldots, (\mathbf{q}_p, \delta_p)) \rightarrow (\mathbf{q}, \delta) \in \mathcal{S}' \Leftrightarrow b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \rightarrow \mathbf{q} \in \mathcal{S}.$$

- Ω' contient tous les ensembles de couples dont la projection sur la première composante donne un élément de Ω.

Correction de la construction.

Pour montrer que $\mathcal{C}_x(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}')$ nous prouvons qu'à tout calcul réussi r dans $\mathcal{A}$ et tout $(L_1, \ldots, L_{n+1}) \in \mathcal{C}_x(\mathcal{L}(\mathcal{A}, r))$, nous pouvons associer un calcul réussi r' de $\mathcal{A}'$ tel que $\mathcal{L}(\mathcal{A}', r') = (L_1, \ldots, L_{n+1})$.

Soit r' défini par :

$$r'(t) = (r(t), 0) \Leftrightarrow t \notin L_x \ ,$$
$$r'(t) = (r(t), 1) \Leftrightarrow t \in L_x \ .$$

Clairement, r' est un calcul dans $\mathcal{A}'$. En effet, c'est une application de T_Σ dans $\mathcal{Q}$. Cette application est compatible avec les règles de $\mathcal{S}$ car pour tout terme $t = b(t_1, \ldots, t_p)$, $r'(t) = (r(t), \delta)$, avec $\delta \in \{0,1\}$. Par construction de $\mathcal{S}'$, $b(r'(t_1), \ldots, r'(t_p)) \rightarrow r'(t) \in \mathcal{S}'$.

De plus, pour tout i dans $\{1,\ldots,x-1\}$, $L_i = L_i(\mathcal{A},r) = L_i(\mathcal{A}',r')$ car

$$\begin{aligned} \forall t \in T_\Sigma \quad t \in L_i(\mathcal{A},r) &\Leftrightarrow r(t) \in F_i \\ &\Leftrightarrow (r(t),1) \in F'_i \wedge (r(t),0) \in F'_i \\ &\Rightarrow r'(t) \in F'_i \end{aligned}$$

Le calcul r' est réussi car la projection de l'ensemble des couples de $r'(T_\Sigma)$ sur la première composante est $r(T_\Sigma) \in \Omega$.

De la même façon, nous remarquons que $\forall i \in \{x+1,\ldots,n+1\}$ $L_i = L_{i-1}(\mathcal{A},r) = L_i(\mathcal{A}',r')$ et finalement, d'après la définition de r', $L_x(\mathcal{A},r) = L_x$.

La propriété inverse, $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{C}_x(\mathcal{L}(\mathcal{A}))$, est vérifiée car à tout calcul r' dans $\mathcal{A}'$ nous pouvons associer un calcul r dans $\mathcal{A}$ défini par :

$$\forall t \in T_\Sigma \quad r'(t) = (\mathbf{q},\delta) \Rightarrow r(t) = \mathbf{q}$$

et un langage $L = L_x(\mathcal{A}',r')$ tels que :

$$\mathcal{L}(\mathcal{A}',r') = (L_1(\mathcal{A},r),\ldots,L,L_{x+1}(\mathcal{A},r),\ldots,L_n(\mathcal{A},r)) \ .$$

□

Proposition 3.13 *La classe $\mathcal{TSA}$ est effectivement close par projection.*

Preuve. Etant donnés un *TSA* $\mathcal{A} = (\Sigma,\mathcal{Q},(F_1,\ldots,F_n),\mathcal{S},\Omega)$ de rang n et un entier x compris entre 1 et n, nous pouvons construire un *TSA* $\mathcal{A}'$ tel que $\mathcal{L}(\mathcal{A}') = \Pi_x(\mathcal{L}(\mathcal{A}))$.

Soit $\mathcal{A}' = (\Sigma,\mathcal{Q},(F'_1,\ldots,F'_{n-1}),\mathcal{S},\Omega)$ où $F'_i = F_i$ si $1 \leq i < x$, $F'_i = F_{i+1}$ si $x \leq i < n$. La preuve de correction de cette construction est directe. □

Nous avons introduit dans le chapitre 2 les automates d'ensembles de mots, les *WSA*. Ce sont en fait des automates d'ensembles d'arbres filiformes. La classe des *WSA*-reconnaissables est une sous-classe de $\mathcal{TSA}$. Par la proposition 2.3, nous savons qu'elle n'est pas close par complémentarité.

Proposition 3.14 *La classe $\mathcal{TSA}$ n'est pas close par complémentarité.*

Preuve. L'ensemble $\mathcal{L}$ des langages L sur l'alphabet $\Sigma = \{a\}$ qui vérifient : $\exists k$ tel que $\forall n > k \quad a^n \in L$, est *WSA*-reconnaissable, donc *TSA*-reconnaissable. Son complémentaire n'est par contre, pas *TSA*-reconnaissable. La preuve est identique à celle du fait 2.2 page 52. □

Dans le but de définir une sous-classe des langages *TSA*-reconnaissables close par complémentarité, nous envisageons d'adapter les *TSA* et de reconnaître des graphes infinis $\{0,1\}^n$-valués. Nous pouvons voir de tels graphes comme les représentations de fonctions caractéristiques de n-uplets de langages d'arbres. Les règles de ces nouveaux automates comporteraient en membre gauche, un label de $\{0,1\}^n$ et seraient de la forme : $(label, symbole, etat_1,\ldots,etat_p) \rightarrow etat$.

Chapitre 4

Contraintes Ensemblistes

4.1 Définitions

4.1.1 Syntaxe des Expressions et Contraintes

Soit B un ensemble de symboles contenant l'union ($\cup$ d'arité 2), l'intersection ($\cap$ d'arité 2), le complémentaire ($\sim$ d'arité 1), le tout et le vide ($\top$ et $\bot$ d'arité 0). Soit Σ un alphabet gradué. Pour chaque lettre a dans Σ_i avec $i \geq 1$, et pour tout j dans $\{1, \ldots, i\}$, $a_{(j)}^{-1}$ est appelé *symbole de projection*. Chaque symbole de projection est unaire et Σ_π est l'ensemble des symboles de projection.

Les lettres de Σ sont représentées par des lettres minuscules $(a, b, c, d, \ldots)$.

Soit $\mathcal{X}$ un ensemble dénombrable de variables, représentées par des majuscules $X, Y, Z, \ldots$ Par $\Sigma + B + \Sigma_\pi$ nous désignons l'union disjointe des ensembles Σ, B et Σ_π.

Définition 4.1 Une *expression ensembliste* sur Σ est un terme de $T_{\Sigma+B+\Sigma_\pi}(\mathcal{X})$.

Les symboles $\cap$ et $\cup$ seront utilisés en notation infixée. Le plus souvent, le parenthésage des expressions autour des symboles d'union, d'intersection et de complémentaire sera simplifié. Pour cela, nous respecterons les règles de priorité de $\sim$ devant $\cap$ et $\cup$.

Exemple 4.1 Si $\Sigma = \{a, b\}$ et $\mathcal{X} = \{X, Y\}$ alors $\cap(b(\cup(a, X), \sim(Y)), X)$ est une expression ensembliste sur Σ que nous écrivons $b(a \cup X, \sim Y) \cap X$.

Définition 4.2 Un *système de contraintes ensemblistes* est une conjonction de *contraintes positives* de la forme $exp_1 \subseteq exp_2$ et de *contraintes négatives* de la forme $exp_1 \not\subseteq exp_2$ avec exp_1 et exp_2 des expressions ensemblistes.

L'ensemble des sous-expressions ensemblistes qui apparaissent dans exp, $E(exp)$, est inductivement défini par :

$$\begin{aligned} E(\bot) &= \{\bot\}; \\ E(\top) &= \{\top\}; \\ E(X) &= \{X\}; \\ E(op(e_1,\ldots,e_p)) &= \{op(e_1,\ldots,e_p)\} \cup (\bigcup_i E(e_i)); \end{aligned}$$

pour tout X de $\mathcal{X}$, et pour tout symbole op de $\Sigma + B + \Sigma_\pi$.

Si C est une contrainte positive $exp \subseteq exp'$, ou négative $exp \not\subseteq exp'$ alors l'ensemble des sous-expressions de C est $E(C) = E(exp) \cup E(exp')$.

Exemple 4.2 Soit C la contrainte $b(b(Y) \cap Z) \cup a \subseteq Z$. Alors $E(C)$ est l'ensemble composé des six expressions ensemblistes :

$$\{b(Y), Z, a, b(Y) \cap Z, b(b(Y) \cap Z), b(b(Y) \cap Z) \cup a\}$$

4.1.2 Classes Syntaxiques

Du fait des études dont elles ont fait l'objet, on a l'habitude de distinguer parmi les systèmes de contraintes certaines classes syntaxiques.

DSC La classe Definite Set Constraints a été étudiée par N. Heintze et J. Jaffar dans [HJ90a] et T. Früwirth, Shapiro, M.Y Vardi et E. Yardeni dans [FSVY91]. Une contrainte de la classe DSC est de la forme $exp \subseteq t$ où $t \in T_\Sigma(\mathcal{X})$ et exp est une expression ensembliste sans symbole de complémentaire.

PSC La classe Positive Set Constraints a été largement étudiée [GTT93a],[AW92], [BGW93],[AKVW93]. Une contrainte de la classe PSC ne comporte pas de symbole de projection, ni de contrainte négative. Elles s'écrivent donc sous la forme $exp_1 \subseteq exp_2$ avec exp_1 et exp_2 des termes de $T_{\Sigma+B}(\mathcal{X})$.

PNSC La classe Positive and Negative Set Constraints ([GTT93b],[AKVW93],[CP93]). Une contrainte de la classe PNSC est soit positive soit négative, mais ne comporte pas de symbole de projection : $exp_1 \subseteq exp_2$ ou $exp_1 \not\subseteq exp_2$ avec exp_1 et exp_2 des termes de $T_{\Sigma+B}(\mathcal{X})$.

4.1.3 Interprétations et Solutions

Soit SC un système de contraintes sur l'alphabet Σ et de variables dans $\mathcal{X}$. Une *interprétation* est une application qui associe à chaque variable une valeur dans 2^{T_Σ}, l'ensemble des parties de T_Σ.

$$I : \mathcal{X} \to 2^{T_\Sigma}.$$

Toute interprétation I des variables peut être étendue de façon à associer un ensemble de termes à une expression ensembliste. C'est cette extension χ_I qui donne une signification aux connecteurs de B.

$$\chi_I(\top) = T_\Sigma$$
$$\chi_I(\bot) = \emptyset$$
$$\chi_I(X) = I(X)$$
$$\chi_I(b(exp_1, \ldots, exp_p)) = \{b(t_1, \ldots, t_p) \mid t_j \in \chi_I(exp_j)\ \forall 1 \leq j \leq p\}$$
$$\chi_I\left(b_i^{-1}(exp)\right) = \{t_i \mid b(t_1, \ldots, t_p) \in \chi_I(exp)\}$$
$$\chi_I(\cap(exp_1, exp_2)) = \chi_I(exp_1) \cap \chi_I(exp_2)$$
$$\chi_I(\cup(exp_1, exp_2)) = \chi_I(exp_1) \cup \chi_I(exp_2)$$
$$\chi_I(\sim(exp)) = T_\Sigma \setminus \chi_I(exp)$$

Soit SC un système à n variables $X_1, \ldots, X_n$. Une interprétation $\mathcal{I}$ des n variables X_i est définie par le n-uplet $(\mathcal{I}(X_1), \ldots, \mathcal{I}(X_n))$.

