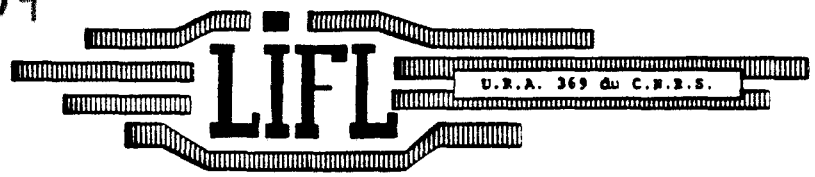
n° d'ordre 1064



LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

Thèse

présentée à

L'Université des Sciences et Technologies de Lille

pour l'obtention du titre de

Docteur en Informatique

par

Anne-Cécile Caron

Structures et Décision en Réécriture

Soutenue le 8 février 1993 devant le jury :

Maurice Nivat, Président.
André Arnold,
Hubert Comon, Rapporteurs.
Max Dauchet,
Harald Ganzinger,
Jean-Pierre Jouannaud,
Michel Parigot,
Michel Sintzoff,
Sophie Tison, Examinateurs.

UNIVERSITE [illegible] LOGIES 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. 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. 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. 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 Maurice Nivat qui me fait l'honneur de présider le jury de cette thèse.

J'exprime toute ma gratitude à André Arnold qui a bien voulu rapporter ces travaux.

Je remercie également Hubert Comon d'avoir accepté d'être rapporteur. Je le remercie de l'intérêt qu'il porte à mon travail et de ses remarques qui m'ont aidée dans la rédaction de ce document.

Mes remerciements vont également à Jean-Pierre Jouannaud, Harald Ganzinger, Michel Parigot et Michel Sintzoff qui me font l'honneur de participer au jury.

Ce travail doit beaucoup à Max Dauchet. Sa passion communicative de la recherche, sa compétence et sa gentillesse m'ont beaucoup aidée. Je lui suis très reconnaissante de la confiance qu'il m'a accordée.

Je tiens également à remercier Sophie Tison et toute l'équipe "automates et langages d'arbres". Merci aux barbus du 332: Bruno Bogaert, Francesco de Comité, Rémi Gilleron et tout particulièrement Jean-Luc Coquidé pour notre collaboration.

Henri Glanc a assuré la reproduction de cette thèse avec la compétence et la sympathie connues de tous. Je lui en suis reconnaissante.

Table des matières

Introduction

Ce mémoire présente trois contributions à l'étude des systèmes de réécriture, pris comme paradigme du calcul symbolique. Il est donc divisé en trois grandes parties, correspondant en fait à trois publications ([Car91],[Car92],[CCD93]).

La première partie présente deux résultats apparemment sans rapport et dont le lien de nature technique sera précisé ultérieurement. Nous y illustrons d'abord l'influence des sortes sur les propriétés de décision. Ainsi la confluence des systèmes noethériens devient indécidable si on ajoute des contraintes de sortes, même très simples. Ensuite nous répondons à une question de N. Dershowitz et J.-P. Jouannaud [DJ90] : la terminaison des systèmes de réécriture de mots de longueur non-croissante est indécidable.

La problématique de la deuxième partie touche à la modularité. L'idée est de décomposer l'étude d'un système de réécriture en sous-systèmes en fonction de propriétés syntaxiques simples liées aux alphabets sur lesquels on travaille. On obtient ainsi la notion d'union disjointe de systèmes de réécriture [Toy87a] et celle plus large de composition de systèmes constructeurs [MT91]. Les propriétés intéressantes sont évidemment celles préservées par recomposition. De ce point de vue, nous présentons ici une palette de résultats. Comme pour les travaux de nos prédécesseurs, le bilan est "mitigé" et confirme qu'il est difficile de conquérir la programmation symbolique en la divisant.

Dans la troisième partie, nous étudions les problèmes d'entourage, c'est-à-dire les propriétés du premier ordre construites à partir des atomes $\text{facteur}_t(x)$: t est facteur de x (i.e. x entoure t). Certaines propriétés d'incrémentalité de spécifications s'expriment ainsi (H. Comon [Com92], D. Plaisted [Pla85], J.-P. Jouannaud et E. Kounalis [JK89], Kapur, Narendran et Zhang [KNZ87]). Nous relions l'approche logique et l'approche automates (W. Thomas [Tho90] M.O. Rabin [Rab77]) ; nos résultats disent en ce sens que les contraintes d'entourage peuvent être manipulées comme des contraintes de sortes classiques. Nous pensons que c'est là l'apport le plus signifiant

du présent mémoire.

Les développements que nous comptons donner à notre travail concernent la dernière partie.

- Concevoir un algorithme de décision empiriquement efficace
- Peaufiner nos automates afin d'obtenir une présentation minimale des solutions.
- Resserrer les liens avec les techniques de résolution de formules équationnelles (H. Comon [Com88], M.J. Maher [Mah88]) et préciser l'idée que les problèmes d'entourage se manipulent comme des sortes en reprenant des algorithmes de résolution dans le cadre des algèbres sortées.
- Définir une classe d'automates plus générale, englobant celle construite par B. Bogaert et S. Tison, en autorisant un nombre non borné de tests d'égalités entre fils d'un même noeud [BT92].
- Etendre nos résultats à des algèbres quotient simples

Surtout, nous aimerions démontrer ce qui pour l'instant n'est qu'une conjecture:

La théorie du premier ordre de la réécriture pas à pas est décidable.

Ce que nous appelons réécriture pas à pas concerne tout problème de réécriture du premier ordre (les quantificateurs portant sur les termes), où l'on s'autorise seulement des relations binaires $t_1 \rightarrow_R t_2$ (t_1 se réécrit en 1 pas en t_2 par une règle du système de réécriture R) et non pas leurs clôtures réflexives et transitives $\stackrel{*}{\rightarrow}_R$. Dans le cas des mots ce problème est une conséquence de la décidabilité de la théorie du premier ordre des relations RR (M. Dauchet et S. Tison [DT90]) ou des relations rationnelles à délai borné (C. Frougny et J. Sakarovitch [FS92]).

Les trois paragraphes suivants présentent les résultats de cette thèse.

A Sortes et problèmes de décision

Nous introduisons ici les résultats dans un ordre différent de celui du chapitre 2. Nous présentons d'abord une propriété générale qui permet de mieux situer un résultat

sur les systèmes de réécriture de mots préservant les longueurs, ce dernier résultat étant en fait la motivation du chapitre.

Nous donnons un exemple de propriété décidable classique, de la forme $\forall x P(x)$, qui devient indécidable quand on restreint x à une certaine sorte. Il s'agit de la confluence des systèmes de réécriture noethériens.

Soit R un système de réécriture noethérien. Notons $P(R,x)$ la confluence à partir de x ($x \in T_\Sigma(\mathcal{X})$) : $P(R,x) = \forall u,v(x \xrightarrow{*}_R u \wedge x \xrightarrow{*}_R v \Rightarrow (\exists y\ u \xrightarrow{*}_R y \wedge v \xrightarrow{*}_R y))$

Le théorème de Newman permet de prouver la décidabilité de $P(R) = \forall x P(R,x)$.

Nous montrons que, pour des systèmes de réécriture de mots, la propriété $P(R,S) = \forall x \in S\ P(R,x)$, où S est une sorte, est indécidable, même si S exprime simplement que x commence par une lettre donnée. On pourrait objecter que les sortes s'expriment dans le cadre des termes clos, et que la confluence close est déjà indécidable pour les systèmes de réécriture noethériens, sans restriction de sorte. Mais on remarquera que, dans le cas des mots, la confluence close et la confluence coïncident (Voir le lemme d'identification du chapitre 2). Notons que F. Otto démontre dans [Ott87] que la confluence sur des classes de congruence est indécidable, mais les classes de congruence ne sont, en général, pas reconnaissables.

Venons-en à notre problème originel. N. Dershowitz et J.-P. Jouannaud posent dans [DJ90] le problème de la décidabilité de la terminaison des systèmes de réécriture de mots de longueur non croissante. Notre idée était d'utiliser l'indécidabilité de la terminaison de machines de Turing particulières afin de démontrer celle des systèmes de longueur non-croissante. En 1966, le mathématicien P.K. Hooper en étudiant la terminaison des machines de Turing [Hoo66], considère le cas des machines dont la taille du ruban est bornée en fonction de la taille de la donnée initiale. Ces machines sont aussi appelées automates linéairement bornés (ALB).

On voit assez facilement comment simuler un ALB avec un système de réécriture. Le tableau suivant donne un exemple de traduction possible :

ALB	SDR
$(q,a) \to (b,q',\text{Droite})$	$qa \to bq'$
$(q,a) \to (b,q',\text{Gauche})$	$\forall c \in \Sigma$ $cqa \to q'cb$

Hooper démontre l'indécidabilité de la terminaison des machines de Turing et des automates linéairement bornés, et suggère d'utiliser ce dernier résultat pour prouver

l'indécidabilité de la terminaison des systèmes de réécriture de mots qui préservent les longueurs. Dans le même ordre d'idées, F. Otto [Ott86] utilise les ALB dans un tout autre but: il réduit le problème du vide des ALB déterministes au problème de l'auto-imbrication des systèmes de mots, afin de démontrer que celui-ci est indécidable.

Mais il ne faut pas oublier qu'une machine de Turing démarre ses calculs par une configuration initiale alors qu'un système de réécriture s'applique à des mots quelconques. En effet, pour simuler un ALB par un système de réécriture de mots préservant les longueurs, il faut commencer les séquences de dérivation sur des mots représentant des configurations initiales de l'automate. Sachant que l'ensemble des configurations initiales d'un automate constitue un langage reconnaissable (i.e. une sorte S), la simulation d'un automate par un système de réécriture revient à restreindre ce système à la sorte S. L'influence des sortes sur les problèmes de décision ne nous garantissant pas l'indécidabilité cherchée comme conséquence directe du résultat de Hooper, nous reprenons dans une preuve directe les même principes de simulation.

B Propriétés modulaires

Il existe des algorithmes de décision pour les propriétés "connues" des systèmes de réécriture, mais ils sont souvent inefficaces pour des systèmes ayant un grand nombre de règles. C'est pourquoi il est intéressant de savoir si, en découpant un gros système en systèmes plus petits, on peut appliquer l'algorithme de décision sur ces sous-systèmes et en déduire la propriété du système initial. Bien entendu, cette idée est utopique en général. Considérons par exemple le système $\{a \to b, b \to a\}$, ce système n'est pas noethérien bien que chacun de ses sous-systèmes (constitués d'une seule règle) le soient. De même, le système $\{a \to b, a \to c\}$ n'est pas confluent alors que ses sous-systèmes le sont. C'est pourquoi il est nécessaire d'apporter des restrictions.

L'exemple de la réécriture dans les mots illustre combien la réunion de systèmes travaillant sur un même alphabet boulverse les propriétés des systèmes: considérons un alphabet Σ et l'opération de concaténation. Dans l'algèbre libre, toute propriété du premier ordre des systèmes de réécriture clos est décidable [DT90]. Si on ajoute le seul axiome d'associativité de la concaténation on obtient la classe des systèmes de réécriture dans les mots qui simulent les machines de Turing et on perd donc toute propriété de décision. On trouvera des résultats partiels dans (M. Nivat, M.

Benois [NB70]) et (Y. Cochet, M. Nivat [CN71]).

Face à ce problème, Y. Toyama a eu l'idée de décomposer un système de réécriture en sous-systèmes travaillant sur des alphabets disjoints. Il a alors posé les définitions suivantes :

Si $\mathcal{F}_1$ et $\mathcal{F}_2$ sont deux alphabets gradués, finis et disjoints, et si $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ sont deux systèmes de réécriture de termes sur $\mathcal{F}_1$ et $\mathcal{F}_2$ alors l'*union disjointe* $\mathcal{R}_1 \oplus \mathcal{R}_2$ de $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ est le système de réécriture $(\mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{R}_1 \cup \mathcal{R}_2)$. Une propriété des systèmes de réécriture est dite *modulaire* si elle est préservée par union disjointe.

Y. Toyama a montré dans [Toy87b] que la confluence était préservée par union disjointe. Il a réécrit depuis une preuve simplifiée de ce résultat avec J.W. Klop, A. Middeldorp et R. de Vrijer [KMTdV91].

Il a également montré que la terminaison n'était pas modulaire, grâce à ce contre-exemple très simple [Toy87a] :

Soient $\mathcal{R}_1 = \{F(0,1,x) \to F(x,x,x)\}$ et $\mathcal{R}_2 = \left\{ \begin{array}{l} \mathrm{ou}(x,y) \to x \\ \mathrm{ou}(x,y) \to y \end{array} \right\}$

Les deux systèmes sont noethériens, mais $\mathcal{R}_1 \oplus \mathcal{R}_2$ admet une dérivation cyclique :

$$\begin{array}{rcl} F(\mathrm{ou}(0,1),\mathrm{ou}(0,1),\mathrm{ou}(0,1)) & \to & F(0,\mathrm{ou}(0,1),\mathrm{ou}(0,1)) \\ & \to & F(0,1,\mathrm{ou}(0,1)) \\ & \to & F(\mathrm{ou}(0,1),\mathrm{ou}(0,1),\mathrm{ou}(0,1)) \end{array}$$

B. Gramlich a étudié plus en détail des contre-exemples minimaux à la modularité de la terminaison [Gra91]. Le contre-exemple de Y. Toyama a amené M. Rusinowitch [Rus87] à rechercher des conditions suffisantes à la modularité de la terminaison. Plus précisemment, il a montré que l'union disjointe de deux systèmes noethériens était noethérienne si ni $\mathcal{R}_1$ ni $\mathcal{R}_2$ n'avait de règle effaçante (i.e. de règle où le membre droit est réduit à une variable) ou si aucun des deux n'avaient de règle duplicante (de règle non linéaire à droite). Dans le même ordre d'idées, A. Middeldorp [Mid89b] a démontré que la terminaison était modulaire si l'un des deux systèmes ne contenait ni règle effaçante, ni règle duplicante. Kurihara et Ohuchi [KO90a] ont prouvé que la terminaison était modulaire si elle pouvait se démontrer par un ordre de simplification.

Une autre propriété étudiée est la complétude, c'est-à-dire la confluence et la terminaison. Barendregt et Klop dans [Toy87b] et Drosten dans [Dro89] donnent des contre-exemples à la modularité de la complétude. Cependant, Toyama, Klop et Ba-

rendregt [TKB89] montrent que l'union disjointe de deux SDR linéaires à gauche et complets est un système complet.

A. Middeldorp étudie dans [Mid89a] certaines propriétés concernant les formes normales et prouve que la propriété de tous les termes de posséder une forme normale unique est modulaire. Il donne dans sa thèse [Mid91] une vue d'ensemble sur les aspects modulaires des systèmes de réécriture de termes et étudie plus particulièrement les systèmes de réécriture conditionnels [Mid91].

Pour notre part, nous étudions la modularité de la décidabilité de l'accessibilité. Le problème de l'accessibilité pour deux termes t et t' et un système de réécriture $\mathcal{R}$ est de savoir si t se réécrit en t' par les règles de $\mathcal{R}$. Ce problème est donc aux systèmes de réécriture de termes ce qu'est le problème du mot pour les systèmes semi-Thue. Il est indécidable en général mais décidable pour des systèmes quasi-clos (M. Oyamaguchi [Oya86]), c'est-à-dire des systèmes clos à droite ou pour des systèmes monadiques. Un algorithme de décision de l'accessibilité pour des systèmes clos utilisant les automates d'arbres a été implémenté en Prolog par A. Deruyver [DD89], ainsi qu'un algorithme permettant de trouver une dérivation qui transforme un terme t en un terme t' (J.-L. Coquidé et R. Gilleron [CG90]).

Nous démontrons d'une part que la décidabilité de l'accessibilité n'est pas une propriété modulaire pour des systèmes de réécritures non linéaires à gauche. D'autre part, pour des systèmes linéaires à gauche, on doit prendre en considération la présence ou non de règles effaçantes, c'est-à-dire de règles de la forme $l \to r$ où r est une variable. La décidabilité de l'accessibilité est conservée par union disjointe si les systèmes sont linéaires à gauche sans règles effaçantes ou linéaires avec règles effaçantes. Pour des systèmes linéaires à gauche avec règles effaçantes, cette propriété n'est pas modulaire. Nous nous sommes aussi intéressés à la décidabilité de l'accessibilité close, malheureusement cette propriété n'est pas modulaire, même pour des systèmes linéaires sans règles effaçantes. La plupart de ces résultats sont parus dans [Car92].

L'union disjointe signifie l'union de systèmes de réécriture travaillant sur des alphabets disjoints, et représente ainsi une forte restriction. Dans le dernier paragraphe, nous présentons des résultats concernant la composition de systèmes constructeurs.

Un *système constructeur* est un système de réécriture de termes tel que l'alphabet $\mathcal{F}$ peut être partitionné en deux ensembles disjoints $\mathcal{D}$ et $\mathcal{C}$ de la façon suivante :

tout terme $f(t_1, \cdots, t_n)$, membre gauche d'une règle de $\mathcal{R}$, satisfait les conditions $(f \in \mathcal{D})$ et $(t_1, \cdots, t_n \in \mathcal{T}_{\mathcal{C}}(\mathcal{X}))$.

$$\mathcal{R} = \begin{cases} 0+x & \to & x \\ S(x)+y & \to & S(x+y) \\ 0 \times x & \to & 0 \\ S(x) \times y & \to & x \times y + y \\ f(0) & \to & 0 \\ f(S(x)) & \to & f(x)+S(x) \end{cases}$$

$\mathcal{R}$ peut se décomposer en

$$\mathcal{R}_1 = \begin{cases} 0+x & \to & x \\ S(x)+y & \to & S(x+y) \\ 0 \times x & \to & 0 \\ S(x) \times y & \to & x \times y + y \end{cases} \quad \text{avec} \quad \begin{array}{l} \mathcal{C}_1 = \{0, s()\} \\ \mathcal{D}_1 = \{+(,), \times(,)\} \end{array}$$

$$\mathcal{R}_2 = \begin{cases} 0+x & \to & x \\ S(x)+y & \to & S(x+y) \\ f(0) & \to & 0 \\ f(S(x)) & \to & f(x)+S(x) \end{cases} \quad \text{avec} \quad \begin{array}{l} \mathcal{C}_2 = \{0, s()\} \\ \mathcal{D}_2 = \{+(,), f()\} \end{array}$$

Figure 0.1 : Décomposition d'un système constructeur

Y. Toyama et A. Middeldorp ont montré dans [MT91] qu'un système constructeur est complet (confluent et noethérien) s'il peut être décomposé en plusieurs systèmes constructeurs complets. Cette notion de décomposition ne nécessite pas que les alphabets soient disjoints, mais si les deux systèmes partagent un symbole défini, ils doivent contenir tous les deux toutes les règles qui "définissent" ce symbole.

Considérons l'exemple de la figure 0.1 donné dans [MT91]. Les deux systèmes $\mathcal{R}_1$ et $\mathcal{R}_2$ sont complets et entrainent la complétude de $\mathcal{R}$. D'autres auteurs ont également cherché à éviter cette contrainte des alphabets disjoints (M. Kurihara et A. Ohuchi [KO90b], B. Gramlich [Gra91]).

Les résultats que nous obtenons sont les suivants : si $\mathcal{R}_1$ et $\mathcal{R}_2$ sont des systèmes constructeurs non noethériens dont l'accessibilité est décidable, l'accessibilité de $\mathcal{R}_1 + \mathcal{R}_2$ devient indécidable. Si les deux systèmes sont noethériens alors l'accessibilité de $\mathcal{R}_1 + \mathcal{R}_2$ est décidable si les deux systèmes sont linéaires à droite, mais indécidable en général.

C Réductibilité et automates d'arbres

Dans cette partie, nous étendons les techniques développées pour décider de la réductibilité inductive d'un terme (D. Plaisted [Pla85]) à la décision de tout problème d'entourage. Nous appelons problème d'entourage toute formule du premier ordre construite à partir des formules atomiques $\text{facteur}_t(x)$ indicées par des termes de $T_\Sigma(\mathcal{X})$ (lire "t est facteur de x" ou "x entoure t"). Nous nous plaçons dans le cadre des algèbres avec sortes ordonnées. Plus précisément, à tout problème d'entourage spécifié en termes logiques, nous associons son pendant algorithmique, qui est un automate déterministe d'un genre nouveau. La génération de cet algorithme à partir de la formule permet de décider de la validité de la formule, car ceci revient à décider si le langage reconnu par l'automate associé est vide ou non.

Informellement, du point de vue des automates, la classe est nouvelle mais l'approche est des plus classiques. En gros, on peut considérer [1] nos automates comme la clôture booléenne des automates classiques et de "nouveaux" automates correspondant aux formules atomiques d'entourages. A partir de là, on obtient les propriétés de décision par un cheminement classique (M.O. Rabin [Rab69],[Rab77], W. Thomas [Tho90]).

On peut aussi considérer que notre travail permet de manipuler toute formule d'entourage comme une sorte habituelle. En effet, les sortes usuelles s'identifient aux automates finis d'arbres standards (H. Comon [Com90]) et ici, nous montrons que les problèmes d'entourage s'identifient à nos automates AFA, qui jouissent des mêmes bonnes propriétés algorithmiques que les automates classiques. Notons d'ailleurs que, dans le cas de problèmes linéaires[2], on obtient des automates standards.

Voyons dans quelle mesure nos résultats pourraient être étendus. Notons d'abord qu'une propriété d'entourage se réduit (pour sa part non reconnaissable, i.e. non exprimable par un automate classique) à la non-linéarité. Nos automates doivent donc tester des égalités entre sous-termes. De plus, dans les automates que nous introduisons, le nombre de tests le long de toute branche est borné. On peut se poser la question de remplacer les tests d'égalités par des tests de différences. Ceci revient (par complémentation) à considérer un nombre non borné de tests d'égalités le long des branches. On obtient alors les ensembles récursivement énumérables de termes (J. Mongy [Mon81]) sauf si les tests se font entre fils d'un même noeud (B. Bogaert et S. Tison [BT92]). Notons que Muchnik avait dans un autre cadre constaté de la même façon l'importance des tests d'égalités entre fils pour les propriétés de

[1] Ce n'est pas formellement le point de vue développé dans le corps de la thèse mais le lecteur familier des automates fera facilement la transcription

[2] c'est-à-dire quant les termes t en indice des formules atomiques sont linéaires

décision. Nous pourrions étendre la définition de nos automate afin d'autoriser un nombre non borné de tests d'égalités entre fils.

Remarquons enfin que toute forme purement existentielle d'un problème équationnel (H. Comon [Com88]) se traduit immédiatement en un de nos automates (voir l'exemple donné au début du chapitre 4). Nous comptons approfondir ce rapprochement.

Chapitre 1

Généralités

Le premier paragraphe présente informellement les notions concernant les systèmes de réécriture de termes et les automates d'arbres. Le deuxième paragraphe introduit les principales définitions et notations utilisées dans cette thèse. Elles proviennent pour la plupart de N. Dershowitz et J.P. Jouannaud [DJ90].

1.1 Quelques notions de bases

Nous présentons ici brièvement les principales notions concernant la réécriture et les automates d'arbres. Dans le premier paragraphe, les principales propriétés des systèmes de réécriture sont introduites à l'aide d'un exemple. Puis, nous parlerons des théories : théories équationnelles, théories du premier ordre, théories inductives. Les automates comme outils de décision sera le thème de la dernière partie.

1.1.1 Principales propriétés des systèmes de réécriture

Considérons l'exemple classique de la spécification des entiers naturels avec zéro, le successeur, l'addition et la multiplication :

$$\begin{cases} x+0 = x & (e1) \\ x+S(y) = S(x+y) & (e2) \\ x\times 0 = 0 & (e3) \\ x\times S(y) = x\times y+x & (e4) \end{cases}$$

Les entiers naturels sont représentés par les termes $0, S(0), S(S(0)), \cdots$

En utilisant les équations de la gauche vers la droite, on peut calculer 2×2 :

$$\begin{array}{rcll} S(S(0)) \times S(S(0)) & = & S(S(0)) \times S(0) + S(S(0)) & (e4) \\ & = & S(S(0)) \times 0 + S(S(0)) + S(S(0)) & e(4) \\ & = & 0 + S(S(0)) + S(S(0)) & (e3) \\ & = & S(0 + S(0)) + S(S(0)) & (e2) \\ & = & S(S(0 + S(0)) + S(0)) & (e2)) \\ & = & S(S(S(0 + S(0) + 0))) & (e2) \\ & = & S(S(S(0 + S(0)))) & (e1) \\ & = & S(S(S(S(0 + 0)))) & (e2) \\ & = & S(S(S(S(0)))) & (e1) \end{array}$$

Cette utilisation uni-directionnelle des équations signifie qu'on utilise le système de réécriture associé :

$$\left\{ \begin{array}{rcl} x + 0 & \rightarrow & x \\ x + S(y) & \rightarrow & S(x + y) \\ x \times 0 & \rightarrow & 0 \\ x \times S(y) & \rightarrow & x \times y + x \end{array} \right.$$

Il faut remarquer que le calcul que nous venons d'écrire n'est pas le seul possible pour obtenir $2 \times 2 = 4$. Mais toutes les séquences de réduction partant de $S(S(0)) \times S(S(0))$ sont finies, i.e. tout calcul s'arrête sur un terme irréductible appelé *forme normale*. Cette propriété s'appelle la *terminaison* (on dit aussi que le système est noethérien). De plus tous les calculs aboutissent au même résultat. Cette propriété est la *confluence*.

Les systèmes de réécriture confluents et noethériens sont très intéressants car, partant d'un terme donné (par exemple $S(S(0)) \times S(S(0))$), quelque soit la stratégie utilisée, on est sûr d'arriver au "bon" résultat.

1.1.2 Historique

Avec le λ-calcul, et la logique combinatoire, les systèmes de réécriture confluents et noethériens sont définis pour formaliser la notion de calculabilité. De nombreux résultats proviennent de cette branche. Par exemple, la propriété de Church-Rosser

(synonyme de confluence) provient de l'étude faite par Church et Rosser pour établir la consistance du λ-calcul et de la logique combinatoire.

Nous nous intéresserons surtout à l'étude des théories équationnelles. Le résultat le plus célèbre, qui est l'une des bases du développement actuel de la réécriture, est le théorème de Knuth et Bendix [KB70]. Le but est de vérifier si une égalité $u = v$ donnée est conséquence d'un ensemble E d'équations données (axiomes). Pour cela, on construit à partir de E un système de réécriture confluent et noethérien grâce à un algorithme de *complétion*. Ce système fournit un algorithme de décision pour la théorie équationnelle E. En effet $u = v$ est valide dans E si et seulement si u et v ont la même forme normale pour R.

Par exemple, partant de l'ensemble d'équations :

$$\begin{cases} x+0 &= x \\ (-x)+x &= 0 \\ (x+y)+z &= x+(y+z) \end{cases}$$

On obtient le système de réécriture confluent et noethérien :

$$\begin{cases} (-x)+(x+y) &\to y \\ x+0 &\to x \\ -0 &\to 0 \\ -(-x) &\to x \\ x+(-x) &\to 0 \\ x+((-x)+y) &\to y \\ -(x+y) &\to (-y)+(-x) \end{cases}$$

Il existe cependant des systèmes noethériens et non confluents qui ne peuvent être complétés en un nombre fini d'étapes.

1.1.3 Théories

Considérons la théorie équationnelle :

$$\begin{cases} x+0 &= x \\ x+S(y) &= S(x+y) \\ x\times 0 &= 0 \\ x\times S(y) &= x\times y+x \\ 0^y &= 1 \\ S(x)^y &= x^y\times x \end{cases}$$

Le système de réécriture R obtenu à partir de E en orientant les équations de la gauche vers la droite est confluent et noethérien. La théorie équationnelle de E est décidable car $u = v$ si et seulement si u et v ont la même forme normale pour R.

Cependant, on peut écrire des formules plus complexes dites *du premier ordre*. Une formule du premier ordre est une formule qui s'écrit à l'aide des connecteurs logiques $(\wedge, \vee, \neg)$, des quantificateurs $(\forall, \exists)$, d'opérateurs et de prédicats définis. Une théorie du premier ordre est dite décidable si l'on peut décider de la validité de toute formule du premier ordre.

Il peut aussi être intéressant de se restreindre aux termes clos. On se situe alors dans l'algèbre initiale, et on peut parler de la *théorie inductive* d'un ensemble d'axiomes E.

Sur l'exemple précédent, l'égalité $(x + y) + z = x + (y + z)$ n'est pas un théorème équationnel mais est un théorème inductif (on peut le prouver par une induction sur la structure des termes clos).

Pour décider de la validité d'un théorème dans l'algèbre initiale, on utilise le principe de la preuve par induction sans induction (ou preuve par consistance). Il consiste à ajouter le théorème à démontrer $u = v$ à l'ensemble E et à vérifier que les relations de congruences sur les termes clos définies par E d'une part et $E \cup \{u = v\}$ d'autre part sont les mêmes. On peut prouver ce résultat en démontrant la confluence close de E et que les termes u et v peuvent se réécrire en un même terme, pour toute substitution close .

1.1.4 Automates comme outils de décision

La théorie des automates et la théorie des langages apparurent dans les années 50. Les motivations étaient diverses: logique, circuits électroniques, compilation, linguistique,... La théorie des automates et des langages reconnaissables devint vite

un domaine de recherche fondamentale. Si au départ les structures étudiées étaient essentiellement les mots, les notions d'automates et de reconnaissabilité furent vite étendues au mots infinis, aux arbres finis ou non, n-uplets, graphes, ...

Les automates finis d'arbres sont, exprimés en un autre vocabulaire, des définitions de signatures d'algèbres sortées (H. Comon [Com90]). A chaque sorte de la signature correspond un état de l'automate. Les inclusions de sortes se traduisent par des ϵ-transitions. Cette remarque est illustrée par la signature et l'automate donnés figure 1.1, permettant de définir les listes d'entiers.

L'idée de considérer les automates comme outil de décision n'est pas neuve puisque, dans les années 60, les automates sur les mots et arbres infinis avaient été introduits comme outil de décision pour des problèmes de logique (Büchi, Doner, Mac Naughton, Rabin, ...). En particulier, Rabin développa la théorie des automates d'arbres infinis pour résoudre le problème de la décidabilité de la théorie monadique du second ordre et en déduisit la décidabilité d'un grand nombre d'autres théories ([Rab69],[Rab77]). W. Thomas fournit dans [Tho90] une étude détaillée des propriétés des automates de mots et d'arbres, finis et infinis, et de leurs connections avec la logique. Plus récemment, M. Dauchet et S. Tison ont démontré grâce aux automates d'arbres finis que la théorie du premier ordre de la réécriture close était décidable [DT90], i.e. toute propriété des systèmes de réécriture clos exprimée par une formule du premier ordre est décidable.

L'idée des preuves "à la Rabin" est d'associer à une formule atomique un langage (de mots, d'arbres,...) reconnaissable, et ainsi de coder l'ensemble des éléments du domaine satisfaisant la formule par un reconnaissable. Chaque connecteur logique correspond alors à un opérateur Booléen ou à une projection et donc si la classe des reconnaissables concernée est fermée par les opérations Booléennes (union, intersection, complémentaire) les projections et projections inverses, chaque formule sera codée par un reconnaissable. Donc, finalement, si le vide est décidable, on peut décider de la validité de toute phrase.

1.2 Définitions et notations

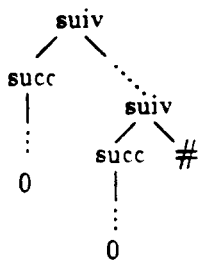
1.2.1 Termes, sous-termes

On considère un *alphabet fini gradué* Σ, c'est-à-dire un alphabet dont toutes les lettres possèdent une arité unique dans $\mathbb{N}$. Les éléments d'arité 0 sont appelés

On définit les sortes Nat (entier naturel), ListeEntiers (liste d'entiers) et ListeNonVide (liste non vide d'entiers) par la signature :

$$\begin{aligned} \mathbf{0}: &\ \text{Nat} \\ \mathbf{succ}: &\ \text{Nat} \rightarrow \text{Nat} \\ \#: &\ \rightarrow \text{ListeEntiers} \\ \mathbf{suiv}: &\ \text{Nat} \times \text{ListeEntiers} \rightarrow \text{ListeNonVide} \\ &\ \text{ListeNonVide sous-sorte de ListeEntiers} \end{aligned}$$

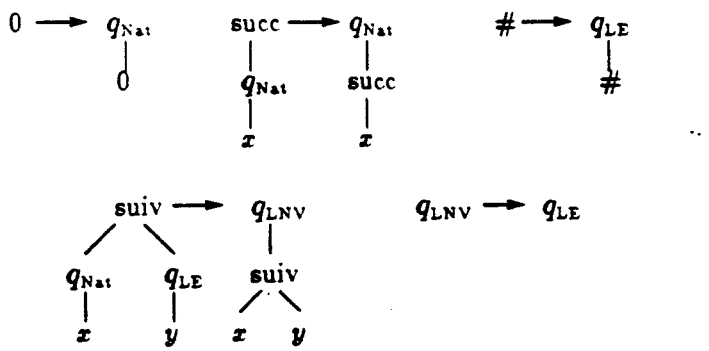
Un terme t de la sorte ListeEntiers a pour forme :



Soit l'automate ascendant $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{Q}_f, \mathcal{R})$ avec

$$\begin{aligned} \Sigma &= \{0, succ(), \#, suiv(,)\} \\ \mathcal{Q} &= \{q_{\text{Nat}}, q_{\text{LE}}, q_{\text{LNV}}\} \\ \mathcal{Q}_f &= \mathcal{Q} \end{aligned}$$

$\mathcal{R}$ est l'ensemble des transitions :



La sorte ListeNonVide est le langage reconnu par $\mathcal{A}$ quand on prend q_{LNV} comme état final.

Figure 1.1 : Correspondance entre sortes et automates

constantes. On suppose que Σ contient toujours au moins une constante. On notera Σ_n le sous-ensemble de Σ qui contient toutes les lettres d'arité n.

Soit $\mathcal{X}$ un ensemble dénombrable de variables.

L'ensemble $T_\Sigma(\mathcal{X})$ des *termes* sur $\Sigma \cup \mathcal{X}$ est défini comme étant le plus petit ensemble pour lequel

- $\mathcal{X} \subseteq T_\Sigma(\mathcal{X})$
- $b(t_1, \cdots, t_n) \in T_\Sigma(\mathcal{X})$ pour tout $b \in \Sigma_n$ avec $n \geq 0$ et $t_1, \cdots, t_n \in T_\Sigma(\mathcal{X})$.

Un terme est vu comme un arbre. Sa *racine* est le symbole qui apparaît à son sommet, c'est-à-dire, le symbole le plus à gauche en notation fonctionnelle. Par exemple, le terme f(g(a),h(x,g(a),y)) a pour racine f et peut être représenté par l'arbre dessiné figure 1.2

L'ensemble $T_\Sigma(\emptyset)$, noté T_Σ, est l'ensemble des *termes clos* sur Σ.

L'ensemble des variables d'un terme t est noté $\mathcal{V}(t)$. Un terme est dit *linéaire* si aucune variable n'apparaît deux fois dans ce terme. Par exemple, le terme de la figure 1.2 est linéaire.

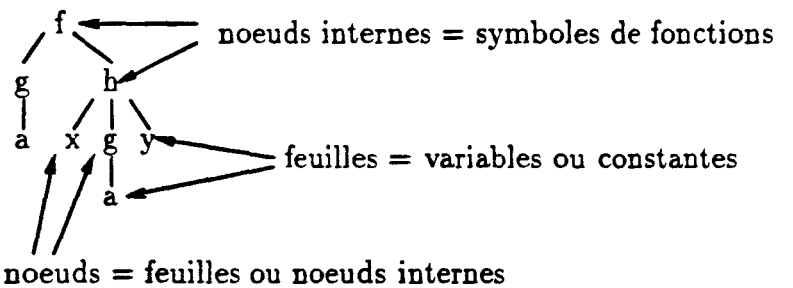


Figure 1.2: arbre de $T_\Sigma(\mathcal{X})$

La *position* d'un *sous-terme* dans un terme est une liste d'entiers qui indique le chemin de la racine jusqu'au sous-terme. Pour représenter la position de la racine, on utilise le symbole ϵ. Le sous-terme à la position l dans le terme t est noté $t|_l$ et est défini récursivement par:

$$\begin{cases} \text{Si } l = \epsilon \text{ alors } t|_l = t \\ \text{Si } l = i.l' \text{ alors } t|_l = t_i|_{l'} \text{ où } t_i \text{ est le ième fils de } t \end{cases}$$

Par exemple, si l'on considère le terme t de la figure 1.2, le sous-terme de t à la position 2.2 est g(a).

Par extension, on parlera de la position d'un noeud a, au lieu de la position du sous-terme de t de racine a.

La *longueur* d'une position p est 0 si p est ϵ, et vaut le nombre d'entiers composant p sinon. Par exemple, la longueur de 1.10.5, notée $|1.10.5|$ vaut 3.

Un sous-terme de t est dit *propre* s'il est différent de t.

Soit p une position dans l'arbre t. *Chemin(p)* est l'ensemble des positions préfixes de p Par exemple, Chemin(1.3.2.1) = $\{\epsilon, 1, 1.3, 1.3.2, 1.3.2.1\}$

La *hauteur* d'un arbre est la plus grande longueur de ses positions et sa *taille* est le nombre de noeuds qu'il comporte. Par exemple, la longueur de l'arbre t (figure 1.2) est 3 et sa taille est 8. On écrira par la suite $| t |= 3$ et $\| t \|= 8$.