Une interprétation I *satisfait*

- une contrainte positive $exp_1 \subseteq exp_2$ si $\chi_I(exp_1) \subseteq \chi_I(exp_2)$ c'est-à-dire $\forall t \in T_\Sigma \quad t \in \chi_I(exp_1) \Rightarrow t \in \chi_I(exp_2)$.

- une contrainte négative $exp_1 \not\subseteq exp_2$ si $\chi_I(exp_1) \not\subseteq \chi_I(exp_2)$ c'est-à-dire $\exists t \in T_\Sigma \quad t \in \chi_I(exp_1) \wedge t \notin \chi_I(exp_2)$.

Une *solution* d'une contrainte C est une interprétation qui satisfait la contrainte. L'ensemble des solutions de C est notée $SOL(C)$.

Une solution d'un système de contrainte SC satisfait la conjonction de toutes les contraintes. L'ensemble des solutions de SC est $SOL(SC)$. Par la suite nous confondrons *solution* et *n-uplet de langages d'arbres.*

Par la suite, nous identifierons χ_I et I.

4.2 Remarques sur le Cas des Mots

Les solutions des systèmes de contraintes sur des ensembles de mots sont des n-uplets de langages de mots et nous avons vu comment les coder dans un arbre infini (voir page 35). Nous montrons comment définir un automate d'arbre infini capable de reconnaître l'ensemble des solutions d'une contrainte dans les mots.

Par souci de clarté, nous modifions légèrement la syntaxe des contraintes ensemblistes. En effet, le mode de fonctionnement des automates d'arbres infinis est plus adapté à la manipulation d'expressions avec concaténation à droite. Nous écrirons par exemple $(Xa \cap Z)ba$ plutôt que $ab(aX \cap Z)$.

Les solutions des systèmes avec concaténation à droite sont équivalentes aux solutions des systèmes avec concaténation à gauche via l'application de l'opération de *miroir* sur les langages[1].

[1] Le miroir du mot $u = u_1 u_2 \ldots u_k$ est le mot $\tilde{u} = u_k u_{k-1} \ldots u_1$

Dans le cas où l'alphabet Σ ne contient qu'une seule lettre, les interprétations des variables sont donc des n-uplets que l'on peut représenter par des mots infinis sur $\{0,1\}^n$.

L'exemple qui suit montre qu'un ensemble des solutions de contraintes très simples (pas de complémentaire ni d'intersection) peut avoir des propriétés non triviales.

Exemple 4.3 Soit le système

$$X \cup Xa = X \cup Xaa \cup a \qquad (E)$$

Nous vérifions que les solutions de (E) sont les interprétations qui vérifient la propriété (P) suivante:

$$\forall i \in \mathbb{N} \quad a^i \notin \mathcal{I}(X) \Rightarrow a^{i+1} \in \mathcal{I}(X) \text{ et } a^{i+2} \in \mathcal{I}(X) \qquad (P)$$

Les solutions minimales sont les ensembles minimaux au sens de l'inclusion qui vérifient (P).

Par exemple, $\mathcal{I}(X) = \{a^{3n} + a^{3n+1} \mid n \in \mathbb{N}\}$, sont des solutions
$\mathcal{I}(X) = \{a^{3n+1} + a^{3n+2} \mid n \in \mathbb{N}\}$
$\mathcal{I}(X) = \{a^{4n} + a^{4n+1} + a^{4n+2} \mid n \in \mathbb{N}\}$
régulières minimales.

Il existe des solutions non régulières minimales correspondant à des constructions non régulières. Si A est une partie de a^*, nous représentons l'ensemble A par son mot caractéristique, i.e., le mot w_A de $\{0,1\}^\infty$ défini par $w_A[i] = 0 \Leftrightarrow a^i \notin A$.

Le mot $w = uvu^2v^2 \ldots u^nv^n \ldots$ avec $u = 011$ et $v = 0111$ est la représentation d'une solution minimale non régulière. Tous les mots infinis $w = w_0w_1w_2 \ldots$ avec $w_i \in \{u, v\}$, sont des mots caractéristiques de solutions minimales. Clairement, il existe une infinité de solutions minimales régulières ou non.

Contraintes dans les mots et Formules Logiques

Soit le système de contraintes dans les mots

$$SC \equiv \begin{cases} (X \cap \sim Y)ba = Z \\ X \cap Z = \bot \end{cases}$$

La première contrainte se décrit aussi par: "pour tout mot m, mba est dans Z si et seulement si m est dans X et m n'est pas dans Y". Soit, plus formellement $\forall m \in \Sigma^* \quad (mba \in Z) \Leftrightarrow (m \in X \wedge m \notin Y)$. Cette remarque se généralise à tout système de contraintes. Les formules associées font partie de la théorie monadique du second ordre à k successeurs SkS. Intuitivement, les seuls prédicats utilisés sont des prédicats d'appartenance, et les k lettres de Σ jouent le rôle de successeurs. Voyons maintenant de façon plus formelle, l'identification des systèmes de contraintes ensemblistes à des formules de SkS.

Soit V un ensemble dénombrable de variables. Soit $\mathcal{TH}$, l'ensemble des formules de *SkS*. Rappelons enfin que $E(SC)$ est l'ensemble des sous-expressions de SC (voir page 106). Soit Φ l'application de $E(SC) \times V$ dans $\mathcal{TH}$ définie par:

$$\begin{gathered}
\Phi(X, v) = v \in X \\
\Phi(\top, v) = 1 \\
\Phi(\bot, v) = 0 \\
\Phi(\varepsilon, v) = (v = \varepsilon) \\
\Phi(a_i, v) = (v = succ_i(\varepsilon)) \\
\Phi(e_1 \cap e_2, v) = (\Phi(e_1, v) \wedge \Phi(e_2, v)) \\
\Phi(e_1 \cup e_2, v) = (\Phi(e_1, v) \vee \Phi(e_2, v)) \\
\Phi(\sim e_1, v) = (\neg\Phi(e_1, v)) \\
\Phi(e_1.a_i, v) = (\exists v'\ v = succ_i(v') \wedge \Phi(e_1, v'))
\end{gathered}$$

avec e_1 et e_2 des expressions ensemblistes et v une variable de V. La dernière règle introduit une nouvelle variable v'. Elle est supposée (syntaxiquement) différente de v et de toutes les autres variables. Les opérations de successeur $v = succ_i(v')$ sont notées, $v = v'.a_i$.

La formule associée à un système de contraintes SC

$$SC \equiv \begin{cases} exp_1 \subseteq exp'_1 \\ \vdots \\ exp_i \subseteq exp'_i \end{cases}$$

est la formule ϕ_{SC}

$$\phi_{SC}(X_1, \ldots, X_n) = \begin{cases} (\forall v_1\ \Phi(exp_1, v_1) \Rightarrow \Phi(exp'_1, v_1)) \\ \wedge\, (\forall v_1\ \Phi(exp_2, v_1) \Rightarrow \Phi(exp'_2, v_1)) \\ \vdots \\ \wedge\, (\forall v_i\ \Phi(exp_i, v_i) \Rightarrow \Phi(exp'_i, v_i)). \end{cases}$$

où $X_1, \ldots, X_n$ sont les n variables ensemblistes qui apparaissent dans le système SC.

Exemple 4.4 Sur l'expression $(X \cap \sim Y)ba$ du système SC page 108:

$$\begin{aligned}
\Phi((X \cap \sim Y)ba, v) &= \exists v_1\ v = v_1.a \wedge \Phi((X \cap \sim Y)b, v_1) \\
&= \exists v_1\ v = v_1.a \wedge \exists v_2\ v_1 = v_2.b \wedge \Phi((X \cap \sim Y), v_2) \\
&= \exists v_1, v_2\ v = v_1.a \wedge v_1 = v_2.b \wedge \Phi(X, v_2) \wedge \Phi(\sim Y, v_2) \\
&= \exists v_1, v_2\ v = v_1.a \wedge v_1 = v_2.b \wedge v_2 \in X \wedge v_2 \notin Y.
\end{aligned}$$

La contrainte $(X \cap \sim Y)ba = Z$ s'écrit donc:

$$\varphi(X) = \forall v\, (\exists v_1, v_2\ v = v_1.a \wedge v_1 = v_2.b \wedge v_2 \in X \wedge v_2 \notin Y) \Leftrightarrow v \in Z\ .$$

Proposition 4.1 *Le langage $L(\phi_{SC})$ est l'ensemble des arbres caractéristiques des solutions de SC.*

Preuve. La partie $\subseteq$ de la preuve est montrée par induction sur le nombre d'appels à Φ, et la partie inverse par induction sur la taille du système (c'est-à-dire, le nombre de sous-expressions ensemblistes du système). □

Résolution de Contraintes sur des Ensembles de Mots

Grâce à la traduction d'un système en une formule de SkS, nous déduisons un algorithme de résolution des contraintes sur les ensembles de mots.

La méthode de résolution se déroule en deux étapes. Soit SC un système de contraintes.

La première étape associe au système SC une formule ϕ_{SC} de la théorie monadique du second ordre à k successeurs, SkS.

La deuxième étape construit l'automate de Rabin A_{SC} tel que $L(A_{SC}) = L(\phi_{SC})$. Cet automate reconnaît alors le langage des arbres caractéristiques des solutions du système SC.

La correction et la complétude de la résolution sont la conséquence de la proposition 4.1 et du théorème de Rabin page 38.

Remarque : A un arbre infini régulier sur $\{0,1\}^n$, correspond un n-uplet de langages réguliers. Puisque dans un langage non vide Rabin-reconnaissable il existe un arbre infini régulier (voir proposition 1.2, alors,

> *toute contrainte sur des ensembles de mots satisfiable admet une solution qui est un n-uplet de langages réguliers.*

Nous pouvons assurer que les automates de Büchi suffisent à l'élaboration de cette méthode. En effet, les formules logiques manipulées s'écrivent sous la forme : $\varphi(X_1, \ldots, X_n)$ où dans φ les seules variables libres sont $X_1, \ldots, X_n$ et aucun quantificateur ne porte sur une variable du second ordre. Ce sont donc essentiellement des formules du second ordre monadique faible. En utilisant le théorème 1.3, nous pouvons conclure que les langages définissables par les formules associées aux systèmes sont Büchi-reconnaissables. Cette dernière remarque nous permet d'avancer qu'un pré-traitement des systèmes et des formules doit certainement faciliter la mise en œuvre de la seconde étape de l'algorithme. Mais, le principal reproche est peut être le manque d'adaptabilité de la méthode à manipuler des contraintes sur des ensembles d'arbres.

4.3 Algorithme de Résolution

Nous montrons la décidabilité de la satisfiabilité des contraintes sans symbole de projection (dans la classe $PNSC$).

Théorème 4.1 *La satisfiabilité des contraintes ensemblistes de la classe PNSC est décidable.*

Nous avons introduit dans la section 1.2.3 le schéma classique de résolution de problèmes de décision à l'aide d'automates. Pour montrer la décidabilité de la satisfiabilité des systèmes de contraintes ensemblistes, nous prouvons qu'à un système de contraintes ensemblistes, il est possible d'associer un automate d'ensembles d'arbres qui reconnaît exactement les les n-uplets de langages d'arbres qui sont les solutions du système. Les propriétés des langages *TSA*-reconnaissables seront transposées dans le cadre des contraintes en vue de simplifier l'exposé de l'algorithme de résolution (ex. clôture - voir section 3.4), et de déterminer d'autres propriétés (de régularité - voir 3.3.1 ; topologiques - voir 3.3.2 ; ...)

Théorème 4.2 *Soit SC un système de contraintes ensemblistes dans PNSC. On peut construire un Automate d'Ensembles d'Arbres $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = SOL(SC)$.*

Soient $X_1, \ldots, X_n$ les variables ensemblistes et Σ l'alphabet du système SC. Rappelons que la formule $\mathcal{L}(\mathcal{A}) = SOL(SC)$ indique que l'ensemble des interprétations $(L_1, \ldots, L_n)$ des n variables qui satisfont SC, ce que nous avons appelé les *solutions* de SC , est exactement l'ensemble reconnu par un *TSA* $\mathcal{A}$. La preuve de ce théorème est détaillée dans les trois sections suivantes.

4.3.1 Remarque Préliminaire

Les propriétés de clôture des *TSA* permettent de réduire la preuve à la construction d'un automate associé à une seule contrainte. En effet, supposons qu'au système

$$SC = \begin{cases} C_1 \\ C_2 \\ \vdots \\ C_m \end{cases}$$

sur les variables $X_1, \ldots, X_n$, correspondent les *TSA* $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$ de telle sorte que

$$\forall i \in \{1, \ldots, m\} \quad \mathcal{L}(\mathcal{A}_i) = SOL(C_i).$$

Une solution de SC satisfait simultanément toutes les contraintes C_i. Puisque les contraintes ne portent pas en général sur les même variables, nous normalisons à l'aide d'opérations de cylindification, les solutions de chaque contrainte de façon à obtenir pour chaque C_i, un ensemble $SOL_n(C_i)$ de n-uplets de langages $(L_1, \ldots, L_n)$ où les L_j représentent les interprétations des variables X_j.

L'ensemble $SOL(SC)$ est alors $\bigcap_{i=1}^{m} SOL_n(C_i)$.

Maintenant, grâce aux opérations de cylindrification, d'intersection et de projection définies en 3.4, nous pouvons construire un automate $\mathcal{A}$ tel que $SOL(SC) = \mathcal{L}(\mathcal{A})$.

4.3.2 Notations

Nous considérons maintenant, et pour la suite de cette section, une contrainte C, portant sur n variables $X_1, \ldots, X_n$.

Regardons maintenant les éléments de $E(C)$ comme des variables propositionnelles. Les opérateurs $\cup$, $\cap$, $\sim$, $\subseteq$, $\not\subseteq$ sont respectivement interprétés par les connecteurs logiques $\vee$ (ou), $\wedge$ (et), $\neg$ (non), $\Rightarrow$ (implique) et $\not\Rightarrow$ (n'implique pas) et les symboles spéciaux $\bot$ et $\top$ par les valeurs de vérité 0 et 1.

Appellons *valuation booléenne* toute application φ de $E(C)$ dans $\mathcal{B}$ qui vérifie les propriétés suivantes :

$$\begin{gathered}\varphi(\bot) = 0;\\ \varphi(\top) = 1;\\ \varphi(\cap(e,e')) \Leftrightarrow (\varphi(e) \wedge \varphi(e'));\\ \varphi(\cup(e,e')) \Leftrightarrow (\varphi(e) \vee \varphi(e'));\\ \varphi(\sim e) \Leftrightarrow \neg\varphi(e);\end{gathered}$$

Etant donnés une contrainte C et une valuation booléenne φ, nous pouvons maintenant associer une valeur de vérité à la formule propositionnelle correspondant à C. Dans le cas d'une contrainte positive $exp \subseteq exp'$ cette valeur est $\varphi(exp) \Rightarrow \varphi(exp')$ et dans le cas d'une contrainte négative $exp \not\subseteq exp'$, c'est $\varphi(exp) \not\Rightarrow \varphi(exp')$.

Un calcul dans un *TSA* de rang n accepte un n-uplet de langages d'arbres. Il est intéressant de considérer qu'un tel calcul est l'interprétation dans 2^{T_Σ} de n variables ensemblistes $X_1, \ldots, X_n$. Nous appelons $\mathcal{I}_r$ l'interprétation ensembliste associée au calcul r. Comme précédemment (voir page 106), cette interprétation est facilement étendue à toute expression ensembliste.

4.3.3 Définition de $\mathcal{A}$

La construction du *TSA* est basée sur l'idée de conserver un historique (fini) des choix effectués ainsi que leurs incidences.

Pour motiver cette remarque, prenons l'exemple suivant. Si C est la contrainte

$$b(b(X)) \subseteq X$$

sur l'alphabet $\{b(),a\}$, alors pour toute solution L, l'appartenance de a à L entraîne l'appartenance de $b(b(a))$ à L. Plus généralement, si t est dans L, alors $b(b(t))$ est aussi dans L.

Puisqu'un automate d'ensembles d'arbres peut être vu comme une machine qui dans un calcul r analyse successivement les termes dans l'ordre "sous-terme", nous devons nous souvenir du choix fait pour t pour décider de l'appartenance de $b(b(t))$ à $L(\mathcal{A}, r)$.

C'est-à-dire que pour tout calcul r, candidat à accepter une solution, si un terme t s'écrit $b(b(t'))$, alors il faut se souvenir de l'appartenance de t' à $L(\mathcal{A}, r)$. Cette mémorisation est limitée. Il suffit de conserver un nombre fini de valeurs booléennes de la forme *t est dans l'expression* e[2] pour effectuer cette construction. Dans notre exemple, il suffit de conserver les informations: *t est dans* X, *t est dans* $b(X)$, et *t est dans* $b(b(X))$ quelque soit l'alphabet Σ et l'ensemble des variables $\mathcal{X}$.

En règle générale, la construction requiert la mémorisation d'un nombre d'informations du type booléen, qui est linéairement dépendant du nombre de symboles dans la contraintes.

Dans la définition qui suit, nous décrivons l'automate *associé* à une contrainte ensembliste C. C'est l'automate qui reconnaît l'ensemble des solutions de C. Les informations sont mémorisées dans l'état atteint par un calcul. Puisque celles-ci sont de la forme *u est dans* e, un état sera une valuation booléenne de telle sorte que, si u atteint l'état φ, alors *u est dans* e si et seulement si $\varphi(e) = 1$.

Remarquons finalement que certaines informations sont superflues. En d'autres termes, la construction n'est pas optimale et certaines règles et certains états ne seront jamais utilisés.

Soient C une contrainte et $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ un automate d'ensemble d'arbres où :

- $\mathcal{Q} = \{\varphi \mid \varphi$: valuation booléenne de $E(C)$ dans $\mathcal{B}\}$;
- $\mathcal{F} = (F_1, \ldots, F_n)$ avec $F_i = \{\varphi \in \mathcal{Q} \mid \varphi(X_i) = 1\}$.
- $\mathcal{S}$ est l'ensemble des règles suivantes :
  - *Règles initiales :* $a \rightarrow \varphi$ où $a \in \Sigma_0$ et φ vérifie :
$$\forall e \in E(C) \quad (e \notin \mathcal{X}) \Rightarrow ((e = a) \Leftrightarrow (\varphi(e) = 1));$$
  - *Règles de progression :* $b(\varphi_1, \ldots, \varphi_p) \rightarrow \varphi$ où $b \in \Sigma_p$ et φ, φ_i vérifient
$$\forall e \in E(C) : (e \notin \mathcal{X}) \Rightarrow \left((\varphi(e) = 1) \Leftrightarrow \left(\begin{array}{l} e = b(e_1, \ldots, e_p) \\ \forall i \quad 1 \leq i \leq p \quad \varphi_i(e_i) = 1 \end{array} \right) \right).$$
- Ω est l'ensemble des parties ω de $\mathcal{Q}$ telles que :
  - Si C est une contrainte positive $exp \subseteq exp'$ alors
$$\forall \varphi \in \omega \quad \varphi(exp) \Rightarrow \varphi(exp');$$
  - Si C est une contrainte négative $exp \not\subseteq exp'$ alors
$$\exists \varphi \in \omega \quad \neg(\varphi(exp) \Rightarrow \varphi(exp'));$$

[2] En fait, il faudrait écrire : *t est dans l'ensemble* $\mathcal{I}_r(e)$ où $\mathcal{I}_r$ est l'interprétation décrite par le calcul r en cours de construction.

4.3.4 Correction de la Construction

La preuve de correction de cette construction est divisée en trois lemmes.

Ce premier lemme caractérise l'interprétation $\mathcal{I}_r$ d'un calcul r dans $\mathcal{A}$. Un terme qui se réécrit en un état φ dans le calcul r, appartient à l'interprétation $\mathcal{I}_r(e)$ d'une expression e qui apparaît dans la contrainte, si et seulement si $\varphi(e)$ est vraie.

Lemme 4.1 $\forall r \in \mathcal{R}_{\mathcal{A}},\ \forall e \in E(C),\ \forall t \in T_\Sigma, (t \in \mathcal{I}_r(e)) \Leftrightarrow (r(t)(e) = 1)$.

Preuve. Soit un calcul r dans l'automate $\mathcal{A}$. Nous allons prouver cette propriété par induction sur la taille $h(e)$ de l'expression $e \in T_{\Sigma+B}(\mathcal{X})$.

- $h(e) = 1$. L'expression e est soit une constante a, soit l'un des deux symboles $\top$ ou $\bot$ soit une variable X_i. Examinons ces quatre cas.

  $e = a$ Nous devons prouver que pour tout terme t, $(t \in \mathcal{I}_r(a)) \Leftrightarrow (r(t)(a) = 1)$. Montrons d'abord $t = a \Leftrightarrow r(t)(a) = 1$. Si $t = a$, alors par définition de $\mathcal{A}$, les seules règles applicables sont de la forme $a \longrightarrow \varphi$ et vérifient :

  $$\forall exp \in E(C) \quad (exp \notin \mathcal{X}) \Rightarrow ((exp = a) \Leftrightarrow (\varphi(exp) = 1));$$

  Donc, $t = a \Rightarrow r(t)(a) = 1$. Maintenant, si un terme t est tel que $r(t)(a) = 1$ alors selon la définition de $\mathcal{S}$, l'état $r(t)$ est en partie droite d'une règle initiale $a \longrightarrow r(t)$, ce qui implique que $t = a$. Nous avons donc $t = a \Leftrightarrow r(t)(a) = 1$.

  L'interprétation $\mathcal{I}_r$, par définition de son extension aux expressions ensemblistes satisfait à : $t \in \mathcal{I}_r(a) \Leftrightarrow t = a$. Finalement, $(t \in \mathcal{I}_r(a)) \Leftrightarrow (r(t)(a) = 1)$.

  $e = \top$ L'ensemble des états $\mathcal{Q}$ est l'ensemble des valuations booléennes de $E(C)$ dans $\mathcal{B}$. Puisque toute valuation booléenne φ est telle que $\varphi(\top) = 1$, et que tout terme de $T(\Sigma)$ appartient à $\mathcal{I}_r(\top)$, alors $\forall t \in T_\Sigma, (t \in \mathcal{I}_r(\top)) \Leftrightarrow (r(t)(\top) = 1)$.

  $e = \bot$ De même, pour tout φ, $\varphi(\bot) = 0$ et aucun terme de $T(\Sigma)$ n'appartient à $\mathcal{I}_r(\bot)$.