1.2.2 Substitution, filtrage, unification

Une *substitution* σ est une application de $\mathcal{X}$ dans $T_\Sigma(\mathcal{X})$ distincte de l'identité pour un nombre fini de variables. σ est déterminée de manière unique par l'image de ces variables. On l'écrit donc sous la forme $\{x_1 \mapsto t_1, \cdots, x_n \mapsto t_n\}$ avec $(\forall i \in [1,n]\ t_i \neq x_i)$.

On étend σ à $T_\Sigma(\mathcal{X})$ en posant : $\forall f \in \Sigma_n$, $\forall t_1 \cdots, t_n \in T_\Sigma(\mathcal{X})$, $\sigma(f(t_1, \cdots, t_n)) = f(\sigma(t_1), \cdots, \sigma(t_n))$

Un exemple est donné figure 1.3

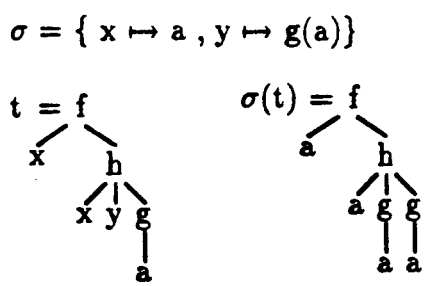


Figure 1.3 : comment appliquer une substitution

Le *domaine* d'une substitution σ est l'ensemble des variables qui ne sont pas envoyées sur elles-mêmes.

Une substitution $\{x_1 \mapsto t_1, \cdots, x_n \mapsto t_n\}$ telle que tous les termes t_i sont clos est appelée *substitution close*. La substitution décrite figure 1.3 en est une.

La composition des substitutions α et β se note $\alpha.\beta$ ou plus simplement $\alpha\beta$ et est définie par $\alpha.\beta(x) = \alpha\beta(x) = \alpha(\beta(x))$.

Un terme t est une *instance* d'un terme s si $t = \sigma(s)$ pour une substitution σ. Dans ce cas on écrit $s \geq t$ et on dit que s *filtre* t. La relation $\geq$ est un préordre sur les termes appelé *filtrage*.

Soient σ une substitution, et t un terme de $T_\Sigma(\mathcal{X})$. Si $\sigma(t)$ est un terme de T_Σ alors σ est appelée *instanciation close* de t.

Soient t et u deux termes. On dira que t est *facteur* de u ou que u *entoure* t s'il existe un sous-terme s de u tel que s est une instance de t. Par exemple, dans la figure 1.4, t' est facteur de t.

t et t' termes de $T_\Sigma(\mathcal{X})$ sont *unifiables* s'il existe une substitution σ telle que $\sigma(t) = \sigma(t')$. Pour tous t et t' unifiables, il existe une substitution σ la plus générale (Plotkin [Plo72]). Cette substitution est couramment appelée *mgu(t,t')*. L'*unifié* de t et t', noté *unifié*(t, t'), est obtenu par application sur t (ou t') de cette substitution σ. Un exemple est donné figure 1.5.

1.2.3 Décidabilité

Un problème P dépendant des données $d_1, \cdots, d_n$ est *décidable* s'il existe un programme capable de calculer si, pour toutes les valeurs de $d_1, \cdots, d_n$, $P(d_1, \cdots, d_n)$ est vraie ou fausse.

Par la suite, pour démontrer qu'un problème P est décidable, on exhibe le programme permettant de calculer la valeur de vérité de P en fonction des données dont elle dépend. Par contre, on montre que P est indécidable en se ramenant à l'existence d'une solution au problème de correspondance de Post.

Problème de correspondance de Post *[Pos46] Le problème de correspondance de Post $P(\varphi, \psi)$ sur un alphabet X est donné par deux morphismes φ et ψ de I^* dans X^*. $P(\varphi, \psi)$ admet une solution si et seulement s'il existe m_I dans I^+ tel que $\varphi(m_I) = \psi(m_I)$.*

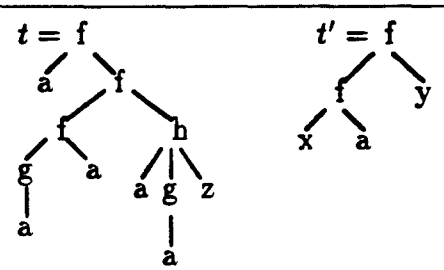
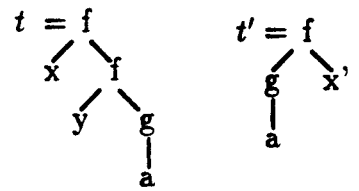


Figure 1.4 : t' est facteur de t



$\text{mgu}(t,t') = \{x \mapsto g(a),\ x' \mapsto f(y,g(a))\}$

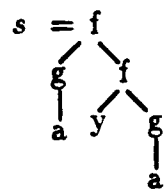


Figure 1.5 : s = unifié(t,t')

E.L. Post a démontré que l'existence d'une solution à ce problème était indécidable [Pos46].

1.2.4 Systèmes de réécriture

Définitions

Une *règle de réécriture* est un couple (l, r) de termes qui se note $l \rightarrow r$, tel que $\mathcal{V}(r) \subset \mathcal{V}(l)$. l est appelé *membre gauche* de la règle et r, *membre droit.*

Un ensemble $\mathcal{R}$ de règles de réécriture est appelé *système de réécriture.* Par la suite, nous considérons toujours des ensembles finis de règles. Nous abrègerons parfois "système de réécriture" par "SDR".

Soit t un terme et $\mathcal{R}$ un système de réécriture. S'il existe une position p dans t et une règle $l \rightarrow r$ de $\mathcal{R}$ telles que $l \geq t|_p$ alors $t|_p$ est appelé *rédex* . Ça signifie qu'il existe une substitution σ telle que $t|_p = \sigma(l)$. On *réécrit* le terme t par application de la règle $l \rightarrow r$ sur le rédex $t|_p$ en remplaçant $t|_p$ par $\sigma(r)$ dans t. La figure 1.6 schématise ce pas de réécriture. On écrit $t \rightarrow_{\mathcal{R}} t'$ si t se réécrit en t' par une règle de $\mathcal{R}$. Quand il n'y a pas d'ambiguité pour $\mathcal{R}$, on écrit souvent $\rightarrow$ au lieu de $\rightarrow_{\mathcal{R}}$.

La relation $\rightarrow_{\mathcal{R}}$ est appelée *relation de réécriture*, ce qui signifie qu'elle est close :

- par application de contexte : l'application d'une règle à un sous-terme ne dépend pas de son contexte. Si $t \rightarrow_{\mathcal{R}} t'$ alors $C[t] \rightarrow_{\mathcal{R}} C[t']$

- par substitution : si $t \rightarrow_{\mathcal{R}} t'$ et σ substitution, alors $\sigma(t) \rightarrow_{\mathcal{R}} \sigma(t')$.

Pour toute relation de réécriture, sa clôture symétrique, sa clôture réflexive et sa clôture transitive sont aussi des relations de réécriture.

Notations utilisées :

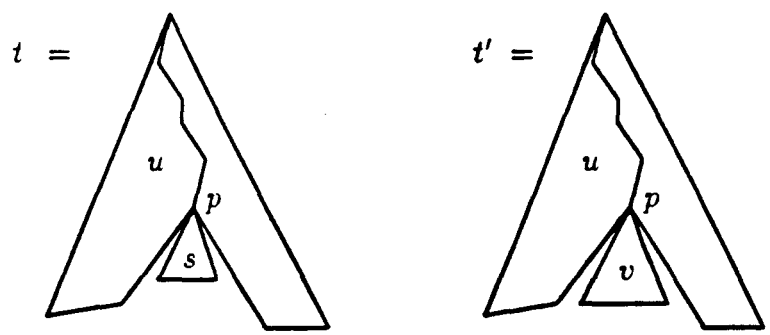
$\rightarrow$	relation de réécriture
$\xrightarrow{*}$	clôture réflexive-transitive
$\xrightarrow{+}$	clôture transitive

Une *dérivation* dans $\mathcal{R}$ est une séquence $t_0 \rightarrow_{\mathcal{R}} t_1 \rightarrow_{\mathcal{R}} \cdots$

$\xrightarrow{*}$ est appelée *dérivabilité.*

$l \to r$ règle de $\mathcal{R}$

$l \geq s$ donc il existe une substitution σ telle que $s = \sigma(l)$



$s = t|_p$ $\qquad$ $v = \sigma(r)$

u contexte de s

Figure 1.6 : $t \to_{\mathcal{R}} t'$

Un système de réécriture *clos* est un système où les membres des règles sont clos, c'est-à-dire sans variables. Ces systèmes ont été largement étudiés dans [DT85], [DHLT87], [DG89], [Koz77], [Oya86], [Oya87], [DT90]...

Un système de réécriture est *linéaire* si les membres des règles sont linéaires.

Un système est *linéaire à gauche* (resp. *linéaire à droite*) si les membres gauches (resp. droits) des règles sont tous linéaires.

Un *système de réécriture de mots* ou système *semi-Thue* est un système dont les membres gauches et droits des règles sont des éléments de $T_{\Sigma_1}(\{x\})$.

Un système de réécriture de mots *de longueur non croissante* est un système où les règles $l \to r$ sont telles que $|l| \geq |r|$

Un système de réécriture de mots *préservant les longueurs* est un système où les règles $l \to r$ sont telles que $|l| = |r|$

Propriétés

Un système est *noethérien* si et seulement s'il n'existe pas de chaine de dérivation infinie $t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \cdots$. On dit aussi que le système *termine*. Cette propriété est indécidable en général, même dans le cas d'une seule règle linéaire à gauche (M. Dauchet [Dau89]), mais est décidable pour un système clos (Huet et Lankford [HL78]). Nachum Dershowitz dans [Der87] décrit des méthodes permettant de prouver qu'un système termine.

Un système est *confluent* si et seulement si

$$\forall x \forall y \forall z \in T_\Sigma(\mathcal{X}) \ \ (z \xrightarrow{*} x \wedge z \xrightarrow{*} y) \Rightarrow \exists t \in T_\Sigma(\mathcal{X}) \ \ (x \xrightarrow{*} t \wedge y \xrightarrow{*} t)$$

Cette propriété est indécidable en général. La confluence est décidable pour des systèmes de réécriture clos. Ce résultat a été démontré par M. Dauchet, T. Heuillard, P. Lescanne et S. Tison d'une part ([DT85], [DHLT87]) et M. Oyamaguchi d'autre part ([Oya87]). Des critères de confluence sont donnés par Huet dans [Hue80] notamment pour des systèmes linéaires à gauche.

Un système est *localement confluent* si et seulement si

$$\forall x \forall y \forall z \in T_\Sigma(\mathcal{X}) \ \ (z \to x \wedge z \to y) \Rightarrow \exists t \in T_\Sigma(\mathcal{X}) \ \ (x \xrightarrow{*} t \wedge y \xrightarrow{*} t)$$

Un système peut être localement confluent sans être pour autant confluent. C'est le cas du système suivant :

$$\begin{cases} a \to b \\ b \to a \\ b \to c \\ a \to d \end{cases}$$

La confluence locale est décidable en générale. Dans le cas de systèmes noethériens, la confluence se ramène à la confluence locale, elle est donc décidable (Théorème de Newman [New42]).

La *confluence close* est la confluence restreinte aux termes clos. Bien sûr, tout système confluent est également clos-confluent, mais la réciproque est fausse. Par exemple, le système suivant sur l'alphabet $\{a, b(), c()\}$ est clos-confluent puisque tout terme clos se réécrit en a mais n'est pas confluent.

$$\begin{cases} c(a) \to a \\ b(x) \to c(x) \\ b(x) \to x \end{cases}$$

Etant donnés un système de réécriture $\mathcal{R}$, et deux termes t_1 et t_2, le *problème d'accessibilité* pour $\mathcal{R}$, t_1 et t_2 est de décider si $t_1 \xrightarrow{*}_{\mathcal{R}} t_2$.

Ce problème est indécidable en général, mais décidable pour les systèmes quasi-clos (i.e. clos à droite) [Oya86] ou pour des systèmes monadiques.

Pour des systèmes de réécriture de mots, ce problème est plus couramment appelé *problème du mot*. Il est indécidable en général.

1.2.5 Automates d'arbres et langages reconnaissables

Automates d'arbres

Un *automate ascendant d'arbres* (a.a.a.) est un quadruplet $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{Q}_f, \mathcal{R})$ où

- Σ alphabet fini gradué
- $\mathcal{Q}$ ensemble fini d'états
- $\mathcal{Q}_f$ ensemble des états finaux. $\mathcal{Q}_f \subset \mathcal{Q}$
- $\mathcal{R}$ ensemble des règles de transition

$\mathcal{R}$ est un système de réécriture où les règles sont de deux formes décrites figure 1.7.

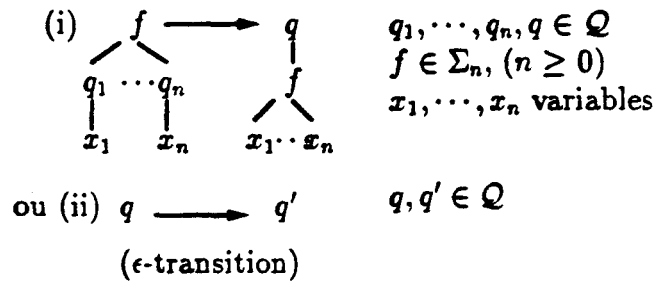


Figure 1.7 : transitions d'un automate ascendant

Dans le cas d'un automate, la relation de réécriture $\rightarrow_{\mathcal{R}}$ sur $T_{\Sigma \cup Q}$ se note $\vdash_{\mathcal{A}}$. La lecture d'un arbre par l'automate se fait des feuilles vers la racine. Le terme d'automate ascendant provient donc de notre façon de dessiner les arbres, avec la racine en haut et les feuilles en bas. Le terme anglais "frontier to root" convient mieux car il est indépendant de cette représentation.

Un a.a.a. est *déterministe* s'il n'existe pas deux règles de même membre gauche et s'il ne contient pas d'ϵ-transition.

Un a.a.a. est *complétement spécifié* si pour toute lettre f de Σ_n et pour tout n-uplet d'états $(q_1, \cdots, q_n)$, il existe une transition de membre gauche $f(q_1(x_1), \cdots, q_n(x_n))$.

Le langage d'arbres reconnu par l'automate $\mathcal{A}$ est $L(\mathcal{A}) = \{t \in T_\Sigma \mid t \stackrel{*}{\vdash}_{\mathcal{A}} q(t),\ q \in Q_f\}$. Par exemple, si l'automate décrit figure 1.8 a pour état final q_b, le langage qu'il reconnait est $L(\mathcal{A}) = \{b(a^n(\overline{a}), a^m(\overline{a})),\ n, m > 0\}$.

Il existe une autre définition des règles de transitions (bien entendu équivalente), où $\mathcal{R}$ est un système de réécriture clos. Il suffit de remplacer dans la figure 1.7 la forme (i) par $f(q_1, \cdots, q_n) \rightarrow q$. Dans ce cas, $L(\mathcal{A})$ devient $\{t \in T_\Sigma \mid t \stackrel{*}{\vdash}_{\mathcal{A}} q,\ q \in Q_f\}$.

Théorème 1.2.1 *Soit $\mathcal{A}$ un automate ascendant d'arbres (a.a.a.). Il existe B a.a.a. sans ϵ-transition tel que $L(B) = L(\mathcal{A})$ et il existe un a.a.a. déterministe C tel que $L(C) = L(\mathcal{A})$.*

Il existe aussi des automates descendants où la reconnaissance de l'arbre s'effectue de la racine vers les feuilles. Je ne détaillerai pas ces automates car ils n'interviennent pas dans le cadre de cette thèse.

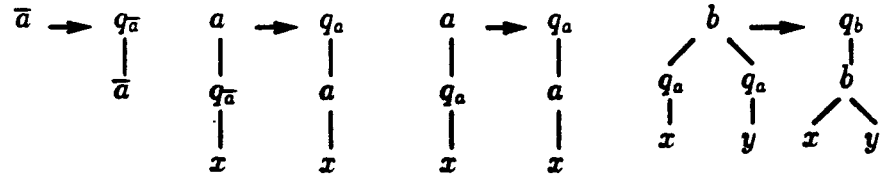


Figure 1.8 : automate ascendant

Forêts reconnaissables d'arbres

Toutes les notions et résultats concernant les automates d'arbres et les forêts reconnaissables sont présentés plus formellement par F. Gecseg et M. Steinby dans [GS84].

Un langage d'arbres $\mathcal{F}$ est *reconnaissable* s'il existe un a.a.a. $\mathcal{A}$ tel que $L(\mathcal{A}) = \mathcal{F}$.

Une *forêt* est un ensemble d'arbres, c'est donc un langage d'arbres.

La classe des forêts reconnues par les a.a.a. est notée REC. En fait les forêts reconnues par les a.a.a. sont aussi celles reconnues par les automates descendants. On les appelle *forêts reconnaissables* .

Théorème 1.2.2 *REC est close par union, intersection, complémentarité.*

Théorème 1.2.3 *Si $\mathcal{F}$ et $\mathcal{F}'$ sont deux forêts reconnaissables, on peut décider si $\mathcal{F} = \mathcal{F}'$, $\mathcal{F} = \emptyset$, $\mathcal{F} \subset \mathcal{F}'$.*

Les automates finis d'arbres sont, exprimés en un autre vocabulaire, des définitions de sortes, c'est-à-dire des signatures d'algèbres sortées (H. Comon [Com90]). A chaque sorte de la signature correspond un état de l'automate. Les inclusions de sortes se traduisent par des ϵ-transitions. Cette remarque est illustrée par la signature et l'automate donnés figure 1.1, permettant de définir les listes d'entiers.

Automates à tests d'égalités

Les automates à tests sont des automates ascendants dont les règles contiennent des conditions utilisant la comparaison de sous-arbres du noeud considéré. Cette

comparaison peut prendre différentes formes, suivant l'endroit où se situent les sous-arbres comparés.

La comparaison peut se faire entre deux sous-termes quelconques, à des profondeurs bornées. J. Mongy définit de tels automates appelés RATEG, dans [Mon81]. Malheureusement, ils n'ont pas toutes les "bonnes" propriétés souhaitées pour des automates : on ne peut pas décider si la forêt reconnue par un automate de RATEG est vide. Si chaque règle ne compare que des sous-termes à profondeur bornée, on arrive à propager les contraintes d'égalités et à relier des sous-arbres arbitrairement éloignés, par des comparaisons de proche en proche. Ce phénomène de "chevauchement d'égalités" permet de montrer que le vide est indécidable. Les automates que nous définissons dans cette thèse permettent aussi des comparaisons entre sous-arbres à profondeur bornée, mais pour obtenir la décidabilité du vide, nous n'autorisons qu'un nombre borné de tests d'égalités le long de tout chemin.

Une autre voie consiste à comparer seulement les fils de la lettre courante. De cette manière chaque arbre n'est comparé qu'à des arbres du même niveau que lui, à profondeur 1, et le chevauchement non borné est évité. C'est le cas du travail de B. Bogaert et S. Tison ([Bog90], [BT92]) qui définissent une classe d'automates dont les règles permettent de tester des égalités et des différence entre fils d'un même noeud. Cette classe est close par union, intersection, complémentaire et le vide est décidable.

M. Tommasi a démontré que, dans le cas des automates autorisant des tests à profondeur 1 (entre fils) et 2 (entre cousins germains), le vide est indécidable. Mais ces automates peuvent tester un nombre non borné d'égalités, ce qui n'est pas le cas de ceux que nous définissons au chapitre 4.

Chapitre 2

Sortes et problèmes de décision

Notre premier but est de démontrer que la terminaison des systèmes de réécriture de mots de longueur non croissante était indécidable. En effet, ce problème est présenté comme ouvert dans [DJ90]. Pour cela, nous utilisons des machines de Turing particulières appelées *automates linéairement bornés*, dont la taille du ruban dépend linéairement de la taille de la donnée (la simulation d'un automate linéairement borné par un système de réécriture avait déjà été utilisée par F. Otto [Ott86]). Mais il ne faut pas oublier qu'une machine de Turing démarre ses calculs par une configuration initiale alors qu'un système de réécriture s'applique à des mots quelconques. En effet, pour simuler un ALB par un système de réécriture de mots préservant les longueurs, il faut commencer les séquences de dérivation sur des mots représentant des configurations initiales de l'automate. Cette réflexion nous a amené à étudier plus généralement le lien entre la décidabilité d'une propriété notée $\mathcal{P}(\mathcal{R}, \mathcal{S})$ sur une classe $\mathcal{R}$ de systèmes de réécriture et sur un ensemble de termes de sorte $\mathcal{S}$ et de la même propriété notée $\mathcal{P}(\mathcal{R})$ où les termes n'ont pas de contrainte de sorte (rappelons qu'une sorte est un langage d'arbres reconnaissable).

Dans ce chapitre, nous démontrons tout d'abord que la terminaison des systèmes de réécriture de mots préservant les longueurs est indécidable, que l'on se restreigne ou non à des configurations initiales de machines de Turing. On a donc, exprimé dans notre formalisme,

Terminaison($\mathcal{R}$ préservant les longueurs, $\mathcal{S}$ configurations initiales d'un ALB) est indécidable,

et Terminaison($\mathcal{R}$ préservant les longueurs) est aussi indécidable.

Cela signifie que l'on a pu déduire de l'indécidabilité de la terminaison des automates linéairement bornés celle des systèmes de réécriture de mots préservant les longueurs.

Notre second résultat montre qu'une propriété décidable dans le cas général, peut devenir indécidable quand on se restreint à des termes d'une sorte donnée. Le théorème de Newman permet de montrer que la confluence est décidable pour des systèmes de réécriture noethériens. Pour les systèmes de réécriture de mots, cette propriété se traduit par :

Confluence(SDR de mots noethériens) est décidable.

Nous prouvons dans le dernier paragraphe que :

Confluence(SDR de mots noethériens, termes commençant par une lettre donnée) est indécidable.

C'est ainsi que, même dans le cas des mots, le théorème de Newman ne s'applique plus quand on se restreint à des mots commençant tous par la même lettre, bien que ce soit une contrainte reconnaissable très simple. Remarquons que la confluence close est indécidable en général mais notre résultat est différent car dans le cas des mots, la confluence close et la confluence représentent la même propriété. F. Otto démontre dans [Ott87] que la confluence sur des classes de congruence est indécidable, mais les classes de congruence ne sont, en général, pas reconnaissables.

Dans certains cas, on peut directement déduire $\mathcal{P}(\mathcal{R}, S)$ de $\mathcal{P}(\mathcal{R})$. Considérons par exemple la théorie de la réécriture close. Cette théorie est définie par un langage du premier ordre où les constantes sont des termes clos et les prédicats sont associés à des relations de réécriture définies par des systèmes clos. On obtient ainsi des formules qui sont habituellement utilisées pour définir les propriétés des systèmes de réécriture (confluence, confluence locale, ...). Par exemple, si R est un système de réécriture clos, la confluence de R est définie dans ce langage par la formule :

$$\forall t \forall u \forall v (R^*(t,u) \wedge R^*(t,v)) \Rightarrow \exists w (R^*(u,w) \wedge R^*(v,w))$$

$R^*(t_1, t_2)$ signifie que t_1 se réécrit en t_2 en utilisant les règles de R. R est confluent si et seulement si la formule ci-dessus est vraie. M. Dauchet et S. Tison ont démontré qu'il existait un algorithme pour décider de la validité de telles formules [DT90], ce qui revient à dire que la théorie de la réécriture close est décidable. Pour reprendre les notations précédentes, V(systèmes clos) est décidable, où V est la propriété de validité des formules. V(systèmes clos, sorte S) est aussi décidable. Ici la décidabilité peut se déduire du cas général. En effet, l'automate qui spécifie la sorte S et l'expression $t \in S$ se traduisent facilement dans le langage de la théorie : Soit A l'automate qui reconnaît la sorte S et soit q_S l'état final de A ; L'expression $t \in S$ est traduite

par la formule $A^*(t, q_S)$, où A est considéré comme un système de réécriture clos.

2.1 Automates linéairement bornés

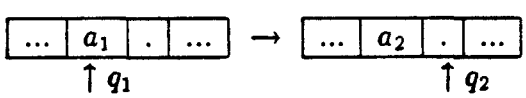
Les automates linéairement bornés ont été créés par J. Myhill [Myh60] en 1960.

Un *automate linéairement borné* (ALB) est une machine de Turing déterministe particulière, où la taille du ruban est linéairement dépendante de la taille de la donnée. En fait, nous considérons que la taille utilisée pour les calculs est la même que celle de la donnée. Plus formellement, un ALB est un sextuplet $(\Sigma, \Gamma, Q, q_0, Q_f, \Delta)$ avec

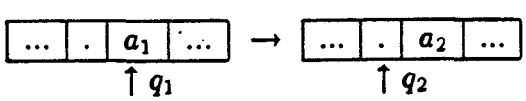
- Σ un alphabet fini d'entrée
- Γ un alphabet fini de travail
- Q un ensemble fini d'états
- q_0 un état initial ($q_0 \in Q$)
- Q_f un ensemble fini d'états finaux ($Q_f \subseteq Q$)
- Δ une fonction de transition, de $Q \times (\Gamma \cup \Sigma)$ dans $(\Gamma \cup \Sigma) \times Q \times \{D, G\}$

A chaque transition, l'automate lit une lettre a_1, écrit une lettre a_2 à la place de a_1 et se déplace ;

déplacement vers la droite : $(q_1, a_1) \rightarrow (a_2, q_2, D)$



déplacement vers la gauche : $(q_1, a_1) \rightarrow (a_2, q_2, G)$



On supposera par la suite que, initialement, le ruban a la forme $\# < d > \#$, où $\#$, $<$, $>$ sont des symboles qui ne sont jamais modifiés et d est la donnée.

On utilise le vocabulaire lié aux machines de Turing : description instantanée, configuration initiale, pas de calcul, calcul.

- Une *description instantanée* (DI) est une écriture $\# < m_1 q a m_2 > \#$ qui signifie que l'automate est dans l'état q, lit la lettre a, le mot m_1 est à gauche et m_2 à droite.

- Une *configuration initiale* est une description instantanée $\# < qm > \#$ où q est un état initial.

- Un *pas de calcul* $DI_1 \rightarrow DI_2$ signifie que l'on passe de DI_1 à DI_2 par une transition de Δ.

- Un *calcul* est une succession de pas de calcul.

2.2 Terminaison des systèmes préservant les longueurs

Pour démontrer que la terminaison des systèmes de réécriture de mots préservant les longueurs est indécidable, nous utilisons les automates linéairement bornés. En effet, Hooper a démontré que leur terminaison était indécidable [Hoo66]. Notons que J.E. Hopcroft et J.D. Ullman ont démontré dans [HU67] que tout ALB peut être simulé par un ALB noethérien équivalent. Nous redémontrons le résultat de Hooper pour une classe de machines particulières qui restaurent leur configuration initiale quand elles bouclent. En effet, cette propriété est importante pour la suite.

L'idée de notre preuve est la suivante : nous construisons d'abord une classe d'automates linéairement bornés, associée au problème de correspondance de Post : à tous morphismes φ et ψ nous associons un ALB qui termine si et seulement si le problème de correspondance de Post pour φ et ψ, noté $P(\varphi, \psi)$ n'a pas de solution. Nous appelons cette classe d'ALB A_{Post}. On construit ensuite une classe de systèmes de réécriture préservant les longueurs, notée R_{Post}, simulant les automates de la classe A_{Post}. Les propriétés de ces deux classes nous permettent de déduire l'indécidabilité de la terminaison des systèmes de R_{Post}.

2.2.1 Construction des ALB associés au problème de Post

Définition 2.2.1 *A_{Post} est un ensemble d'ALB déterministes associé au problème de correspondance de Post.*

$A_{\text{Post}} = \{A_{\varphi\psi} = (\Sigma, \Gamma, Q_{\varphi\psi}, \{q_0\}, \emptyset, \Delta) \mid \varphi$ *et* ψ *deux morphismes de* I^* *vers* $X^*\}$, *avec*

- $\Sigma = I \cup X \cup \{<,>\}$, I *et* X *sont deux alphabets disjoints, qui ne contiennent aucun des deux symboles* $<$ *et* $>$.

- $\Gamma = \Sigma \cup \overline{I} \cup \overline{X}$, *où* $\overline{I}$ *et* $\overline{X}$ *sont deux nouveaux alphabets construits à partir de* I *et* X : $\forall x \in X, \overline{x} \in \overline{X}$ *et* $\forall i \in I, \overline{i} \in \overline{I}$

- $Q_{\varphi\psi}$ *est l'ensemble des états,* q_0 *est l'état initial et il n'y a pas d'état final.*

$Q_{\varphi\psi} = \{q_0, q_1, q_2, q_{\text{ret}}, q_{\text{reinit}}, q_{\text{recom}}, q'_{\text{ret}}, q'_{\text{reinit}}, q_{\text{fin}}\}$
$\cup\{q_{\varphi(i)_j} \mid i \in I$ *et* $\varphi(i) = \varphi(i)_1 ... \varphi(i)_{k_i}$, $j \in [1, k_i]$ *si* $\varphi(i) \neq \epsilon\} \cup \{q_{\varphi(i)_0}$ *si* $\varphi(i) = \epsilon\}$
$\cup\{q_{\psi(i)_j} \mid i \in I$ *et* $\psi(i) = \psi(i)_1 ... \psi(i)_{r_i}$, $j \in [1, r_i]$ *si* $\psi(i) \neq \epsilon\} \cup \{q_{\psi(i)_0}$ *si* $\psi(i) = \epsilon\}$.

- Δ *est la fonction de transition de l'automate.*

Le programme de la figure 2.1 décrit sous une autre forme (plus lisible, je l'espère) que les règles de transition le comportement de l'automate $A_{\varphi\psi}$. Ce programme a pour donnée un mot m de $(I \cup X)^*$ et boucle si et seulement si $m = i_n \cdots i_2 i_1 x_1 x_2 \cdots x_p$ avec $\varphi(i_1 \cdots i_n) = x_1 \cdots x_p = \psi(i_1 \cdots i_n)$ (cette propriété sera démontrée par la suite).

Au départ, on suppose que l'automate lit la lettre juste à droite de $<$. On utilise une variable lettre qui représente la lettre courante lue par l'automate. Le prédicat I(a) (resp. X(a), Isurligné(a), Xsurligné(a)) est vrai si et seulement si a est une lettre de I (resp. de X, de $\overline{I}$, de $\overline{X}$). La procedure surligner(a) remplace sur le ruban la lettre a par $\overline{a}$. La procédure réécrire($\overline{a}$) fait le contraire. Les alphabets $\overline{I}$ et $\overline{X}$ permettent d'indiquer les lettres qui sont déjà traitées. La procedure rechercher$\varphi(a)$ recherche le mot $\varphi(a)$ sur le ruban et met le booléen stop à Vrai s'il ne le trouve pas.

Définition 2.2.2 *Soit* $A_{\varphi\psi}$ *un automate de* A_{Post}. *Une configuration initiale de* $A_{\varphi\psi}$ *est* convenable *si et seulement si elle est de la forme* $\# < q_0 m_I m_X > \#$, *avec* m_X *mot de* X^*, m_I *mot de* I^+.

Lemme 2.2.3 *Soit* $A_{\varphi\psi}$ *un automate de* A_{Post}. *Si la configuration initiale de* $A_{\varphi\psi}$ *n'est pas convenable, alors la machine s'arrête.*

PREUVE : Par définition, la machine initialement est dans l'état q_0 et lit la première lettre à droite du symbole $<$.

```
Programme A_{φψ} ;
   stop := faux ; état := q0 ;
   TantQue non stop faire
      * recherche de la lettre de I la plus à droite *
05    Si X(lettre) alors stop := Vrai finSi
      TantQue I(lettre) faire état := q1 ; direction := droite ; finTQ
      Si lettre = ">" alors stop := Vrai ; sinon direction := gauche ; état := q2 ; finSi
      * on vérifie que m = i_n ··· i_2 i_1 x_1 x_2 ··· x_p avec φ(i_1 ··· i_n) = x_1 ··· x_p *
      * la tête de lecture est sur la lettre i_1 ou bien stop vaut Vrai *
10    TantQue (lettre ≠ "<") et non stop faire
         i := lettre ; surligner(lettre) ;
         Si longueurφ(i) = 0 alors direction := gauche ; état := q_{φ(i)_0} ;
         sinon état := q_{φ(i)_1} ;
            * on recherche la première lettre de X non surlignée, sur la droite *
15          TantQue Isurligné (lettre) faire direction := droite ; finTQ
            TantQue Xsurligné (lettre) faire direction := droite ; finTQ
            * le ruban est de la forme i_n ··· i_j i_{j-1} ··· i_1 x_1 ··· x_{k-1} x_k ··· x_p *
            * si x_k ··· x_{k+q} = φ(i_j) on surligne x_k ··· x_{k+q} *
            rechercherφ(i) * stop = Vrai si φ(i) n'est pas à droite *
20          * on recherche la première lettre de I non surlignée, sur la gauche *
            TantQue Xsurligné(lettre) faire direction := gauche ; finTQ
            TantQue Isurligné(lettre) faire direction := gauche ; finTQ
         finSi
      finTQ
25    état := qreinit ; Si lettre = "<" alors direction := droite finSi
      * on remplace les lettres surlignées par les non-surlignées *
      TantQue Isurligné(lettre) faire réécrire(lettre) ; direction := droite ; finTQ
      TantQue Xsurligné(lettre) faire réécrire(lettre) ; direction := droite ; finTQ
      Si lettre ≠ ">" alors stop := Vrai ;
30    sinon état := qrecom ; direction := gauche ; * on se positionne sur i_1 *
         TantQue X(lettre) faire direction := gauche ; finTQ
      finSi
      * On fait exactement la même chose avec ψ *
      TantQue (lettre ≠ "<") et non stop faire
35       ...
      Si lettre ≠ ">" alors stop := Vrai ;
      sinon état := qfin ; * on restaure la configuration initiale *
         TantQue lettre ≠ "<" faire direction := gauche ; finTQ
         direction := droite ; état := q0 ;
40    finSi
   finTQ
finProgramme
```

Figure 2.1 : Automate $A_{\varphi\psi}$

```
Procédure recherche_φ(i) ;
    l := longueur_φ(i) ;
    Pour k := 1 jusqu'à l faire
        * On vérifie que la lettre lue est bien la kième lettre de φ(i) *
        Si lettre = φ(i)_k alors
            surligner(lettre) ;
            Si k ≠ l alors
                état := q_φ(i)_k+1 ;
                direction := droite ;
            sinon
                état := qret ;
                direction := gauche ;
            finSi
        sinon stop := Vrai ; exit ; finSi
    finPour
finProcédure
```

Figure 2.2 : Procédure de recherche de $\varphi(i)$

- Si la première lettre n'est pas dans I alors la machine s'arrête.
- Sinon,
  - Si le ruban ne contient que des lettres de I, alors la tête va se déplacer à droite, rencontrer le symbole >, et s'arrêter.
  - Sinon, le ruban est de la forme $\# < i_p \cdots i_1 x_1 \cdots x_j i \cdots > \#$. On vérifie alors que $\varphi(i_1) = x_1 \cdots x_j$, $\varphi(i_2) = x_{j+1} \cdots x_q$, ..., $\varphi(i_p) = x_{r+1} \cdots, x_f$. Si ce n'est pas le cas, alors l'automate stoppe (propriété de la procédure rechercheφ). Sinon la description instantanée de la machine est de la forme $\# < \overline{i_p} \cdots \overline{i_1 x_1} \cdots \overline{x_f} i \cdots > \#$. En désurlignant les lettres, l'automate s'arrêtera sur la lettre i.

Donc, si la configuration initiale n'est pas convenable, la machine s'arrête. □

Par la suite, $\tilde{m}_I$ représente le mot miroir de m_I.

Lemme 2.2.4 *Soit $A_{\varphi\psi}$ un automate de A_{Post} démarrant d'une configuration initiale convenable $\# < q_0 \tilde{m}_I m_X > \#$. $A_{\varphi\psi}$ boucle (en repassant par sa configuration initiale) si et seulement si $m_X = \varphi(m_I) = \psi(m_I)$.*

PREUVE : Si la donnée de départ est le mot $m = \tilde{m}_I m_X$ avec $\varphi(m_I) = \psi(m_I) = m_X$, il est facile de vérifier avec le programme de la figure 2.1 que le programme boucle en repassant par sa configuration initiale.

Par contre, si $\varphi(m_I) \neq m_X$ ou $\psi(m_I) \neq m_X$, le programme s'arrêtera. En effet, supposons que $\varphi(m_I) \neq m_X$ avec $m = i_1 \cdots i_n$ et $m_X = x_1 \cdots x_p$.

– 1er cas : $x_1 \cdots x_p$ préfixe de $\varphi(m_I)$.

La procédure **recherche**φ détectera que, pour une lettre i de m_I, $\varphi(i)$ n'est pas sur le ruban. Donc la variable **stop** sera mise à Vrai et le programme s'arrêtera.

– 2ème cas : $x_1 \cdots x_p$ suffixe de $\varphi(m_I)$.