  $e = X_i \in \mathcal{X}$ D'après la Définition 3.4, si $(L_1, \ldots, L_n)$ est le n-uplet accepté par le calcul r, alors pour tout terme t de T_Σ, $(t \in L_i) \Leftrightarrow (r(t) \in F_i)$. Or, $F_i = \{\varphi \in \mathcal{Q} \mid \varphi(X_i) = 1\}$. Donc $t \in \mathcal{I}_r(X_i) \Leftrightarrow r(t)(X_i) = 1$.

- Supposons le lemme vrai pour toute expression de taille inférieure ou égale à k. Une expression e de taille $k+1$ est soit de la forme $b(e_1, \ldots, e_p)$, ou $\cup(e_1, e_2)$, ou $\cap(e_1, e_2)$, ou $\sim(e_1)$.

Considérons le premier cas $e = b(e_1, \ldots, e_p)$. Nous devons vérifier que $\forall t \in T_\Sigma, (t \in \mathcal{I}_r(e)) \Leftrightarrow (r(t)(e) = 1)$. Par la définition de l'extension de l'interprétation $\mathcal{I}_r$, nous savons que

$$t \in \mathcal{I}_r(b(e_1, \ldots, e_p) \Leftrightarrow (t = b(t_1, \ldots, t_p) \wedge \forall i \; t_i \in \mathcal{I}_r(e_i))$$

Par hypothèse d'induction, $t_i \in \mathcal{I}_r(e_i)$ si et seulement si $r(t_i)(e_i) = 1$. La définition des règles de progression est telle que si $t = b(t_1, \ldots, t_p)$ et $r(t_i)(e_i) = 1$ alors $r(t)(e) = 1$.

Dans les trois cas suivants, $e = \cup(e_1, e_2)$, $e = \cap(e_1, e_2)$, $e = \sim(e)$ la propriété se déduit directement de l'interprétation des opérateurs et des hypothèses d'induction. □

Le lemme suivant associe un calcul dans $\mathcal{A}$ à une solution de C.

Lemme 4.2 *Soit $\mathcal{I}$ une solution de C . L'application r de T_Σ dans $\mathcal{Q}$ définie telle que $\forall e \in E(C), (r(t)(e) = 1) \Leftrightarrow (t \in \mathcal{I}(e))$ est un calcul dans $\mathcal{A}$.*

Preuve.

Soit $\mathcal{I}$ une solution de C. Vérifions que r défini par $\forall e \in E(C), (r(t)(e) = 1) \Leftrightarrow (t \in \mathcal{I}(e))$ est un calcul de $\mathcal{A}$.

Clairement, r est une application de T_Σ dans $\mathcal{Q}$. C'est un calcul si elle vérifie pour tout $t = b(t_1, \ldots, t_p)$ de T_Σ : $b(r(t_1), \ldots, r(t_p)) \rightarrow r(b(t_1, \ldots, t_p)) \in \mathcal{S}$.

Rappelons la définition des règles de progression : $b(\varphi_1, \ldots, \varphi_p) \rightarrow \varphi$ où $b \in \Sigma_p$ et φ, φ_i sont tels que :

$$\forall e \in E(C) \quad (e \notin \mathcal{X}) \Rightarrow \left((\varphi(e) = 1) \Leftrightarrow \left(\begin{array}{l} e = b(e_1, \ldots, e_p) \\ \forall i \quad 1 \leq i \leq p \quad \varphi_i(e_i) = 1 \end{array} \right) \right).$$

Soit $t = b(t_1, \ldots, t_p)$. D'après la construction de r, pour tout i de $\{1, \ldots, p\}$ et pour toute expression e de EC), $r(t_i)(e_i) = 1 \Leftrightarrow t_i \in \mathcal{I}(e_i)$, et $r(t)(e) = 1 \Leftrightarrow t \in \mathcal{I}(e)$.

Nous devons donc montrer que pour toute sous expression e de $E(C)$, soit $e \in \mathcal{X}$, soit $r(t)(e) = 1$ si et seulement si e est de la forme $b(e_1, \ldots, e_p)$ et $r(t_i)(e_i) = 1$.

Par l'extension de $\mathcal{I}$, nous avons :

$$\left. \begin{array}{l} e = b(e_1, \ldots, e_p) \\ \forall i \quad 1 \leq i \leq p \quad t_i \in \mathcal{I}(e) \end{array} \right\} \Rightarrow t \in \mathcal{I}(e).$$

Pour les même raisons,

$$t \in \mathcal{I}(e) \Rightarrow (e \in \mathcal{X}) \text{ ou } \left\{ \begin{array}{l} e = b(e_1, \ldots, e_p) \\ \forall i \quad 1 \leq i \leq p \quad t_i \in \mathcal{I}(e) \end{array} \right.$$

Ce qui prouve que r est bien compatible avec les règles de $\mathcal{S}$. □

Le théorème suivant prouve la correction de la construction de l'automate $\mathcal{A}$ associé à la contrainte C donnée en section 4.3.3.

Lemme 4.3 $\mathcal{L}(\mathcal{A}) = SOL(C)$.

Preuve. La preuve se divise en deux cas, selon la nature de la contrainte C.

- C est une contrainte positive : $exp \subseteq exp'$

  Nous montrons d'abord que $\mathcal{L}(\mathcal{A}) \subseteq SOL(C)$. En d'autres termes, chaque n-uplet $(L_1, \ldots, L_n)$ accepté par un calcul réussi r de $\mathcal{A}$, donc toute interprétation $\mathcal{I}_r$, doit vérifier $\mathcal{I}_r(exp) \subseteq \mathcal{I}_r(exp')$. D'après la définition de Ω, l'image de T_Σ par r est un ensemble d'états ω qui vérifie $\forall \varphi \in \omega \quad \varphi(exp) \Rightarrow \varphi(exp')$. Le Lemme 4.1 assure maintenant que l'inclusion est bien vérifiée puisque :

$$\begin{array}{lll} & \forall t \in T_\Sigma \quad \forall e \in E(C) \quad t \in \mathcal{I}_r(e) \Leftrightarrow r(t)(\epsilon) = 1 & \text{Lemme 4.1} \\ & \forall t \in T_\Sigma \quad r(t)(exp) \Rightarrow r(t)(exp') & \text{Définition de } \Omega \\ \Rightarrow & \forall t \in T_\Sigma \quad t \in \mathcal{I}_r(exp) \Rightarrow t \in \mathcal{I}_r(exp') & \\ \Rightarrow & \mathcal{I}_r(exp) \subseteq \mathcal{I}_r(exp'). & \end{array}$$

  L'inclusion inverse $SOL(C) \subseteq \mathcal{L}(\mathcal{A})$ est directe par le Lemme 4.2. En effet, si $\mathcal{I}$ est une solution de la contrainte C alors, le calcul r défini par $\forall e \in E(C), (r(t)(e) = 1) \Leftrightarrow (t \in \mathcal{I}(e))$ induit une interprétation $\mathcal{I}_r$ égale à $\mathcal{I}$. Remarquons que r est bien un calcul réussi puisque $\forall t \in T_\Sigma \quad t \in \mathcal{I}_r(exp) \Rightarrow t \in \mathcal{I}_r(exp')$ ce qui implique à l'aide du Lemme 4.1 que $\forall t \in T_\Sigma \quad r(t)(exp) \Rightarrow r(t)(exp')$.

- C est une contrainte négative : $exp \not\subseteq exp'$

  La preuve est similaire à celle du cas positif. C'est la condition de succès d'un calcul dans $\mathcal{A}$ qui implique que $\mathcal{I}_r(exp) \not\subseteq \mathcal{I}_r(exp')$ si r est un calcul réussit de $\mathcal{A}$. En effet, si $\exists t \in T_\Sigma \quad \neg(r(t)(exp) \Rightarrow r(t)(exp'))$ alors, d'après le Lemme 4.1, $\exists t \in T_\Sigma \quad \neg(r(t)(exp) \Rightarrow r(t)(exp'))$, ce qui implique $\mathcal{I}_r(exp) \not\subseteq \mathcal{I}_r(exp')$.

  Comme précédemment, le Lemme 4.2 termine cette preuve en associant un calcul réussit à une solution de C. □

4.4 Exemples

Nous allons traiter un problème simple conformément à la méthode qui a été proposée précédemment. Ensuite, nous verrons, sans le définir formellement, comment optimiser cette procédure.

Soit Σ l'alphabet gradué composé de la constante a et du symbole binaire b. Soit SC le système, construit sur Σ, suivant :

$$SC = \begin{cases} X & \subseteq & b(\sim X, \sim X) \\ X & \neq & \bot \end{cases}$$

Nommons les deux contraintes $C1$ et $C2$.

Résolution de $C1$ L'ensemble des sous-expressions de $C1$, est $E(C1)$.

$$E(C1) = \{X, \sim X, b(\sim X, \sim X)\} \ .$$

Nous représentons une valuation booléenne $\varphi : E(C1) \rightarrow \mathcal{B}$, par les valeurs qu'elles prend respectivement pour $X, b(\sim X, \sim X)$. Cette notation est suffisante puisque, par définition d'une valuation booléenne, nous avons $\varphi(X) = \neg\varphi(\sim X)$. Par exemple, φ_1, définie par :

$$\varphi_1(X) = 1 \quad \varphi_1(\sim X) = 0 \quad \varphi_1(b(\sim X, \sim X)) = 0 \ ,$$

est représentée par $[1,0]$.

Soit $\mathcal{A}_1 = (\Sigma, \mathcal{Q}_1, \mathcal{F}_1, \mathcal{S}_1, \Omega_1)$, le *TSA* associé à $C1$.

- $\mathcal{Q}_1 = \{[0,0], [0,1], [1,0], [1,1]\}$
- $\mathcal{F}_1 = \{[1,0], [1,1]\}$.
- $\mathcal{S}_1$ est l'ensemble des règles suivantes :

$$\begin{array}{rcl|rcl}
a & \rightarrow & [0,0] \mid [1,0] \ , & b([0,0],[0,0]) & \rightarrow & [1,1] \mid [0,1] \ , \\
b([0,0],[0,1]) & \rightarrow & [1,1] \mid [0,1] \ , & b([0,1],[0,0]) & \rightarrow & [1,1] \mid [0,1] \ , \\
b([0,1],[0,1]) & \rightarrow & [1,1] \mid [0,1] \ , & b([0,0],[1,1]) & \rightarrow & [0,0] \mid [1,0] \ , \\
b([0,0],[1,0]) & \rightarrow & [0,0] \mid [1,0] \ , & b([1,1],[0,0]) & \rightarrow & [0,0] \mid [1,0] \ , \\
b([1,1],[1,1]) & \rightarrow & [0,0] \mid [1,0] \ , & b([1,1],[0,1]) & \rightarrow & [0,0] \mid [1,0] \ , \\
b([1,1],[1,0]) & \rightarrow & [0,0] \mid [1,0] \ , & b([0,1],[1,1]) & \rightarrow & [0,0] \mid [1,0] \ , \\
b([0,1],[1,0]) & \rightarrow & [0,0] \mid [1,0] \ , & b([1,0],[0,0]) & \rightarrow & [0,0] \mid [1,0] \ , \\
b([1,0],[1,1]) & \rightarrow & [0,0] \mid [1,0] \ , & b([1,0],[0,1]) & \rightarrow & [0,0] \mid [1,0] \ , \\
b([1,0],[1,0]) & \rightarrow & [0,0] \mid [1,0] \ . & & &
\end{array}$$

- $\Omega_1 = \{\omega \subseteq \mathcal{Q}_1 \mid [1,0] \notin \omega\}$.

La construction de l'automate est facilement optimisable. Premièrement, le nombre des états est réduit en supprimant ceux qui correspondent une valuation ne validant pas la contrainte. En d'autres termes $\mathcal{Q}_1 = \{\varphi \mid \varphi(C1) = 1\}$. Deuxièmement, l'information $\varphi(b(\sim X, \sim X))$ n'est pas nécessaire, puisqu'elle n'est pas propagée.

En fait, dans cet exemple, il suffit de conserver dans un état l'unique information $\varphi(X)$. La construction simplifiée donne :

- $\mathcal{Q}_1 = \{[0], [1]\}$

- $\mathcal{F}_1 = \{[1]\}$.
- $\mathcal{S}_1$ est l'ensemble des règles suivantes :

$$\begin{aligned} a &\rightarrow [0] \ , \\ b([0],[0]) &\rightarrow [1] \mid [0] \ , \\ b([0],[1]) &\rightarrow [0] \ , \\ b([1],[0]) &\rightarrow [0] \ , \\ b([1],[1]) &\rightarrow [0] \ . \end{aligned}$$

- $\Omega = \{\omega \subseteq \mathcal{Q}_1\}$.

Résolution de C2. Les sous-expressions de $C2$ sont $E(C2) = \{X, \bot\}$. Encore une fois, nous simplifions les notations en ne faisant pas apparaître la valeur d'une valuation booléenne en $\bot$. Le *TSA* associé à $C2$ est $\mathcal{A}_2 = (\Sigma, \mathcal{Q}_2, \mathcal{F}_2, \mathcal{S}_2, \Omega_2)$.

- $\mathcal{Q}_2 = \{[0],[1]\}$
- $\mathcal{F}_2 = \{[1]\}$.
- $\mathcal{S}_2$ est l'ensemble des règles suivantes :

$$\begin{aligned} a &\rightarrow [0] \mid [1] \ , \\ b([0],[0]) &\rightarrow [1] \mid [0] \ , \\ b([0],[1]) &\rightarrow [1] \mid [0] \ , \\ b([1],[0]) &\rightarrow [1] \mid [0] \ , \\ b([1],[1]) &\rightarrow [1] \mid [0] \ . \end{aligned}$$

- $\Omega = \{\omega \subseteq \mathcal{Q}_2 \mid [1] \in \omega\}$.

Résolution de SC L'automate d'ensembles d'arbres qui reconnaît l'ensemble des solutions de SC est obtenu en calculant "l'intersection" $\mathcal{A}_1 \cap \mathcal{A}_2$. Après simplifications,

$\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$.

- $\mathcal{Q} = \{[0],[1]\}$
- $\mathcal{F} = \{[1]\}$.
- $\mathcal{S}$ est l'ensemble des règles suivantes :

$$\begin{aligned} a &\rightarrow [0] \ , \\ b([0],[0]) &\rightarrow [1] \mid [0] \ , \\ b([0],[1]) &\rightarrow [0] \ , \\ b([1],[0]) &\rightarrow [0] \ , \\ b([1],[1]) &\rightarrow [0] \ . \end{aligned}$$

- $\Omega = \{\omega \subseteq \mathcal{Q} \mid [1] \in \omega\}$.

4.5 Propriétés des Solutions et des Systèmes

Outre la décision de la satisfiabilité, les propriétés des automates d'ensembles d'arbres induisent d'autres résultats dans le cadre des contraintes ensemblistes. Nous les présentons dans cette section.

4.5.1 Solutions Régulières

Une solution régulière est un n-uplet dont chaque composante est un langage d'arbre régulier.

Proposition 4.2 *Tout système de contraintes ensemblistes satisfiable de la classe PNSC admet une solution régulière.*

Preuve. Soit SC un système de contraintes dans $PNSC$. Par le théorème 4.2, on peut construire un TSA qui reconnaît exactement l'ensemble des solutions de SC.

Puisque SC est satisfiable, cet ensemble est non vide et par la proposition 3.6, il contient un n-uplet de langages réguliers. □

Proposition 4.3 *Soit SC un système de contraintes de la classe $PNSC$. L'appartenance d'un n-uplet de langages réguliers à $SOL(SC)$ est décidable.*

Preuve. Soit SC un système de contraintes de $PNSC$. Puisque l'appartenance d'un n-uplet de langages réguliers au langage reconnu par un TSA est décidable (théorème 3.2), par le théorème 4.2, nous déduisons que son appartenance à l'ensemble des solutions de SC l'est aussi. □

4.5.2 Conditions d'Acceptation et non Inclusion

Nous montrons maintenant les connexions étroites entre la classe des contraintes positives[3] PSC et les TSA sans condition d'acceptation.

Proposition 4.4 *Soit SC une contrainte dans PSC. Il existe un TSA $\mathcal{A}$ sans condition d'acceptation tel que $\mathcal{L}(\mathcal{A}) = SOL(SC)$.*

Preuve. La preuve est similaire à celle du théorème 4.2. Seul l'ensemble des états est construit de façon différente.

En utilisant les propriétés de clôture des TSA, nous réduisons la construction au cas d'une seule contrainte positive (voir page 111) sur les variables $X_1, \ldots, X_n$. Rappelons que $E(C)$ est l'ensemble des sous-expressions de C.

[3] sans symbole de non inclusion

Construction

Soit C une contrainte positive. Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ l'automate d'ensemble d'arbres où :

- $\mathcal{Q} = \{\varphi \mid \varphi$ est une valuation booléenne $\wedge\, \varphi(C) = 1\}$;
- $\mathcal{F} = (F_1, \ldots, F_n)$ avec $F_i = \{\varphi \in \mathcal{Q} \mid \varphi(X_i) = 1\}$.
- $\mathcal{S}$ est l'ensemble des règles suivantes :
  - *Règles initiales :* $a \rightarrow \varphi$ où $a \in \Sigma_0$ et φ vérifie :
$$\forall e \in E(C) \quad (e \notin \mathcal{X}) \Rightarrow ((e = a) \Leftrightarrow (\varphi(e) = 1));$$
  - *Règles de progression :* $b(\varphi_1, \ldots, \varphi_p) \rightarrow \varphi$ où $b \in \Sigma_p$ et φ , φ_i vérifient
$$\forall e \in E(C) : (e \notin \mathcal{X}) \Rightarrow \left((\varphi(e) = 1) \Leftrightarrow \begin{pmatrix} e = b(e_1, \ldots, e_p) \\ \forall i \quad 1 \leq i \leq p \quad \varphi_i(e_i) = 1 \end{pmatrix}\right).$$
- $\Omega = 2^{\mathcal{Q}}$.

La preuve de correction de cette construction est menée de façon identique à celle du théorème 4.2.

Lemme 4.4 $\forall r \in \mathcal{R}_{\mathcal{A}}$, $\forall e \in E(C)$, $\forall t \in T_\Sigma, (t \in \mathcal{I}_r(e)) \Leftrightarrow (r(t)(e) = 1)$.

Lemme 4.5 *Soit $\mathcal{I}$ une solution de C . L'application r de T_Σ dans $\mathcal{Q}$ définie telle que $\forall e \in E(C), (r(t)(e) = 1) \Leftrightarrow (t \in \mathcal{I}(e))$ est un calcul dans $\mathcal{A}$.*

Voir page 114 et 115 les preuves de ces deux lemmes.

Lemme 4.6 $\mathcal{L}(\mathcal{A}) = SOL(C)$.

Preuve. Montrons que $\mathcal{L}(\mathcal{A}) \subseteq SOL(C)$. Soit $C = exp \subseteq exp'$. Pour un n-uplet $(L_1, \ldots, L_n)$ accepté par un calcul r, donc pour une interprétation $\mathcal{I}_r$, nous devons avoir $\mathcal{I}_r(exp) \subseteq \mathcal{I}_r(exp')$.

Soit $t \in \mathcal{I}_r(exp)$. Alors $r(t)(exp) = 1$. D'après la définition de $\mathcal{Q}$ nous savons que $r(t)(C) = 1$ Donc, $r(t)(exp) = 1$, $r(t)(C) = 1$ et $C = exp \subseteq exp'$. Nous déduisons donc $r(t)(exp') = 1$, i.e. $t \in \mathcal{I}_r(exp')$.

L'inclusion inverse $\supseteq$ est prouvée par le lemme précédent. □

Soit SC un système de PSC sur les variables ensemblistes $X_1, \ldots, X_n$ et E un sous-ensemble de $\{X_1, \ldots, X_n\}$. Par la suite, $SOL(SC)_{|E}$ représentera l'ensemble des solutions d'un système de contraintes restreint aux variables de E.

Théorème 4.3 *Soit $\mathcal{A}$ un TSA. Il existe un système de contraintes positives SC et un ensemble E tel que $SOL(SC)_{|E} = \mathcal{L}(\mathcal{A})$.*

Preuve. Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ un automate d'ensembles d'arbres avec $\mathcal{F} = (F_1, \ldots, F_n)$ et $\Omega = 2^{\mathcal{Q}}$.

Construction.

Soit $\{Z_{\mathbf{q}} \mid \mathbf{q} \in \mathcal{Q}\}$ un ensemble de variables et $E = \{X_1, \ldots, X_n\}$. L'ensemble de tous les membres gauches de règle constructibles à partir de Σ et de $\mathcal{Q}$ est G.

$$G = \{b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \mid b \in \Sigma_p \wedge \mathbf{q}_1, \ldots, \mathbf{q}_p\}.$$

L'ensemble de tous les membres droits de règle de membre gauche donné g est D_g.

$$D_g = \{\mathbf{q} \mid g \to \mathbf{q} \in \mathcal{S}\}$$

Le système SC est la conjonction de quatre ensembles de contraintes: SC_1, SC_2, SC_3, SC_4. Pour simuler un calcul, les variables $Z_{\mathbf{q}}$ vont capturer les termes d'image $\mathbf{q}$. Les contraintes de SC_1 exprimeront les règles de l'automate, celles de SC_2 assureront qu'il n'est pas possible d'être dans deux états à la fois. Finalement les contraintes de SC_3 construiront les composantes de la solution et celles de SC_4 signifieront que le calcul est une application, c'est-à-dire que chaque terme a une image.

$$\begin{aligned}
SC_1 &\equiv \bigwedge_{g=b(q_1,\ldots,q_p)\in G} \left(b(Z_{\mathbf{q}_1}, \ldots, Z_{\mathbf{q}_p}) \subseteq \bigcup_{\mathbf{q}\in D_g} Z_{\mathbf{q}}\right) ; \\
SC_2 &\equiv \bigwedge_{\mathbf{q},\mathbf{q}'\in\mathcal{Q},\ \mathbf{q}\neq\mathbf{q}'} (Z_{\mathbf{q}} \cap Z_{\mathbf{q}'} = \emptyset) ; \\
SC_3 &\equiv \bigwedge_{i=1}^{n} \left(X_i = \bigcup_{\mathbf{q}\in F_i} Z_{\mathbf{q}}\right) ; \\
SC_4 &\equiv \left(\bigcup_{\mathbf{q}\in\mathcal{Q}} Z_{\mathbf{q}} = \top\right) .
\end{aligned}$$

Correction

Vérifions que $SOL(SC)_{|E} = \mathcal{L}(\mathcal{A})$. Si r est un calcul de $\mathcal{A}$, soit

$$L_r(\mathbf{q}) = r^{-1}(\mathbf{q}) = \{t \mid r(t) = \mathbf{q}\}.$$

Fait 4.1 $SOL(SC)_{|E} \subseteq \mathcal{L}(\mathcal{A})$.