Le programme vérifie qu'il n'y a pas d'autres lettres à droite que celles de $\varphi(m_I)$. Donc, si $x_1 \cdots x_p$ suffixe de $\varphi(m_I)$, la variable **stop** sera mise à Vrai grâce à l'instruction de la ligne 28.

– 3ème cas : $x_1 \cdots x_p$ et $\varphi(m_I)$ sont incomparables. On arrive alors à la configuration :

$$\# < i_n \cdots i_{k+1} \overline{i_k} \cdots \overline{i_1 \varphi(i_1)} \cdots \overline{\varphi(i_{k-1}) x_j} \cdots \overline{x_{j+q}} q_{\varphi(i_k)_q} x_{j+q+1} \cdots x_p > \#$$

avec x_{j+q+1} différente de la $q$ième lettre de $\varphi(i_k)$. La variable **stop** sera mise à Vrai dans la procédure **recherche**φ et le programme s'arrêtera. □

On obtient en corollaire de ces lemmes la proposition suivante :

Proposition 2.2.5 *1. La terminaison est indécidable pour la classe* A_{Post}.

2. Si un automate $A_{\varphi\psi}$ *de* A_{Post} *boucle pour la donnée d, alors il repasse par sa configuration initiale.*

PREUVE :

1. D'après les lemmes 2.2.3 et 2.2.4, $A_{\varphi\psi}$ ne termine pas si et seulement s'il existe m_I et m_X tels que $m_X = \varphi(m_I) = \psi(m_I)$. Or ce problème est indécidable (problème de correspondance de Post). Donc la terminaison est indécidable pour les automates de la classe A_{Post}.
2. D'après le lemme 2.2.4, si la machine boucle elle repasse par sa configuration initiale.

□

Lemme 2.2.6 *S'il existe un calcul qui boucle à partir d'une configuration quelconque (pas nécessairement accessible) alors il existe un début de calcul partant d'une configuration initiale convenable qui boucle.*

PREUVE : Soit DI une configuration à partir de laquelle la machine $A_{\varphi\psi}$ boucle.

– 1er cas : DI est accessible. Cela signifie qu'il existe une configuration initiale à partir de laquelle on l'a atteinte. Donc il existe un calcul partant d'une configuration initiale qui boucle. D'après le lemme 2.2.3, cette configuration initiale est convenable.

– 2ème cas : DI n'est pas accessible.

Examinons le graphe de dépendance des états de la machine $A_{\varphi\psi}$, décrit figure 2.3. Ce graphe est construit de la manière suivante : si $A_{\varphi\psi}$ passe de l'état q à l'état q' par une transition, alors il existe une flèche reliant le sommet q au sommet q'. Pour simplifier le graphe, on construit un seul sommet pour l'ensemble des états $q_{\varphi(i)_j}$, noté Q_φ et un autre pour l'ensemble Q_ψ des $q_{\psi(i)_j}$. Notons que dans le programme

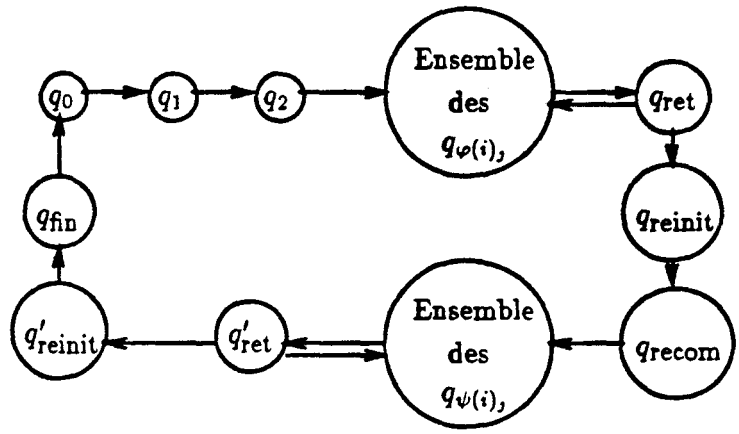


Figure 2.3 : Graphe de dépendance des états

de la figure 2.1, les points de suspension de la ligne 35 sont à remplacer par les lignes 9 à 27, où l'on écrit ψ à la place de φ, q'_{ret} au lieu de q_{ret}, et q'_{reinit} au lieu de q_{reinit}. De plus, on suppose qu'il existe une procédure rechercheψ construite sur le même modèle que rechercheφ.

1. La machine ne boucle pas sur un seul état, ni sur l'un des ensembles Q_φ et Q_ψ.

   - Lorsque la machine applique une transition en restant dans le même état, alors la tête de l'automate se déplace toujours dans la même direction. Or le ruban est fini, à droite comme à gauche. Donc on ne peut pas rester dans le même état en effectuant un nombre infini de pas de calcul.

   - Pour l'ensemble Q_φ :

   * On reste dans un état $q_{\varphi(i)_j}$ si et seulement si $j = 1$. On se ramène alors au cas précédent.

   * Si $j > 1$ alors on passe dans l'état $q_{\varphi(i)_{j+1}}$ (ou alors on sort de Q_φ). Comme j ne prend qu'un nombre fini de valeurs, on fait un nombre fini de pas de calcul en restant dans l'ensemble Q_φ.

   * Si j = 0 alors on remplace une lettre de I par une lettre de $\overline{I}$, en restant dans un état de Q_φ. Or, quand la machine est dans un état de Q_φ, jamais elle ne transforme une lettre de $\overline{I}$ en une lettre de I. Donc le nombre de lettres de I sur le ruban va décroître. De plus, s'il n'y a plus de lettre de I sur le ruban, on

ne peut plus appliquer de transition en étant dans un état $q_{\varphi(i)_0}$ et en restant dans un état de Q_φ. Donc on fait un nombre fini de pas de calcul en restant dans Q_φ.

- Il en est de même pour Q_ψ.

2. La machine ne boucle pas sur Q_φ et q_{ret} :

   Quand on passe de Q_φ à q_{ret}, on a barré au moins une lettre du ruban. Il en est de même pour le passage de q_{ret} à Q_φ. Il y a un nombre fini de lettres sur le ruban et le nombre de lettres non barrées décroît strictement donc on ne peut pas boucler entre Q_φ et q_{ret}. On peut tenir le même raisonnement pour Q_ψ et q'_{ret}

3. De 1 et 2, on déduit que la machine boucle en repassant par q_0, q_1, q_2, q_{ret}, q_{reinit}, q_{recom}, q'_{ret}, q'_{reinit}, q_{fin}, et par les ensembles Q_φ et Q_ψ. Or, la seule façon de repasser par q_0 est de restaurer la configuration initiale $\# < q_0 m > \#$ (lignes 37 à 40 dans le programme). Enfin, d'après le lemme 2.2.3, si la machine boucle à partir d'une configuration initiale, alors celle-ci est convenable.

□

2.2.2 Systèmes préservant les longueurs

On construit maintenant une classe R_{Post} de systèmes de réécriture préservant les longueurs à partir des automates linéairement bornés définis au paragraphe précédent.

I, X, Σ sont les alphabets déjà utilisés pour les ALB de la classe A_{Post}. On construit de nouveaux alphabets : $\overleftarrow{I}$, $\overrightarrow{I}$, $\overleftarrow{X}$, $\overrightarrow{X}$, $\overleftarrow{\Sigma}$, $\overrightarrow{\Sigma}$ à partir de I, X et Σ, exactement comme on avait défini $\overline{I}$ et $\overline{X}$ à partir de I et X.

On associe à toute machine $A_{\varphi\psi}$ de A_{Post} un système $\mathcal{R}_{\varphi\psi}$ de R_{Post}, de la manière suivante :

$$(q_1, a_1) \rightarrow_{A_{\varphi\psi}} (a_2, q_2, D) \Leftrightarrow \forall a, b \in \Sigma \ \overleftarrow{a}\, q_1\, \overrightarrow{a_1}\, \overrightarrow{b} \rightarrow_{\mathcal{R}_{\varphi\psi}} \overleftarrow{a}\, \overleftarrow{a_2}\, q_2\, \overrightarrow{b}$$

$$(q_1, a_1) \rightarrow_{A_{\varphi\psi}} (a_2, q_2, G) \Leftrightarrow \forall a, b \in \Sigma \ \overleftarrow{a}\, \overleftarrow{b}\, q_1\, \overrightarrow{a_1} \rightarrow_{\mathcal{R}_{\varphi\psi}} \overleftarrow{a}\, q_2\, \overrightarrow{b}\, \overrightarrow{a_2}$$

(intuitivement, $\overleftarrow{a}$ est une lettre située à gauche de la tête de l'automate et $\overrightarrow{a}$, à droite).

R_{Post} est bien une classe de systèmes de réécriture de mots qui préservent les longueurs. Soient φ et ψ deux morphismes de I dans X^*. Pour simplifier les notations, on écrit $\mathcal{R}$ au lieu de $\mathcal{R}_{\varphi\psi}$, A pour $A_{\varphi\psi}$ et Q pour $Q_{\varphi\psi}$. Le lemme suivant signifie que tout mot sur l'alphabet $\vec{\Sigma} \cup \overleftarrow{\Sigma} \cup Q$ contient un ou plusieurs facteurs représentant des rubans d'ALB. Ces facteurs sont des mots de $\overleftarrow{\Sigma}^+.Q.\vec{\Sigma}^+$, ou plus généralement, de $(\overleftarrow{\Sigma} \cup Q)^*.(\vec{\Sigma} \cup Q)^*$. Nous démontrons par la suite que les réécritures se font sur ces "rubans" de manières indépendantes. Ainsi, la non-terminaison du système de réécriture, sur des mots représentant plusieurs "rubans" d'automates linéairement bornés est équivalente à la non-terminaison de la machine associée qui ne comprend qu'un seul ruban.

Lemme 2.2.7 *Pour tout m de $(\vec{\Sigma} \cup \overleftarrow{\Sigma} \cup Q)^*$*

- soit $m \in (\overleftarrow{\Sigma} \cup Q)^*.(\vec{\Sigma} \cup Q)^*$

- soit m peut s'écrire de façon unique $m_1\ \vec{a_1}\ u_1\ \overleftarrow{b_1}\ m_2\ \vec{a_2}\ u_2\ \overleftarrow{b_2} \cdots m_n\ \vec{a_n}\ u_n\ \overleftarrow{b_n}\ m_{n+1}$ avec $n > 0$, $m_1, \cdots, m_{n+1} \in (\overleftarrow{\Sigma} \cup Q)^*.(\vec{\Sigma} \cup Q)^*$, $u_1, \cdots, u_n \in Q^*$, $\vec{a_1}, \cdots, \vec{a_n} \in \vec{\Sigma}$, $\overleftarrow{b_1}, \cdots, \overleftarrow{b_n} \in \overleftarrow{\Sigma}$.

PREUVE : Par induction sur la longueur de m.

1. Si $|m| = 1$ alors m appartient soit à $\overleftarrow{\Sigma}$, soit à $\vec{\Sigma}$, soit à Q.

   Donc $m \in (\overleftarrow{\Sigma} \cup Q)^*.(\vec{\Sigma} \cup Q)^*$.

2. Supposons que le lemme est vrai pour tout mot m de longueur k fixée, et démontrons qu'il reste vrai pour tout mot $m.a$ de longueur $k+1$.

   - 1er cas : $a \in \vec{\Sigma} \cup Q$.
     - si $m \in (\overleftarrow{\Sigma} \cup Q)^*.(\vec{\Sigma} \cup Q)^*$ alors $m.a \in (\overleftarrow{\Sigma} \cup Q)^*.(\vec{\Sigma} \cup Q)^*$.
     - sinon, $m = m_1\ \vec{a_1}\ u_1\ \overleftarrow{b_1}\ m_2\ \vec{a_2}\ u_2\ \overleftarrow{b_2} \cdots m_n\ \vec{a_n}\ u_n\ \overleftarrow{b_n}\ m_{n+1}$ avec $n > 0$, $m_1, \cdots, m_{n+1} \in (\overleftarrow{\Sigma} \cup Q)^*.(\vec{\Sigma} \cup Q)^*$, $u_1, \cdots, u_n \in Q^*$, $\vec{a_1}, \cdots, \vec{a_n} \in \vec{\Sigma}$, $\overleftarrow{b_1}, \cdots, \overleftarrow{b_n} \in \overleftarrow{\Sigma}$. Comme $m_{n+1}.a \in (\overleftarrow{\Sigma} \cup Q)^*.(\vec{\Sigma} \cup Q)^*$, $m.a = m_1\ \vec{a_1}\ u_1\ \overleftarrow{b_1}\ m_2\ \vec{a_2}\ u_2\ \overleftarrow{b_2} \cdots m_n\ \vec{a_n}\ u_n\ \overleftarrow{b_n}\ \mu_{n+1}$, où $\mu_{n+1} = m_{n+1}.a$.

- 2ème cas : $a \in \overleftarrow{\Sigma}$.
  - si $m \in (\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$, soit $m \in (\overleftarrow{\Sigma} \cup Q)^*$ et alors $m.a \in (\overleftarrow{\Sigma} \cup Q)^*$, soit $m \notin (\overleftarrow{\Sigma} \cup Q)^*$ et alors m contient au moins une lettre de $\overrightarrow{\Sigma}$. Dans ce dernier cas, $m = v_1 v_2 \overrightarrow{\sigma} v_3$ avec $v_1 \in (\overleftarrow{\Sigma} \cup Q)^*$, $v_2 \in (\overrightarrow{\Sigma} \cup Q)^*$, $\overrightarrow{\sigma} \in \overrightarrow{\Sigma}$ et $v_3 \in Q^*$. Donc $m.a = v_1 v_2 \overrightarrow{\sigma} v_3 a$ et peut s'écrire de manière unique $m_1 \overrightarrow{a_1} u_1 \overleftarrow{b_1} m_2$ avec $m_1, m_2 \in (\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$, $\overrightarrow{a_1} \in \overrightarrow{\Sigma}$, $u_1 \in Q^*$ et $\overleftarrow{b_1} \in \overleftarrow{\Sigma}$. En effet, on choisit $m_1 = v_1 v_2$, $\overrightarrow{a_1} = \overrightarrow{\sigma}$, $u_1 = v_3$, $\overleftarrow{b_1} = a$ et $m_2 = \epsilon$.
  - sinon $m = m_1 \overrightarrow{a_1} u_1 \overleftarrow{b_1} m_2 \overrightarrow{a_2} u_2 \overleftarrow{b_2} \cdots m_n \overrightarrow{a_n} u_n \overleftarrow{b_n} m_{n+1}$ avec $n > 0$, $m_1, \cdots, m_{n+1} \in (\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$, $u_1, \cdots, u_n \in Q^*$, $\overrightarrow{a_1}, \cdots, \overrightarrow{a_n} \in \overrightarrow{\Sigma}$, $\overleftarrow{b_1}, \cdots, \overleftarrow{b_n} \in \overleftarrow{\Sigma}$. Soit $m_{n+1} \in (\overleftarrow{\Sigma} \cup Q)^*$ et alors $m_{n+1}.a \in (\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$. Dans ce cas $m.a = m_1 \overrightarrow{a_1} u_1 \overleftarrow{b_1} m_2 \overrightarrow{a_2} u_2 \overleftarrow{b_2} \cdots m_n \overrightarrow{a_n} u_n \overleftarrow{b_n} \mu_{n+1}$ avec $\mu_{n+1} = m_{n+1}.a$. Soit $m_{n+1} \notin (\overleftarrow{\Sigma} \cup Q)^*$ et alors m_{n+1} contient au moins une lettre de $\overrightarrow{\Sigma}$: $m_{n+1} = v_1 v_2 \overrightarrow{\sigma} v_3$ avec $v_1 \in (\overleftarrow{\Sigma} \cup Q)^*$, $v_2 \in (\overrightarrow{\Sigma} \cup Q)^*$, $\overrightarrow{\sigma} \in \overrightarrow{\Sigma}$ et $v_3 \in Q^*$. Donc $m.a = m_1 \overrightarrow{a_1} u_1 \overleftarrow{b_1} m_2 \cdots m_n \overrightarrow{a_n} u_n \overleftarrow{b_n} \mu_{n+1} \overrightarrow{a_{n+1}} u_{n+1} \overleftarrow{b_{n+1}} m_{n+2}$. En effet, on pose $\mu_{n+1} = v_1 v_2$, $\overrightarrow{a_{n+1}} = \overrightarrow{\sigma}$, $u_{n+1} = v_3$, $\overleftarrow{b_{n+1}} = a$ et $m_{n+2} = \epsilon$. Cette décomposition est unique.

□

Corollaire 2.2.8 *Tout mot m qui n'appartient pas à $(\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$ peut être décomposé de manière unique en $w_1 w_2 \cdots w_{n+1}$ avec $w_i \in \overleftarrow{\Sigma}.(\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*.\overrightarrow{\Sigma}.Q^*$ pour $1 < i \leq n$, et $w_1, w_{n+1} \in (\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$.*

PREUVE : Dans la décomposition donnée par le lemme 2.2.7, on suppose $w_1 = m_1 \overrightarrow{a_1} u_1$, $w_i = \overleftarrow{b_i} m_{i+1} \overrightarrow{a_{i+1}} u_{i+1}$ pour $1 < i \leq n$, et $w_{n+1} = \overleftarrow{b_n} m_{n+1}$. □

Définition 2.2.9 *L'occurrence d'une lettre x_i dans un mot $x_1 \cdots x_n$ est la longueur du mot $x_1, \cdots, x_i$, c'est-à-dire i.*

Définition 2.2.10 *Soit m un mot de $(\overrightarrow{\Sigma} \cup \overleftarrow{\Sigma} \cup Q)^*$. On définit la* signature *de m par*

- Si $m \in (\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$ alors sa signature est $\emptyset$.

– Sinon, sa signature est l'ensemble des occurrences des lettres $\vec{a_i}$ dans la décomposition de m décrite par le lemme 2.2.7.

EXEMPLES :

- $\Sigma = \{a, b, c, d, e, f, g\}$; $m = acfgc$. L'occurrence de la première lettre c dans m est 2, l'occurrence de la deuxième lettre c est 5.
- $\Sigma = \{0, 1\}$; $Q = \{q\}$.
  $m = \overleftarrow{0}\,\overleftarrow{1}\,\overleftarrow{0}\ q\ \overleftarrow{0}\ q\ \overrightarrow{0}\,\overrightarrow{1}$; la signature de m est $\emptyset$.
  $m = \overleftarrow{0}\,\overleftarrow{1}\,\overleftarrow{0}\ q\ \overrightarrow{0}\ q\ \overleftarrow{0}\,\overleftarrow{1}\ q\ \overleftarrow{0}\,\overleftarrow{0}\,\overleftarrow{1}$; la signature de m est $\{5, 11\}$.

Lemme 2.2.11 *La taille et la signature d'un mot m restent inchangées quand m est réécrit par $\mathcal{R}$.*

PREUVE : Soit m un mot de $(\overrightarrow{\Sigma} \cup \overleftarrow{\Sigma} \cup Q)^*$.

1. Si $m \in (\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$ alors sa signature est $\emptyset$. On applique sur m une règle de $\mathcal{R}$.
   (a) Si c'est une règle de la forme $\overleftarrow{a}\ q_1\ \overrightarrow{a_1}\,\overrightarrow{b} \rightarrow_{\mathcal{R}} \overleftarrow{a}\,\overleftarrow{a_2}\ q_2\ \overrightarrow{b}$, elle ne peut s'appliquer que si $m = m_1\ \overleftarrow{a}\ q_1\ \overrightarrow{a_1}\,\overrightarrow{b}\ m_2$ avec $m_1 \in (\overleftarrow{\Sigma} \cup Q)^*$ et $m_2 \in (\overrightarrow{\Sigma} \cup Q)^*$. Alors, en appliquant la règle, on obtient le mot $m' = m_1\ \overleftarrow{a}\,\overleftarrow{a_2}\ q_2\ \overrightarrow{b}\ m_2$. Donc $|m| = |m'| = |m_1| + |m_2| + 4$ et $m' \in (\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$ donc sa signature est vide.
   (b) Si c'est une règle de la forme $\overleftarrow{a}\,\overleftarrow{b}\ q_1\ \overrightarrow{a_1} \rightarrow_{\mathcal{R}} \overleftarrow{a}\ q_2\ \overrightarrow{b}\,\overrightarrow{a_2}$, elle ne peut s'appliquer que si $m = m_1\ \overleftarrow{a}\,\overleftarrow{b}\ q_1\ \overrightarrow{a_1}\ m_2$ avec $m_1 \in (\overleftarrow{\Sigma} \cup Q)^*$ et $m_2 \in (\overrightarrow{\Sigma} \cup Q)^*$. On obtient par réécriture le mot $m' = m_1\ \overleftarrow{a}\ q_2\ \overrightarrow{b}\,\overrightarrow{a_2}\ m_2$. Donc $|m| = |m'| = |m_1| + |m_2| + 4$ et $m' \in (\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$ donc sa signature est vide.
2. Sinon, m s'écrit $m_1\ \overrightarrow{a_1}\ u_1\ \overleftarrow{b_1}\ m_2\ \overrightarrow{a_2}\ u_2\ \overleftarrow{b_2} \cdots m_n\ \overrightarrow{a_n}\ u_n\ \overleftarrow{b_n}\ m_{n+1}$, décomposition donnée par le lemme 2.2.7. On applique une seule règle de $\mathcal{R}$ sur m. Celle-ci ne peut s'appliquer que sur un mot de $(\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$. Donc on se ramène au cas précédent. □

Lemme 2.2.12 *Soit m un mot qui n'appartient pas à $(\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$. Ce mot s'écrit $w_1 \cdots w_{n+1}$, décomposition donnée dans le corollaire 2.2.8. On a alors :*

$$(m \xrightarrow{*}_{\mathcal{R}} t) \Leftrightarrow (t = t_1 t_2 \cdots t_{n+1} \text{ tel que } \forall i\ w_i \xrightarrow{*}_{\mathcal{R}} t_i)$$

PREUVE :

$\Rightarrow$) Soit $m = w_1 \cdots w_{n+1}$. On a démontré grâce au lemme 2.2.11 que les règles de $\mathcal{R}$ ne peuvent s'appliquer que sur les w_i.

Donc $(m \xrightarrow{*}_{\mathcal{R}} t) \Rightarrow (t = t_1 t_2 \cdots t_{n+1}$ tel que $\forall i\ w_i \xrightarrow{*}_{\mathcal{R}} t_i)$.

$\Leftarrow$) Si $t = t_1 \cdots t_{n+1}$, et si $\forall i\ w_i \xrightarrow{*}_{\mathcal{R}} t_i$ alors il est clair que $w_1 \cdots w_{n+1} \xrightarrow{*}_{\mathcal{R}} t_1 \cdots t_{n+1}$. Donc $m \xrightarrow{*}_{\mathcal{R}} t$. □

Définition 2.2.13 *Soit m un mot de la forme $\overleftarrow{a}\overleftarrow{v}\ q\ \overrightarrow{w}\overrightarrow{b}$ avec $q \in Q$, $\overleftarrow{a} \in \overleftarrow{\Sigma}$, $\overleftarrow{v} \in \overleftarrow{\Sigma}^*$, $\overrightarrow{b} \in \overrightarrow{\Sigma}$ et $\overrightarrow{w} \in \overrightarrow{\Sigma}^*$. On associe à m la configuration de la machine A : $C(m) = avqwb$.*

REMARQUE : a et b sont des caractères qui ne sont jamais lus par la machine et sont juste des marqueurs de fin de bande (à la place du symbole #).

Lemme 2.2.14 $m \in \overleftarrow{\Sigma}^+.Q.\overrightarrow{\Sigma}^+$, $m \rightarrow_{\mathcal{R}} m' \Leftrightarrow C(m) \rightarrow_A C(m')$.

PREUVE : Soit m un mot de $\overleftarrow{\Sigma}^+.Q.\overrightarrow{\Sigma}^+$.

$\Rightarrow$) On associe au mot m la configuration $C(m)$. Les règles de m sont de deux formes possibles :

(1) $\overleftarrow{a}\ q_1\ \overrightarrow{a_1}\overrightarrow{b} \rightarrow_{\mathcal{R}} \overleftarrow{a}\overleftarrow{a_2}\ q_2\ \overrightarrow{b}$ pour a et b dans Σ.

(2) $\overleftarrow{a}\overleftarrow{b}\ q_1\ \overrightarrow{a_1} \rightarrow_{\mathcal{R}} \overleftarrow{a}\ q_2\ \overrightarrow{b}\overrightarrow{a_2}$ pour a et b dans Σ.

- 1er cas : on applique une règle r_1 de type (1). Pour appliquer r_1 sur m, m doit être de la forme $\overleftarrow{v}\overleftarrow{a}\ q_1\ \overrightarrow{a_1}\overrightarrow{b}\overrightarrow{w}$ avec $\overleftarrow{v} \in \overleftarrow{\Sigma}^*$ et $\overrightarrow{w} \in \overrightarrow{\Sigma}^*$. Alors $C(m) = vaq_1a_1bw$. Le mot m' obtenu en appliquant la règle r_1 sur m est $\overleftarrow{v}\overleftarrow{a}\overleftarrow{a_2}\ q_2\ \overrightarrow{b}\overrightarrow{w}$. Mais r_1 a été construite à partir de la transition $(q_1, a_1) \rightarrow_A (a_2, q_2, D)$. Donc $C(m) \rightarrow_A C(m')$.

- 2ème cas : on applique une règle r_2 de type (2). Pour appliquer r_2 sur m, m doit être de la forme $\overleftarrow{v}\,\overleftarrow{a}\,\overleftarrow{b}\ q_1\ \overrightarrow{a_1}\overrightarrow{w}$ avec $\overleftarrow{v}\in\overleftarrow{\Sigma}^*$ et $\overrightarrow{w}\in\overrightarrow{\Sigma}^*$. Alors $C(m) = vabq_1a_1w$. Donc m' tel que $m \rightarrow_{r_2} m'$ est $m' = \overleftarrow{v}\,\overleftarrow{a}\ q_2\ \overrightarrow{b}\,\overrightarrow{a_2}\overrightarrow{w}$. Mais r_2 a été construite à partir de la transition $(q_1, a_1) \rightarrow_A (a_2, q_2, G)$. Donc $C(m) \rightarrow_A C(m')$.

$\Leftarrow$) On suppose maintenant qu'il existe m' tel que $C(m) \rightarrow_A C(m')$. Les transitions de A sont de deux formes :

$(1bis)$ $(q_1, a_1) \rightarrow_A (a_2, q_2, D)$

$(2bis)$ $(q_1, a_1) \rightarrow_A (a_2, q_2, G)$

$C(m) = \alpha\sigma q_1 a_1 \omega\beta$ avec $\alpha, \beta \in \Sigma$ et $\sigma, \omega \in \Sigma^*$. (En fait les lettres α et β servent de marqueurs de fin de ruban).

- 1er cas : on applique sur $C(m)$ une transition t_1 de type (1bis). On associe à t_1 un nombre fini de règles de type (1), en faisant varier les valeurs de a et b. m est un mot de $\overleftarrow{\Sigma}^+.Q.\overrightarrow{\Sigma}^+$ donc parmi ces règles, il n'y en a qu'une applicable sur m (imposée par les valeurs de a et b). Soit r_1 cette règle. Le mot m de configuration $C(m)$ peut s'écrire $\overleftarrow{\alpha}\overleftarrow{\sigma}\ q_1\ \overrightarrow{a_1}\overrightarrow{\omega}\overrightarrow{\beta}$. Le mot m' tel que $m \rightarrow_A m'$ est $\overleftarrow{\alpha}\overleftarrow{\sigma}\overleftarrow{a_2}\ q_2\ \overrightarrow{\omega}\overrightarrow{\beta}$. Donc, en supposant que $\overleftarrow{v}\,\overleftarrow{a} = \overleftarrow{\alpha}\overleftarrow{\sigma}$ et $\overrightarrow{b}\overrightarrow{w} = \overrightarrow{\omega}\overrightarrow{\beta}$ on obtient $m' = \overleftarrow{v}\,\overleftarrow{a}\overleftarrow{a_2}\ q_2\ \overrightarrow{b}\overrightarrow{w}$ et $m = \overleftarrow{v}\,\overleftarrow{a}\ q_1\ \overrightarrow{a_1}\,\overrightarrow{b}\overrightarrow{w}$, donc $m \rightarrow_{r_1} m'$.
- 2ème cas : on applique sur $C(m)$ une transition t_2 de type (2bis). Comme pour le cas précédent, on associe à t_2 un nombre fini de règles de type (2) mais il n'y en a qu'une applicable sur m. Soit r_2 cette règle. Le mot m de configuration $C(m)$ peut s'écrire $\overleftarrow{\alpha}\overleftarrow{\sigma}\ q_1\ \overrightarrow{a_1}\overrightarrow{\omega}\overrightarrow{\beta}$ (comme pour le cas précédent). En posant $\overleftarrow{v}\,\overleftarrow{a} = \overleftarrow{\alpha}\overleftarrow{\sigma}$ et $\overrightarrow{b}\overrightarrow{w} = \overrightarrow{\omega}\overrightarrow{\beta}$ on a $m = \overleftarrow{v}\,\overleftarrow{a}\ q_1\ \overrightarrow{a_1}\,\overrightarrow{b}\overrightarrow{w}$ et le mot m' tel que $C(m) \rightarrow_{t_2} C(m')$ s'écrit $\overleftarrow{v}\ q_2\ \overrightarrow{a}\,\overrightarrow{a_2}\,\overrightarrow{b}\overrightarrow{w}$. Comme $C(m)$ est une configuration $\overleftarrow{v}$ contient au moins une lettre (le marqueur de fin de bande). Donc on peut apliquer la règle r_2 sur m et $m \rightarrow_{r_2} m'$. □

Lemme 2.2.15 *Si l'automate A ne termine pas alors le système $\mathcal{R}$ ne termine pas non plus.*

PREUVE : Supposons que A ne termine pas, alors il existe un calcul infini partant d'une configuration initiale $C_0 \rightarrow_A C_1 \rightarrow_A \cdots$. Mais pour toute configuration C_i, il

existe un mot m_i de $\overleftarrow{\Sigma}^+.Q.\overrightarrow{\Sigma}^+$ tel que $C_i = C(m_i)$. Donc il existe $m_0, m_1, \cdots$ dans $\overleftarrow{\Sigma}^+.Q.\overrightarrow{\Sigma}^+$ tels que $C(m_0) \rightarrow_A C(m_1) \rightarrow_A \cdots$. D'après le lemme 2.2.14, ceci signifie qu'il existe une dérivation infinie $m_0 \rightarrow_{\mathcal{R}} m_1 \rightarrow_{\mathcal{R}} \cdots$. Donc $\mathcal{R}$ ne termine pas. □

Lemme 2.2.16 *Si le système $\mathcal{R}$ ne termine pas alors il existe un calcul infini $C_0 \rightarrow_A C_1 \rightarrow_A \cdots$*

PREUVE : Supposons que $\mathcal{R}$ ne termine pas. Il existe alors une dérivation infinie $m \rightarrow_{\mathcal{R}} \cdots$. Soit $m \in (\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$, soit $m = m_1 \overrightarrow{a_1} u_1 \overleftarrow{b_1} m_2 \overrightarrow{a_2} u_2 \overleftarrow{b_2} \cdots m_n \overrightarrow{a_n} u_n \overleftarrow{b_n} m_{n+1}$

1. Si $m \in (\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$, on peut appliquer une règle de $\mathcal{R}$. Ce qui veut dire que $m = v_1 \sigma v_2$ avec $v_1 \in (\overleftarrow{\Sigma} \cup Q)^*.Q^+$ ou $v_1 \in Q^*$, $v_2 \in Q^+.(\overrightarrow{\Sigma} \cup Q)^*$ ou $v_2 \in Q^*$, et $\sigma \in \overleftarrow{\Sigma}^+.Q.\overrightarrow{\Sigma}^+$). On associe à σ la configuration $C(\sigma)$ de l'automate A. On ne peut pas appliquer de règle sur v_1 ni sur v_2. Donc on réécrit σ et il existe une dérivation infinie $\sigma \rightarrow_{\mathcal{R}} \cdots$. Le lemme 2.2.14 nous permet de déduire qu'il existe une dérivation infinie $C(\sigma) \rightarrow_A \cdots$.

2. Sinon $m = m_1 \overrightarrow{a_1} u_1 \overleftarrow{b_1} m_2 \overrightarrow{a_2} u_2 \overleftarrow{b_2} \cdots m_n \overrightarrow{a_n} u_n \overleftarrow{b_n} m_{n+1} = w_1 w_2 \cdots w_{n+1}$. D'après le lemme 2.2.12, $(m \xrightarrow{*}_{\mathcal{R}} t) \Leftrightarrow (t = t_1 t_2 \cdots t_{n+1}$ tel que $\forall i\ w_i \xrightarrow{*}_{\mathcal{R}} t_i)$. Donc s'il existe une dérivation infinie partant de m, il existe un w_i à partir duquel il existe aussi une dérivation infinie. Or ce w_i appartient à $(\overleftarrow{\Sigma} \cup Q)^*.(\overrightarrow{\Sigma} \cup Q)^*$. On peut donc se ramener au point précédent.

□

Théorème 2.2.17 *La terminaison des systèmes de réécriture préservant les longueurs est indécidable.*

PREUVE : D'après la proposition 2.2.5, le lemme 2.2.6 et le lemme 2.2.16, si $\mathcal{R}_{\varphi\psi}$ ne termine pas alors $A_{\varphi\psi}$ ne termine pas non plus. Le lemme 2.2.15 nous donne la réciproque. Donc la terminaison est indécidable pour la classe R_{Post}, car elle se ramène à la terminaison de A_{Post}. Or les systèmes de réécriture de R_{Post} préservent les longueurs. Ceci achève la preuve du théorème. □

REMARQUE : $A_{\varphi\psi}$ aurait pu boucler à partir d'une configuration non accessible et s'arrêter à partir de n'importe quelle configuration initiale. Ainsi elle aurait terminé au sens des machines de Turing, sans assurer la terminaison de $\mathcal{R}_{\varphi\psi}$. Le lemme 2.2.6 évite ce problème.

Corollaire 2.2.18 *La terminaison des systèmes de réécriture de mots de longueur non croissante est indécidable*

PREUVE : Les systèmes de réécriture préservant les longueurs sont des cas particuliers des systèmes de longueur non croissante. □

2.3 Confluence restreinte à une sorte

Nous voulons démontrer dans ce paragraphe qu'une propriété décidable dans le cas général, peut devenir indécidable quand on se restreint à des termes d'une sorte donnée. Le théorème de Newman dit que la confluence est décidable pour des systèmes de réécriture noethériens. Pour les systèmes de réécriture de mots, cette propriété se traduit par :

Confluence(SDR de mots noethériens) est décidable.

Nous prouvons que :

Confluence(SDR de mots noethériens, termes commençant par une lettre donnée) est indécidable.

Ceci signifie que, même dans le cas des mots, le théorème de Newman devient faux quand on se restreint à des mots commençant tous par la même lettre, ce qui est une contrainte reconnaissable très simple.

REMARQUE : La confluence close est indécidable pour des systèmes noethériens, mais notre résultat est différent. Il est bien connu que, sur les termes, la confluence et la confluence close ne sont pas équivalentes. Si un système R est confluent alors il est confluent sur les termes clos, mais le contraire est faux. Cependant, le lemme suivant montre que l'on peut identifier la confluence d'un système semi-Thue à la confluence et la confluence close du système de réécriture de termes S' correspondant.

Lemme 2.3.1 *Soit S un système semi-Thue sur un alphabet Σ. On associe à S le système de réécriture S' sur l'alphabet gradué $\Sigma' = \{a()|a \in \Sigma\} \cup \{\$\}$. ($a()$ est une lettre d'arité 1 et \$ une constante). $S' = \{l(x) \to r(x)|l \to r \in S\}$*

Alors $t \xrightarrow{*}_S u \Leftrightarrow t(x) \xrightarrow{*}_{S'} u(x) \Leftrightarrow t(\$) \xrightarrow{*}_{S'} u(\$)$.

PREUVE : Evidente. □

Comme corollaire, la confluence de S, la confluence et la confluence close de S' coïncident.

Nous définissons dans ce paragraphe une classe de systèmes de réécriture de mots sur un alphabet Λ. Ces systèmes ne sont pas confluents sur Λ^* mais leur confluence devient indécidable sur $q\Lambda^*$, q étant un symbole fixé de Λ.

Soit $A_{\varphi\psi}$ une machine de la classe A_{Post} étudiée précédemment. On associe à cet ALB un alphabet $\Lambda = Q_{\varphi\psi} \cup \Sigma \cup \{q, q_{\text{oui}}, O, N\}$. On modifie la machine $A_{\varphi\psi}$: on enlève les transitions qui font boucler la machine. Donc, la donnée s'écrit $\tilde{m}m'$ avec $m' = \varphi(m) = \psi(m)$ si et seulement si la machine s'arrête sur la configuration $\#q_{\text{fin}} < \tilde{m}m' > \#$. On associe à ce nouvel automate un système de réécriture R_1 qui le simule.

$(q, a) \rightarrow_{A_{\varphi\psi}} (b, q', G) \Leftrightarrow hqa \rightarrow_{R_1} q'hb$, avec $a, b, h \in \Lambda, q, q' \in Q_{\varphi\psi}$

$(q, a) \rightarrow_{A_{\varphi\psi}} (b, q', D) \Leftrightarrow qa \rightarrow_{R_1} bq'$, avec $a, b \in \Lambda, q, q' \in Q_{\varphi\psi}$

De plus, on ajoute à R_1 les règles :

$q_{\text{fin}} <\!\!\rightarrow_{R_1} q_{\text{oui}}$
$\forall a \in \Lambda - \{\#\}\ \ q_{\text{oui}}a \rightarrow_{R_1} q_{\text{oui}}$
$q_{\text{oui}}\# \rightarrow_{R_1} O$
$q \rightarrow_{R_1} < q_0$

On considère maintenant un deuxième système de réécriture R_2 :

$\forall f \in \Lambda - \{O, N\}\ \ f \rightarrow_{R_2} N$
$\forall a \in \Lambda\ \ Oa \rightarrow_{R_2} N$
$\forall a \in \Lambda\ \ Na \rightarrow_{R_2} N$

R est le système constitué par les règles de R_1 et de R_2. Notons que R est noethérien.

Lemme 2.3.2 $m \in I^+$, $m' \in X^*$, $q\tilde{m}m' > \# \xrightarrow{*}_R O \Leftrightarrow \varphi(m) = \psi(m) = m'$.

PREUVE :

$\Leftarrow$) Si $\varphi(m) = \psi(m) = m'$ alors l'automate $A_{\varphi\psi}$ partant de la description instantanée $\# < q_0\tilde{m}m' > \#$ arrive à la descrition instantanée $\#q_{\text{fin}} < \tilde{m}m' > \#$. Donc, on peut

réécrire $q\tilde{m}m' > \# \text{ en } < q_0\tilde{m}m' > \#$ puis en $q_{\text{fin}} < \tilde{m}m' > \#$. En appliquant la règle $q_{\text{fin}} <\!\!\rightarrow_R q_{\text{oui}}$ on obtient $q_{\text{oui}}\tilde{m}m' > \#$. Par des applications répétées de règles de la forme $q_{\text{oui}}a \rightarrow_R q_{\text{oui}}$ on arrive au mot $q_{\text{oui}}\#$. Enfin, grâce à la règle $q_{\text{oui}}\# \rightarrow O$ on obtient O.

$\Rightarrow$) Supposons que $q\tilde{m}m' > \# \rightarrow_R O$. m et m' ne contiennent pas O ni q_{oui} ni aucun état de $Q_{\varphi\psi}$. Il n'y a qu'une façon d'obtenir O: utiliser la règle $q_{\text{oui}}\# \rightarrow_R O$. De plus, l'état q_{oui} apparaît en appliquant la règle $q_{\text{fin}} <\!\!\rightarrow_R q_{\text{oui}}$. Pour engendrer q_{fin} on doit appliquer les règles qui simulent l'automate.

$q\tilde{m}m' > \# \rightarrow_R < q_0\tilde{m}m' > \# \overset{*}{\rightarrow}_R q_{\text{fin}} < \tilde{m}m' > \#$. Mais on a vu dans les paragraphes précédents que $\# < q_0\tilde{m}m' > \# \overset{*}{\rightarrow}_A \#q_{\text{fin}} < \tilde{m}m' > \#$ si et seulement si $\varphi(m) = \psi(m) = m'$. □

Définition 2.3.3 *Un mot w* est une forme normale *s'il n'existe pas de mot v tel que $w \rightarrow_{\mathcal{R}} v$*

Un mot w a une forme normale *v si $w \overset{*}{\rightarrow}_{\mathcal{R}} v$ et v est une forme normale.*

L'ensemble des formes normales du mot w, appelées aussi formes irréductibles de w, est noté IRR(w).

Lemme 2.3.4 *Pour tout u dans $\Lambda^+ - \{O\}$, N est dans IRR(u).*

PREUVE :

- Si u commence par une lettre f qui n'est ni N ni O, alors en appliquant la règle $f \rightarrow N$, et éventuellement des règles de la forme $Na \rightarrow N$ on obtient une dérivation $u \overset{*}{\rightarrow}_{\mathcal{R}} N$. Donc N est dans IRR(u).

- Si u commence par un N : $u = N.w$. Si $w = \epsilon$ alors $u = N$ et u est irréductible. Sinon, on applique plusieurs fois des règles du type $Na \rightarrow N$ et on prouve bien que N est dans IRR(u).

- Si u commence par O : $u = O.w$ tel que $w \neq \epsilon$. On peut donc appliquer une règle $Oa \rightarrow N$ et éventuellement des règles $Na \rightarrow N$. Donc N est dans IRR(u). □

Lemme 2.3.5 $\forall m \in \Lambda^*$, $IRR(m) \subset \{O, N, \epsilon\}$. $\forall m \in \Lambda^+$, $IRR(m) \subset \{O, N\}$.

PREUVE : Supposons qu'il existe u dans IRR(m), u différent de N, O et ϵ. D'après le lemme 2.3.4, u peut se réécrire en N, on obtient donc une contradiction car u ne

peut pas appartenir à IRR(m). Donc, la première partie du lemme est vraie. Pour la deuxième partie, il est évident que $\epsilon \in$ IRR(m) si et seulement si $m = \epsilon$. □

Lemme 2.3.6 *Soit $q.w$ un mot dans $q.\Lambda^*$.*

Si *qw a la forme $q\tilde{m}m' > \#$ avec $m \in I^+$ et $m' \in X^*$* **alors**

1. *IRR(qw) = $\{O, N\}$ ssi $\varphi(m) = \psi(m) = m'$*
2. *IRR(qw) = $\{N\}$ ssi $\varphi(m) \neq m'$ ou $\psi(m) \neq m'$*

Sinon *IRR(qw) = $\{N\}$*

PREUVE :

1er cas : qw a la forme $q\tilde{m}m' > \#$ avec $m \in I^+$ et $m' \in X^*$.

1. D'après le lemme 2.3.2, $q\tilde{m}m' > \# \xrightarrow{*}_{\mathcal{R}} O$ si et seulement si $\varphi(m) = \psi(m) = m'$. De plus, le lemme 2.3.4 nous dit que $q\tilde{m}m' > \# \xrightarrow{*}_{\mathcal{R}} N$.

   Donc $\{O, N\} \subset$ IRR(qw) si et seulement si $\varphi(m) = \psi(m) = m'$. Le lemme 2.3.5 montre que IRR(qw) = $\{O, N\}$ si et seulement si $\varphi(m) = \psi(m) = m'$.

2. $\varphi(m) \neq m'$ ou $\psi(m) \neq m'$ si et seulement si $O \notin$ IRR(qw).

   Mais $q\tilde{m}m' > \# \xrightarrow{*}_{\mathcal{R}} N$ et d'après le lemme 2.3.5, IRR(qw) $\subset \{O, N\}$. Donc IRR(qw) = $\{N\}$.

2ème cas : qw n'est pas de la forme $q\tilde{m}m' > \#$ avec $m \in I^+$ et $m' \in X^*$. Alors qw ne correspond pas à une configuration initiale convenable de $A_{\varphi\psi}$. Donc on applique les règles de R sans jamais obtenir de mot du type $q_{\text{fin}} < m$. C'est pourquoi $O \notin$ IRR(qw). Mais d'après le lemme 2.3.5 IRR(qw) $\subset \{O, N\}$. Donc IRR(qw) = $\{N\}$. □

Théorème 2.3.7 *q est un symbole fixé d'un alphabet fini Λ. Le problème suivant est indécidable :*

Instance : système de réécriture semi-Thue R, de longueur non croissante, noethérien, non confluent.

Question : R est-il confluent sur $q.\Lambda^*$?

PREUVE : D'après le lemme 2.3.6, R n'est pas confluent sur $q.\Lambda^*$ si et seulement s'il existe m tel que $\varphi(m) = \psi(m)$. Donc la confluence de R sur $q.\Lambda^*$ est équivalente au problème de correspondance de Post pour φ et ψ. Il est donc indécidable en général.

On peut considérer tout système de réécriture de mots comme un système de réécriture de termes (lemme 2.3.1). On obtient alors le corollaire 2.3.8

Corollaire 2.3.8 *Le problème suivant est indécidable :*

Instance : R est un système de réécriture de termes noethérien. S est une sorte.

Question : R est-il confluent sur S ?

Remarquons que S peut être choisie très simple, par exemple S est l'ensemble des termes qui commencent tous par le même symbole.

Chapitre 3

Propriétés modulaires

Si $\mathcal{F}_1$ et $\mathcal{F}_2$ sont deux alphabets gradués, finis et disjoints, et si $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ sont deux systèmes de réécriture de termes sur $\mathcal{F}_1$ et $\mathcal{F}_2$ alors l'union disjointe $\mathcal{R}_1 \oplus \mathcal{R}_2$ de $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ est le système de réécriture $(\mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{R}_1 \cup \mathcal{R}_2)$. Une propriété des systèmes de réécriture est dite modulaire si elle est préservée par union disjointe.

En programmation, la modularité permet de décomposer un problème en problèmes plus faciles à résoudre. Pour déduire des propriétés des systèmes de réécriture, comme la confluence ou la terminaison, il existe beaucoup de méthodes mais elles ont en général de plus grandes chances de succès si le système considéré a peu de règles. C'est pourquoi il est intéressant de savoir si un système de réécriture qui a une propriété P peut être partagé en systèmes plus petits, qui garderont cette propriété P. Beaucoup d'auteurs étudient les propriétés modulaires, et en particulier la confluence et la terminaison. Le premier d'entre eux est Y. Toyama. Il a montré dans [Toy87a] que la confluence était préservée par union disjointe. Une preuve simplifiée de ce résultat a été écrite depuis par [KMTdV91]. Il a également trouvé des contre-exemples à la modularité de la terminaison [Toy87b]. B. Gramlich a étudié plus en détail des contre-exemples minimaux à la modularité de la terminaison [Gra91]. M. Rusinowitch [Rus87] et A. Middeldorp [Mid89b] ont trouvé des conditions suffisantes pour que l'union disjointe de deux systèmes noethériens soit noethérienne. Kurihara et Ohuchi [KO90a] ont prouvé que la terminaison était modulaire si elle pouvait se démontrer par un ordre de simplification. Une autre propriété étudiée est la complétude, c'est-à-dire la confluence et la terminaison. Barendregt et Klop dans [Toy87b] et Drosten dans [Dro89] donnent des contre-exemples à la modularité de la complétude. Cependant, Toyama, Klop et Barendregt [TKB89] montrent que

l'union disjointe de deux systèmes de réécriture linéaires à gauche et complets est un système complet. Middeldorp [Mid89a] étudie certaines propriétés concernant les formes normales et prouve que la propriété de tous les termes de posséder une forme normale unique est modulaire. Il donne dans sa thèse [Mid91] une vue d'ensemble sur les aspects modulaires des systèmes de réécriture de termes et étudie plus particulièrement les systèmes de réécriture conditionnels [Mid91].

Dans ce chapitre, nous étudions la modularité de la décidabilité de l'accessibilité. Le problème de l'accessibilité pour deux termes t et t' et un système de réécriture $\mathcal{R}$ est de savoir si t se réécrit en t' par les règles de $\mathcal{R}$. Ce problème est donc aux systèmes de réécriture de termes ce qu'est le problème du mot pour les systèmes semi-Thue. Il est indécidable en général mais décidable pour des systèmes quasi-clos [Oya86], c'est-à-dire des systèmes clos à droite ou pour des systèmes monadiques. Un algorithme de décision de l'accessibilité pour des systèmes clos utilisant les automates d'arbres a été implémenté en Prolog par A. Deruyver [DD89], ainsi qu'un algorithme permettant de trouver une dérivation qui transforme un terme t en un terme t' [CG90].

Notre premier résultat est la non modularité de la décidabilité de l'accessibilité pour des systèmes de réécritures qui ne sont pas linéaires à gauche. Pour des systèmes linéaires à gauche, on doit prendre en considération la présence ou non de règles effaçantes, c'est-à-dire de règles de la forme $l \to r$ où r est une variable. La décidabilité de l'accessibilité est conservée par union disjointe si les systèmes sont linéaires à gauche sans règles effaçantes ou linéaires avec règles effaçantes. Pour des systèmes linéaires à gauche avec règles effaçantes, cette propriété n'est pas modulaire. Nous nous sommes aussi intéressés à la décidabilité de l'accessibilité close, malheureusement cette propriété n'est pas modulaire, même pour des systèmes linéaires sans règles effaçantes. La plupart de ces résultats sont parus dans [Car92].

L'union disjointe signifie l'union de systèmes travaillant sur des alphabets disjoints, et représente ainsi une forte restriction. Dans le dernier paragraphe, nous présentons les résultats concernant la composition de systèmes constructeurs. Y. Toyama et A. Middeldorp ont montré dans [MT91] qu'un système constructeur est complet (confluent et noethérien) s'il peut être décomposé en plusieurs systèmes constructeurs complets. Cette notion de décomposition ne nécessite pas que les alphabets soient disjoints. D'autres auteurs ont également cherché à éviter cette contrainte des alphabets disjoints ([KO90b], [Gra91]).

Nous obtenons les résultats suivants: si $\mathcal{R}_1$ et $\mathcal{R}_2$ sont des systèmes constructeurs non noethériens dont l'accessibilité est décidable, l'accessibilité de $\mathcal{R}_1 + \mathcal{R}_2$ devient indécidable. Si les deux systèmes sont noethériens alors l'accessibilité de $\mathcal{R}_1 + \mathcal{R}_2$ est décidable si les deux systèmes sont linéaires à droite, mais indécidable en général.

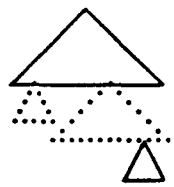


Figure 3.1 : structure des termes sur $\mathcal{F}_1 \cup \mathcal{F}_2$

3.1 Union disjointe de SDR

3.1.1 Définitions

La plupart de ces définitions proviennent de Y. Toyama [Toy87a] et A. Middeldorp [Mid90].

Un *contexte* $C[,\cdots,]$ est un "terme" qui contient au moins une occurrence d'un symbole de constante spécial $\Box$. Si $C[,\cdots,]$ est un contexte avec n occurrences de $\Box$ et si $t_1,\cdots,t_n$ sont des termes alors $C[t_1,\cdots,t_n]$ est le terme obtenu en remplaçant de la gauche vers la droite les occurrences de $\Box$ par $t_1,\cdots,t_n$. Un contexte qui ne contient qu'une seule occurrence de $\Box$ est noté $C[\,]$.

L'égalité littérale de termes est notée $\equiv$.

Pour préciser sur quel alphabet on travaille, le système de réécriture $\mathcal{R}$ agissant sur des termes de $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$ sera noté $(\mathcal{F},\mathcal{R})$.

Etant donnés deux systèmes de réécriture $(\mathcal{F}_1,\mathcal{R}_1)$ et $(\mathcal{F}_2,\mathcal{R}_2)$, l'union disjointe $\mathcal{R}_1 \oplus \mathcal{R}_2$ de $(\mathcal{F}_1,\mathcal{R}_1)$ et $(\mathcal{F}_2,\mathcal{R}_2)$ est le système de réécriture $(\mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{R}_1 \cup \mathcal{R}_2)$.

Une propriété P est dite modulaire si :

$\mathcal{R}_1 \oplus \mathcal{R}_2$ a la propriété $P \Leftrightarrow (\mathcal{F}_1,\mathcal{R}_1)$ et $(\mathcal{F}_2,\mathcal{R}_2)$ ont la propriété P.

Soient $(\mathcal{F}_1,\mathcal{R}_1)$ et $(\mathcal{F}_2,\mathcal{R}_2)$ des systèmes de réécriture disjoints. Tout terme de t appartenant à $\mathcal{T}_{\mathcal{F}_1\cup\mathcal{F}_2}(\mathcal{X})$ peut être décomposé en parties de $\mathcal{F}_1$ et en parties de $\mathcal{F}_2$ (figure 3.1). Pour simplifier, $\mathcal{F}_\oplus$ représente l'alphabet $\mathcal{F}_1 \cup \mathcal{F}_2$ et $\mathcal{T}_\oplus$ l'ensemble des termes sur cet alphabet. De même, on abrégera $\mathcal{T}_{\mathcal{F}_1}$ par $\mathcal{T}_1$ et $\mathcal{T}_{\mathcal{F}_2}$ par $\mathcal{T}_2$.

Soit $t \equiv C[t_1,\cdots,t_n]$ tel que $C[,...,] \not\equiv \Box$.
On écrit $t \equiv C[\![t_1,\cdots,t_n]\!]$ si $C[,\cdots,] \in \mathcal{T}_{\mathcal{F}_a\cup\{\Box\}}(\mathcal{X})$ et si $\forall i \in [1,n]$ racine$(t_i) \in \mathcal{F}_{3-a}$

pour une valeur de a dans $\{1,2\}$. Les termes t_i sont appelés *sous-termes principaux* de t.

Le *rang* d'un terme $t \in \mathcal{T}_\oplus$ est défini par

$$\begin{cases} \text{rang}(t) = 1 & \text{si } t \in \mathcal{T}_1 \cup \mathcal{T}_2 \\ \text{rang}(t) = 1 + \max\{\text{rang}(t_i), 1 \leq i \leq n\} & \text{si } t = C[\![t_1, \cdots, t_n]\!] \end{cases}$$

Remarquons que si $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ alors $\text{rang}(s) \geq \text{rang}(t)$.

Le multi-ensemble $S(t)$ des *sous-termes spéciaux* d'un terme $t \in \mathcal{T}_\oplus$ est défini par

$$\begin{cases} S_1(t) & =< t > \\ S_{n+1}(t) & =< > \text{ si } \text{rang}(t) = 1 \\ & = S_n(t_1) \cup \cdots \cup S_n(t_m) \text{ si } t \equiv C[\![t_1, \cdots, t_m]\!] \end{cases}$$

$S(t) = \bigcup_{i \geq 1} S_i(t)$

La *partie homogène au sommet* d'un terme $t \in \mathcal{T}_\oplus$, notée $\text{top}(t)$, s'obtient en remplaçant les sous-termes principaux de t par $\Box$, i.e.

$$\begin{cases} \text{top}(t) & = t \text{ si } \text{rang}(t) = 1 \\ & = C[,...,] \text{ si } t \equiv C[\![t_1, \cdots, t_n]\!] \end{cases}$$

On définit également top'(t) par:

- si $\text{rang}(t) = 1$ alors top'$(t) = t$.

- sinon, top'(t) est obtenu à partir de $\text{top}(t)$ en remplaçant les p occurrences de $\Box$ dans $\text{top}(t)$ par p variables distinctes $x_1, \cdots, x_p$ de telle sorte que pour tout $i \in [1,p]$, x_i n'apparaît pas dans $\text{top}(t)$.

(top'(t) est défini modulo l'α-conversion).

Le multi-ensemble $P(t)$ des *parties* d'un terme $t \in \mathcal{T}_\oplus$ est défini par

$$P(t) = \{\text{top'}(s) | s \in S(t)\} = \text{top'}(S(t))$$

Le multi-ensemble $E(t)$ des *parties effaçables* d'un terme $t \in \mathcal{T}_\oplus$ est défini par

$$E(t) = P(t) - (\mathcal{T}_{\mathcal{F}_1} \cup \mathcal{T}_{\mathcal{F}_2})$$

i.e. $E(t)$ est le multi-ensemble des termes non clos de $P(t)$.

La figure 3.2 illustre sur un exemple toutes ces définitions.

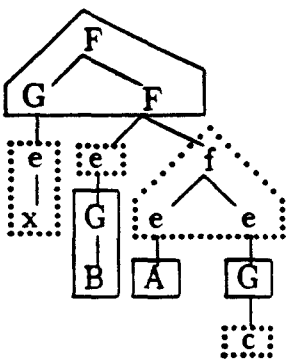
Soit s un terme de $\mathcal{T}_\oplus$ et soient $e_1, \cdots, e_k$ des parties effaçables de s. $\mathcal{R}_1$ et $\mathcal{R}_2$ sont

Soit t le terme :

$\{F, G, A, B\} \subseteq \mathcal{F}_1$

$\{e, f, c\} \subseteq \mathcal{F}_2$

$x \in \mathcal{X}$



- Le rang de t est 4.
- $t \equiv C[\![t_1, t_2, t_3]\!]$ avec $C \equiv F(G(\Box), F(\Box, \Box))$,
$t_1 \equiv e(x)$, $t_2 \equiv e(G(B))$, $t_3 \equiv f(e(A), e(G(c)))$.

- *sous-termes spéciaux de t :*
$S_1(t) =< t >$
$S_2(t) =< e(x),\ e(G(B)),\ f(e(A), e(G(c))) >$
$S_3(t) =< G(B),\ A,\ G(c) >$
$S_4(t) =< c >$
$S(t) = S_1(t) \cup S_2(t) \cup S_3(t) \cup S_4(t)$

- *parties de t et parties effaçables de t :*
$E(t) =< F(G(x), F(y, z)),\ e(x),\ e(x),\ f(e(x), e(y)),\ G(x) >$
$P(t) = E(t) \cup < G(B),\ A,\ c >$

Figure 3.2 : Exemple de terme de $\mathcal{T}_\oplus$

des systèmes linéaires à gauche. On dit qu'on obtient s' à partir de s *en effaçant les parties* $e_1, \cdots, e_k$ s'il existe k variables $x_1, \cdots, x_k$ respectivement dans $e_1, \cdots, e_k$ telles que $(\forall i \in [1,k]\ e_i \xrightarrow{*}_{\mathcal{R}_1} x_i \ \vee \ e_i \xrightarrow{*}_{\mathcal{R}_2} x_i)$ et s' est obtenu à partir de s en réécrivant les e_i en x_i.

Par exemple, si on considère le terme de la figure 3.2, le terme t' construit à partir de t en effaçant la partie $f(e(x), e(y))$ grâce à la règle $f(e(x), e(y)) \xrightarrow{*}_{\mathcal{R}_2} x$ est $t' \equiv F(G(e(x)), F(e(G(B)), A))$.

Nous nous intéressons par la suite à la modularité de la décidabilité du problème d'accessibilité. Ce problème a été défini au premier chapitre. Il est l'équivalent du problème du mot pour les systèmes de réécriture de termes.

3.1.2 Systèmes non linéaires à gauche

Théorème 3.1.1 *La décidabilité de l'accessibilité n'est pas une propriété modulaire des systèmes de réécriture de termes non linéaires à gauche.*

Pour démontrer ce théorème, on construit deux systèmes de réécriture sur des alphabets disjoints. L'accessibilité est décidable pour ces deux systèmes et on prouve que la décision de l'accessibilité pour $\mathcal{R}_1 \oplus \mathcal{R}_2$ se ramène à celle du problème de correspondance de Post.

Le premier système $\mathcal{R}_1$ sur l'alphabet $\mathcal{F}_1 = \{\mathrm{f}(,), \heartsuit\}$, où f est une lettre d'arité 2 et $\heartsuit$ une constante, ne contient qu'une règle donnée figure 3.3. Cette règle n'est pas linéaire à gauche.

Lemme 3.1.2 *L'accessibilité est décidable pour* $(\mathcal{F}_1, \mathcal{R}_1)$.

PREUVE : $(\mathcal{F}_1, \mathcal{R}_1)$ est strictement décroissant, donc l'accessibilité est décidable. □

Nous allons maintenant définir le deuxième système. Soit Σ un alphabet fini gradué dont les lettres ont l'arité 1. Soit $\mathcal{F}_2 = \{\#(), \phi(), \psi(), \$\} \cup \Sigma$, tel que $\{\#(), \phi(), \psi(), \$\}$ et Σ soient des alphabets disjoints.

Soient ϕ et ψ deux morphismes de Σ dans Σ^*. On définit à partir de ces morphismes un ensemble de règles de réécriture $\mathcal{R}_2$, décrit figure 3.4.

Lemme 3.1.3 *L'accessibilité est décidable pour* $(\mathcal{F}_2, \mathcal{R}_2)$.

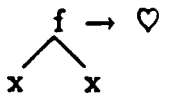


Figure 3.3 : Règle du système $(\mathcal{F}_1, \mathcal{R}_1)$

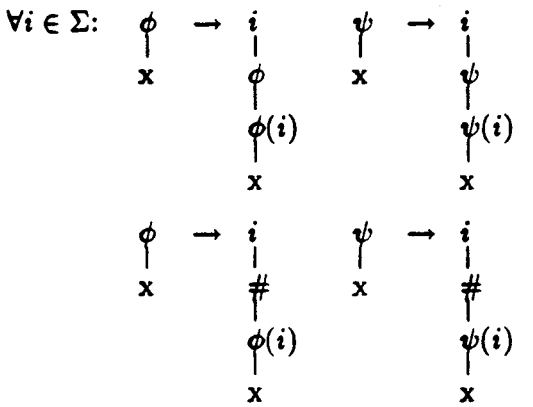
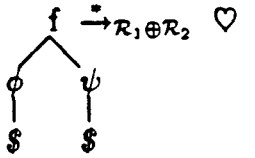


Figure 3.4 : Règles du système $(\mathcal{F}_2, \mathcal{R}_2)$

PREUVE :

On considère le problème d'accessibilité $t \xrightarrow{*}_{\mathcal{R}_2} t'$ pour t et t' termes de $\mathcal{T}_2$. Soit n la hauteur de t'. Il existe un nombre fini de dérivations partant de t et donnant un terme de hauteur n, car Σ est un alphabet fini. Donc l'accessibilité est décidable pour $(\mathcal{F}_2, \mathcal{R}_2)$. □

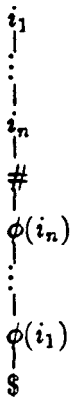
Lemme 3.1.4 *Le problème d'accessibilité* 

est équivalent au problème de correspondance de Post pour ϕ et ψ.

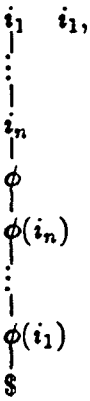
PREUVE :

Il faut démontrer que $\phi(\$)$ et $\psi(\$)$ ont un réduit commun si et seulement si le problème de correspondance de Post est satisfait pour ϕ et ψ.

A partir de $\phi(\$)$ le système peut engendrer tous les termes de la forme



et les termes



$i_1, \cdots, i_n \in \Sigma, n \geq 0$

C'est la même chose pour $\psi(\$)$, donc $\phi(\$)$ et $\psi(\$)$ ont un réduit commun si et seulement s'il existe un mot $m = i_1 \cdots i_n$ tel que $\phi(m) = \psi(m)$. On en déduit donc immédiatement que le problème d'accessibilité du lemme 3.1.4 se ramène au problème de correspondance de Post pour ϕ et ψ. □

PREUVE DU THÉORÈME : Le problème de correspondance de Post est indécidable. Donc, d'après le lemme précédent, l'accessibilité est indécidable pour $\mathcal{R}_1 \oplus \mathcal{R}_2$. Cependant, elle était décidable pour $(\mathcal{F}_1, \mathcal{R}_1)$ et pour $(\mathcal{F}_2, \mathcal{R}_2)$. Donc la décidabilité du problème de l'accessibilité n'est pas une propriété modulaire des systèmes de réécriture de termes. □

Nous pouvons remarquer que, dans la preuve que nous donnons, les systèmes ne contiennent pas de règles effaçantes et sont tous les deux linéaires à droite.

3.1.3 Systèmes linéaires à gauche, sans règles effaçantes

Une règle $l \to r$ est *effaçante* si r est une variable.

Définition 3.1.5 *Supposons que $s \to t$ par l'application de la règle de réécriture $l \to r$. On écrit $s \to^{i} t$ si $l \to r$ est appliquée sur l'un des sous-termes principaux de s et on écrit $s \to^{o} t$ sinon. La relation $\to^{i}$ est appelée réduction* interne *et $\to^{o}$ est appelée réduction* externe.

Proposition 3.1.6 (Middeldorp) *Soient $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ des systèmes de réécriture de termes disjoints, sans règles effaçantes. Pour toute séquence de réduction $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ il existe un terme s' tel que $s \xrightarrow{*}{}^{o} s' \xrightarrow{*}{}^{i} t$.*

Théorème 3.1.7 *La décidabilité de l'accessibilité est une propriété modulaire des systèmes de réécriture sans règles effaçantes.*

PREUVE : Soient $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ deux systèmes de réécriture linéaires à gauche, sans règle effaçante. On suppose que l'accessibilité est décidable pour $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$.

1. Tout d'abord, nous démontrons que si $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ alors $s \equiv C[\![s_1, \cdots, s_n]\!]$ et $t \equiv C'[\![t_1, \cdots, t_p]\!]$ pour des valeurs positives ou nulles de n et p, et pour des contextes C et C' sur le même alphabet $F_a, (a \in \{1,2\})$.

   Si $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ alors il existe s' tel que $s \xrightarrow{*}{}^{\circ} s' \xrightarrow{*}{}^{\mathrm{i}} t$ (proposition 3.1.6).

   Donc $s \equiv C[\![s_1, \cdots, s_n]\!] \xrightarrow{*}{}^{\circ} s' \equiv C'[\![s_{i_1}, \cdots, s_{i_p}]\!] \xrightarrow{*}{}^{\mathrm{i}} t \equiv C'[\![t_1, \cdots, t_p]\!]$, où $\{s_{i_1}, \cdots, s_{i_p}\} \subset \{s_1, \cdots, s_n\}$.

   Donc C et C' sont des contextes sur le même alphabet $\mathcal{F}_a$ et il existe n nouvelles variables $x_1, \cdots, x_n$ telles que $C[x_1, \cdots, x_n] \xrightarrow{*}_{\mathcal{R}_a} C'[x_{i_1}, \cdots, x_{i_p}]$ et telles que $s_{i_j} \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t_j \ \forall j \in [1,p]$. C' est un contexte différent de $\Box$ car $\mathcal{R}_a$ ne contient pas de règles effaçantes.

2. Nous prouvons par récurrence sur rang(s) que le problème $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ est décidable.

   a) Supposons que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ avec rang$(s) = 1$. L'accessibilité est décidable puisque rang$(s) =$ rang$(t) = 1$ et elle est décidable pour $(\mathcal{F}_1, \mathcal{R}_1)$ ou $(\mathcal{F}_2, \mathcal{R}_2)$.

   b) Supposons que, pour k fixé, l'accessibilité est décidable à partir de s tel que rang$(s) < k$. Nous prouvons qu'elle reste décidable pour rang$(s) = k$. Le problème est donc de répondre à la question $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$? avec $s \equiv C[\![s_1, \cdots, s_n]\!]$ et $t \equiv C'[\![t_1, \cdots, t_p]\!]$

   - Si C et C' sont des contextes sur des alphabets différents, la réponse est NON.

   - Sinon, $C, C' \in \mathcal{T}(\mathcal{F}_a \cup \{\Box\}, \mathcal{X})$ pour un $a \in \{1,2\}$.

   $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t \Leftrightarrow \exists s' \ s \xrightarrow{*}{}^{\circ} s' \xrightarrow{*}{}^{\mathrm{i}} t$ avec $s' \equiv C'[\![s_{i_1}, \cdots, s_{i_p}]\!]$ pour $\{s_{i_1}, \cdots, s_{i_p}\} \subset \{s_1, \cdots, s_n\}$. Soit $\{x_1, \cdots, x_n\}$ un ensemble de nouvelles variables distinctes. Pour trouver le terme s', on choisit p variables non nécessairement distinctes dans l'ensemble $\{x_1, \cdots, x_n\}$. Il y a un nombre fini de cas à considérer. On dit que $< x_{i_1}, \cdots, x_{i_p} >$ est un multi-ensemble solution si $C[x_1, \cdots, x_n] \xrightarrow{*}_{\mathcal{R}_a} C'[x_{i_1}, \cdots, x_{i_p}]$.

- **Simplifie:** $(\{s \stackrel{?}{\rightarrow} s\} \cup P; A) \Rightarrow (P; T)$
- **Vérifie:** $(\{C[\![s_1, \cdots, s_n]\!] \stackrel{?}{\rightarrow} C'[\![t_1, \cdots, t_p]\!]\} \cup P; A) \Rightarrow (\emptyset; F)$
  Si C est un contexte sur $\mathcal{F}_a$ et C' sur $\mathcal{F}_{3-a}$, $a \in \{1,2\}$.
- **Echoue:** $(\{C[\![s_1, \cdots, s_n]\!] \stackrel{?}{\rightarrow} C'[\![t_1, \cdots, t_p]\!]\} \cup P; A) \Rightarrow (\emptyset; F)$
  Si C et C' sont des contextes sur $\mathcal{F}_a$ et s'il n'existe pas de multi-ensemble $< x_{i_1}, \cdots, x_{i_p} >, i_j \in [1,n]$ tel que $C[x_1, \cdots, x_n] \stackrel{*}{\rightarrow}_{\mathcal{R}_a} C'[x_{i_1}, \cdots, x_{i_p}]$
- **Décompose:** $(\{C[\![s_1, \cdots, s_n]\!] \stackrel{?}{\rightarrow} C'[\![t_1, \cdots, t_p]\!]\} \cup P; A)$
  $\Rightarrow (\{s_{i_1} \stackrel{?}{\rightarrow} t_1\} \cup \cdots \cup \{s_{i_p} \stackrel{?}{\rightarrow} t_p\} \cup P; T)$
  Si C et C' sont des contextes sur $\mathcal{F}_a$ et si $< x_{i_1}, \cdots, x_{i_p} >, i_j \in [1,n]$ est un multi-ensemble tel que $C[x_1, \cdots, x_n] \stackrel{*}{\rightarrow}_{\mathcal{R}_a} C'[x_{i_1}, \cdots, x_{i_p}]$

Figure 3.5 : règles d'inférence

Nous obtenons finalement que $s \stackrel{*}{\rightarrow}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ si et seulement s'il existe un multi-ensemble solution $< x_{i_1}, \cdots, x_{i_p} >$ tel que $s_{i_j} \stackrel{*}{\rightarrow}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t_j \ \forall j \in [1,p]$. Comme $\text{rang}(s_{i_j}) < \text{rang}(s)$ pour tous j, les problèmes d'accessibilité $s_{i_j} \stackrel{*}{\rightarrow}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t_j$ sont décidables, d'après l'hypothèse de récurrence. □

On définit un ensemble de règles d'inférence qui agissent sur des paires $(P; A)$. P est un ensemble de problèmes d'accessibilités et A est une réponse temporaire ($A \in \{T, F\}$). Ces règles sont données figure 3.5. Cet ensemble de règles est tel que $s \stackrel{*}{\rightarrow}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ si et seulement si, partant de $(\{s \stackrel{?}{\rightarrow} t\}; T)$ et appliquant les règles tant que cela est possible, il existe une séquence de dérivation qui aboutit à $(\emptyset; T)$. Il faut remarquer que, quelque soient s et t, il existe un nombre fini de dérivations partant de $(\{s \stackrel{?}{\rightarrow} t\}; T)$.

3.1.4 Systèmes linéaires avec règles effaçantes

Théorème 3.1.8 *La décidabilité de l'accessibilité est une propriété modulaire des systèmes de réécriture linéaires, même s'ils contiennent des règles effaçantes.*

Pour la démonstration de ce théorème, nous avons besoin de quatre lemmes.

Lemme 3.1.9 *Soient $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ deux systèmes de réécriture linéaires à gauche. Supposons que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ sans effacer de parties. Alors, il existe un terme s' tel que $s \xrightarrow{*}{}^{0} s' \xrightarrow{*}{}^{1} t$*

PREUVE : C'est une simple extension de la proposition 3.1.6. □

Le lemme suivant permet de réécrire les parties d'un terme indépendamment les unes des autres, à condition de ne pas effacer de partie.

Lemme 3.1.10 *Soient $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ deux systèmes de réécriture disjoints et linéaires. Soit s un terme comprenant n parties, numérotées arbitrairement. Si $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ sans effacer de partie alors il existe $s_1, \cdots, s_{n-1}$ tels que :*

$s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_1$ en ne réécrivant que la partie 1.

$s_1 \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_2$ en ne réécrivant que la partie 2.

...

$s_{n-1} \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ en ne réécrivant que la partie n.

PREUVE :

Quand on réécrit une partie p, on peut détruire d'autres parties en dessous de p, mais aucune partie ne peut être dupliquée car $\mathcal{R}_1 \oplus \mathcal{R}_2$ est linéaire. $N(t)$ est l'ensemble des numéros de toutes les parties de t. Quand une partie p est réécrite en p' alors on suppose que (le numéro de p) = (le numéro de p'). Ce qui signifie que le pas de réduction $t \rightarrow_{\mathcal{R}_1 \oplus \mathcal{R}_2} t'$ transforme l'ensemble $N(t)$ en $N(t')$ tel que $N(t') \subset N(t)$.

Nous prouvons maintenant que les parties de t peuvent être réécrites dans n'importe quel ordre.

– Considérons deux parties p_1 et p_2 telles que p_1 est sous-terme de p_2. Pour réécrire p_1 et p_2, on peut soit réécrire p_1 et p_2 simultanément, ou ne réécrire que p_1 puis ne réécrire que p_2, soit ne réécrire que p_2 puis réécrire p_1 s'il n'a pas été détruit pendant la réduction de p_2. Si p_1 est effacé, on dira que l'on réécrit p_1 par une réduction vide.

–Nous considérons deux parties p_1 et p_2 telles que p_1 n'est pas sous-terme de p_2 et p_2 n'est pas non plus sous-terme de p_1. Il est évident que dans ce cas, on peut réécrire p_1 et p_2 de manière indépendante. □

Le lemme suivant signifie que, si l'on efface une et une seule partie dans une séquence de réduction, alors on peut commencer la réduction par la réécriture de cette partie effaçable.