Preuve. Chaque calcul r de $\mathcal{A}$ accepte le langage $\mathcal{L}(\mathcal{A}, r) = (L_1, \ldots, L_n)$ avec $L_i = \bigcup_{\mathbf{q}\in F_i} L_r(\mathbf{q})$. Vérifions que $(L_1, \ldots, L_n)$ est aussi solution de SC.

- Les contraintes de SC_1 sont satisfaites.

$$\forall r \in \mathcal{R}_{\mathcal{A}} \quad b(L_r(\mathbf{q}_1), \ldots, L_r(\mathbf{q}_p)) \subseteq \bigcup_{\mathbf{q}\in D_{b(\mathbf{q}_1,\ldots,\mathbf{q}_p)}} L_r(\mathbf{q}) .$$

En effet, si r est un calcul alors $\forall\ g = b(\mathbf{q}_1, \ldots, \mathbf{q}_p)\ \ \forall t_i \in L_r(\mathbf{q}_i)$ alors $r(b(t_1, \ldots, t_p))$ existe et appartient à D_g. Donc, $b(t_1, \ldots, t_p) \in \bigcup_{\mathbf{q}\in D_g} L_r(\mathbf{q})$.

- Les contraintes de SC_2 sont satisfaites.

$$\forall r \in \mathcal{R}_{\mathcal{A}} \text{ et } \forall \mathbf{q}, \mathbf{q}' \in \mathcal{Q} \text{ avec } \mathbf{q} \neq \mathbf{q}' \quad L_r(\mathbf{q}) \cap L_r(\mathbf{q}') = \emptyset \ .$$

Supposons qu'il existe deux états **q** et **q'** tels que $L_r(\mathbf{q}) \cap L_r(\mathbf{q}') \neq \emptyset$. Alors il existe un terme t tel que $r(t) = \mathbf{q}$ et $r(t) = \mathbf{q}'$. Ceci contredit le fait que r est une application de T_Σ dans $\mathcal{Q}$.

- Les contraintes de SC_4 sont satisfaites.

$$\bigcup_{\mathbf{q} \in \mathcal{Q}} L_r(\mathbf{q}) = \top$$

La preuve est directe puisque r est une application de T_Σ dans $\mathcal{Q}$.

Finalement, grâce aux contraintes de SC_3 chaque calcul accepte un n-uplet de langages qui appartient à l'ensemble des solutions de SC restreint à E. □

Fait 4.2 $SOL(SC)_{|E} \supseteq \mathcal{L}(\mathcal{A})$.

Preuve. Nous prouvons que si $(L_1, \ldots, L_n)$ est un n-uplet de $SOL(SC)_{|E}$, alors il existe un calcul r tel que $\mathcal{L}(\mathcal{A}, r) = (L_1, \ldots, L_n)$.

Soit $\mathcal{I}$ une interprétation des variables de SC solution du système. Soit r tel que

$$\forall t \in T_\Sigma \quad t \in \mathcal{I}(Z_{\mathbf{q}}) \Leftrightarrow r(t) = \mathbf{q}.$$

- r est une application de T_Σ dans $\mathcal{Q}$. En effet, d'après les contraintes de SC_3 et SC_4, r est définie pour tout terme de T_Σ et $r(t) = r(t')$ si et seulement si $t = t'$.

- r est compatible avec les règles de $\mathcal{S}$.

$$\forall b \in \Sigma_p \ \forall t_1, \ldots, t_p \in T_\Sigma \quad b(r(t_1), \ldots, r(t_p)) \rightarrow r(b(t_1, \ldots, t_p))$$

En effet, si $r(t_i) = \mathbf{q}_i$ pour tout i de $\{1, \ldots, p\}$, alors $t_i \in \mathcal{I}(Z_{\mathbf{q}_i})$ et

$$b(t_1, \ldots, t_p) \in \left(\mathcal{I}(Z_{\mathbf{q}_1}), \ldots, \mathcal{I}(Z_{\mathbf{q}_p})\right).$$

De plus $\mathcal{I}$ est une solution de SC, donc la contrainte suivante est satisfaite :

$$b(Z_{\mathbf{q}_1}, \ldots, Z_{\mathbf{q}_p}) \subseteq \bigcup_{\mathbf{q}' \in D_g} Z_{\mathbf{q}'} \text{ avec } g = b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \ .$$

L'ensemble $b(\mathcal{I}(Z_{\mathbf{q}_1}), \ldots, \mathcal{I}(Z_{\mathbf{q}_p}))$ est non vide et il existe un état **q** de D_g tel que $b(t_1, \ldots, t_p) \in \mathcal{I}(Z_{\mathbf{q}})$. Il existe donc une règle $b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \rightarrow \mathbf{q}$ dans $\mathcal{S}$. □

4.5.3 Les Classes *PSC* et *PNSC*

Nous montrons que la classe *PNSC* a une puissance d'expression strictement plus forte que la classe *PSC*. Nous utilisons pour cela les résultats de la section 3.3.2.

D'après le théorème 4.3, à toute contrainte positive *PC* correspond un automate sans condition d'acceptation reconnaissant les solutions de *PC*. On déduit alors grâce à la proposition 3.7, que

Proposition 4.5 *L'ensemble des solutions d'un système de contraintes positives est compact.*

Dans le cas de contraintes négatives, cette propriété n'est pas toujours vérifiée. Soit Σ l'alphabet composé d'une constante a et d'une lettre b d'arité 1. Considérons la contrainte $X \neq \bot$ construite sur Σ. Alors l'ensemble des solutions de $X \neq \bot$ est reconnu par le *TSA* $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ avec :

- $\mathcal{Q} = \{q_X, q_{\bar{X}}\}$;
- $\mathcal{F} = \{q_X\}$;
- $\Omega = \{\{q_X\}, \{q_X, q_{\bar{X}}\}\}$;
- $\mathcal{S}$ est l'ensemble

$$\begin{array}{rcl} a & \rightarrow & q_X \mid q_{\bar{X}} \ , \\ b(q_X) & \rightarrow & q_X \mid q_{\bar{X}} \ , \\ b(q_{\bar{X}}) & \rightarrow & q_X \mid q_{\bar{X}} \ . \end{array}$$

or, l'ensemble $\mathcal{L}(\mathcal{A}) = \{L \mid L \neq \emptyset\}$ n'est pas compact. En effet, la suite $(\mathcal{L}_i)_{i \geq 0}$ définie par

$$\begin{array}{rcl} L_0 & = & \{a\} \ , \\ L_1 & = & \{b(a)\} \ , \\ L_2 & = & \{b(b(a))\} \ , \\ & \vdots & \\ L_i & = & \{b^i(a)\} \ , \\ & \vdots & \end{array}$$

est une suite infinie extraite de $\mathcal{L}(\mathcal{A})$, car $\forall i \ L_i \neq \emptyset$, qui converge vers $\emptyset$. Or, $\emptyset$ n'est pas dans $\mathcal{L}(\mathcal{A})$, et donc $\mathcal{L}(\mathcal{A})$ n'est pas compact. Nous en déduisons,

Proposition 4.6 *La puissance d'expression des contraintes de la classe PNSC est strictement supérieure à celle des contraintes de la classe PSC.*

4.5.4 Equivalence de systèmes de contraintes

Deux systèmes de contraintes ensemblistes sont *équivalents* si ils possèdent les mêmes solutions. Des résultats de la section 3.3.2, nous déduisons directement :

Proposition 4.7 *Deux systèmes de contraintes positives sont équivalents si ils possèdent les mêmes solutions régulières.*

Une procédure de décision de l'équivalence de deux systèmes de contraintes est obtenue à l'aide des propriétés de clôture des *TSA* et du théorème 4.2.

Soit C une contrainte de la classe *PNSC*. Soit $\bar{C}$ la *négation* de la contrainte C. Syntaxiquement, si $C \equiv exp \subseteq exp'$ alors $\bar{C} \equiv exp \not\subseteq exp'$; inversement, si $C \equiv exp \not\subseteq exp'$ alors $\bar{C} \equiv exp \subseteq exp'$.

La négation d'un système de contraintes ensemblistes SC de *PNSC* à k contraintes, positives ou négatives, $C_1, \ldots, C_k$ est la disjonction des négations de chaque contrainte C_i.

$$SC \equiv C_1 \wedge C_2 \wedge C_3 \wedge \ldots \wedge C_k$$
$$\overline{SC} \equiv \bar{C}_1 \vee \bar{C}_2 \vee \bar{C}_3 \vee \ldots \vee \bar{C}_k$$

L'ensemble des solutions de SC est le complémentaire de l'ensemble des solutions de la négation de SC. Notons

$$\overline{SOL(SC)} = \{(L_1, \ldots, L_n) \mid (L_1, \ldots, L_n) \notin SOL(SC)\} \ .$$

Fait 4.3 *Soit SC un système de contraintes de PNSC à n variables.*

$$SOL(SC) = \overline{SOL(\overline{SC})} \ .$$

Preuve. Vérifions dans un premier temps que pour toute contrainte C de *PNSC*, nous avons $SOL(C) = \overline{SOL(\bar{C})}$. En effet, soit $\mathcal{L}$ un n-uplet de langages dans $SOL(C)$. Alors, à $\mathcal{L}$ est associée une interprétation $\mathcal{I}$ des variables de C. Dans le cas où C est de la forme $exp \subseteq exp'$, alors cette interprétation vérifie :

$$\forall t \in T_\Sigma \quad t \in \mathcal{I}(exp) \Rightarrow t \in \mathcal{I}(exp') \ .$$

Donc, toute interprétation qui ne satisfait pas C vérifie :

$$\exists t \in T_\Sigma \quad t \in \mathcal{I}(exp) \wedge t \notin \mathcal{I}(exp') \ ,$$

ce qui implique donc pour toute contrainte positive C.

$$SOL(C) = \overline{SOL(\bar{C})} \ .$$

La preuve est identique pour une contrainte négative.

Le fait est maintenant direct puisque par définition, la négation d'une conjonction de contraintes $C_1 \wedge \ldots \wedge C_k$ est la disjonction $\bar{C}_1 \vee \ldots \vee \bar{C}_k$ des négations de chaque contrainte. □

Fait 4.4 *Soit SC un système de PNSC.*
On peut construire un TSA $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = SOL(\overline{SC})$.

Preuve. La preuve est une conséquence de la clôture par union des *TSA*. En effet, si C est une contrainte de *PNSC*, alors il en est de même de $\bar{C}$. Par le théorème 4.2 il existe un *TSA* $\mathcal{A}_C$ tel que $SOL(\bar{C}) = \mathcal{L}(\mathcal{A}_C)$.

Maintenant, si SC est un système de *PNSC* $SC \equiv C_1 \wedge C_2 \wedge \ldots \wedge C_k$, alors $\overline{SC} \equiv \bar{C}_1 \vee \ldots \vee \bar{C}_k$.

Nous déduisons donc de la clôture des *TSA* par les opérations d'union et de cylindrification (voir page 102) que l'on peut construire un *TSA* $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = SOL(\overline{SC})$. □

Théorème 4.4 *Soient SC_1 et SC_2 deux systèmes de PNSC. Alors, l'inclusion $SOL(SC_1) \subseteq SOL(SC_2)$ est décidable.*

Preuve. Soient SC_1 et SC_2 deux systèmes de *PNSC*. Soient $\mathcal{A}_1$, $\overline{\mathcal{A}_2}$, les *TSA* associés respectivement aux systèmes SC_1, $\overline{SC_2}$.