Lemme 3.1.11 *Soient* $(\mathcal{F}_1, \mathcal{R}_1)$ *et* $(\mathcal{F}_2, \mathcal{R}_2)$ *des systèmes de réécritures disjoints et linéaires. Si* $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ *est une séquence de réduction qui efface exactement une seule partie* $e \in E(s)$, *alors il existe* s' *tel que* $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ *avec* $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'$ *ne réécrivant que* e *et l'effaçant et* $s' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ *n'effaçant aucune partie.*

PREUVE :

On suppose que s est un terme comprenant n parties numérotées arbitrairement. On suppose aussi que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ en effaçant la partie portant le numéro j, $j \in [1, n]$.

Cela signifie qu'il existe deux termes s', s'' tels que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s' \rightarrow_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ tels que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'$ sans effacer de parties, $s' \rightarrow_{\mathcal{R}_1 \oplus \mathcal{R}_2} s''$ en effaçant la partie j et $s'' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ sans effacer de parties.

D'après le lemme 3.1.10, on peut réécrire s en s' en terminant par les pas de réduction appliqués sur j. Donc il existe s''' tel que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s''' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'$ avec $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'''$ en réécrivant tout sauf la partie j, et $s''' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'$ en ne réécrivant que la partie j.

On obtient ainsi $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s''' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s' \rightarrow_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$.

Les réductions faites dans $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'''$ peuvent être faites après avoir réécrit la partie j : chaque pas $a \rightarrow b$ sur une partie k dans la réduction $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'''$ peut soit être faite après avoir réécrit j si la partie k n'est pas détruite par la réécriture de j, soit être supprimée si la partie k est détruite lors de la réécriture de la partie j.

On peut donc en déduire qu'il existe des termes t', t'' tels que

$$s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t' \rightarrow_{\mathcal{R}_1 \oplus \mathcal{R}_2} t'' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t,$$

avec $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t'$ réécrivant uniquement la partie j, $t' \rightarrow_{\mathcal{R}_1 \oplus \mathcal{R}_2} t''$ effaçant la partie j et $t'' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ n'effaçant aucune partie. □

Le dernier lemme signifie que, si un terme se réécrit en un autre en effaçant des parties, alors on peut commencer la réduction par la réduction et l'effacement de ces parties puis réécrire les parties restantes.

Lemme 3.1.12 *Soient $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ deux systèmes de réécriture linéaires et disjoints. Si $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ en effaçant k parties $e_1, \cdots, e_k$ dans cet ordre, alors il existe k termes $s_1, \cdots, s_k$ tels que*

$s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_1 \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_2 \cdots \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_k \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ *avec*

$s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_1$ *réécrivant et effaçant* e_1

$s_1 \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_2$ *réécrivant et effaçant* e_2

...

$s_k \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ *sans supprimer de partie effaçable.*

PREUVE :

On démontre ce lemme par récurrence sur $k \geq 0$.

• Si $k = 0$ le lemme est bien-sûr vrai..

• Supposons que le lemme est vrai pour k fixé, positif ou nul. Soient s et t deux termes tels que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ en effaçant $k+1$ parties. Alors il existe un terme s_1 tel que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_1 \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ où $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_1$ est une séquence de réduction qui efface une seule partie e. On peut en déduire, grâce au lemme 3.1.11, qu'il existe s'_1 tel que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'_1 \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_1$ où $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'_1$ réécrit et efface la partie e.

Donc $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'_1 \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_1 \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ où $s'_1 \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ est une réduction qui efface k parties. Les hypothèses de récurrence nous permettent de conclure. □

PREUVE DU THÉORÈME 3.1.8 :

Soient $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ deux systèmes de réécriture linéaires disjoints. Nous supposons que l'accessibilité est décidable pour $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ et nous démontrons qu'elle reste décidable pour $\mathcal{R}_1 \oplus \mathcal{R}_2$. Soient s et t deux termes. On veut résoudre le problème d'accessibilité $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$.

Si $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$, soit on ne détruit pas de partie effaçable, soit on en efface.

• Premier cas: On cherche à résoudre le problème sans effacer de partie.

D'après le lemme 3.1.9, $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ si et seulement s'il existe un terme s' tel que $s \xrightarrow{*}^{\circ} s' \xrightarrow{*}^{i} t$ On démontre que l'accessibilité est décidable, de la même façon que pour le théorème 3.1.7.

• Second cas: On résoud le problème $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ en effaçant au moins une partie. On

prouve par récurrence sur le nombre de parties de s que le problème est décidable.

- Si s ne contient qu'une seule partie alors $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ en effaçant s si et seulement si t est une variable qui apparaît dans s. Dans ce cas, $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ est un problème d'accessibilité sur $(\mathcal{F}_1, \mathcal{R}_1)$ ou sur $(\mathcal{F}_2, \mathcal{R}_2)$. Il est donc décidable.

- Soit NP(s) le nombre de parties de s.

On suppose que $u \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} v$ est décidable si NP(u) $\leq p$, p étant une valeur fixée. On résoud le problème $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ en effaçant au moins une partie, avec NP(s) = $p+1$. Cela signifie qu'il existe un terme s' et un entier $k \in [1, n]$ où n est le nombre de parties effaçables de s, tels que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'$ en effaçant k parties et $s' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ avec NP(s') $\leq p$. Donc il existe $e_1, \cdots, e_k$ dans $E(s)$ telles que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s'$ en effaçant $e_1, \cdots, e_k$ dans cet ordre. D'après le lemme 3.1.12, il existe $s_1, \cdots, s_k$ tels que $s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_1 \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_2 \cdots \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_k$ où

$s \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_1$ réécrivant et effaçant e_1

$s_1 \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_2$ réécrivant et effaçant e_2

...

$s_{k-1} \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} s_k$ réécrivant et effaçant e_k

On choisit $s' \equiv s_k$.

Il y a un nombre fini de cas à considérer, i.e. un nombre fini de k et pour chaque k un nombre fini de k-uplets $(e_1, \cdots, e_k)$. Pour chacun des cas, on peut décider si chaque e_i peut être effacé car il s'agit de problèmes d'accessibilité sur $(\mathcal{F}_1, \mathcal{R}_1)$ ou sur $(\mathcal{F}_2, \mathcal{R}_2)$. Si tous les e_i sont effaçables, on obtient le terme s_k et on doit alors résoudre le problème $s_k \equiv s' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ avec NP(s') $\leq p$. D'après l'hypothèse de récurrence, le problème $s' \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ est décidable. □

L'ensemble des règles d'inférence de la figure 3.6 fournit une procédure de décision pour l'accessibilité sur $\mathcal{R}_1 \oplus \mathcal{R}_2$, si on suppose que $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ sont des systèmes de réécriture disjoints et linéaires. ces règles d'inférence agissent sur des quadruplets (P_1;P_2;P_3;A). P_1 contient un problème d'accessibilité. P_2 est un ensemble de problèmes d'accessibilité de la forme $s \xrightarrow{?} t$ qui sont à résoudre sans effacer de parties. P_3 est un ensemble de problèmes d'effacement, i.e. de problèmes d'accessibilité de la forme $s \xrightarrow{?} x$ où x est une variable apparaissant dans s. A est une réponse temporaire au problème P_1 ($A \in \{T, F\}$).

- Avec la règle 1, on choisit de résoudre le problème sans effacement.

1. $(\{s \xrightarrow{?} t\}; \emptyset; \emptyset; A) \Rightarrow (\emptyset; \{s \xrightarrow{?} t\}; \emptyset; A)$

2. $(\{s \xrightarrow{?} t\}; \emptyset; \emptyset; A) \Rightarrow (\{s' \xrightarrow{?} t\}; \emptyset; \{s_1[x_1] \xrightarrow{?} x_1\} \cup \cdots \cup \{s_k[x_k] \xrightarrow{?} x_k\}; A)$
   Si s' est obtenu à partir de s en effaçant $s_1[x_1], \cdots, s_k[x_k]$
   avec $1 \leq k \leq \text{Card}(E(s))$.

3. $(\emptyset; \{s \xrightarrow{?} s\} \cup P_2; P_3; A) \Rightarrow (\emptyset; P_2; P_3; T)$

4. $(\emptyset; \{C[\![s_1, \cdots, s_n]\!] \xrightarrow{?} C'[\![t_1, \cdots, t_p]\!]\} \cup P_2; P_3; A) \Rightarrow (\emptyset; \emptyset; \emptyset; F)$
   Si C contexte sur $\mathcal{F}_a$ et C' contexte sur $\mathcal{F}_{3-a}$, $a \in \{1,2\}$.

5. $(\emptyset; \{C[\![s_1, \cdots, s_n]\!] \xrightarrow{?} C'[\![t_1, \cdots, t_p]\!]\} \cup P_2; P_3; A) \Rightarrow (\emptyset; \emptyset; \emptyset; F)$
   Si C et C' sont des contextes sur $\mathcal{F}_a$ et s'il n'existe pas de multi-ensemble $< x_{i_1}, \cdots, x_{i_p} > i_j \in [1,n]$ tel que $C[x_1, \cdots, x_n] \xrightarrow{*}_{\mathcal{R}_a} C'[x_{i_1}, \cdots, x_{i_p}]$

6. $(\emptyset; \{C[\![s_1, \cdots, s_n]\!] \xrightarrow{?} C'[\![t_1, \cdots, t_p]\!]\} \cup P_2; P_3; A)$
   $\Rightarrow (\emptyset; \{s_{i_1} \xrightarrow{?} t_1\} \cup \cdots \cup \{s_{i_p} \xrightarrow{?} t_p\} \cup P_2; P_3; T)$
   Si C et C' sont des contextes sur $\mathcal{F}_a$ et si $< x_{i_1}, \cdots, x_{i_p} >, i_j \in [1,n]$ est un multi-ensemble tel que $C[x_1, \cdots, x_n] \xrightarrow{*}_{\mathcal{R}_a} C'[x_{i_1}, \cdots, x_{i_p}]$

7. $(P_1; P_2; \{s[x] \xrightarrow{?} x\} \cup P_3; A) \Rightarrow (P_1; P_2; P_3; T)$
   Si $s[x]$ est un terme de $\mathcal{T}_a$ et $s[x] \xrightarrow{*}_{\mathcal{R}_a} x$

8. $(P_1; P_2; \{s[x] \xrightarrow{?} x\} \cup P_3; A) \Rightarrow (\emptyset; \emptyset; \emptyset; F)$
   Si $s[x]$ est un terme de $\mathcal{T}_a$ et on n'a pas $s[x] \xrightarrow{*}_{\mathcal{R}_a} x$

Figure 3.6 : règles d'inférence

- Avec la règle 2, on efface k parties et on obtient le terme s'. On doit donc résoudre k problèmes d'effacement,et un nouveau problème d'accessibilité $s' \stackrel{?}{\rightarrow} t$.

- Les règles 3, 4, 5, 6 permettent de résoudre le problème sans effacer de parties. Ces règles sont les mêmes que celles données au paragraphe précédent (figure 3.5).

- Les règles 7 et 8 permettent de résoudre les problèmes d'effacement.

Cet ensemble de règles est tel que $s \stackrel{*}{\rightarrow}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t$ si et seulement s'il existe une séquence d'inférences partant de $(\{s \stackrel{?}{\rightarrow} t\}; \emptyset; \emptyset; T)$ et appliquant les règles tant que cela est possible, qui aboutit au résultat $(\emptyset, \emptyset; \emptyset; T)$. Bien entendu, il n'existe qu'un nombre fini de séquences d'inférences partant de $(\{s \stackrel{?}{\rightarrow} t\}; \emptyset; \emptyset; T)$.

Dans le cas de systèmes de réécriture de termes, il est clair que nos algorithmes ont une complexité exponentielle, même dans le cas de systèmes linéaires à gauche sans règles effaçantes, et en considérant que la résolution de l'accessibilité pour un seul système se fait en temps constant. En effet, quand on doit résoudre un problème d'accessibilité de la forme $C[\![s_1, \cdots, s_n]\!] \stackrel{*}{\rightarrow}_{\mathcal{R}_1 \oplus \mathcal{R}_2} C'[\![t_1, \cdots, t_p]\!]$, on doit trouver les multi-ensembles $< x_{i_1}, \cdots, x_{i_p} >$ tels que $C[\![x_1, \cdots, x_n]\!] \rightarrow_{\mathcal{R}_a} C'[\![x_{i_1}, \cdots, x_{i_p}]\!]$ (si C et C' sont des contextes sur le même alphabet $\mathcal{F}_a$). On obtient ainsi n^p cas possibles, et donc n^p problèmes d'accessibilité sur $(\mathcal{F}_a, \mathcal{R}_a)$ à examiner. De plus, pour chaque multi-ensemble solution, on doit traiter p nouveaux problèmes d'accessibilité sur $\mathcal{R}_1 \oplus \mathcal{R}_2$: $s_{i_j} \stackrel{*}{\rightarrow}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t_j$, $j \in [1, p]$.

Etudions plutôt la complexité de notre algorithme dans le cas particulier de systèmes de réécriture de mots. Considérons deux systèmes $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ dont les règles ont la forme $m_1 \rightarrow m_2$ où m_1 et m_2 sont des mots. Le symbole ϵ représente le mot vide. Nous calculons le nombre d'appels à la procédure qui permet de résoudre les problèmes d'accessibilité sur $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$. C'est pourquoi la complexité que nous obtenons est à multiplier par la complexité de cette procédure (c'est une complexité relative).

PREMIER CAS : S'il n'y a pas de règles effaçantes, i.e. de règles $m_1 \rightarrow \epsilon$, l'algorithme de décision du problème $m \stackrel{*}{\rightarrow}_{\mathcal{R}_1 \oplus \mathcal{R}_2} m'$ a une complexité linéaire en fonction du rang de m. En effet :

- si $m \equiv \epsilon$ alors $m \stackrel{*}{\rightarrow}_{\mathcal{R}_1 \oplus \mathcal{R}_2} m' \Leftrightarrow m' \equiv \epsilon$

- supposons que m commence par une lettre de $\mathcal{F}_a$, $a \in \{1, 2\}$. Alors m s'écrit $m \equiv u_1 u_2 \cdots u_n$, $n \geq 1$, avec u_i mot de $\mathcal{F}_a^*$ si i impair et u_i mot de $\mathcal{F}_{3-a}^*$ sinon.

- Si m' commence par une lettre de $\mathcal{F}_{3-a}$ alors la réponse au problème d'accessibilité est NON.

- Sinon, $m' \equiv v_1 v_2 \cdots v_p$, $p \geq 0$, avec v_i mot de $\mathcal{F}_a^*$ si i est impair et v_i mot de $\mathcal{F}_{3-a}^*$ sinon. On a donc ($m \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} m' \Leftrightarrow (p = n)$ et $u_i \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} v_i \ \forall i \in \{1, \ldots, n\}$). Chaque problème d'accessibilité $u_i \rightarrow v_i$ est un problème sur $(\mathcal{F}_1, \mathcal{R}_1)$ or $(\mathcal{F}_2, \mathcal{R}_2)$. Il y a donc n problèmes d'accessibilité "simples" à résoudre, où n est le rang de m. Donc la complexité de notre algorithme dépend linéairement du rang de m.

SECOND CAS : S'il y a des règles effaçantes, la complexité devient exponentielle.

- si $m \equiv \epsilon$, alors $m \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} m' \Leftrightarrow m' \equiv \epsilon$

- supposons que $m \equiv u_1 u_2 \cdots u_n$, $n \geq 1$ où u_i est un mot de $\mathcal{F}_a^*$ quand i est impair et u_i mot de $\mathcal{F}_{3-a}^*$ sinon ($a \in \{1,2\}$). Posons également que $m' \equiv v_1 v_2 \cdots v_p$, $p \geq 0$, avec v_i mot de $\mathcal{F}_b^*$ si i est impair et v_i mot de $\mathcal{F}_{3-b}^*$ sinon, pour $b \in \{1,2\}$.

  - si $n = p$, nous nous plaçons dans le cas précédent, donc la complexité est linéaire en fonction du rang de m.

  - si $n < p$ alors la réponse au problème est immédiatement NON.

  - si $n > p$ alors on doit choisir $n - p$ mots parmi n. L'ordre des mots est important (à cause de notre algorithme qui est prévu pour des arbres) et les mots ne sont pas forcément distincts. Il y a donc n^{n-p} cas à considérer. Pour chaque cas, il y a n problèmes d'accessibilité "simples" à résoudre. En effet, il y a $n - p$ problèmes de la forme $u_i \rightarrow \epsilon$ et p problèmes de la forme $u_i \rightarrow v_j$. C'est pourquoi, dans les mots, la complexité de notre algorithme permettant de résoudre $m \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} m'$ est $\mathcal{O}(n^{n-p+1})$ si on suppose que la complexité de la procédure de résolution de l'accessibilité pour $(\mathcal{F}_1, \mathcal{R}_1)$ et $(\mathcal{F}_2, \mathcal{R}_2)$ est constante. La complexité de notre algorithme de résolution de $m \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} m'$ est donc exponentielle en fonction des rangs de m et m'.

3.1.5 Systèmes linéaires à gauche avec règles effaçantes

Dans [Car92], on conjecturait que la décidabilité de l'accessibilité n'était plus modulaire si les systèmes étaient linéaires à gauche et non-linéaires à droite, avec des règles effaçantes. Dans ce paragraphe, nous donnons une preuve de cette conjecture.

Théorème 3.1.13 *La décidabilité de l'accessibilité n'est pas préservée par l'union disjointe de deux systèmes de réécriture linéaires à gauche contenant des règles effaçantes.*

Les deux systèmes de réécriture suivants permettent de prouver le théorème.

On considère d'abord un système $\mathcal{R}_1$ sur l'alphabet $\mathcal{F}_1 = \{g(,)\}$. Il ne contient que deux règles : $g(x,y) \to x$ and $g(x,y) \to y$

Lemme 3.1.14 *L'accessibilité est décidable pour* $(\mathcal{F}_1, \mathcal{R}_1)$.

PREUVE : $(\mathcal{F}_1, \mathcal{R}_1)$ est strictement décroissant, donc l'accessibilité est décidable. □

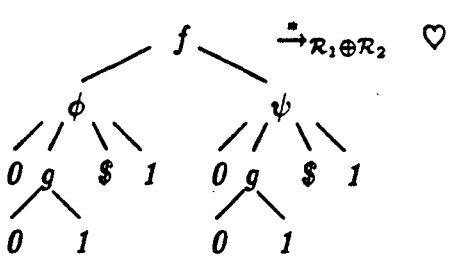
Nous considérons maintenant un alphabet fini gradué Σ dont chaque lettre a l'arité 1. Soit $\mathcal{F}_2 = \{f(,), \#(), \phi(), \psi(), \$\} \cup \Sigma$, tel que $\{f(,), \#(), \phi(), \psi(), \$\}$ et Σ sont des alphabets disjoints.

Soient ϕ et ψ deux morphismes de Σ dans Σ^*. $(\mathcal{F}_2, \mathcal{R}_2)$ est un système de réécriture linéaire à gauche associé aux morphismes ϕ et ψ. $\mathcal{R}_2$ est l'ensemble des règles données par la figure 3.7.

Lemme 3.1.15 *L'accessibilité est décidable pour* $(\mathcal{F}_2, \mathcal{R}_2)$.

PREUVE : $(\mathcal{F}_2, \mathcal{R}_2)$ est noethérien donc l'accessibilité est décidable. □

Lemme 3.1.16 *Le problème d'accessibilité*



est équivalent au problème de correspondance de Post pour ϕ *et* ψ.

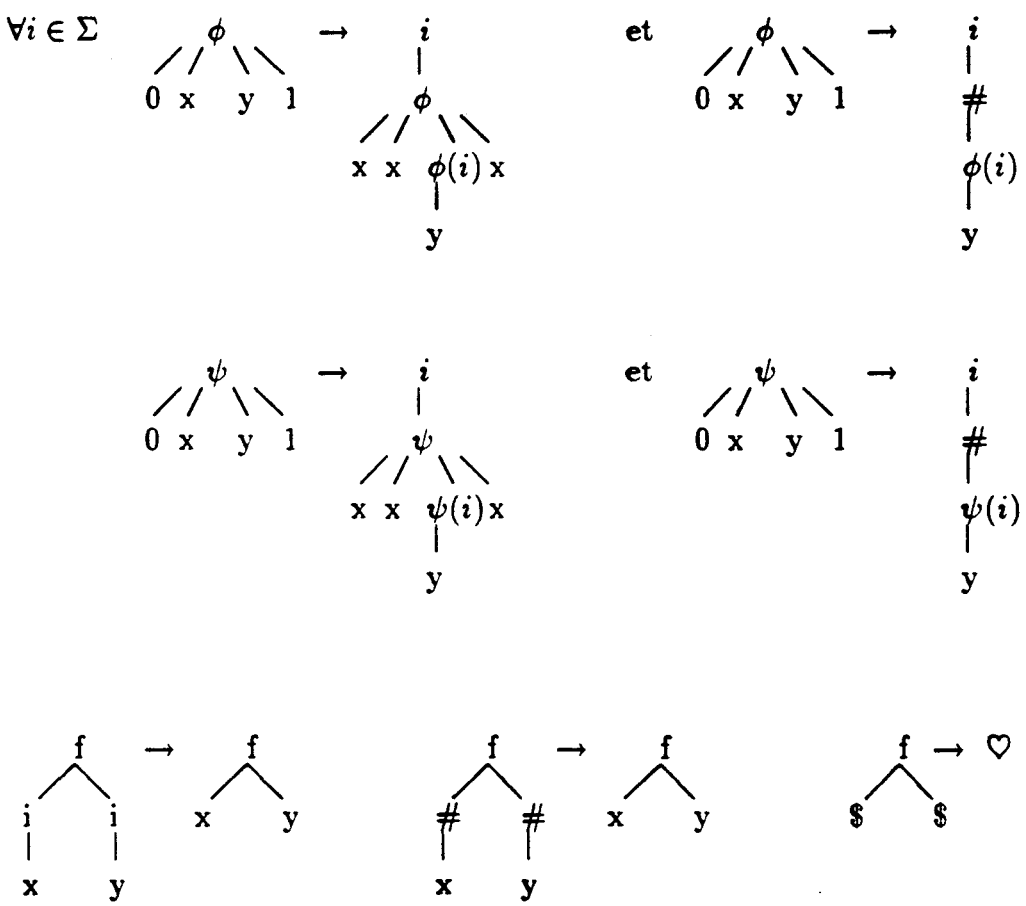


Figure 3.7 : règles du système $(\mathcal{F}_2, \mathcal{R}_2)$

PREUVE: Identique à celle de la section 3.1.2. □

PREUVE DU THÉORÈME 3.1.13:

Le problème de correspondance de Post est indécidable. C'est pourquoi le lemme précédent nous permet d'affirmer que l'accessibilité est indécidable pour $\mathcal{R}_1 \oplus \mathcal{R}_2$. Mais elle était décidable pour $(\mathcal{F}_1, \mathcal{R}_1)$ et pour $(\mathcal{F}_2, \mathcal{R}_2)$. Donc la décidabilité de l'accessibilité n'est pas une propriété modulaire des systèmes de réécriture linéaires à gauche pouvant contenir des règles effaçantes. □

3.1.6 Accessibilité close

Théorème 3.1.17 *L'accessibilité close n'est pas une propriété modulaire des systèmes de réécriture de termes linéaires sans règles effaçantes.*

Pour démontrer ce théorème, nous construisons des systèmes particuliers pour lesquels l'accessibilité close est décidable et nous prouvons que, pour leur union disjointe, la décidabilité de l'accessibilité est équivalente à celle du problème de correspondance de Post.

Soit $\mathcal{F}_1 = \{\alpha(),\ 0\}$ (α est une lettre d'arité 1 et 0 est une constante). Nous considérons un premier système $(\mathcal{F}_1, \mathcal{R}_1)$. $\mathcal{R}_1$ ne contient aucune règle donc il est évident que l'accessibilité est décidable pour $(\mathcal{F}_1, \mathcal{R}_1)$.

Le deuxième système est plus compliqué. Soit $\mathcal{F}_2 = \{c(,,),\ c'(,,),\ c''(,,),\ a(),\ a'(),\ a''(),\ \#\}$ (c, c' et c'' sont des lettres d'arité 3 , a, a' et a'', d'arité 1 et # est une constante). $\mathcal{F}_2$ et $\mathcal{F}_1$ sont disjoints. $\mathcal{G}_1$ est l'ensemble des termes clos sur $\mathcal{F}_1$ et $\mathcal{G}_2$ celui des termes clos sur l'alphabet $\mathcal{F}_2$. Soient ϕ et ψ deux morphismes de I dans $\{a, a'\}^*$. On définit $(\mathcal{F}_2, \mathcal{R}_2)$ en fonction de ces morphismes.

- $\mathcal{R}_2$ contient des règles qui permettent de réduire tout terme clos de $\mathcal{G}_2$ en # (règles 1 à 6 de la figure 3.8).
- $\mathcal{R}_2$ contient également des règles qui permettent de réécrire # en n'importe quel terme clos (règles 7 à 12 de la figure 3.8).

  On a donc $t \xrightarrow{*}_{\mathcal{R}_2} \# \xrightarrow{*}_{\mathcal{R}_2} t'$ quelque soient les termes clos t et t' de $\mathcal{G}_2$.

  Donc l'accessibilité close est décidable pour $(\mathcal{F}_2, \mathcal{R}_2)$.
- Enfin, $\mathcal{R}_2$ contient toutes les règles associées au problème de correspondance de Post pour ϕ et ψ (règles 13 à 18 figure 3.9).

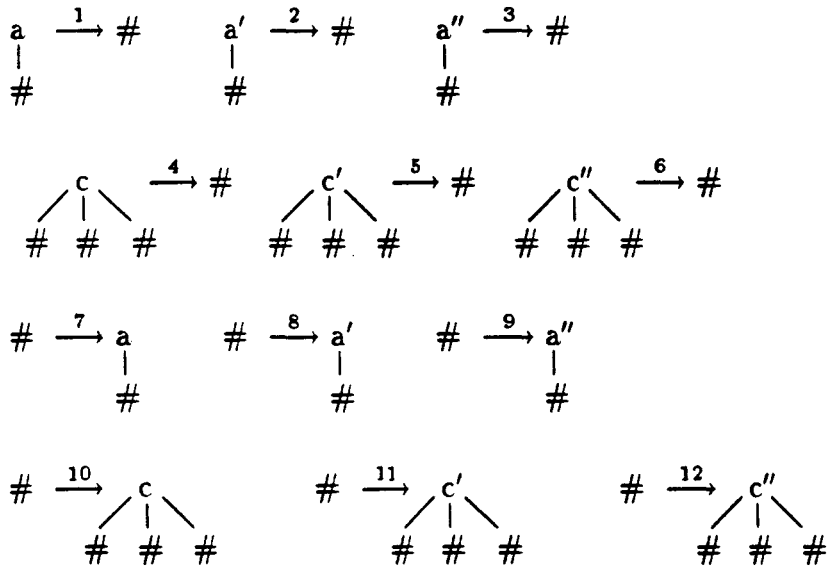


Figure 3.8 : règles de $(F_2, \mathcal{R}_2)$, 1ère partie

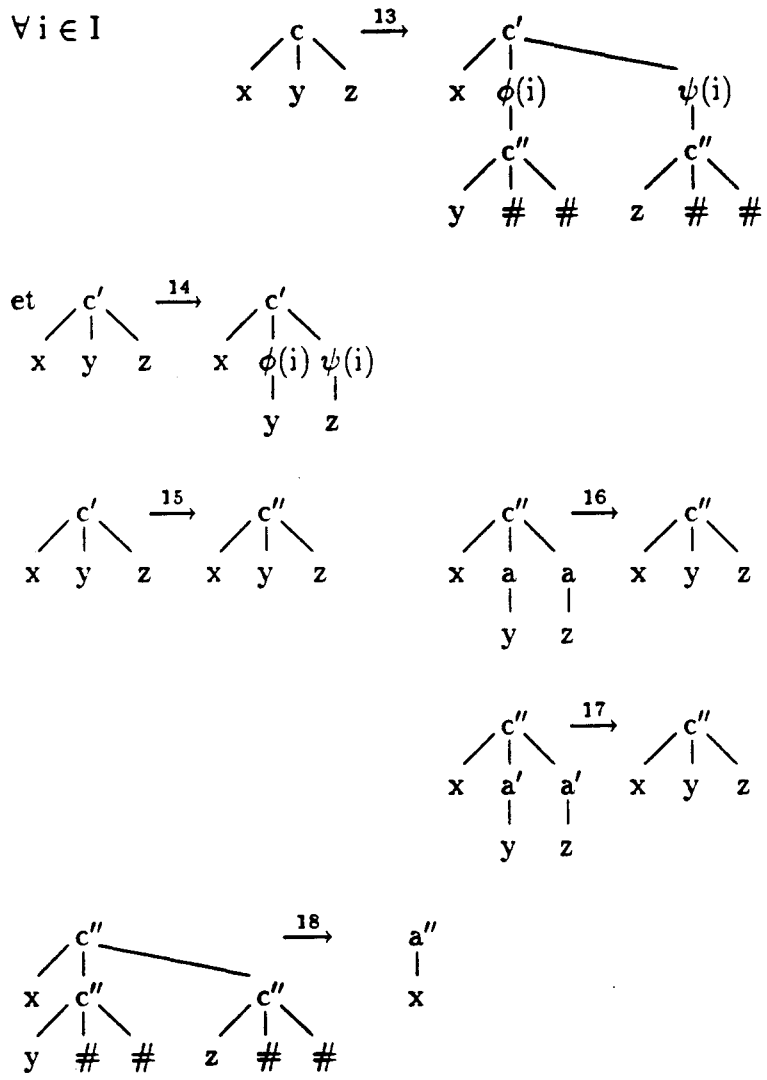
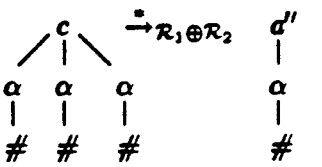


Figure 3.9 : règles de $(F_2, \mathcal{R}_2)$, 2ème partie

Toutes les règles de $\mathcal{R}_2$ sont linéaires, et non-effaçantes.

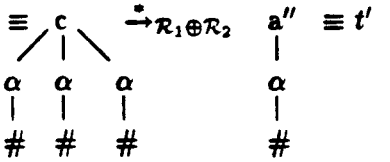
Lemme 3.1.18 *Le problème d'accessibilité close*



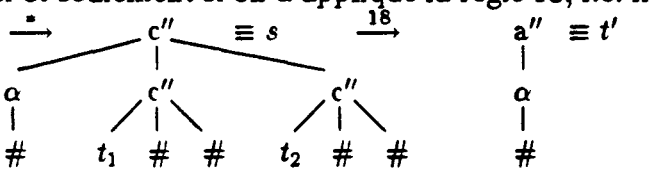
est équivalent au problème de correspondance de Post pour ϕ and ψ.

PREUVE :

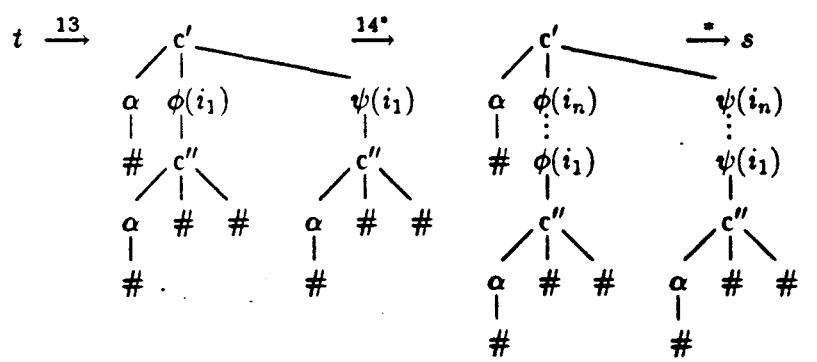
Nous nous intéressons au problème : $t \equiv$



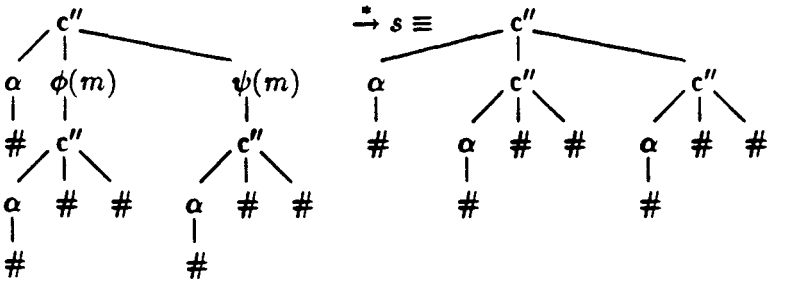
Les deux systèmes ne contiennent pas de règles effaçantes, c'est pourquoi, d'après le lemme 3.1.10, on peut décomposer le problème d'accessibilité. On en déduit donc que $t \xrightarrow{*}_{\mathcal{R}_1 \oplus \mathcal{R}_2} t'$ si et seulement si on a appliqué la règle 18, i.e. il existe deux termes t_1 et t_2 tels que $t \xrightarrow{*}$



Etudions le nouveau problème d'accessibilité $t \xrightarrow{*} s$. Il est satisfait si et seulement si $t_1 \equiv t_2 \equiv \alpha(\#)$ et il existe $i_1, \cdots, i_n, n \geq 1$ tels que



Il faut remarquer que, à cause des symboles α, on ne peut pas appliquer de règle permettant de réduire un terme dont la racine est c, c' ou c'' en #. s est un terme de racine c''. On réécrit un terme de racine c' en un terme de racine c'' en appliquant la règle 15. Donc $t \xrightarrow{*} s$ si et seulement s'il existe $m = i_n \cdots i_1$ $(n \geq 1)$ tel que



On supprime les lettres de $\phi(m)$ et $\psi(m)$ grâce aux règles 16 ou 17. Donc on en déduit que $t \xrightarrow{*} s$ si et seulement s'il existe $m = i_n \cdots i_1 (n \geq 1)$ tel que $\phi(m) = \psi(m)$. C'est pourquoi $t \xrightarrow{*} t'$ si et seulement si $t \xrightarrow{*} s$ ce qui est équivalent à dire que le problème de correspondance de Post pour ϕ et ψ a une solution. Donc le problème d'accessibilité $t \xrightarrow{*} t'$ est indécidable. □

PREUVE DU THÉORÈME 3.1.17 : Soient ϕ et ψ deux morphismes de I dans {a,a'}*. $(\mathcal{F}_2, \mathcal{R}_2)$ est le système de réécriture associé à ϕ et ψ. Nous savons que l'accessibilité close est décidable pour $(\mathcal{F}_1, \mathcal{R}_1)$ et pour $(\mathcal{F}_2, \mathcal{R}_2)$. Mais d'après le lemme précédent, elle est indécidable pour $\mathcal{R}_1 \oplus \mathcal{R}_2$. Ceci achève la preuve du théorème. □

3.2 Composition de systèmes constructeurs

3.2.1 Cas général

Un *système constructeur* (ou SC) est un système de réécriture de termes $(\mathcal{F}, \mathcal{R})$ tel que l'alphabet $\mathcal{F}$ peut être partitionné en deux ensembles disjoints $\mathcal{D}$ et $\mathcal{C}$ de la façon suivante :

tout terme $f(t_1, \cdots, t_n)$, membre gauche d'une règle de $\mathcal{R}$, satisfait les conditions $(f \in \mathcal{D})$ et $(t_1, \cdots, t_n \in \mathcal{T}_{\mathcal{C}}(\mathcal{X}))$.

Les symboles de fonctions de $\mathcal{D}$ sont appelés symboles ***définis*** et ceux de $\mathcal{C}$, *constructeurs.*

Pour mettre en évidence la partition de $\mathcal{F}$ en $\mathcal{D}$ et $\mathcal{C}$, on écrit $(\mathcal{D},\mathcal{C},\mathcal{R})$ au lieu de $(\mathcal{F},\mathcal{R})$.

Comme le comportement d'une machine de Turing peut être simulé par un SC [Klo91], les systèmes constructeurs ont une puissance de calcul universelle.

Définition 3.2.1 *[MT91]*

1. *Soit (D,C,R) un système constructeur et soit $\mathcal{D}' \subset \mathcal{D}$.*

   L'ensemble $\{l \to r \in \mathcal{R} \mid racine(l) \in \mathcal{D}'\}$ se note $\mathcal{R}|\mathcal{D}'$.

2. *Deux systèmes constructeurs $(\mathcal{D}_1, \mathcal{C}_1, \mathcal{R}_1)$ et $(\mathcal{D}_2, \mathcal{C}_2, \mathcal{R}_2)$ sont* composables *si $\mathcal{D}_1 \cap \mathcal{C}_2 = \mathcal{D}_2 \cap \mathcal{C}_1 = \emptyset$ et $\mathcal{R}_1|\mathcal{D}_2 = \mathcal{R}_2|\mathcal{D}_1$. Cette seconde condition signifie que, si les deux systèmes partagent un symbole défini, ils doivent contenir tous les deux toutes les règles qui "définissent" ce symbole.*

   La réunion de SC composables $SC_1, \cdots, SC_n$ se note $SC_1 + \cdots + SC_n$ et on dit que $SC_1, \cdots, SC_n$ est une décomposition de $SC_1 + \cdots + SC_n$.

3. *Une propriété P est dite* décomposable *si pour tous systèmes constructeurs $SC_1, \cdots, SC_n$ composables deux à deux et possédant la propriété P le système $SC_1 + \cdots + SC_n$ a aussi cette propriété P.*

Proposition 3.2.2 *[MT91] Soit P une propriété des systèmes constructeurs. Les affirmations suivantes sont équivalentes :*

1. *P est décomposable.*

2. *Pour tous systèmes constructeurs composables SC_1 et SC_2 qui ont la propriété P, le système $SC_1 + SC_2$ a aussi la propriété P.*

Dans ce paragraphe, nous étudions la composition de systèmes constructeurs dont le problème d'accessibilité est décidable. Malheureusement, dans le cas général, l'accessibilité n'est plus décidable pour le système obtenu par composition.

Théorème 3.2.3 *La décidabilité de l'accessibilité n'est pas une propriété composable des systèmes de réécriture constructeurs.*

PREUVE :

La preuve donnée dans le paragraphe 3.1.2 peut s'appliquer à ce théorème.

Dans cette preuve, on donne deux systèmes pour lesquels l'accessibilité est décidable. Ces systèmes sont des systèmes constructeurs.

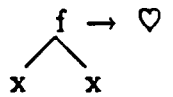


Figure 3.10 : Règle du système $(\mathcal{D}_1, \mathcal{C}_1, \mathcal{R}_1)$

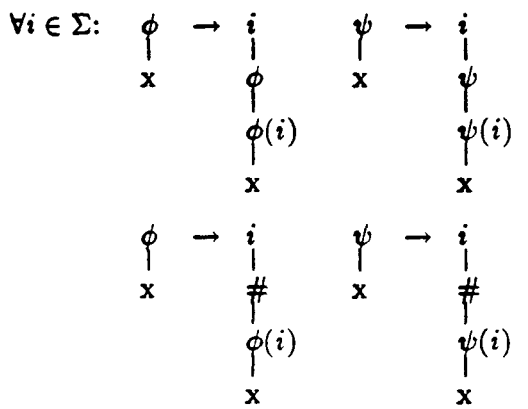


Figure 3.11 : Règles du système $(\mathcal{D}_2, \mathcal{C}_2, \mathcal{R}_2)$

En effet, pour le premier système, on pose $\mathcal{D}_1 = \{f(,)\}$ et $\mathcal{C}_1 = \{\heartsuit\}$. L'ensemble des règles est donné figure 3.10.

Σ étant un alphabet fini dont toutes les lettres ont l'arité 1, $\mathcal{D}_2 = \{\phi(), \psi()\}$ et $\mathcal{C}_2 = \{\#(), \$\} \cup \Sigma$, tels que $\{\#(), \phi(), \psi(), \$\}$ et Σ sont des alphabets disjoints.

Soit ϕ et ψ deux morphismes de Σ dans Σ^*. $(\mathcal{D}_2, \mathcal{C}_2, \mathcal{R}_2)$ est un système constructeur non noethérien, associé à ϕ and ψ. $\mathcal{R}_2$ est l'ensemble des règles de la figure 3.11.

D'après le paragraphe 3.1.2, l'accessibilité est décidable pour chacun des deux systèmes. D'après le lemme 3.1.4, il existe un problème d'accessibilité qui se ramène au problème de correspondance de Post pour ϕ et ψ. Donc l'accessibilité est indécidable pour $\mathcal{R}_1 + \mathcal{R}_2$.

Il faut remarquer que, dans cette preuve, la non-linéarité gauche de $\mathcal{R}_1$ n'est pas importante. En effet, on peut remplacer $\mathcal{R}_1$ par les règles de la figure 3.12, l'alphabet $\mathcal{D}_1$ ne change pas et $\mathcal{C}_1$ devient $\{\heartsuit, \#(), \$\} \cup \Sigma$

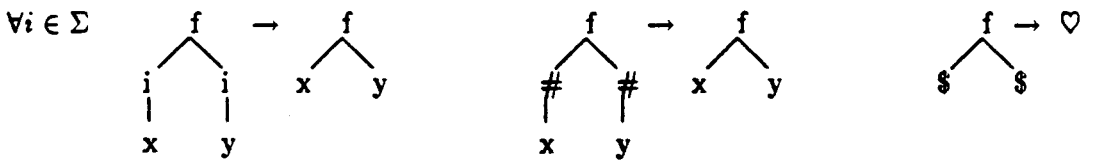


Figure 3.12: Transformation de $\mathcal{R}_1$

Par contre, dans le paragraphe 3.1.2, la non-linéarité gauche était obligatoire car les alphabets $\mathcal{F}_1$ et $\mathcal{F}_2$ devaient être disjoints.

3.2.2 Systèmes noethériens quelconques

La preuve donnée pour le cas général utilise un système $(\mathcal{D}_2, \mathcal{C}_2, \mathcal{R}_2)$ qui ne termine pas. C'est pourquoi nous avons voulu étudier le cas des systèmes noethériens.

Théorème 3.2.4 *La décidabilité de l'accessibilité n'est pas une propriété décomposable pour des systèmes constructeurs noethériens.*

Pour démontrer ce théorème, nous donnons deux systèmes noethériens. Leur accessibilité est bien sûr décidable (c'est une propriété des systèmes noethériens). Malheureusement, la composition de ces systèmes ne termine pas et en plus l'accessibilité n'est plus décidable.

Le premier système $(\mathcal{D}_1, \mathcal{C}_1, \mathcal{R}_1)$ a pour alphabets $\mathcal{C}_1 = \{\$, \clubsuit, \#(), 0, 1\} \cup \Sigma$, où Σ est un alphabet monadique (chaque lettre a l'arité 1) et $\mathcal{D}_1 = \{f(,), \alpha\}$. Les règles de $\mathcal{R}_1$ sont décrites dans la figure 3.13.

Le second système $(\mathcal{D}_2, \mathcal{C}_2, \mathcal{R}_2)$ est défini par les alphabets $\mathcal{D}_2 = \{\phi(,,,), \psi(,,,)\}$ et $\mathcal{C}_2 = \{0, 1, \#()\} \cup \Sigma$. La figure 3.14 contient les règles de $\mathcal{R}_2$.

Ces deux systèmes sont composables et leur accessibilité est décidable car ils sont noethériens.

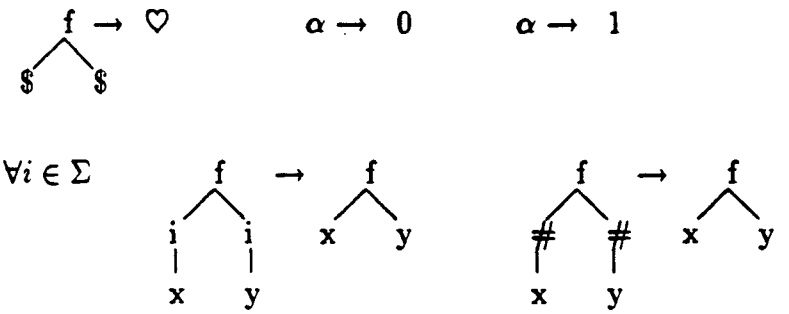


Figure 3.13 : règles du système constructeur $(\mathcal{D}_1, \mathcal{C}_1, \mathcal{R}_1)$

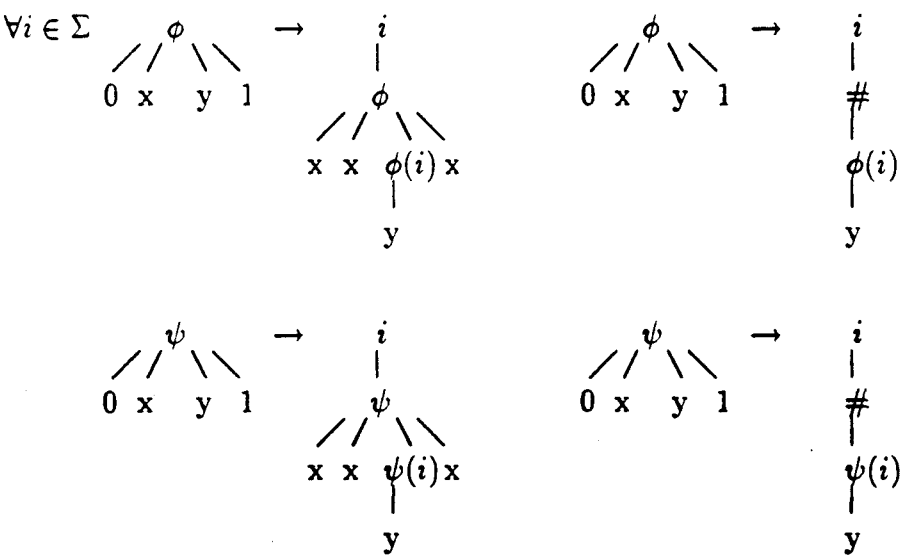
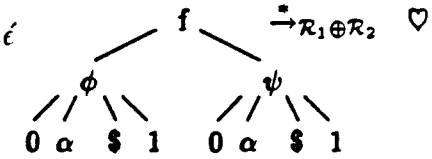


Figure 3.14 : règles du système constructeur $(\mathcal{D}_2, \mathcal{C}_2, \mathcal{R}_2)$

Lemme 3.2.5 *Le problème d'accessibilité*



est équivalent au problème de correspondance de Post pour ϕ et ψ.

PREUVE : La preuve est similaire à celle de la section 3.1.2. □

PREUVE DU THÉORÈME 3.2.4 : Le problème de correspondance de Post étant indécidable, on en déduit l'indécidabilité de l'accessibilité pour $\mathcal{R}_1 + \mathcal{R}_2$. Donc, la décidabilité de l'accessibilité n'est pas conservée par la composition de deux systèmes constructeurs noethériens. □

3.2.3 Systèmes noethériens linéaires à droite

La preuve précédente utilise des systèmes linéaires à gauche mais $(\mathcal{D}_2, \mathcal{C}_2, \mathcal{R}_2)$ n'est pas linéaire à droite. Nous démontrons ici que, dans le cas de systèmes noethériens et linéaires à droite, la décidabilité de l'accessibilité est préservée par composition de SC.

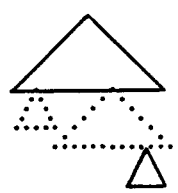
Définitions

Soient $(\mathcal{D}_1, \mathcal{C}_1, \mathcal{R}_1)$ et $(\mathcal{D}_2, \mathcal{C}_2, \mathcal{R}_2)$ deux SC.

Nous définissons les ensembles $\mathcal{D}' = \mathcal{D}_1 \cap \mathcal{D}_2$, $\mathcal{D}'_1 = \mathcal{D}_1 - \mathcal{D}'$, $\mathcal{D}'_2 = \mathcal{D}_2 - \mathcal{D}'$.

Nous transformons les définitions utilisées pour l'union disjointe de systèmes de réécriture afin de les adapter aux systèmes constructeurs.

Ici, les alphabets ne sont pas disjoints, néanmoins, nous pouvons quand même donner une décomposition des termes en parties qui ne contiennent pas de lettres de $\mathcal{D}'_1$ et en parties qui ne contiennent pas de lettres de $\mathcal{D}'_2$:



..... Sans symboles de $\mathcal{D}'_2$

___ Sans symboles de $\mathcal{D}'_1$

A cause des propriétés des systèmes constructeurs, sur une partie qui ne contient pas de symbole de $\mathcal{D}'_2$, on ne peut appliquer que des règles de $\mathcal{R}_1$ (en effet, les règles de $\mathcal{R}_2$ que l'on pourrait appliquer sont aussi dans $\mathcal{R}_1$).

- Soit $t \equiv C[t_1, \cdots, t_n]$ avec $C[, ...,] \not\equiv \Box$. On écrit $t \equiv C((t_1, \cdots, t_n))$ si $C[, \cdots,]$ ne contient pas de symbole de $\mathcal{D}'_a$ et si racine(t_1),..., racine(t_n) $\in \mathcal{D}'_a$ pour un $a \in \{1, 2\}$. Les termes t_i's sont appelés *SC-sous-termes principaux* de t.
- Le *SC-rang* d'un terme t est défini par

$$\begin{cases} \text{SC-rang}(t) = 1 & \text{si } t \in \mathcal{T}_1 \cup \mathcal{T}_2 \\ \text{SC-rang}(t) = 1 + \max\{\text{SC-rang}(t_i), 1 \leq i \leq n\} & \text{si } t = C((t_1, \cdots, t_n)) \end{cases}$$

- La *SC-partie homogène au sommet* d'un terme t, notée SC-top(t), est le terme obtenu en remplaçant tous les SC-sous-termes principaux de t par $\Box$, i.e.

$$\begin{cases} \text{SC-top}(t) = t \text{ si SC-rang}(t) = 1 \\ \qquad\qquad = C[, ...,] \text{ si } t \equiv C((t_1, \cdots, t_n)) \end{cases}$$

Résultat de décidabilité

Nous démontrons que, dans le cas de systèmes constructeurs linéaires à droite, la terminaison est préservée par composition. Comme tout système noethérien a une accessibilité décidable, on en déduit immédiatement que la décidabilité de l'accessibilité est conservée par composition de systèmes constructeurs noethériens.

Lemme 3.2.6 *La composition de deux SC noethériens est un SC noethérien.*

PREUVE : Nous démontrons qu'il n'existe pas de dérivation infinie partant d'un terme t, par induction sur le SC-rang de t.

- Soit t un terme de SC-rang 1. $(\mathcal{D}_1, \mathcal{C}_1, \mathcal{R}_1)$ et $(\mathcal{D}_2, \mathcal{C}_2, \mathcal{R}_2)$ sont noethériens par hypothèse. Si t est de SC-rang 1 alors t ne contient pas de lettres de $\mathcal{D}'_1$ ou pas de lettre de $\mathcal{D}'_2$. Donc on ne peut appliquer sur t que des règles de $\mathcal{R}_2$ ou que des règles de $\mathcal{R}_1$. Donc il n'existe pas de dérivation infinie partant de t.

- Maintenant, on suppose que le lemme 3.2.6 est vrai pour des termes de SC-rang $n-1$ pour n fixé. Supposons qu'il existe une dérivation infinie partant d'un terme t tel que SC-rang(t)=n. Par hypothèse d'induction, il n'y a pas de dérivation infinie partant d'un SC-sous-terme principal de t car le SC-rang de ces sous-termes est au

maximum $n-1$. De plus, les systèmes de réécriture sont linéaires à droite. Ce qui veut dire que si $t \equiv C((t_1,\cdots,t_k)) \xrightarrow{*} s \equiv C'((s_1,\cdots,s_p))$ alors $p \leq k$ et il existe un ensemble $\{i_1,\cdots,i_p\} \subset \{1,\cdots,k\}$ tel que $t_{i_1} \xrightarrow{*} s_1,\cdots,t_{i_p} \xrightarrow{*} s_p$. Ceci signifie qu'aucun sous-terme principal n'est dupliqué lors de la réduction. Donc, après un nombre fini de pas, on obtient un terme t' de rang n tel que toutes les règles ne s'appliquent que sur le SC-top de t'. Alors, à partir du terme SC-top(t') il existe une dérivation infinie. On peut supposer que SC-top(t') ne contient pas de lettre de $\mathcal{D}'_2$. Alors, sur SC-top(t') on ne peut appliquer que des règles de $\mathcal{R}_1$. Comme $(\mathcal{D}_1, \mathcal{C}_1, \mathcal{R}_1)$ est noethérien, on arrive à une contradiction et on en déduit qu'il n'existe pas de dérivation infinie partant de t. □

Nous obtenons comme corollaire de ce lemme le théorème suivant :

Théorème 3.2.7 *La décidabilité de l'accessibilité est conservée par la composition de SC noethériens et linéaires à droite.*

PREUVE : La preuve est triviale car on sait que l'accessibilité est décidable pour tout système noethérien et on vient de démontrer que la terminaison était décomposable pour des SC linéaires à droite. □

3.3 Récapitulatif des résultats obtenus

Aussi bien pour l'union disjointe de systèmes que pour la composition de systèmes constructeurs, nous avons obtenu plusieurs résultats par une étude exhaustive de tous les cas possibles. Le tableau suivant permet de les résumer. En effet, on indique dans chaque case si la décidabilité de l'accessibilité est conservée par union. En colonne, on précise la (non-)linéarité des systèmes considérés et en ligne, le type d'union (union disjointe, composition de SC, noethériens ou non). L'abréviation "r.e." signifie "règle effaçante".

	Linéaire droit		Non linéaire droit		
	Non linéaire à gauche	Linéaire à gauche	Non linéaire à gauche	Linéaire à gauche	
Union disjointe de SDR	Non	Oui	Non	Avec r.e.	Sans r.e.
				Non	Oui
Composition de SC	Non	Non	Non	Non	
Composition de SC noethériens	Oui	Oui	Non	Non	

Chapitre 4

Réductibilité et automates d'arbres

Nous définissons dans ce chapitre une classe d'automates qui fournit un cadre algébrique et algorithmique au théorème de Plaisted de décidabilité de la réductibilité inductive [Pla85][1]. Plus précisément, notre but est de savoir résoudre toutes les formules du premier ordre avec sortes ordonnées, écrites à partir de prédicats de la forme "t est facteur de x", notés $\mathsf{facteur}_t(x)$. Une autre façon de voir notre résultat est de considérer que l'on peut traiter comme des sortes des ensembles de termes ayant des contraintes d'égalité entre sous-termes (le nombre de contraintes le long d'un chemin est borné). Nos automates sont donc des formes résolues de ces contraintes. Le point le plus technique de notre résultat est la preuve de la décision du vide. L'idée est d'obtenir un lemme de pompage qui généralise la preuve de D. Plaisted. Nous proposons d'appeler ce résultat "théorème de Plaisted pour les automates d'arbres".

Nous définissons la classe de nos automates progressivement : nous partons de définitions générales et posons des restrictions dès qu'elles sont nécessaires. Ceci permet de mettre en valeur le rôle de chacune de ces restrictions. Avec ces automates, nous voulons résoudre les formules du premier ordre appelées *formules d'entourage*, écrites avec des formules atomiques $\mathsf{facteur}_t(x)$ ($t \in T_\Sigma(\mathcal{X})$). La classe d'automates que nous construisons, nommée AFA (automates de filtrage dans les arbres) a les propriétés suivantes :

[1] Un terme t est inductivement réductible si toutes ses instances closes sont réductibles. La réductibilité inductive est aussi appelée quasi-reductibilité ou réductibilité close

0. Pour tout automate non déterministe, on peut construire un automate complétement spécifié et déterministe qui reconnaît le même langage.
1. La classe AFA est effectivement close par les opérations booléennes.
2. Le vide est décidable.
3. Pour tout t dans $T_\Sigma(\mathcal{X})$, la formule atomique $\mathsf{facteur}_t(x)$ est définissable par un automate, i.e. on peut construire un automate qui reconnaît l'ensemble des termes qui ont la propriété $\mathsf{facteur}_t(x)$. De plus, t peut être sorté.
4. Quand t est linéaire, l'automate qui reconnaît l'ensemble des termes qui entourent t est un automate standard de REC.

Le point 0 est utilisé pour exprimer le complément. Les points 1, 2 et 3 permettent de construire un automate de AFA $\mathcal{A}_F$ qui spécifie l'ensemble des termes satisfaisant une formule d'entourage donnée. Par exemple, si la formule F est :

$$\mathsf{facteur}_t(x) \Rightarrow (\mathsf{facteur}_{l_1}(x) \vee \cdots \vee \mathsf{facteur}_{l_p}(x)) \tag{4.1}$$

on peut déduire $\mathcal{A}_F$ des automates spécifiant $\mathsf{facteur}_t(x)$, $\mathsf{facteur}_{l_1}(x)$, ..., $\mathsf{facteur}_{l_p}(x)$. En utilisant des produits d'ensembles, on peut étendre le résultat aux formules $F(x_1, \cdots, x_n)$. Ces techniques sont utilisées entre autres par M. O. Rabin [Rab77] et W. Thomas [Tho90]. Si F est une formule close, on peut décider de sa validité. Les automates peuvent donc être considérés comme des formes résolues des formules d'entourage.

En résumé, nos automates spécifient des sortes usuelles et des contraintes d'entourage en nombre borné le long de tout chemin. Si on s'autorise un nombre non borné de contraintes le long d'une branche, on perd les bonnes propriétés 1 et 2 (J. Mongy [Mon81]). Notons que dans le cas particulier d'égalités entre fils d'un même noeud, avec un nombre non borné de contraintes le long des chemins, les propriétés 1 et 2 restent vraies (B. Bogaert et S. Tison [BT92]).

Notre résultat permet d'unifier des résultats connus : on obtient immédiatement la décidabilité de la réductibilité inductive dans le cas des sortes ordonnées et d'autres résultats qui en découlent (D. Plaisted [Pla85], D. Kapur, P. Narendran, H. Zhang [KNZ87]). Dans le cas linéaire (quand les termes t des prédicats $\mathsf{facteur}_t(x)$ sont linéaires), la construction donne des automates standards ce qui confirme les résultats de N. Dershowitz [Der85] et J.P. Jouannaud et E. Kounalis [JK89].

Nous donnons maintenant quelques exemples d'utilisation de nos automates :

- Soit R un système de réécriture. Si $R = \{l_1 \rightarrow r_1, \cdots, l_p \rightarrow r_p\}$ alors on peut exprimer la réductibilité inductive d'un terme t par rapport à R grâce à la formule $\forall x\ F(x)$ où F est la formule (4.1) donnée précédemment.

- Une spécification équationnelle E est dite suffisamment complète (pour un ensemble de constructeurs C) si et seulement si pour tout terme clos g de T_Σ il existe un terme clos c composé uniquement de constructeurs de C, tel que $g \leftrightarrow^* c$. Un ensemble R de règles de réécriture est dit suffisamment complet si l'ensemble d'équations E associé est suffisamment complet.

  Un système de réécriture préserve les constructeurs si pour toute règle $l \to r$ telle que l ne contient que des constructeurs alors r ne contient également que des constructeurs.

  Théorème de Kapur, Narendran, Zhang [KNZ87] Un ensemble R de règles préservant les constructeurs est suffisamment complet si et seulement si pour tout f dans $\Sigma - C$, $f(x_1, \cdots, x_n)$ est inductivement réductible pour R.

  Donc dans le cas d'un système de réécriture $\{l_1 \to r_1, \cdots, l_p \to r_p\}$ préservant les constructeurs, où l'ensemble des symboles définis (i.e. non constructeurs) est $\Sigma - C = \{f_1, \cdots, f_n\}$, R est suffisamment complet si et seulement si la formule suivante est vraie :

$$\forall x(\mathsf{facteur}_{f_1(x_1,\cdots,x_{i_1})}(x) \vee \cdots \vee \mathsf{facteur}_{f_n(x_1,\cdots,x_{i_n})}(x) \Rightarrow (\mathsf{facteur}_{l_1}(x) \vee \cdots \vee \mathsf{facteur}_{l_p}(x)))$$

- Nous pouvons construire un automate qui décrit l'ensemble des termes tels que "il n'y a aucune occurrence de $b(b(x,x),b(y,y))$ et deux occurrences de $b(b(x,x),x)$ au plus pour tout chemin, et exactement deux pour au moins un chemin". L'alphabet Σ est $\{b(,),a\}$ et l'automate associé à cette contrainte est donné figure 4.1

  On peut construire un automate complétement spécifié et déterministe qui reconnaît le même langage.

Nous définissons les automates AFA progressivement, et pour cette raison, nous passons par deux classes d'automates intermédiaires AC (automates avec contraintes) et ACCCB (automates à contraintes closes par contexte et bornées). Dans ces deux classes d'automates, les contraintes sont quelconques (i.e. pas nécessairement des contraintes d'entourage) mais le lecteur peut avoir en tête l'exemple canonique des contraintes d'entourage $\mathsf{facteur}_t(x)$ comme contraintes de bases. Les contraintes de bases linéaires peuvent toujours être supprimées et controlées par des automates d'arbres standard (Hoffmann et O'Donnell [HO82]), c'est pourquoi nous ne considérons que des contraintes non linéaires. Plus généralement, les contraintes sont des combinaisons booléennes des contraintes de base et sont obtenues par composition

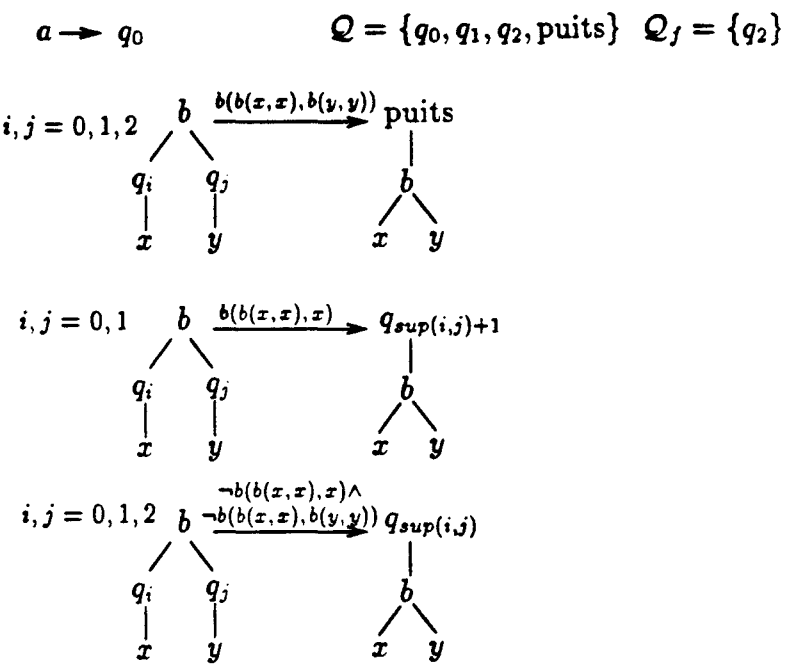


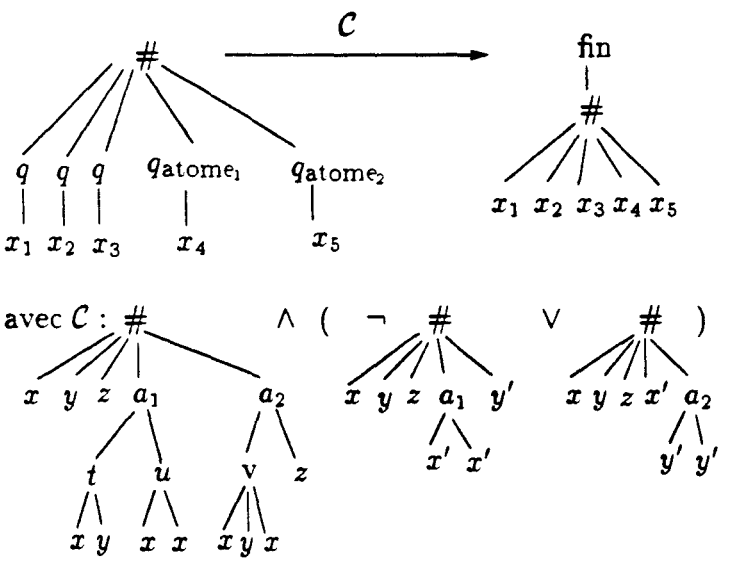
Figure 4.1 : Automate de AFA

d'automates. Les applications ω sont introduites dans la classe ACCCB pour contrôler le nombre de tests de contraintes de base le long d'un chemin. Il faut remarquer que les contraintes de base et leurs négations jouent un rôle dissymétrique. En effet, on ne teste qu'un nombre borné de contraintes de bases le long de toute branche alors qu'on s'autorise un nombre non borné de tests de leurs négations. Ceci peut s'expliquer par la transitivité de l'égalité et la non-transitivité de sa négation.

Notons enfin que toute forme purement existentielle d'un problème équationnel (H. Comon [Com92]) se traduit en un automate de AFA. Par exemple, la formule

$$\exists x, y, z\ t(x,y) \neq u(x,x) \vee v(x,y,x) = z$$

peut se traduire en un automate dont l'une des règles sera :



4.1 Automates avec contraintes

4.1.1 Définition

AC représente la classe des automates avec contraintes.

Définition 4.1.1 *Un automate de AC est un quintuplet* $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{Q}_f, \mathcal{E}, \mathcal{R})$

- Σ *est un alphabet fini gradué.*
- $\mathcal{Q}$ *est un ensemble fini d'états.*
- $\mathcal{Q}_f$ *est un ensemble d'états finaux* ($\mathcal{Q}_f \subseteq \mathcal{Q}$).
- $\mathcal{E}$ *est un ensemble fini de contraintes de base portant sur le terme lu par* $\mathcal{A}$.
- $\mathcal{R}$ *est un ensemble fini de règles. Ces règles sont de la forme:*
  $f(q_1(x_1), \cdots, q_n(x_n)) \xrightarrow{c} q(f(x_1, \cdots, x_n))$ *où c est une combinaison booléenne d'éléments de* $\mathcal{E}$. *c sera appelée contrainte de la règle.*

FONCTIONNEMENT DES AUTOMATES AVEC CONTRAINTES :

Soit $t = f(t_1, \cdots, t_n)$

$t \overset{*}{\vdash}_{\mathcal{A}} q(t)$ si et seulement s'il existe une règle

$f(q_1(x_1), \cdots, q_n(x_n)) \xrightarrow{c} q(f(x_1, \cdots, x_n))$

dans $\mathcal{R}$ tels que c est vérifiée par t et $\forall i \in [1, n]$, $t_i \overset{*}{\vdash}_{\mathcal{A}} q_i(t_i)$

Définition 4.1.2 *Pour tout état* q

$$L_q(\mathcal{A}) = \{t \in T_\Sigma \mid t \overset{*}{\vdash}_{\mathcal{A}} q(t)\}$$

Ainsi, le langage reconnu par $\mathcal{A}$ *est:*

$$L(\mathcal{A}) = \bigcup_{q \in Q_f} L_q(\mathcal{A})$$

Définition 4.1.3 *Un langage d'arbres $\mathcal{F}$ est* AC-reconnaissable *s'il existe un automate $\mathcal{A}$ dans AC tel que $L(\mathcal{A}) = \mathcal{F}$*

La classe des langages AC-reconnaissables sera notée RAC.

Remarque 4.1.4 Les automates ascendants d'arbres classiques sont des automates de AC, où $\mathcal{E}$ est vide.

Sauf précision contraire, les automates considérés par la suite seront toujours des automates avec contraintes.

On note $\mathcal{E}_-$ l'ensemble $\{\neg P \mid P \in \mathcal{E}\}$.

4.1.2 Quelques transformations

Algorithme de suppression du "ou"

Lemme 4.1.5 *A tout automate $\mathcal{A}$ de base $\mathcal{E}$ on peut associer un automate $\mathcal{A}'$ équivalent dont les contraintes sont des conjonctions d'éléments de $\mathcal{E}_- \cup \mathcal{E}$. $\mathcal{A}'$ sera appelé automate* sans "ou".

PREUVE : On met les contraintes sous forme normale disjonctive. Puis on applique la règle de transformation :

$$\frac{l \xrightarrow{c_1 \vee c_2} r}{l \xrightarrow{c_1} r,\ l \xrightarrow{c_2} r}$$

On obtient un automate $\mathcal{A}'$ tel que pour tout q $L_q(\mathcal{A}) = L_q(\mathcal{A}')$. □

Par la suite, on ne considère plus que des automates sans "ou".

Définition 4.1.6 *Un automate est* complètement spécifié *si pour tout terme $t \in T_\Sigma$, il existe un état q tel que $t \vdash_{\mathcal{A}}^* q(t)$.*

Définition 4.1.7 *Un automate $\mathcal{A}$ est* déterministe *si pour tout couple de règles ayant mêmes parties gauches les contraintes sont incompatibles (i.e. leur conjonction est insatisfiable) .*

Remarque 4.1.8 La définition ci-dessus coïncide avec celle du déterminisme dans le cas des automates de REC.

Algorithme de spécification complète

Lemme 4.1.9 *Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{Q}_f, \mathcal{E}, \mathcal{R})$ un automate sans "ou", on peut construire un automate $\mathcal{A}' = (\Sigma, \mathcal{Q}', \mathcal{Q}_f, \mathcal{E}, \mathcal{R}')$ sans "ou" complétement spécifié tel que pour tout état q de $\mathcal{Q}$, $L_q(\mathcal{A}) = L_q(\mathcal{A}')$.*

PREUVE :

1. Pour toute lettre f de Σ_n et pour tous $q_1, \cdots, q_n$ dans $\mathcal{Q}$,
   (a) S'il n'y a pas de règle de partie gauche $f(q_1(x_1), \cdots, q_n(x_n))$. On crée alors la règle
   $f(q_1(x_1), \cdots, q_n(x_n)) \rightarrow \mathsf{Puits}(f(x_1, \cdots, x_n))$
   (b) Si les règles de parties gauches $f(q_1(x_1), \cdots, q_n(x_n))$ sont :
   $f(q_1(x_1), \cdots, q_n(x_n)) \xrightarrow{c_1} q'_1(f(x_1, \cdots, x_n))$,
   $\vdots$
   $f(q_1(x_1), \cdots, q_n(x_n)) \xrightarrow{c_p} q'_p(f(x_1, \cdots, x_n))$,
   alors, on met la condition $\neg c_1 \wedge \ldots \wedge \neg c_p$ sous forme normale disjonctive $\gamma_1 \vee \cdots \vee \gamma_k$ où les γ_i sont des conjonctions d'éléments de $\mathcal{E} \cup \mathcal{E}_-$ et pour tout i dans [1,k] on ajoute la règle
   $f(q_1(x_1), \cdots, q_n(x_n)) \xrightarrow{\gamma_i} \mathsf{Puits}(f(x_1, \cdots, x_n))$.
2. Pour tout f dans Σ_n, on ajoute toutes les règles
   $f(s_1(x_1), \cdots, s_n(x_n)) \rightarrow \mathsf{Puits}(f(x_1, \cdots, x_n))$
   où au moins l'un des s_i vaut Puits.

$\mathcal{R}'$ est l'ensemble des règles obtenues à partir de $\mathcal{R}$ en appliquant les transformations ci-dessus et $\mathcal{Q}' = \mathcal{Q} \cup \{\mathsf{Puits}\}$.

On a alors $t \overset{*}{\vdash}_{\mathcal{A}'} \mathsf{Puits}(t) \Leftrightarrow \nexists q \in \mathcal{Q}\ t \overset{*}{\vdash}_{\mathcal{A}} q(t)$

et $\forall q \in \mathcal{Q}\ (t \overset{*}{\vdash}_{\mathcal{A}} q(t) \Leftrightarrow t \overset{*}{\vdash}_{\mathcal{A}'} q(t))$.

L'automate $\mathcal{A}'$ est sans "ou" et complétement spécifié.

□

Algorithme de déterminisation

Lemme 4.1.10 *Soit $\mathcal{A} = (\Sigma, Q, Q_f, \mathcal{E}, \mathcal{R})$ complétement spécifié sans "ou", on peut construire $\mathcal{A}' = (\Sigma, Q', Q'_f, \mathcal{E}, \mathcal{R}')$ déterministe complétement spécifié sans "ou" tel que $L(\mathcal{A}) = L(\mathcal{A}')$ (à condition de pouvoir tester la satisfiabilité des combinaisons booléennes de contraintes).*

PREUVE : Comme dans le cas des automates sans contraintes, $Q' \subseteq P(Q)$

1. initialisation : $Q' = \emptyset$, $\mathcal{R}' = \emptyset$.
2. itération : $\forall f \in \Sigma_n,\ \forall k_1, \cdots, k_n \in Q'$

   On considère toutes les règles de parties gauches du type $f(q_1(x_1), \cdots, q_n(x_n))$ avec $\forall i,\ q_i \in k_i$:

   $f(q_1^1(x_1), \cdots, q_n^1(x_n)) \xrightarrow{c_1} s_1(f(x_1, \cdots, x_n))$

   $\vdots$

   $f(q_1^p(x_1), \cdots, q_n^p(x_n)) \xrightarrow{c_p} s_p(f(x_1, \cdots, x_n))$

   On construit l'ensemble Γ des conjonctions satisfiables maximales à partir de $c_1, \cdots, c_p$:

   $$\Gamma = \{\gamma = c_{i_1} \wedge \cdots \wedge c_{i_m} \mid \gamma \text{ satisfiable et } \forall j \notin \{i_1, \cdots, i_m\}\ \gamma \wedge c_j \text{ insatisfiable}\}$$

   On ajoute dans $\mathcal{R}'$ toutes les règles

   $f(k_1(x_1), \cdots, k_n(x_n)) \xrightarrow{\gamma} k_\gamma(f(x_1, \cdots, x_n))$

   avec $\gamma \in \Gamma$, $\gamma = c_{i_1} \wedge \cdots \wedge c_{i_m}$ et $k_\gamma = \{s_{i_1}, \cdots, s_{i_m}\}$.

   On ajoute à Q' les éléments k_γ.

On réitère le deuxième point jusqu'à ce qu'on ne puisse plus ajouter de règle.

L'ensemble Q'_f des états finaux est $\{k \mid Q_f \cap k \neq \emptyset\}$.

On vérifie par récurrence sur la taille t des termes que

$$t \in L_k(\mathcal{A}') \Leftrightarrow t \in (\wedge_{q \in k} L_q(\mathcal{A}))$$

D'autre part, pour tout $t = f(t_1, \cdots, t_n)$, supposons qu'il existe un et un seul mouvement dans $\mathcal{A}$ tel que $t_1 \overset{*}{\vdash}_{\mathcal{A}} k_1(t_1), \cdots, t_n \overset{*}{\vdash}_{\mathcal{A}} k_n(t_n)$, ce mouvement se prolonge de façon unique par $f(k_1(t_1), \cdots, k_n(t_n)) \vdash_{\mathcal{A}} k_\gamma(f(t_1, \cdots, t_n))$ avec $\gamma = \{\wedge c_i \mid c_i(t)\}$.