Remarquons que

$$SOL(SC_1) \subseteq SOL(SC_2) \Leftrightarrow SOL(SC_1) \cap \overline{SOL(SC_2)} = \emptyset \ .$$

Par le fait 4.3, nous avons :

$$SOL(\overline{SC_2}) = \overline{SOL(SC_2)} \ .$$

Donc,

$$SOL(SC_1) \subseteq SOL(SC_2) \Leftrightarrow SOL(SC_1) \cap SOL(\overline{SC_2}) = \emptyset \ .$$

Grâce aux opérations de clôture de $\mathcal{TSA}$ et au fait 4.4, on peut construire un *TSA* $\mathcal{A}$ tel que :

$$\mathcal{L}(\mathcal{A}) = \left(\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\overline{\mathcal{A}_2})\right) \ .$$

Ce qui entraîne :

$$SOL(SC_1) \subseteq SOL(SC_2) \Leftrightarrow \mathcal{L}(A) = \emptyset \ .\square$$

Corollaire 4.1 *L'équivalence de deux systèmes de contraintes ensemblistes dans PNSC est décidable.*

4.5.5 Résultats de Complexité

La complexité du problème de la satisfiabilité des systèmes de contraintes ensemblistes a été largement étudiée par Aiken Kozen Vardi et Wimmers [AKVW93] et Stefansson [Ste93].

Stefansson a montré que la complexité de la satisfiabilité des contraintes de la classe *PNSC* est un problème NEXPTIME-complet. Les résultats de Aiken et al sont résumés dans le tableau page 22 et 129.

Dans notre algorithme, nous avons préféré exposer la résolution d'une seule contrainte. Les propriétés de clôture des langages *TSA*-reconnaissables nous permettent ensuite de résoudre des systèmes entiers. L'opération d'intersection de deux *TSA* n'est pas performante et il est possible de s'en passer.

Si SC est un système de contraintes ensemblistes, l'ensemble $E(SC)$ des sous-expressions de SC est l'union de tous les ensembles $E(C)$ pour chaque contrainte C du système. Les spécifications des ensembles d'états, d'états finaux et de l'ensemble des règles de l'automate associé à SC sont alors les mêmes que dans le cas d'une seule contrainte. Seule, la définition de Ω est modifiée. La collection Ω contient tous les sous-ensembles d'états ω tels que :

$$\left(\bigwedge_{exp \subseteq exp' \in SC} \forall \varphi \in \omega \ \varphi(exp) \Rightarrow \varphi(exp')\right)$$
$$\wedge \left(\bigwedge_{exp \not\subseteq exp' \in SC} \exists \varphi \in \omega \ \neg(\varphi(exp) \Rightarrow \varphi(exp'))\right)$$

La taille d'un système de contrainte SC est le nombre de symboles qui apparaissent dans SC. Le nombre de sous-expressions de SC est linéairement dépendant de la taille de SC.

En général, la construction de l'automate associé à un système SC de contraintes est exponentielle par rapport à la taille de SC. En effet, le nombre d'états est le nombre de valuations booléennes, donc le nombre d'applications de $E(SC)$ dans $\mathcal{B}$ qui vérifient les propriétés données page 112. Le nombre d'états est donc de l'ordre de 2^k avec $k = Card(E(SC))$.

La complexité du problème du vide dépend fortement des conditions d'acceptation.

Complexité de l'Existence d'un Calcul

Rappelons que ce problème est équivalent au problème du vide dans le cas "sans condition d'acceptation".

Proposition 4.8 *Le problème de l'existence d'un calcul dans un Automate d'Ensembles d'Arbres est NP-Complet.*

Preuve. Ce problème est clairement NP. La vérification qu'un ensemble ω satisfait à la condition COND(ω) est réalisée en un temps polynômial par rapport à n_ω la cardinalité de ω.

Si b est un symbole d'arité p, alors, pour vérifier que $\forall \mathbf{q}_1, \ldots, \mathbf{q}_p \in \omega \; \exists \mathbf{q} \in \omega \quad b(\mathbf{q}_1, \ldots, \mathbf{q}_p) \rightarrow \mathbf{q} \in \mathcal{S}$ il est nécessaire de construire ${n_\omega}^p$ n-uplets d'états. Vérifier COND(ω) prend donc un temps polynômial de l'ordre de $\sum_{i=0}^{amax} Card\,(\Sigma_i) \times {n_\omega}^i \times Card(\mathcal{S})$.

Nous allons réduire *SAT*, le problème NP-Complet de satisfiabilité d'une expression booléenne dans le problème de l'existence d'un calcul dans un TSA. Soit P une instance de *SAT*, exprimée à l'aide des connecteurs $\neg$ et $\wedge$. Supposons que P comporte n variables propositionnelles nommées $a_1, \ldots a_n$. Nous pouvons représenter P sous la forme d'un arbre t_P dont les feuilles sont labellées par les variables a_i et les nœuds sont labellés par les connecteurs logiques $\neg$ (d'arité 1) et $\wedge$ (d'arité 2). Alors une sous-expression de P sera un sous-terme de t_P. Rappelons que l'ordre "sous-terme de" est noté $\trianglelefteq$. La réduction *Red* construit le *TSA* $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{F}, \mathcal{S}, \Omega)$ suivant à partir de P.

- $\Sigma_0 = \{a_1, a_2, \ldots, a_k\}$, $\Sigma_1 = \{\neg\}$, $\Sigma_2 = \{\wedge\}$, $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$;
- $\mathcal{Q} = \{V_t, F_t \mid t \trianglelefteq t_P\} \cup \{OK\}$;
- Pour toute constante $a_i \in \Sigma_0 \quad a_i \rightarrow V_{a_i} \mid F_{a_i} \in \mathcal{S}$

si $\neg(t) \trianglelefteq t_P$ alors $\begin{cases} \neg(V_t) \rightarrow F_{\neg(t)} \in \mathcal{S} \\ \neg(F_t) \rightarrow V_{\neg(t)} \in \mathcal{S} \end{cases}$

sinon $\begin{cases} \neg(V_t) \rightarrow OK \in \mathcal{S} \\ \neg(F_t) \rightarrow OK \in \mathcal{S} \end{cases}$

si $\wedge(t, t') \trianglelefteq t_P$ alors $\begin{cases} \wedge(F_t, F_{t'}) \rightarrow F_{\wedge(t,t')} \in \mathcal{S} \\ \wedge(F_t, V_{t'}) \rightarrow F_{\wedge(t,t')} \in \mathcal{S} \\ \wedge(V_t, F_{t'}) \rightarrow F_{\wedge(t,t')} \in \mathcal{S} \\ \wedge(V_t, V_{t'}) \rightarrow V_{\wedge(t,t')} \in \mathcal{S} \end{cases}$

sinon $\begin{cases} \wedge(F_t, F_{t'}) \rightarrow OK \in \mathcal{S} \\ \wedge(F_t, V_{t'}) \rightarrow OK \in \mathcal{S} \\ \wedge(V_t, F_{t'}) \rightarrow OK \in \mathcal{S} \\ \wedge(V_t, V_{t'}) \rightarrow OK \in \mathcal{S} \end{cases}$

$$\left.\begin{array}{c} V_{t_P} \in \{q_1, q_2\} \text{ ou } OK \in \{q_1, q_2\} \\ \text{et} \\ F_{t_P} \notin \{q_1, q_2\} \end{array}\right\} \Rightarrow \begin{cases} \neg(q_1) \rightarrow OK \in \mathcal{S} \\ \wedge(q_1, q_2) \rightarrow OK \in \mathcal{S} \end{cases}$$

Les ensembles Ω et $\mathcal{F}$ sont quelconques. Ils n'ont aucune incidence sur l'existence d'un calcul. Remarquons que le seul non déterminisme dans cet ensemble de règles apparaît lorsque le membre gauche est une constante. Remarquons enfin qu'il n'existe aucune règle avec l'état F_{t_P} en membre gauche. Intuitivement, l'application des règles initiales (celles dont le membre gauche est une constante) fournit une inteprétation pour les variables propositionnelles. L'état F_{a_i} signifie $a_i = Faux$ et l'état V_{a_i} signifie $a_i = Vrai$. C'est, comme nous l'avons dit, le seul

choix dans l'application des règles. Ensuite, une valeur de vérité est donnée à chaque sous-expression t de P par l'application des autres règles (F_t ou V_t). Si lors d'un début de calcul la valeur *Faux* est attribuée à P, l'état F_{t_P} est atteint et le calcul ne peut se poursuivre puisqu'aucune règle avec F_{t_P} en membre gauche n'existe.

Pour prouver cette réduction, montrons d'abord que *Red* est une fonction polynômiale en temps de la taille de la donnée P. La taille de P, notée n, est le nombre symboles qui occurent dans P. Elle est encore égale (aux parenthèses près) au nombre de nœuds dans l'arbre t_P. Maintenant, puisque $\mathcal{Q} = \{V_t, F_t \mid t \trianglelefteq t_P\} \cup \{OK\}$, nous avons $Card(\mathcal{Q}) = 2n + 1$. Un rapide examen montre que le nombre de règles de $\mathcal{S}$ est de l'ordre de $Card(\mathcal{Q})^2$. Le temps de calcul pour la définition de $\mathcal{A}$ est donc polynômial.

Terminons la preuve en montrant que ce codage est bien correct, c'est-à-dire :

Fait 4.5 *P est satisfiable si et seulement si il existe un calcul dans l'automate $\mathcal{A}$.*

Preuve.

$\Rightarrow$ Supposons P satisfiable par l'interprétation I des variables propositionnelles $\{a_1, \ldots, a_k\}$. Un calcul r peut être construit de la façon suivante : $r(a_i) = V_i \Leftrightarrow I(a_i) = V$. La définition des règles de $\mathcal{S}$ implique que tout arbre t de T_Σ correspondant donc à une expression booléenne U a pour image par r : V_t si $t \trianglelefteq t_P$ et U est vraie sous I ; F_t si $t \trianglelefteq t_P$ et U est fausse sous I ; et OK sinon.

$\Leftarrow$ Supposons qu'il existe un calcul r dans $\mathcal{A}$. Alors l'arbre t_P aura pour image par r un état de $\mathcal{Q}$. Par définition des règles de $\mathcal{S}$, $r(t_P)$ ne peut valoir que V_{t_P}. Donc P est satisfiable. □

Complexité des Contraintes

La méthode de résolution de Aiken et al est très proche de la nôtre.

Soit SC un système de contraintes positives, H l'hypergraphe associé à SC et $\mathcal{A}$ le *TSA* associé à SC.

Informellement, l'hypergraphe qui est construit dans [AKVW93, AKW93] correspond au *TSA* construit ici. Dans une première étape, Aiken et al transforment SC dans le but d'obtenir un système sous forme normale. Cette opération applatit le système et introduit de nouvelles variables. Il est clair qu'alors, ces nouvelles variables représentent les sous-expressions du système original. Les nœuds et les relations de l'hypergraphe H, construit à partir de SC, sont alors intimement liés aux états et règles du *TSA* $\mathcal{A}$, associé à SC. Finalement, un sous-hypergraphe clos induit par H, correspond à un sous-ensemble ω d'états, qui vérifie $COND(\omega)$ dans $\mathcal{A}$.

La taille de l'hypergraphe, comme celui du *TSA* est en général exponentielle par rapport à la taille du système.