□

4.1.3 Clôture par les opérations booléennes

Lemme 4.1.11 *La classe AC est close par les opérations booléennes : union, intersection, complémentaire.*

PREUVE :

- Il est facile de voir que la classe AC est close par complémentaire :

  Si $\mathcal{A}$ est un automate de AC, on peut construire un automate $\mathcal{A}'$ déterministe et complétement spécifié qui reconnaît le même langage. Si $\mathcal{A}' = (\Sigma, Q, Q_f, \mathcal{E}, \mathcal{R})$ alors $\mathcal{B} = (\Sigma, Q, Q - Q_f, \mathcal{E}, \mathcal{R})$ est le complémentaire de $\mathcal{A}'$, donc de $\mathcal{A}$.

- Pour montrer que AC est close par union et intersection, on va démontrer qu'elle est close par produit.

  Soient $\mathcal{A}$ et $\mathcal{B}$ deux automates de la classe AC. $\mathcal{A}'$ et $\mathcal{B}'$ sont des automates déterministes et complétement spécifiés qui reconnaissent respectivement $\mathrm{L}(\mathcal{A})$ et $\mathrm{L}(\mathcal{B})$.

  On pose $\mathcal{A}' = (\Sigma, Q_A, Q_A^f, \mathcal{E}_A, \mathcal{R}_A)$ et $\mathcal{B}' = (\Sigma, Q_B, Q_B^f, \mathcal{E}_B, \mathcal{R}_B)$

  L'automate produit $\mathcal{A}' \times \mathcal{B}' = (\Sigma, Q_A \times Q_B, F, \mathcal{E}_A \cup \mathcal{E}_B, \mathcal{R})$.

  L'ensemble $\mathcal{R}$ est construit de la façon suivante :

  $\forall f \in \Sigma_n, f((q_1, q_1')(x_1), \cdots, (q_n, q_n')(x_n)) \xrightarrow{c \wedge c'} (q, q')(f(x_1, \cdots, x_n)) \in \mathcal{R}$ si et seulement si

  $f(q_1(x_1), \cdots, q_n(x_n)) \xrightarrow{c} q(f(x_1, \cdots, x_n)) \in \mathcal{R}_A$ et

  $f(q_1'(x_1), \cdots, q_n'(x_n)) \xrightarrow{c'} q'(f(x_1, \cdots, x_n)) \in \mathcal{R}_B$

  avec $(c \wedge c')$ satisfiable.

  L'automate produit ainsi construit est déterministe et complétement spécifié. C'est bien un automate de la classe AC.

On en déduit que $\mathcal{A} \cup \mathcal{B}$ est l'automate $\mathcal{A}' \times \mathcal{B}'$ en prenant comme ensemble d'états finaux $(Q_A^f \times Q_B) \cup (Q_A \times Q_B^f)$.

De même, $\mathcal{A} \cap \mathcal{B}$ est l'automate $\mathcal{A}' \times \mathcal{B}'$ en prenant comme ensemble d'états finaux $Q_A^f \times Q_B^f$.

□

Malheureusement, le vide n'est en général pas décidable pour la classe AC, même si les contraintes sont du filtrage. En effet, lorsqu'on s'autorise un nombre non borné de contraintes le long d'un chemin, le vide est indécidable (J. Mongy [Mon81]). Pour espérer obtenir la décision du vide, il faut donc restreindre la classe d'automates étudiée.

4.1.4 Automates à contraintes closes par contexte et bornées

On veut mémoriser dans les états les contraintes de base testées et satisfaites le long des branches, avec leur ordre de multiplicité. Dans ces conditions, n'ayant qu'un nombre fini d'états, on ne teste qu'un nombre fini de contraintes le long d'une branche. On peut avoir en tête qu'on a des ressources bornées et que satisfaire une contrainte coûte. L'exemple type est de considérer comme contraintes de base l'entourage de facteurs sortés donnés. Par exemple, l'application de la règle $f(q_1(x_1), q_2(x_2)) \xrightarrow{f(a(x),x)} q'(f(x_1,x_2))$ sur le terme $f(t_1,t_2)$ signifie que $f(a(x),x)$ filtre $f(t_1,t_2)$, que t_1 est de sorte q_1 et t_2 de sorte q_2. La construction va mémoriser la satisfaction de cette contrainte de base dans q'. On va mémoriser dans l'état les multi-ensembles de contraintes de base ainsi satisfaites le long d'une branche. Les propriétés considérées seront closes par contexte, ce qui est cohérent avec l'idée que l'état mémorise une propriété du terme qu'il pointe. Très précisément, une propriété de base close par contexte sera l'application d'une certaine règle lors de la reconnaissance du terme :

$P_r(t) \Leftrightarrow$ il existe un mouvement de reconnaissance de t qui applique la règle r

Les propriétés élémentaires que nous devons spécifier par la donnée d'un automate, et qui sont par définition nécessaires pour étudier l'ordre d'entourage, sont de la forme $f(q_1(x_1), \cdots, q_n(x_n)) \xrightarrow{c} q(f(x_1, \cdots, x_n))$, c étant une contrainte de base de filtrage par un terme. L'extension de la problématique à la théorie de l'entourage, qui revient à la clôture booléenne des propriétés considérées, nous amène à traiter des

règles avec des contraintes plus générales. Nous voulons, à partir de ces règles plus générales ne mémoriser que les règles liées aux contraintes de base. On comprend alors que l'automate doit être sans "ou" : en effet, dans une règle

$$f(q_1(x_1),\cdots,q_n(x_n)) \xrightarrow{c_1 \vee c_2} q(f(x_1,\cdots,x_n))$$

on ne peut pas savoir si on doit mémoriser la satisfaction de la contrainte c_1 ou c_2. De plus, l'automate ne peut pas avoir de règles ne différant que par les contraintes :

$$f(q_1(x_1),\cdots,q_n(x_n)) \xrightarrow{c_1} q(f(x_1,\cdots,x_n))$$
$$f(q_1(x_1),\cdots,q_n(x_n)) \xrightarrow{c_2} q(f(x_1,\cdots,x_n))$$

pour la même raison.

Définition 4.1.12 *Une propriété P sur $T_\Sigma(\mathcal{X})$ est* close par contexte *si*

$$(P(t) \text{ et } t \text{ sous-terme de } s) \Rightarrow P(s)$$

Par exemple, la propriété $P(x)$ = "t est facteur de x" est close par contexte. De même, la propriété $P'(x)$ = "x contient au moins une occurence de la lettre a" est aussi une propriété close par contexte. Par contre $P''(x)$ = "x contient exactement une occurence de a" n'est pas une propriété close par contexte.

Dans ce paragraphe, les contraintes de base sont des propriétés closes par contexte. Notons que, quelque soit l'automate considéré, les ensembles $\mathcal{E}$ et $\mathcal{E}_-$ sont disjoints car la négation d'une propriété close par contexte n'est pas close par contexte.

Définition 4.1.13 *Si c est une conjonction d'éléments $l_1,\cdots,l_n$ de $\mathcal{E} \oplus \mathcal{E}_-$, on note $\mathcal{E}(c)$ l'ensemble des propriétés positives de c, i.e. $\mathcal{E}(c) = (\cup_{i=1}^n \{l_i\}) \cap \mathcal{E}$. Une règle de contrainte c est dite* positive *si $\mathcal{E}(c)$ n'est pas vide.*

Définition 4.1.14 *Un automate à contraintes closes par contexte et bornées est un automate de AC noté $(\Sigma, \mathcal{Q}, \mathcal{Q}_f, \mathcal{E}, \mathcal{R}, \omega)$.*

- *$\mathcal{E}$ est un ensemble fini de propriétés closes par contextes*
- *ω est une application de $\mathcal{Q}$ dans un sup-demi-treillis fini T telle que :*
  $\forall\, f(q_1(x_1),\cdots,q_n(x_n)) \xrightarrow{c} q(f(x_1,\cdots,x_n)) \in \mathcal{R}$
  $\omega(q)$ *majore* $\{\omega(q_1),\cdots,\omega(q_n)\}$
  La majoration est stricte si la règle est positive.

La classe des automates à contraintes closes par contexte et bornées sera notée ACCCB. Dans la suite de ce paragraphe, on ne considère plus que des automates de ACCCB.

Propriété fondamentale *Card(T) majore, pour tout mouvement, le long de toute branche, le nombre d'occurrences des règles positives.*

EXEMPLE 1 : Soit $\mathcal{E} = \{c_1 \cdots, c_p\}$.

$T = [0, n_{c_1}] \times \cdots \times [0, n_{c_p}]$. T est muni de l'ordre produit $\preceq$:

$(n_1, \cdots, n_p) \preceq (n'_1, \cdots, n'_p)$ si $\forall i \in [1, p]$ $n_i \leq n'_i$,

où $\leq$ est la relation d'ordre sur $\mathbb{N}$.

On considère l'application ω définie de la façon suivante :

$f(q_1(x_1), \cdots, q_n(x_n)) \xrightarrow{c} q(f(x_1, \cdots, x_n)) \Rightarrow \omega(q) = \sup\{\omega(q_i) | i \leq n\} + V(c)$

$V(c)$ est le vecteur de $\{0,1\}^p$ tel que sa ième composante vaut 1 si et seulement si c_i est dans $\mathcal{E}(c)$.

Cette application ω permet donc de compter les contraintes de base satisfaites le long d'une branche. On s'autorise au maximum n_{c_i} occurences de la contrainte c_i, donc le nombre d'occurrences des règles positives est bien borné le long d'une branche mais n'est pas borné globalement.

EXEMPLE 2 : On prend le même ensemble T et la même relation d'ordre que pour l'exemple précédent.

$f(q_1(x_1), \cdots, q_n(x_n)) \xrightarrow{c} q(f(x_1, \cdots, x_n)) \Rightarrow \omega(q) = \Sigma_{i=1}^{n} \omega(q_i) + V(c)$

Cette fois, le nombre d'occurrences des règles positives est borné globalement.

Lemme 4.1.15 *Soit $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{Q}_f, \mathcal{E}, \mathcal{R}, \omega)$, il existe $\mathcal{A}' = (\Sigma, \mathcal{Q}', \mathcal{Q}_f, \mathcal{E}, \mathcal{R}', \omega')$ dans ACCCB complétement spécifié tel que $\forall q \in \mathcal{Q}$ $L_q(\mathcal{A}) = L_q(\mathcal{A}')$.*

PREUVE : La construction du lemme 4.1.9 est valide, en ajoutant à T un majorant strict ω(Puits). En effet, les points 1.a et 2 de l'algorithme engendrent des règles sans conditions, et le point 1.b engendre une règle telle que ω(Puits) majore $\{\omega(q_1), \cdots, \omega(q_n)\}$. □

Lemme 4.1.16 *Soit $\mathcal{A} = (\Sigma, Q, Q_f, \mathcal{E}, \mathcal{R}, \omega)$ un automate complétement spécifié. On peut construire $\mathcal{A}' = (\Sigma, Q', Q'_f, \mathcal{E}, \mathcal{R}', \omega')$ déterministe et complétement spécifié tel que $L(\mathcal{A}) = L(\mathcal{A}')$.*

PREUVE : La construction du lemme 4.1.10 est légèrement modifiée : on construit de nouveaux états dans $P(Q) \times [0, \text{Card}(T)]$.

Au départ, pour toute constante a, $a \longrightarrow (k, 0)(a)$ où k est $\{q \mid a \vdash_{\mathcal{A}} q(a)\}$

(on peut toujours supposer qu'il n'y a pas de contrainte sur les règles ayant une constante comme membre gauche).

Sous les même hypothèses que pour la construction du lemme 4.1.10 on crée les règles $f((k_1, i_1)(x_1), \cdots, (k_n, i_n)(x_n)) \xrightarrow{\gamma} (k_\gamma, i_\gamma)(f(x_1, \cdots, x_n))$

k_γ est défini comme dans la précédente construction, et i_γ vaut $(\sup_{j=1}^{n} i_j + 1)$ si la règle est positive et $(\sup_{j=1}^{n} i_j)$ sinon.

On peut alors choisir comme application $\omega' : Q' \longrightarrow [0, \text{Card}(T)]$ telle que $\omega'((k, i)) = i$. La relation d'ordre du treillis T' est $\leq$. □

Lemme 4.1.17 *La classe ACCCB est close par les opérations booléennes : union, intersection, complémentaire.*

PREUVE : On peut utiliser la même preuve que pour le lemme 4.1.11.

- On peut supposer que $\mathcal{A}$ est déterministe et complétement spécifié. Comme dans le cas des automates sans contraintes, on construit le complémentaire de $\mathcal{A}$ en changeant l'ensemble des états finaux. Ceci ne modifie en rien l'application ω.

- Pour montrer que ACCCB est close par union et intersection, on va démontrer qu'elle est close par produit.

  Soient $\mathcal{A}$ et $\mathcal{B}$ deux automates déterministes et complétement spécifiés de la classe ACCCB.

  On pose $\mathcal{A} = (\Sigma, Q_A, Q_A^f, \mathcal{E}_A, \mathcal{R}_A, \omega_A)$ et $\mathcal{B} = (\Sigma, Q_B, Q_B^f, \mathcal{E}_B, \mathcal{R}_B, \omega_B)$

  L'automate produit $\mathcal{A} \times \mathcal{B} = (\Sigma, Q_A \times Q_B, F, \mathcal{E}_A \cup \mathcal{E}_B, \mathcal{R}, \omega)$ construit comme dans la preuve du lemme 4.1.11 est un automate de ACCCB.

En effet, on définit l'application ω de la façon suivante: $\forall (q,q') \in \mathcal{Q}_A \times \mathcal{Q}_B$, $\omega((q,q')) = (\omega_A(q), \omega_B(q'))$. L'ordre est l'ordre produit de $\preceq_A$ et $\preceq_B$: $(q_1,q_2) \preceq (q_1',q_2') \Leftrightarrow (q_1 \preceq_A q_1' \wedge q_2 \preceq_B q_2')$.

On en déduit que $\mathcal{A} \cup \mathcal{B}$ est l'automate $\mathcal{A}' \times \mathcal{B}'$ en prenant comme ensemble d'états finaux $(\mathcal{Q}_A^f \times \mathcal{Q}_B) \cup (\mathcal{Q}_A \times \mathcal{Q}_B^f)$.

De même, $\mathcal{A} \cap \mathcal{B}$ est l'automate $\mathcal{A}' \times \mathcal{B}'$ en prenant comme ensemble d'états finaux $\mathcal{Q}_A^f \times \mathcal{Q}_B^f$.

□

4.2 Automates de filtrage dans les arbres

Nous définissons ici une sous-classe des automates à contraintes closes par contexte et bornées. Maintenant, les propriétés qui nous intéressent sont toutes de la forme "t est facteur de x", que l'on abrégera par $\mathsf{facteur}_t(x)$ où t est un terme quelconque de $T_\Sigma(\mathcal{X})$. Le nombre borné de contraintes de filtrage le long de toute branche nous permet de montrer que le vide est décidable. Nous appelerons cette sous-classe, la classe AFA (Automates de Filtrage dans les Arbres).

4.2.1 Définition et propriétés de la classe AFA

Définition 4.2.1 *La classe AFA est une sous-classe de ACCCB. C'est l'ensemble des automates $\mathcal{A} = (\Sigma, \mathcal{Q}, \mathcal{Q}_f, \mathcal{E}, \mathcal{R}, \omega)$, avec*

- *$\mathcal{E}$ un ensemble fini de propriétés closes par contextes de la forme $\mathsf{facteur}_t(x)$ où t est un terme de $T_\Sigma(\mathcal{X})$ $T_\Sigma \mapsto \mathsf{facteur}_t(s)$ ssi t est un facteur de s.*
- *Soit E l'ensemble des termes apparaissant en indice des contraintes de $\mathcal{E}$. ω est une application de $\mathcal{Q}$ dans 2^E (2^E muni de l'inclusion est un treilli fini)*

  $\forall q \in \mathcal{Q}$, $\omega(q) = \{t_1, \cdots, t_n\}$ *tel que* $t \overset{*}{\vdash}_{\mathcal{A}} q(t) \Leftrightarrow \mathsf{facteur}_{t_1}(t) \wedge \cdots \wedge \mathsf{facteur}_{t_n}(t)$

On appelle *dimension* de l'automate la cardinalité de $\mathcal{E}$, c'est-à-dire le nombre de facteurs que l'automate peut repérer.

La contrainte $\mathbf{facteur}_t$ où t est un terme de $T_\Sigma(\mathcal{X})$ peut se décomposer en une partie "reconnaissable" (au sens des automates standards) et une partie "non reconnaissable". Considérons par exemple le terme $t \equiv b(a(x,y),b(x,y)$. La contrainte $\mathbf{facteur}_t$ est vraie pour le terme clos s si et seulement si

1- $b(a(x,y),a(z,t))$ filtre un sous-terme s' de s

et 2- $s'|_{1.1} = s'|_{2.1}$ et $s'|_{1.2} = s'|_{2.2}$

Le premier point peut être exprimé par un automate standard, sans contrainte. C'est pourquoi, par la suite, la règle $f(q_1(x_1),\cdots,q_n(x_n)) \xrightarrow{c} q(f(x_1,\cdots,x_n))$ avec $c = \mathbf{facteur}_{t_1} \wedge \cdots \wedge \mathbf{facteur}_{t_k} \wedge \neg\mathbf{facteur}_{t_{k+1}} \wedge \cdots \wedge \neg\mathbf{facteur}_{t_l}$ sera représentée par $f(q'_1(x_1),\cdots,q'_n(x_n)) \xrightarrow{c'} q'(f(x_1,\cdots,x_n))$ avec $c' = (e_1,\cdots,e_l)$. Les e_i s'expriment en fonction d'égalités ou de différence entre positions (indices de Dewey). Les états q'_i mémorisent les lettres rencontrées à une profondeur bornée par celle des facteurs à repérer.

EXEMPLE : considérons la contrainte $\mathbf{facteur}_t \wedge \neg\,\mathbf{facteur}_{t'}$, avec $t \equiv b(a(x,y),b(z,t))$ et $t' \equiv b(a(x',x'),b(y',y'))$.

Cette contrainte sera représentée par la règle $b(q_1(x_1),q_2(x_2)) \xrightarrow{(e_1,e_2)} q_3(b(x_1,x_2))$ où q_1 mémorise la lettre a rencontrée au pas précédent et q_2 mémorise la lettre b. $(e_1,e_2) = (1.1 = 2.2 \wedge 1.2 = 2.2\ ,\ 1.1 \neq 1.2 \vee 2.1 \neq 2.2)$.

Grâce à cette remarque, il est facile de voir que l'automate $\mathcal{A} = (\Sigma, Q, Q_f, \mathcal{E}, \mathcal{R}, \omega)$ où $\mathcal{E}$ ne contient que des contraintes de base dont les termes t sont linéaires est un automate standard.

Un exemple complet d'automate de AFA est donné dans la figure 4.2. Pour simplifier l'écriture, on représente les transitions comme des règles de réécriture close. Bien sûr, la transition $b(q,q') \rightarrow q$ est à comprendre comme $b(q(x),q'(y)) \rightarrow q(b(x,y))$.

Lemme 4.2.2 *Soit $\mathcal{A}$ un automate de AFA. On peut construire un automate $\mathcal{A}'$ déterministe et complétement spécifié tel que $L(\mathcal{A}) = L(\mathcal{A}')$.*

PREUVE : C'est une conséquence immédiate du lemme 4.1.16 (même résultat pour la classe ACCCB). □

Lemme 4.2.3 *La classe AFA est close par union, intersection, complémentaire.*

PREUVE : Ce lemme est un simple corollaire du lemme 4.1.17 □

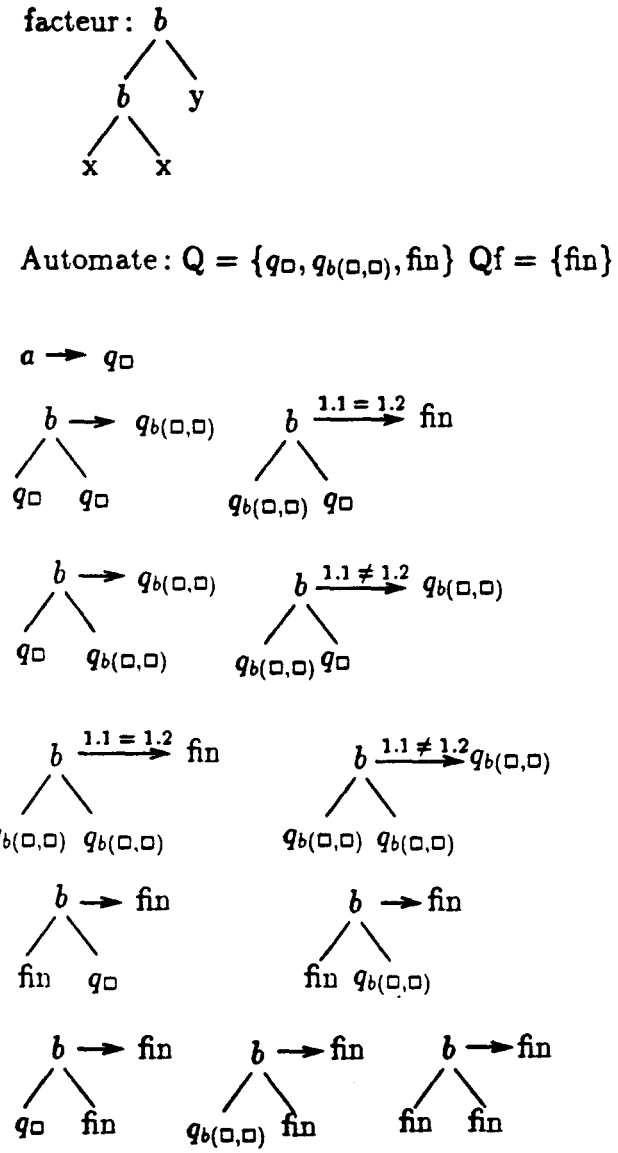


Figure 4.2 : Automate qui recherche le facteur $b(b(x,x),y)$

4.2.2 Décision du vide

Le problème est de décider, pour un automate $\mathcal{A}$ de la classe AFA, si le langage reconnu par cet automate est vide.

La preuve de la décision du vide est fortement inspirée de la preuve de la décidabilité de l'inductive réductibilité de D. Plaisted [Pla85]. Le principe est le suivant : on démontre que si $\mathcal{A}$ reconnaît au moins un terme alors il en reconnaît un de profondeur bornée. La borne s'exprime en fonction des facteurs que l'on peut repérer, et de leur nombre.

Nous allons considérer un terme reconnu par l'automate et minimal pour un certain ordre, dit ordre de pompage. Lorsque l'on pompe sur un terme, il ne faut pas remettre en cause les égalités qui ont été reconnues sur ce terme. Il faut donc associer à une position donnée toutes les positions liées à elle par le test d'une égalité afin de conserver ces égalités en cas de pompage. C'est pourquoi nous définissons et étudions une relation d'équivalence sur les positions des noeuds d'un arbre t de T_Σ. Deux positions seront équivalentes si l'automate vérifie l'égalité des sous-termes en ces positions. Nous bornons la distance séparant deux positions équivalentes, ainsi que la cardinalité des classes d'équivalence. Nous pouvons ensuite définir un ordre sur les termes, dit "ordre de pompage". Intuitivement, un terme t est inférieur à un terme s au sens de cet ordre, si t s'obtient à partir de s par pompage. A partir d'un terme minimal pour cet ordre, nous étudions les nouvelles contraintes qui apparraissent par pompage sur ce terme. Pour terminer, nous construisons un arbre T_θ associé à un chemin θ d'un terme t minimal. En bornant la taille de cet arbre, nous prouvons que la hauteur d'un terme minimal est bornée.

Définitions

Voici quelques définitions utilisées dans la preuve de la décision du vide.

Soit $\mathcal{A}$ un automate de la classe AFA. On le suppose déterministe et complétement spécifié.

- Le degré d'un état est le nombre de contraintes de bases satisfaites c'est-à-dire :

$$\deg(q) = \mathrm{Card}(\omega(q))$$

- A tout noeud de position ν d'un arbre t on peut associer l'état de l'automate quand il atteint ce noeud. On le note état(t,ν).

Comme $\mathcal{A}$ est déterministe, complétement spécifié, état(t,ν) est défini pour tout t et pour tout ν dans t.

- Le degré d'un terme t est égal à deg(état(t,ε)), ε étant la position de la racine.
- Soit H la hauteur maximale des facteurs à repérer.
- Si u est un terme et n un entier naturel, on définit récursivement top(u,n) par

$$\begin{cases} top(f(x_1,\cdots,x_k),1) = f \\ \text{si } n > 1, f \in \sum - \sum_0, \\ \qquad top(f(t_1,\cdots,t_k),n) = f(u_1,\cdots,u_k) \\ \qquad\qquad \text{où } u_i = top(t_i, n-1). \\ \text{si } a \in \sum_0,\ top(a,n) = a \end{cases}$$

top(u,H) = top(u)

- Soit $\mathcal{A}_{top}$ l'automate obtenu à partir de t en mémorisant les tops:

  au noeud de position ν, l'automate $\mathcal{A}_{top}$ mémorise top($t|_\nu$).

- Soit D la dimension de $\mathcal{A}$, D est aussi la dimension de $\mathcal{A}_{top}$.
- On note K le nombre maximal d'égalités testées dans un top, c'est à dire le nombre maximal d'égalités testées par une règle de l'automate.

  Par exemple, la règle $f(q_1(x_1),\cdots,q_n(x_n)) \xrightarrow{c} q$,
  avec $c = ((1 = 1.3) \wedge (2 = 4), (1 = 3.2))$, teste au maximum 3 égalités.

Relation d'equivalence sur les positions

On définit une relation d'équivalence sur les positions d'un terme. Intuitivement, deux positions α et β seront équivalentes si l'égalité $t|_\alpha = t|_\beta$ est vérifiée par l'automate (directement ou indirectement, par propagation des égalités le long des branches et par transitivité de l'égalité). Nous démontrons que la différence de hauteur entre deux noeuds équivalents ainsi que la cardinalité des classes d'équivalence sont bornées en fonction des paramètres D et H de l'automate. Cette relation d'équivalence est définie de la façon suivante:

PREMIÈRE ÉTAPE: On définit la relation $\overset{0}{\sim}_{top}$ telle que deux positions γ et ρ sont en relations si et seulement si l'égalité entre les sous-termes $t|_\gamma$ et t_ρ est testée par l'automate pour appliquer une règle.

1. Soit la relation $\overset{0}{\sim}_{top}$ définie par :

   $\gamma \overset{0}{\sim}_{top} \rho \Leftrightarrow$ il existe une règle de $\mathcal{A}_{top}$ $f(q_1(x_1), \cdots, q_p(x_p)) \xrightarrow{c} q(f(x_1, \cdots, x_p))$ telle que

   $$t \overset{*}{\vdash}_{\mathcal{A}_{top}} t'(f(q_1(t_1), \cdots, q_p(t_p))) \text{ avec } pos(t, f(t_1, \cdots, t_p)) = \alpha$$

   et il existe β, η, η' des positions telles que

   $$\begin{cases} \gamma = \alpha.\beta.\eta & \text{où } \eta \text{ et } \eta' \text{ sont non nulles } (\neq \varepsilon) \\ \rho = \alpha.\beta.\eta' & \text{et } \eta_{(1)} \neq \eta'_{(1)} \end{cases}$$

   $(\beta\eta = \beta\eta')$ étant une égalité appartenant à la condition c.

   La figure 4.3 schématise cette définition.

2. A partir de $\overset{0}{\sim}_{top}$, on définit une autre relation $\overset{0}{\sim}$, par "propagation des égalités" ; deux positions γ et ρ sont en relation si et seulement si $t|_\gamma = t|_\rho$ est conséquence d'une égalité testée par l'automate $t|_{\gamma_0} = t|_{\rho_0}$

   $$\gamma \overset{0}{\sim} \rho \Leftrightarrow (\exists \gamma_0, \rho_0, \beta \ / \ \gamma = \gamma_0.\beta,\ \rho = \rho_0.\beta \text{ et } \gamma_0 \overset{0}{\sim}_{top} \rho_0)$$

   Par exemple, dans la figure 4.4, les positions γ et ρ sont en relation par $\overset{0}{\sim}$

Posons $E_0 = \{(\gamma, \rho) \ / \ \gamma \overset{0}{\sim} \rho\} \cup \{(\gamma, \gamma)\}$

E_0 est le graphe d'une relation symétrique et réflexive.

En effet, $\forall \gamma \ \ (\gamma, \gamma) \in E_0$ et $\forall (\gamma, \rho) \in E_0$, on a $(\rho, \gamma) \in E_0$ car $\overset{0}{\sim}$ est symétrique.

On note cette relation $\overset{0}{\simeq}$ car c'est la clôture réflexive de $\overset{0}{\sim}$.

Lors de cette première étape, nous avons défini une relation symétrique et réflexive $\overset{0}{\simeq}$.

Nous allons maintenant réaliser la clôture transitive de $\overset{0}{\simeq}$.

DEUXIÈME ÉTAPE : clôture transitive de $\overset{0}{\simeq}$.

On pose de manière classique :

$$\forall i \geq 1 \quad E_i = \{(\gamma, \rho) \ / \ \exists \eta \ \begin{matrix} (\gamma, \eta) \in E_{i-1} \\ (\eta, \rho) \in E_{i-1} \end{matrix} \}$$

$E = \cup_{i \geq 0} E_i$

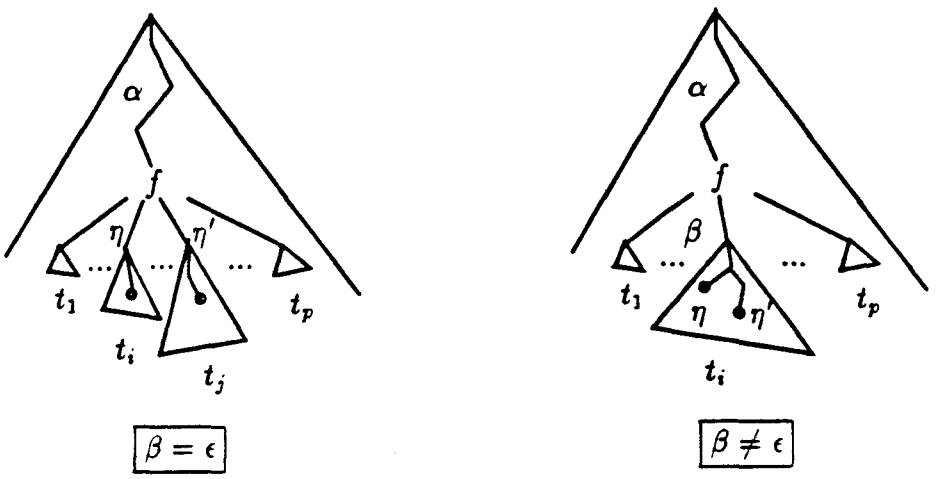


Figure 4.3 : $\gamma \overset{0}{\sim}_{top} \rho$ avec $\gamma = \alpha.\beta.\eta$ et $\rho = \alpha.\beta.\eta'$

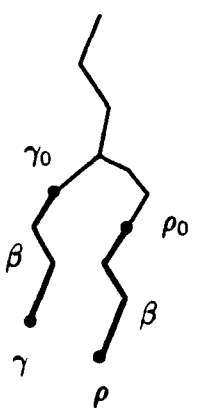


Figure 4.4 : $\gamma \overset{0}{\sim} \rho$

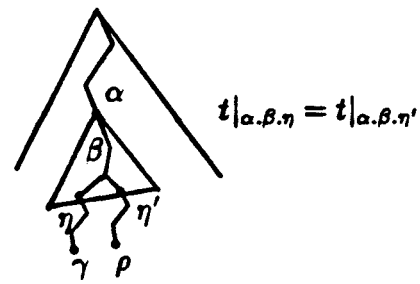


Figure 4.5 : augmentation du degré au noeud de position α

Nous allons prouver que cette construction termine.

Nous posons, par convention, $E_{-1} = \emptyset$

Lemme 4.2.4 *Soit t un terme de degré supérieur ou égal à 1*

$\forall i \in I\!N,\ \forall (\gamma, \rho) \in E_i - E_{i-1},\ \exists \alpha_0,\ \alpha_0 < \gamma,\ \alpha_0 < \rho$ *tel que*

$$deg(t|_{\alpha_0}) > deg(t|_\gamma)+i$$

et $\forall \alpha > \alpha_0,\ deg(t|_\alpha) < deg(t|_{\alpha_0})$

PREUVE :

1. si $i = 0$, $(\gamma, \rho) \in E_0$

   On a alors $\gamma = \alpha\beta\eta$ et $\rho = \alpha\beta\eta'$ d'où $\deg(t|_\alpha) > \deg(t|_\gamma)$ et $\forall \alpha' > \alpha$, $\deg(t|_{\alpha'}) < \deg(t|_\alpha)$.

   Le degré augmente donc strictement en α, ce qui est illustré par la figure 4.5.

2. Supposons que ce soit vrai pour tout $k \leq i-1$, pour i fixé ($i \geq 1$)

   Par hypothèse $(\gamma, \rho) \in E_i - E_{i-1}$, donc $\exists \eta\ \ (\gamma, \eta) \in E_{i-1}$ et $(\eta, \rho) \in E_{i-1}$

   Comme $(\gamma, \eta) \in E_{i-1}$ alors $\exists \alpha_0,\ \alpha_0 < \gamma$ et $\alpha_0 < \eta$ tels que
   $\begin{cases} \deg(t|_{\alpha_0}) > \deg(t|_\eta)+i-1 \\ \text{et} \\ \forall \alpha > \alpha_0\ \deg(t|_\alpha) < \deg(t|_{\alpha_0}) \end{cases}$

De plus, comme $(\eta, \rho) \in E_{i-1}$ on a $\exists \alpha_1$, $\alpha_1 < \eta$ et $\alpha_1 < \rho$ tels que
$\begin{cases} \deg(t|_{\alpha_1}) > \deg(t|_\eta)+i-1 \\ \text{et} \\ \forall \alpha > \alpha_1 \ \deg(t|_\alpha) < \deg(t|_{\alpha_1}) \end{cases}$
On peut supposer que $\alpha_0 < \alpha_1$.
On a alors $\deg(t|_{\alpha_0}) > \deg(t|_{\alpha_1})$ d'où $\deg(t|_{\alpha_0}) \geq \deg(t|_{\alpha_1})+1 > \deg(t|_\eta)+i$
et $\forall \alpha > \alpha_0 \ \deg(t|_\alpha) < \deg(t|_{\alpha_0})$. □

Lemme 4.2.5 *il existe i_0 inférieur ou égal à D tel que $E = E_{i_0}$, D étant la dimension de $\mathcal{A}$.*

PREUVE :

$\forall i \in \mathbb{N}$, s'il existe $(\gamma, \rho) \in E_i - E_{i-1}$, on a $\deg(t|_\epsilon) > \deg(t|_\gamma)+i$.

Or deg(t) $\geq$ D d'où $\deg(t|_\gamma)+i <$ D.

Comme $\deg(t|_\gamma) \geq 0$, on en déduit que $i < D$.

Donc il existe un i_0 tel que $\forall i > i_0, E_i - E_{i-1} = \emptyset$ et $E = E_{i_0}$.

□

Soit $\simeq$ la relation ayant pour graphe E_{i_0}. Cette relation est réflexive, symétrique et transitive (c'est la clôture transitive de $\overset{0}{\simeq}$), c'est donc une relation d'équivalence sur les positions d'un terme t. On note $[\![\gamma]\!]$ la classe d'équivalence de la position γ pour cette relation. Intuitivement, $[\![\gamma]\!]$ est le plus grand ensemble de positions $\{\gamma, \alpha_1, \cdots, \alpha_n\}$ telles que $\forall i \in [1, n]$, l'égalité $t|_\gamma = t|_{\alpha_i}$ a été vérifiée par l'automate (directement ou indirectement, par transitivité de l'égalité).

Nous allons maintenant borner la différence de hauteur entre deux positions équivalentes ainsi que la cardinalité de $[\![\gamma]\!]$.

Lemme 4.2.6 $\forall i \in \mathbb{N}$, $\forall (\gamma, \rho) \in E_i \ \ ||\gamma| - |\rho|| < 2^i.H$.