Pour montrer la satisfiabilité d'un système SC de contraintes ensemblistes, la construction complète de l'automate d'ensembles d'arbres associé $\mathcal{A}_{SC}$ n'est pas utile. Il suffit de montrer que l'automate donné par sa spécification, c'est-à-dire les règles de construction de $\mathcal{A}_{SC}$, ne reconnaît pas le vide. Dans [AKVW93], le même raisonnement est tenu. L'existence d'un sous-hypergraphe clos induit, est montrée simplement à partir de la spécification de l'hypergraphe par des formules boolénnes.

Les résultats de complexité de [AKVW93] sont donc directement transposés ici, puisque les preuves basées sur l'hypergraphe peuvent être vues comme des preuves sur le *TSA*. Nous rappelons ces résultats :

Nombre d'élments dans Σ			Complexité du problème de la satisfiabilité
d'arité 0	d'arité 1	d'arité ≥ 2	
0	0 ou plus	0 ou plus	trivial
1 ou plus	0	0	NP-complet
1 ou plus	1	0	PSPACE-complet
1 ou plus	2 ou plus	0	EXPTIME-complet
1 ou plus	0 ou plus	1 ou plus	NEXPTIME-complet

TAB. 4.1 - : Complexité des contraintes

Les auteurs donnent une bonne idée intuitive des ces résultats.

> "Si Σ ne contient que des symboles constantes, alors le problème est essentiellement un problème de satisfiabilité boolénne. Un système avec un symbole unaire a la puissance de compter en unaire, ce qui est suffisant pour simuler un calcul PSPACE. Un système avec au moins deux symboles unaires peut coder le fonctionnement d'un machine alternante, ce qui donne PSPACE sur une machine alternante ou encore EXPTIME. Finalement, dans un système avec au moins un symbole d'arité deux ou plus, il peut exister une contrainte $b(X, Y) \subseteq \bot$, qui est satisfiable si et seulement si $X \subseteq \bot$ ou $Y \subseteq \bot$. Ceci permet de coder le non déterminisme, élevant la complexité à NEXPTIME."

Extensions

Nous avons présenté un algorithme de résolution pour les systèmes de contraintes ensemblistes positives et négatives à l'aide d'un outil original : les Automates d'Ensembles d'Arbres ou *TSA*.

Nous souhaitons étudier plus en détail les *TSA*. Nous envisageons de leur donner une puissance d'expression plus grande en modifiant leurs conditions d'acceptation. Par exemple, la réussite d'un calcul peut-être contrôlée par la finitude du nombre d'occurrences de certains états. Nous espérons ainsi obtenir des résultats de décision pour la classe des contraintes avec symboles de projection.

D'autre part, l'intégration de contraintes d'égalités et de différences aux règles des *TSA* permet d'exprimer des relations plus fines entre ensembles de termes et peut donc conduire à des contraintes ensemblistes plus expressives. Cette extension se rapproche de celles données aux automates d'arbres finis par Bogaert et Tison [BT92] ou encore Caron, Coquidé et Dauchet [CCD93b].

Dans le but de définir une sous-classe des langages *TSA*-reconnaissables close par complémentarité, nous envisageons d'adapter les *TSA* et de reconnaître des graphes infinis $\{0,1\}^n$-valués. Nous pouvons voir de tels graphes comme les représentations de fonctions caractéristiques de n-uplets de langages d'arbres. Les règles de ces nouveaux automates comporteraient en membre gauche, un label de $\{0,1\}^n$ et seraient de la forme : $(label, symbole, etat_1, \ldots, etat_p) \rightarrow etat$.

Enfin, les procédures de décision que nous avons présentées ne sont pas facilement implantables. La complexité (dans le pire des cas) est telle qu'il est nécessaire de proposer des heuristiques pour une résolution efficace. Nous tenons aussi à montrer qu'une implantation basée sur les *TSA* est réalisable.

Index

Bibliographie

[Ack54] W. Ackermann. *Solvable Cases of the Decision Problem.* North-Holland, Amsterdam, 1954.

[AKVW93] A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The complexity of set constraints. In *Proceedings of Computer Science Logic*, 1993. Techn. Report 93-1352, Cornell University.

[AKW93] A. Aiken, D. Kozen, and E. Wimmers. Decidability of systems of set constraints with negative constraints. Technical Report 93-1362, Computer Science Department, Cornell University, 1993.

[AM91] A. Aiken and B. R. Murphy. Implementing regular tree expressions. In *Proceedings of the ACM conf. on Functional Programming Languages and Computer Architecture*, pages 427–447, 1991.

[Ard61] D.N. Arden. Delayed logic and finite state machines. In *Proceedings of the* 2^{nd} *Ann. Symp. on Switching circuit Theory and Logical Design*, pages 133–151, Detroit, 1961.

[AW92] A. Aiken and E.L. Wimmers. Solving Systems of Set Constraints. In *Proceedings of the* 7^{th} *Symposium on LICS*, pages 329–340, 1992.

[AW93] A. Aiken and E.L. Wimmers. Type inclusion constraints and type inference. In *Proceedings of the ACM conf. on Functional Programming Languages and Computer Architecture*, pages 31–41, 1993.

[AWL94] A. Aiken, E.L. Wimmers, and T.K. Lakshman. Soft typing with conditional types. In *To appear in Proceedings of the* 21^{th} *Symp. on Principle on Programming Languages*, 1994.

[BGW92] L. Bachmair, H. Ganzinger, and U. Waldmann. Solving Set Constraints by Ordered Resolution with Simplification. Technical Report MPI-I-92-240, Max Plank Institute, December 1992.

[BGW93] L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Proceedings of the* 8^{th} *Symposium on LICS*, pages 75–83, 1993.

[BL80] J.A. Brzozowski and E. Leiss. On equations for regular languages, finite automata, and sequential networks. *Theoretical Computer Science*, 10:19–35, 1980.

[BT92] B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In *Lectures Notes in Computer Science*, volume 577, pages 161–171, Paris, 1992. Symposium on Theorical Aspects of Computer Science.

[Büc60] J.R. Büchi. On a decision method in a restricted second order arithmetic. In Stanford Univ. Press., editor, *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science*, pages 1–11, 1960.

[CCD93a] A.C. Caron, J.L. Coquidé, and M. Dauchet. Encompassment properties and automata with constraints. Technical report, Laboratoire d'Informatique Fondamentale de Lille, 1993. to appear.

[CCD93b] A.C. Caron, J.L. Coquidé, and M. Dauchet. Encompassment properties and automata with constraints. In C. Kirchner, editor, *LNCS 690*, pages 328–342, Montréal, 1993. 5th international conference on Rewriting Techniques and Applications.

[CF91] R. Cartwright and M. Fagan. Soft typing. In *Proceeding of the ACM SIGPLAN'91*, pages 278–292, 1991.

[Com90] H. Comon. Equational formulas on order-sorted algebras. In *Proceedings of ICALP'90*, pages 674–688, 1990.

[Cou83] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.

[Cou89] B. Courcelle. *On Recognizable Sets and Tree Automata*, chapter Resolution of Equations in Algebraic Structures. Academic Press, m. nivat and ait-kaci edition, 1989.

[CP93] W. Charatonik and L. Pacholski. Negative set constraints: an easy proof of decidability. Draft Version, Dec 1993.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction and polymorphism. *ACM computing surveys*, 17(4):472–522, 1985.

[Dau53] M. Dauchet. Cours d'informatique. Université de Lille, 1953.

[DJ90] N. Dershowitz and J.P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter Rewrite Systems, pages 243–320. Elsevier, 1990.

[DT90] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proceedings of 5th annual IEEE symposium on Logic in Computer Science.*, pages 242–248, June 1990.

[FSVY91] T. Früwirth, E. Shapiro, M.Y. Vardi, and E. Yardeni. Logic Programs as Types for Logic Programs. In *Proceedings of the* 6^{th} *Symposium on LICS*, 1991.

[GS84] F. Gecseg and M. Steinby. *Tree Automata.* Akademiai Kiado, 1984.

[GTT93a] R. Gilleron, S. Tison, and M. Tommasi. Solving System of Set Constraints using Tree Automata. In *Lectures Notes in Computer Science*, volume 665, pages 505–514, February 1993. Symposium on Theorical Aspects of Computer Science.

[GTT93b] R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints with negated subset relationships. In *Proceedings of the* 34^{th} *Symp. on Foundations of Computer Science*, pages 372–380, 1993.

[Hei92] N. Heintze. *Set Based Program Analysis.* PhD thesis, Carnegie Mellon University, 1992.

[HJ90a] N. Heintze and J. Jaffar. A Decision Procedure for a Class of Set Constraints. In *Proceedings of the* 5^{th} *Symposium on LICS*, pages 42–51, 1990.

[HJ90b] N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. In *Proceedings of the* 17^{th} *ACM Symp. on Principles of Programming Languages*, pages 197–209, 1990. Full version in the IBM tech. rep. RC 16089 (#71415).

[HU79] J.E. Hopcroft and J.D. Ullman. *Intorduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979.

[JM79] N.D. Jones and S.S. Muchnick. Flow Analysis and Optimization of LISP-like Structures. In *Proceedings of the* 6^{th} *ACM Symposium on Principles of Programming Languages*, pages 244–246, 1979.

[Kla90] N. Klarlund. *Automates d'arbres avec tests d'égalités.* PhD thesis, Cornell University, 1990.

[Koz76] D. Kozen. On parallelism in Turing machines. In *Proceedings of the* 17^{th} *Ann. Symp. on Foundations of Computer Science*, pages 89–97, 1976.

[Low15] L. Lowenheim. Uber moglichkeit im relativkalkul. *Mathematische Annalen*, 76:447–470, 1915. English traduction in "From Frege to Godel". Van Heijenoort editor.

[Mil78] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and Systm Sciences*, 17:348–375, 1978.

[Mis84] P. Mishra. Towards a Theory of Types in PROLOG. In *Proceedings of the* 1^{st} *IEEE Symposium on Logic Programming*, pages 456–461, Atlantic City, 1984.

[MO84] A. Mycroft and R. A. O'Keefe. A polymorphic type system for prolog. *Journal of Artificial Intelligence*, 23:295–307, 1984.

[MR85] P. Mishra and U. Reddy. Declaration-free type checking. In *Proceedings of the* 12^{th} *Annual ACM Symp. on the Principles of programming languages*, pages 7–21, 1985.

[Pin93] J.E. Pin. Logic and automata on words. In *Support de cours de l'école de jeunes chercheurs "Automates et logique"*, 1993.

[PS93] M. Plantel and P. Sallé. Typage souple pour le langage FOL. In *Actes des journées du GDR programmation du CNRS, Contraintes et programmation logique*, 1993.

[Rab69] M.O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[Rey69] J.C. Reynolds. Automatic Computation of Data Set Definition. *Information Processing*, 68:456–461, 1969.

[Ste93] K. Stefansson. Systems of set constraints with negative constraints are nexptime-complete. Technical Report 93-1380, Cornell University, 1993.

[Tho90] W. Thomas. *Handbook of Theoretical Computer Science*, volume B, chapter Automata on Infinite Objects, pages 134–191. Elsevier, 1990.

[Uri92] T. E. Uribe. Sorted Unification Using Set Constraints. In D. Kapur, editor, *Proceedings of the* 11^th *International Conference on Automated Deduction*, New York, 1992.

[YO88] J. Young and P. O'Keefe. *Experience with a type evaluator*, chapter Partial Evaluation and Mixed Computation, pages 573–581. North-Holland, d. bjxrner and n. d. jones edition, 1988.