PREUVE : par récurrence sur i

1. Si $i = 0$:
   (a) si $\gamma = \rho$, c'est trivial car H est non nul

(b) si $\gamma \neq \rho$

i. si $\gamma \overset{0}{\sim}_{top} \rho$ alors $\gamma = \alpha\beta\eta$ et $\rho = \alpha\beta\eta'$ où $\beta\eta = \beta\eta'$ est une condition de c.
Il y a donc égalité entre des positions non nulles d'un facteur à repérer or la hauteur de ces facteurs est strictement inférieure à H.
Donc $||\gamma| - |\rho|| < H$

ii. si non$(\gamma \overset{0}{\sim}_{top} \rho)$ alors $\gamma = \gamma_0\beta$ et $\rho = \rho_0\beta$ avec $\gamma_0 \overset{0}{\sim}_{top} \rho_0$
Donc $||\gamma_0| - |\rho_0|| < H \Rightarrow ||\gamma| - |\rho|| < H$

2. supposons que ce soit vrai pour tout k, $k \leq i-1$

$$(\gamma, \rho) \in E_i \Leftrightarrow \exists \eta \begin{array}{l}(\gamma, \eta) \in E_{i-1} \\ (\eta, \rho) \in E_{i-1}\end{array}$$

Par hypothèse de récurrence, on a:

$$\left.\begin{array}{l}||\gamma| - |\eta|| < 2^{i-1}.H \\ ||\eta| - |\rho|| < 2^{i-1}.H\end{array}\right\} \Rightarrow ||\gamma| - |\rho|| < 2^i.H$$

car $||\gamma| - |\rho|| = ||\gamma| - |\eta| + |\eta| - |\rho||$ et donc $||\gamma| - |\rho|| \leq ||\gamma| - |\eta|| + ||\eta| - |\rho|| < 2 \times 2^i.H$ □

Lemme 4.2.7 *étant données deux positions γ_1, γ_2 dans un arbre t, on a:*

$$\gamma_1 \simeq \gamma_2 \Rightarrow ||\gamma_1| - |\gamma_2|| < 2^D.H$$

PREUVE:

$(\gamma_1, \gamma_2) \in E \Leftrightarrow (\gamma_1, \gamma_2) \in E_{i_0}$ avec $i_0 < D$.

Or $||\gamma_1| - |\gamma_2|| < 2^{i_0}H$ d'où $||\gamma_1| - |\gamma_2|| < 2^D H$.

□

On abrège par la suite $2^D.H$ par L.

Lemme 4.2.8 $\forall \alpha$, *Card*$([\gamma]) \leq (K+1)^D$, *K étant le nombre maximal d'égalités $(\alpha = \beta)$ dans une condition.*

PREUVE :

soit γ une position dans l'arbre t. γ est sur un chemin où l'automate a testé au maximum D fois des égalités, où D est la dimension de l'automate, en des noeuds de positions $\alpha_1, \cdots, \alpha_D$ avec $\alpha_1 > \cdots > \alpha_D$.

On note $[\gamma]_i$ l'ensemble des positions δ telles que $t|_\delta = t|_\gamma$ à cause des contraintes d'égalités imposées par l'application des règles de transition de l'automate jusqu'à la position α_i.

On a donc $[\gamma]_D = [\![\gamma]\!]$.

On montre par récurrence sur i que $\text{card}([\gamma]_i) \leq (K+1)^i$

1. Si $i = 0$, $[\gamma]_0 = \{\gamma\}$, on a donc $\text{card}([\gamma]_0) \leq (K+1)^0$.
2. Supposons que $\text{card}([\gamma]_j) \leq (K+1)^j$, pour j fixé, $j \in [1, D]$, et montrons que $\text{card}([\gamma]_{j+1}) \leq (K+1)^{j+1}$.

   On teste p égalités en α_{j+1} avec $p \leq K$. On a donc en α_{j+1} p égalités $(\beta_i = \beta'_i)$, $i \in [1, p]$ d'où $t|_{\alpha_{j+1}\beta_i} = t|_{\alpha_{j+1}\beta'_i}$, $\forall i \in [1, p]$.

   Soit I le sous-ensemble de $[1, p]$ des indices i tels que $\alpha_{j+1}\beta_i$ ou $\alpha_{j+1}\beta'_i$ est préfixe de γ.

   Supposons que $\gamma \geq \alpha_{j+1}\beta_i$. $\exists \xi_i$, $\gamma = \alpha_{j+1}\beta_i\xi_i$. En posant $\gamma'_i = \alpha_{j+1}\beta'_i\xi_i$, on a $t|_\gamma = t|_{\gamma'_i}$. Or $\text{card}([\gamma]_j) \leq (K+1)^j$ et $\forall i$ $\text{card}([\gamma'_i]_j) \leq (K+1)^j$.

   On a $[\gamma]_{j+1} = [\gamma]_j \cup (\cup_{i \in I}[\gamma'_i]_j)$ donc $\text{card}([\gamma]_{j+1}) \leq (K+1)^j + K \times (K+1)^j \leq (K+1)^{j+1}$

   Comme $[\gamma]_D = [\![\gamma]\!]$, on a bien $\forall \gamma$, $\text{Card}([\![\gamma]\!]) \leq (K+1)^D$.

□

Pompage sur les termes

Définition 4.2.9 (ordre de pompage) .

On définit un ordre strict partiel $\prec_{\mathcal{A}_{top}}$ sur les termes clos. Cet ordre est appelé ordre de pompage et ne dépend que de $\mathcal{A}_{top}$.

Intuitivement, $t \prec_{\mathcal{A}_{top}} t'$ si t se déduit de t' en pompant et $deg(t) = deg(t')$ (cf figure 4.6).

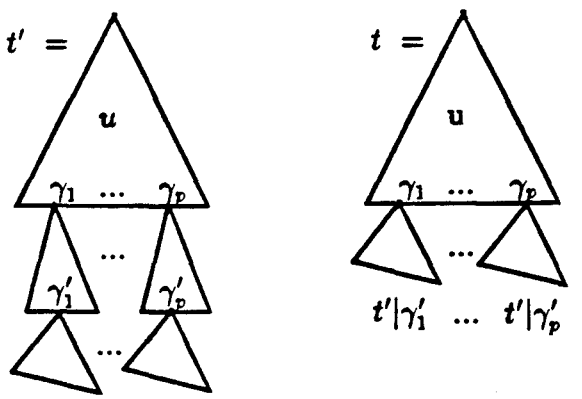


Figure 4.6 : t obtenu par pompage sur t'

Formellement, $t \prec_{\mathcal{A}_{top}} t'$ si et seulement si :

- $t' = u[t'|_{\gamma_1}, \cdots, t'|_{\gamma_p}]$, *où les* γ_i *sont des positions telles que* $[\![\gamma_1]\!] = \{\gamma_1, \cdots, \gamma_p\}$
- *il existe des positions* $\gamma'_1, \cdots, \gamma'_p$ *équivalentes telles que* $\forall i \in [1,p]$ γ_i *est préfixe de* γ'_i *et* $état(t',\gamma_i) = état(t',\gamma'_i)$
- $t = u[t|_{\gamma_1}, \cdots, t|_{\gamma_p}]$ *avec* $\forall i \in [1,p]$ $t|_{\gamma_i} = t'|_{\gamma'_i}$ *et* $\forall \alpha \in u$, $état(t,\alpha) = état(t',\alpha)$.

Par la suite, on représentera ce pompage en écrivant $t = t'[[\gamma_1]] \leftarrow t'|_{\gamma'_1}]$.

Le lemme suivant signifie que si t est un terme minimal au sens de l'ordre de pompage, alors tout pompage sur t crée un terme où apparaissent de nouvelles égalités entre sous-termes, testées par l'automate.

Lemme 4.2.10 (lemme de minimalité) *Soit t un terme minimal selon $\prec_{\mathcal{A}_{top}}$,*

soit γ une position dans t,

soit γ' une position de t telle que $\begin{cases} \exists \xi \neq \varepsilon \ \gamma' = \gamma.\xi \\ état(t,\gamma) = état(t,\gamma') \end{cases}$,

et soit $t' = t[[\gamma]] \leftarrow t|_{\gamma'}]$,

alors il existe α dans $[\![\gamma]\!]$ et η, η_1, η_2 des positions telles que

$$\begin{cases} \eta < \alpha \\ t|_{\eta\eta_1} \neq t|_{\eta\eta_2} \\ t'|_{\eta\eta_1} = t'|_{\eta\eta_2} \end{cases}$$

De plus, on est dans l'un des deux cas suivants :

1) soit $\eta\eta_1 < \alpha$ *et l'égalité entre* $t'|\eta\eta_1$ *et* $t'|\eta\eta_2$ *s'écrit* $c = \uparrow (\eta\eta_1, \eta\eta_2, \alpha, \alpha')$ *pour* $\alpha' = \alpha.\xi$. *On est dans le cas d'une* égalité lointaine *de* α.

2) soit $\eta\eta_1 \geq \alpha$ *et l'égalité entre* $t'|\eta\eta_1$ *et* $t'|\eta\eta_2$ *s'écrit* $c = \downarrow (\eta\eta_1, \eta\eta_2, \alpha, \alpha')$ *pour* $\alpha' = \alpha.\xi$. *Cette égalité est dite* proche *de* α.

Dans le cas général, on écrira $= (\eta\eta_1, \eta\eta_2, \alpha, \alpha')$ au lieu de $\uparrow (\eta\eta_1, \eta\eta_2, \alpha, \alpha')$ ou $\downarrow (\eta\eta_1, \eta\eta_2, \alpha, \alpha')$.

Remarque : si $\alpha = \gamma$ alors on dira que c'est une égalité *concernant* γ sinon c'est une égalité *étrangère à* γ.

PREUVE DU LEMME DE MINIMALITÉ :

Par définition de $\simeq$, $[\![\gamma]\!] = \{\gamma_1, \cdots, \gamma_m\}$. Posons $[\![\gamma']\!] = \{\gamma'_1, \cdots, \gamma'_m\}$ tel que $\forall i \in [1, n]$, $\gamma'_i = \gamma_i.\xi$. Soient $t_1, \cdots, t_m$ les sous-termes de t aux positions $\gamma_1, \cdots, \gamma_m$ et soient $t'_1, \cdots, t'_m$ ceux de t' aux mêmes positions .

$$t \overset{*}{\vdash}_{\mathcal{A}_{top}} u(q_1(t_1), \cdots, q_m(t_m)) \Leftrightarrow t' \overset{*}{\vdash}_{\mathcal{A}_{top}} u(q_1(t'_1), \cdots, q_m(t'_m))$$

car $t'_i = t_i[\gamma_i \leftarrow t|_{\gamma'_i}]$ et $\forall i$ état(t, γ_i)=état(t, γ'_i).

Comme t est minimal, le pompage effectué aux positions $\gamma_1, \cdots, \gamma_m$ a modifié les états lors de la lecture de u par l'automate :

$$\exists \eta \in u, \text{état}(t,\eta) \neq \text{état}(t',\eta)$$

Posons $t|_\eta = f(s_1, \cdots, s_n)$ et $t'|_\eta = f(s'_1, \cdots, s'_n)$

On a donc $t \overset{*}{\vdash}_{\mathcal{A}_{top}} s(f(q'_1(s_1), \cdots, q'_n(s_n))) \vdash_{\mathcal{A}_{top}} s(q(f(s_1, \cdots, s_n)))$

et $t' \overset{*}{\vdash}_{\mathcal{A}_{top}} s(f(q'_1(s'_1), \cdots, q'_n(s'_n))) \vdash_{\mathcal{A}_{top}} s(q'(f(s'_1, \cdots, s'_n)))$

Il y a donc dans $\mathcal{A}_{top}$ les règles suivantes :

$$f(q'_1(x_1), \cdots, q'_n(x_n)) \xrightarrow{c} q$$

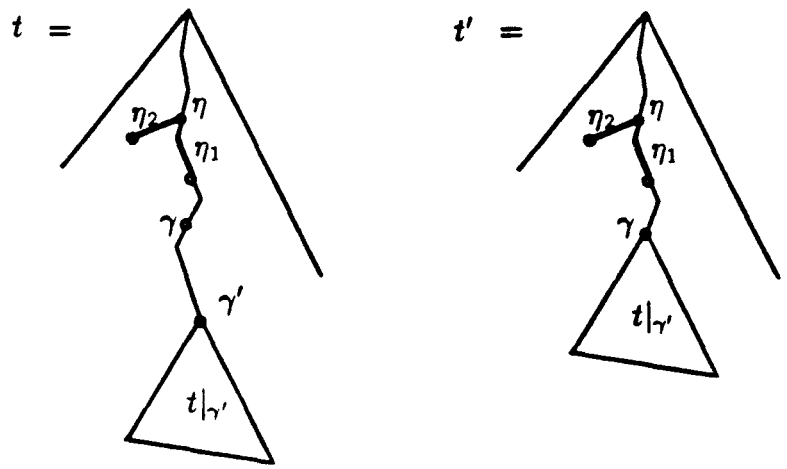


Figure 4.7 : égalité concernant γ, lointaine de γ

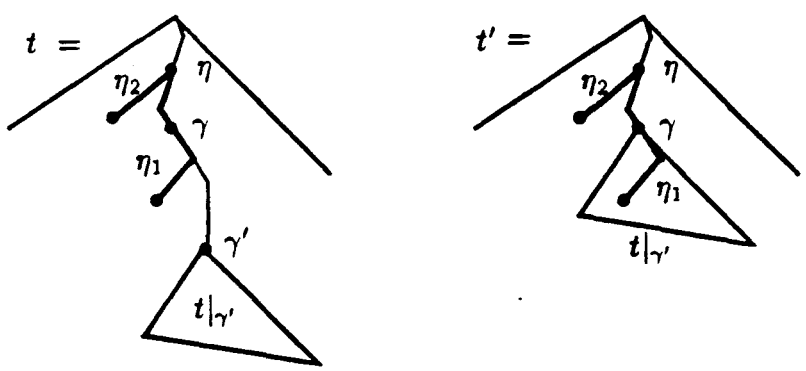


Figure 4.8 : égalité concernant γ, proche de γ

$$f(q'_1(x_1), \cdots, q'_n(x_n)) \xrightarrow{c'} q'$$

où c et c' sont des conditions incompatibles. En fait, par construction des automates de AFA, $c' = (c'_1, \cdots, c'_n)$ et $c = (c_1, \cdots, c_n)$ avec soit $c_i = c'_i$, soit $c_i = \neg c'_i$.

Montrons d'abord que toutes les conjonctions d'égalités vraies dans c le sont aussi dans c' en utilisant une preuve par l'absurde.

Supposons que $(\bigwedge_{i \in I} \beta_i = \beta'_i)$ est une condition élémentaire de c vraie dans t mais fausse dans t' à la position η.

Donc il existe $i \in I$ tel que $t|_{\eta\beta_i} = t|_{\eta\beta'_i}$ et $t'|_{\eta\beta_i} \neq t'|_{\eta\beta'_i}$.

Cela signifie qu'au moins un des deux arbres $t|_{\eta\beta_i}$ ou $t|_{\eta\beta'_i}$ a été modifié par pompage. Supposons que ce soit $t|_{\eta\beta_i}$. On peut en déduire que $\exists \alpha \in [\![\gamma]\!],\ \eta\beta_i < \alpha$. Posons $\alpha = \eta\beta_i\xi$.

- Premier cas : $t|_{\eta\beta'_i}$ n'a pas été modifié par pompage. Cela signifie qu'il n'existe pas $\alpha' \in [\![\gamma]\!],\ \eta\beta_i < \alpha'$.

  Ceci est impossible car $\eta\beta_i \simeq \eta\beta'_i \Rightarrow \eta\beta_i\xi \simeq \eta\beta'_i\xi$ donc on a un élément $\eta\beta'_i\xi$ dans $[\![\gamma]\!]$ suffixe de $\eta\beta'_i$.

- Deuxième cas : $t|_{\eta\beta'_i}$ a été modifié par pompage. On a donc $\exists \alpha' \in [\![\gamma]\!],\ \eta\beta'_i < \alpha'$. On a vu qu'alors $\alpha' = \eta\beta'_i\xi$.

  Donc $t'|_{\eta\beta_i} = t|_{\eta\beta_i}[\alpha \leftarrow t|_{\gamma'}] = t|_{\eta\beta'_i}[\alpha' \leftarrow t|_{\gamma'}] = t'|_{\eta\beta'_i}$ et $t'|_\alpha = t'|_{\alpha'}$. Le pompage a donc conservé les contraintes d'égalité entre $t|_{\eta\beta_i}$ et $t|_{\eta\beta'_i}$ et l'hypothèse de départ était fausse.

Toutes les conjonctions d'égalités vraies dans c sont donc vraies dans c'.

Comme $c' = (c'_1, \cdots, c'_n)$ et $c = (c_1, \cdots, c_n)$ avec soit $c_i = c'_i$, soit $c_i = \neg c'_i$, c' contient une conjonction d'égalités que ne contenait pas c. Puisque toutes les conjonctions d'égalités (i.e. les contraintes de base) dans c sont vraies dans c', il existe $\eta < \alpha$, $\eta \neq \alpha$ et η_1, η_2 des positions telles que :

$$t|_{\eta\wedge\alpha_1} \neq t|_{\eta\eta_2} \text{ et } t'|_{\eta\eta_1} = t'|_{\eta\eta_2}.$$

$|\eta_1| \leq H$ car les conditions apparaissant dans les règles agissent sur des positions de longueur $\leq H$. Donc $|\eta\eta_1| \leq |\eta| + 1 < |\alpha| + H$.

On est donc

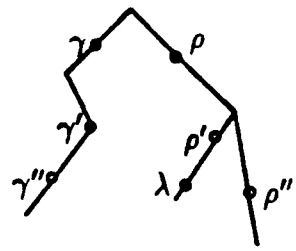


Figure 4.9 : positions équivalentes sur des chemins

- soit dans le cas d'une égalité lointaine de α : $\eta\eta_1 < \alpha$

- soit dans le cas d'une égalité proche de α : $\eta \leq \alpha < \eta\eta_1$ où $\eta\eta_1$ est dans top(α).

□

Dans le cas d'une égalité lointaine de α, notée $\uparrow (\eta\eta_1, \eta\eta_2, \alpha, \alpha')$, on dit que α' est *super-contraint par* α.

Dans le cas d'une égalité proche de α, notée $\downarrow (\eta\eta_1, \eta\eta_2, \alpha, \alpha')$, on dit que α' est *contraint-proche par* α.

Dans l'algorithme de construction de l'arbre T_θ à partir d'un chemin θ dans un terme t, on utilise comme invariant le fait que les noeuds regroupés dans un même ensemble sont sur un même chemin dans t. Comme nous raisonnons avec des classes d'équivalence de positions, il nous faut prouver que le choix de positions sur un même chemin est toujours réalisable. C'est le but du lemme suivant.

Lemme 4.2.11 *Soit t un terme minimal au sens de $\prec_{\mathcal{A}_{top}}$. Soient deux positions γ et ρ équivalentes au sens de $\simeq$. Soient γ' et γ'' deux positions telles que $\gamma'' > \gamma' > \gamma$*

$t[[\gamma]] \leftarrow t_{\gamma'}]$ crée une égalité $= (\eta\eta_1, \eta\eta_2, \rho, \rho')$ avec $\rho < \rho'$

$t[[\gamma]] \leftarrow t_{\gamma''}]$ crée une égalité $= (\xi\xi_1, \xi\xi_2, \rho, \rho'')$ avec $\rho < \rho''$

Alors il existe $\lambda \in [\rho'']$ tel que $\rho' < \lambda$

La figure 4.9 est une illustration de ce lemme.

PREUVE :

Par le lemme précédent, on a $\gamma' \simeq \rho'$ et $\gamma'' \simeq \rho''$.

On a alors $(\gamma' \simeq \rho' \text{ et } \gamma'' > \gamma') \Rightarrow (\exists \lambda \in [\![\gamma']\!] \text{ tel que } \lambda = \rho'\eta \text{ avec } \gamma'' = \gamma'\eta)$

D'où $t|_\lambda = t|_{\gamma''} = t|_{\rho''}$ et donc $\lambda \in [\![\rho']\!]$ et $\rho' < \lambda$.

□

Preuve de la décision du vide

Nous prouvons que le problème du vide est décidable en démontrant que si $\mathcal{A}$ reconnaît au moins un terme, il en reconnaît un de profondeur bornée, minimal au sens de l'ordre de pompage. Dans un premier temps, nous définissons un algorithme qui à tout chemin θ dans un terme t minimal pour $\prec_{\mathcal{A}_{top}}$ associe un arbre T_θ. Puis nous bornons la taille de T_θ en fonction de D, H, K qui ne dépendent que de $\mathcal{A}$ et en déduisons que la hauteur de t est bornée en fonction de ces mêmes paramètres.

Soit θ un chemin de la racine jusqu'à une feuille a. E_θ est l'ensemble de toutes les positions des noeuds de ce chemin, c'est-à-dire Chemin(pos(a)). On peut partitionner E_θ en $E_{q_1}, \cdots, E_{q_m}$ où $\{q_1, \cdots, q_m\} = Q_{\mathcal{A}_{top}}$ de la façon suivante :

$$\forall \gamma \in E_\theta \ (\gamma \in E_q \Leftrightarrow \text{état}(t, \gamma) = q)$$

L'arbre T_θ associé à θ est de la forme $\text{tête}(t_1, \cdots, t_m)$, où **tête** est un nouveau symbole d'arité m et $t_1, \cdots, t_m$ sont des arbres construits en parallèle et dépendants respectivement de $E_{q_1}, \cdots, E_{q_m}$.

Dans l'algorithme suivant, permettant de construire les arbres $t_1, \cdots, t_m$, on a un ensemble E de positions partitionné en $E_1 \oplus \cdots \oplus E_n$. En fait, E est l'ensemble des positions de E_θ non encore traitées.

A chaque composante E_i de la partition de E on associe sa *caractéristique* qui est un ensemble d'égalités entre positions $\{\alpha_1 = \beta_1, \cdots, \alpha_p = \beta_p\}$ que l'on abrégera par $\{c_1, \cdots, c_p\}$.

Si γ est une position de E_θ, $\boxed{\gamma}$ représente un noeud de l'arbre T_θ, labellé par γ (mais pas de position γ).

Si E_i est une composante de la partition de E, $\boxed{\boxed{E_i}}$ représente un noeud de l'arbre T_θ, labellé par l'ensemble E_i. Ces noeuds sont utilisés pendant l'exécution de l'algorithme mais n'apparaissent pas dans les arbres $t_1, \cdots, t_m$ solutions.

ALGORITHME DE CONSTRUCTION DE L'ARBRE T_θ.

Au départ, $E = E_0, E_i = E_{q_i}$ et $m = n$.

1) On recherche le noeud $\gamma \in E$ le plus en haut. De part le partitionnement de E, il existe i tel que $\gamma \in E_i$ avec E_i de caractéristique $c_1, \ldots, c_p$.

2) On fait un pas de construction dans l'arbre.

On crée un noeud $\boxed{\gamma}$ que l'on accroche à la place de $\boxed{\boxed{E_i}}$.

$E_i \leftarrow E_i - \gamma$.

Pour tout noeud γ' dans E_i

l'arbre $t[[\gamma]] \leftarrow t_{\gamma'}]$ contient au moins une égalité testée par l'automate que ne contenait pas l'arbre t de part le lemme de minimalité.

i) si on crée une égalité c lointaine de γ, on ajoute un fils $\boxed{\gamma'}$ à $\boxed{\gamma}$ et $E_i \leftarrow E_i - \gamma'$.

ii) si on crée une égalité c proche de γ, on a une nouvelle contrainte c. Deux cas peuvent se présenter:

*) s'il existe déjà un noeud $\boxed{\boxed{E_{i,k}}}$ fils de $\boxed{\gamma}$ de caractéristique $c_1, \ldots, c_p, c$, alors on ajoute γ' à $\boxed{\boxed{E_{i,k}}}$

*) sinon, on crée un noeud $\boxed{\boxed{E_{i,k'}}}$ fils de $\boxed{\gamma}$ de caractéristique $c_1, \ldots, c_p, c$ et on met γ' dans $\boxed{\boxed{E_{i,k'}}}$.

$E_i \leftarrow E_i - \gamma'$.

iii) s'il n'y a pas d'égalité concernant γ, on a une égalité c étrangère à γ présente dans $t[[\gamma]] \leftarrow t_{\gamma'}]$ et absente dans t.

Il existe donc un noeud $\rho \in [\gamma]$ tel que c soit une égalité concernée par ρ.

On crée, s'il n'existe pas, un noeud $\boxed{\rho}$ fils de $\boxed{\gamma}$.

Comme $\rho \in [\gamma]$ et $\gamma' = \gamma.\xi$, on a donc $\rho' = \rho.\xi$ et $\rho' \in [\gamma']$.

*) si c est une égalité lointaine de ρ, ρ' est supercontraint par ρ.

Si c'est le premier super-contraint en dessous de ρ, on crée $\boxed{\rho'}$ feuille de $\boxed{\rho}$, sinon on recherche sur quel chemin C sont les autres noeuds supercontraints par ρ et on crée un noeud $\boxed{\lambda}$ fils de $\boxed{\rho}$ tel que $\lambda \sim \rho'$ et λ sur le chemin C.

*) si c est une égalité proche de ρ, on a une nouvelle contrainte c.

Si le noeud $\boxed{\boxed{A}}$ de caractéristique c existe déjà, on met dans cet ensemble le noeud λ tel que $\lambda \sim \rho'$ et λ est sur le même chemin que les éléments de $\boxed{\boxed{A}}$

sinon on crée l'ensemble $\boxed{\boxed{A}}$ en fils de $\boxed{\rho}$ et on met dedans ρ'.

$E_i \leftarrow E_i - \gamma'$.

fin Pour.

FIN DE L'ALGORITHME

Invariants :

1) Tous les noeuds $\boxed{\alpha_1}, \ldots, \boxed{\alpha_n}$ supercontraints par un même noeud α et en feuilles de $\boxed{\alpha}$ sont tous sur un même chemin C dans t.

2) Quel que soit l'ensemble A de caractéristique $c_1, \ldots, c_p$, tous les noeuds de A sont sur un même chemin C dans t.

Rappelons que $\mathrm{card}([\![\gamma_1]\!])$ est borné par $(K+1)^D$.

Nous allons maintenant borner la taille de l'arbre T_θ. Pour ce faire, nous allons montrer que le nombre de noeuds créés sous un noeud $\boxed{\gamma}$ est borné.

Lemme 4.2.12 *Soit* $\gamma \in E_{q_i}$.

$Card(\{\gamma' \in E_{q_i} \mid \gamma < \gamma',\ \exists \rho' \in [\![\gamma']\!],\ \exists \rho \in [\![\gamma]\!],\ \rho'$ *supercontraint par* $\rho\})$

$\leq (K+1)^D.K.(L+|\gamma|)$.

PREUVE :

Deux noeuds ρ'_1 et ρ'_2 supercontraints par ρ ne peuvent engendrer une même nouvelle égalité.

Supposons que $\rho < \rho'_1 < \rho'_2$.

Comme ρ'_1 est supercontraint par ρ, il existe ν, ν_1, ν_2 tels que $t[\rho \leftarrow t|_{\rho'_1}]|_{\nu\nu_1} = t|_{\nu\nu_2}$

Si ρ'_2, supercontraint par ρ engendrait la même égalité, on aurait $t[\rho \leftarrow t|_{\rho'_2}]|_{\nu\nu_1} = t|_{\nu\nu_2}$

D'où $t|_{\rho'_1} = t|_{\rho'_2}$ ce qui est impossible car $t|_{\rho'_2}$ est un sous-terme de $t|_{\rho'_1}$.

Ceci signifie que pour ρ fixé, le nombre de ρ' est inférieur ou égal au nombre d'égalités possibles lointaines de ρ.

Il y a $|\rho|$ valeurs possibles pour ν or $|\rho| \leq L + |\gamma|$

Pour chaque ν, il y a K égalités possibles dans top(ν).

Comme il y a $(K+1)^D$ valeurs au maximum dans $[\![\gamma]\!]$, on a donc

Card($\{\gamma' \mid \gamma < \gamma', \exists \rho' \in [\![\gamma']\!], \exists \rho \in [\![\gamma]\!], \rho'$ supercontraint par $\rho\}$)

$\leq (K+1)^D.K.(L + |\gamma|)$. □

Nous affirmons dans l'algorithme que lors d'un pompage créant une égalité proche on obtient une nouvelle contrainte. Le lemme suivant en fait la preuve.

Lemme 4.2.13 *Soit t minimal et soient γ_1, γ_2 des éléments de E_i de caractéritique $\{c_1, \cdots, c_p\}$. On suppose que γ_1 est préfixe de γ_2. Soit $t' = t[\rho_1 \leftarrow t|_{\rho_2}]$ avec $\rho_1 \in [\![\gamma_1]\!]$ et $\rho_2 \in [\![\gamma_2]\!]$. Si t' vérifie une égalité c proche de ρ_1 alors $c \notin \{c_1, \cdots, c_p\}$.*

PREUVE :

La preuve utilise les notations de la figure 4.10.

Supposons qu'il existe une contrainte c' dans E_i telle que $c = c'$. Ceci signifie que $\lambda_2 = \lambda_1$

Par la contrainte c', on a :

$t[\gamma \leftarrow t|_{\gamma_1}]|_{\gamma.\lambda_2} = t|_{\xi.\xi_1} = t[\gamma \leftarrow t|_{\gamma_2}]|_{\gamma.\lambda_2}$

Par la contrainte c, on a :

$t[\rho_1 \leftarrow t|_{\rho_2}]|_{\rho_1.\lambda_1} = t|_{\nu.\nu_2}$

De plus, $\rho_1 \sim \gamma_1$ et $\rho_2 \sim \gamma_2$

On en déduit donc que

$t|_{\rho_1.\lambda_2} = t[\rho_1 \leftarrow t|_{\rho_2}]|_{\rho_1.\lambda_2} = t[\rho_1 \leftarrow t|_{\rho_2}]|_{\rho_1.\lambda_1} = t|_{\nu.\nu_1}$

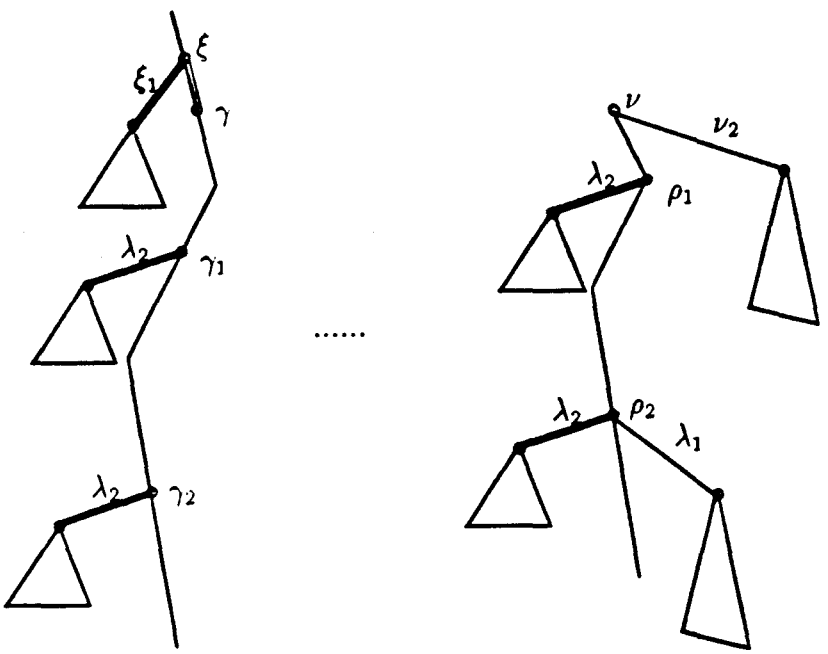


Figure 4.10 : apparition d'une nouvelle contrainte

Donc $t|_{\nu.\nu_2} = t|_{\rho_1.\lambda_2}$ et la contrainte c existerait avant le pompage (dans top(ν)) ce qui contredit le lemme de minimalité.

□

Lemme 4.2.14 *Soit γ fixé dans t.*

Le nombre d'égalités proches de γ est borné par $H.K$.

PREUVE :

Pour une égalité proche de γ, on a : $\exists \nu,\ \nu_1, \nu_2,\ \nu < \gamma < \nu\nu_1$ ou $\nu < \gamma < \nu\nu_2$.

Il y a donc au maximum H valeurs pour ν et pour chaque ν il y a K égalités possibles. □

Lemme 4.2.15 *Soit $\gamma \in E_{q_i}$.*

Card$(\{\gamma' \in E_{q_i} \mid \gamma < \gamma',\ \exists \rho' \in [\![\gamma]\!],\ \exists \rho \in [\![\gamma]\!],\ \rho'$ *contraint-proche par* $\rho\})$

$\leq\ (K+1)^D.K.H.$

PREUVE :

Il y a au maximum $(K+1)^D$ valeurs possibles pour ρ et le nombre d'égalités proches de ρ est borné par $H.K$. □

Nous pouvons maintenant borner le nombre de noeuds internes à l'arbre T_θ, tous ces noeuds étant des noeuds contraints-proches. Le nombre de noeuds contraints-proches est borné en fonction du nombre maximal de contraintes proches dans un top.

Pour un état donné, on a au plus $(K+1)^D - 1$ noeuds équivalents à un γ donné et $(K+1)^D.K.H$ ensembles $\boxed{\boxed{E_i}}$ en dessous de $\boxed{\gamma}$. Comme le nombre de contraintes possibles pour un état donné q_i est inférieur ou égal à K, il y a au maximum K niveaux pour les $\boxed{\boxed{E_i}}$.

Par conséquent le nombre de pas utilisés lors de la construction de E_{q_i} est borné par $[(K+1)^D.K.H]^K$ et le nombre de pas utilisés lors de la construction de T_θ est borné par $N = \text{Card}(Q).[(K+1)^D.K.H]^K$.

Bornons pour terminer le nombre total de noeuds contenus dans T_θ.

On note T_n l'arbre T_θ en cours de construction après n pas de calculs.

Nous allons compter le nombre de noeuds définitifs, c'est-à-dire les feuilles obtenues par les noeuds super-contraints et les noeuds du type $\boxed{\gamma}$.

Au premier tour, le nombre de noeuds créés est borné par $(K+1)^D+1$, γ étant à la racine de l'arbre t et donc aucun noeud n'étant super-contraint par γ.

$||T_1|| \leq (K+1)^D + 1$

Lors du (n+1)-ième pas de construction, on ajoute à l'arbre T_n au maximum $(K+1)^D$ noeuds équivalents $\boxed{\rho}$, $(K+1)^D.K.(L+|\gamma|)$ noeuds super-contraints à ces noeuds et les ensembles provisoires $\boxed{\boxed{E_i}}$.

$||T_{n+1}|| \leq ||T_n|| + (K+1)^D + (K+1)^D.K.(L+|\gamma|)$

Par construction de l'algorithme, γ étant le noeud le plus en haut et non traité sur le chemin θ, on a :

$|\gamma| \leq ||T_n||$

On a donc : $||T_{n+1}|| \leq ||T_n|| + (K+1)^D + (K+1)^D.K.(L+||T_n||)$

soit $||T_{n+1}|| \leq ||T_n||[(K+1)^D.K+1] + (K+1)^D.(K.L+1)$

que nous notons $||T_{n+1}|| \leq A.||T_n|| + B$

N étant le nombre maximal de pas de calculs, le nombre de noeuds de T_θ est donc borné par :

$A^N.||T_1|| + B(A^N-1) \ / \ (A-1)$.

Cette majoration peut être affinée mais notre but est simplement d'en trouver une.

□

On peut enfin énoncer le théorème suivant :

Théorème 4.2.16 (Théorème de Plaisted pour les automates d'arbres) *On peut décider si le langage reconnu par un automate de AFA est vide.*

PREUVE :

Nous avons borné la taille de l'arbre T_θ associé au chemin θ dans le terme t minimal au sens de l'ordre de pompage. Nous avons ainsi borné la hauteur de t. Sachant que,

si l'automate reconnaît un terme, il en reconnaît un minimal, il suffit de tester tous les arbres de hauteur inférieure à la borne pour déterminer si le langage reconnu est vide ou non. □

4.2.3 Résolution des formules d'entourage

Définitions

Nous allons préciser comment produire des formules d'entourage.

ELEMENTS DE BASE

X est un ensemble de variables et Σ est un alphabet fini gradué.

ATOMES

On considère des symboles de prédicats unaires facteur$_t$ où $t \in T_\Sigma(X)$.

$\forall s \in T_\Sigma \cup X$, facteur$_t(s)$ est un atome.

FORMULES D'ENTOURAGE

Tout atome est une formule d'entourage.

Si A et B sont des formules alors $\neg A$, $A \wedge B$, $A \vee B$, $\forall x\ A$, $\exists x\ A$ sont des formules.

Nous allons maintenant expliquer comment calculer la valeur de vérité d'une formule d'entourage sans variable libre.

Résolution des formules d'entourage

Soit φ une formule d'entourage sans variable libre. Pour décider si φ est vraie ou fausse, il faut d'abord la transformer en une autre formule équivalente appelée forme normale prénexe disjonctive. Ensuite, les automates de AFA permettent de décider de la validité de cette nouvelle formule.

1. Transformation de la formule

   - On peut mettre φ sous forme prénexe, c'est-à-dire sous la forme: $Q_1 x_1 \ldots Q_n x_n \varphi'(x_1, \ldots, x_n)$

où $Q_1, \ldots, Q_n$ sont des quantificateurs et $\varphi'(x_1, \ldots, x_n)$ est une formule où toutes les variables sont libres.

- On peut alors transformer φ' en φ'', sa forme normale disjonctive:
  $\varphi''(x_1, \ldots, x_n) = A_1 \vee \ldots \vee A_m$
  où chaque A_i est de la forme:
  $\text{facteur}_{t_1}(x_1^i) \wedge \ldots \wedge \text{facteur}_{t_k}(x_k^i) \wedge \neg\text{facteur}_{t_{k+1}}(x_{k+1}^i) \wedge \ldots \wedge \neg\text{facteur}_{t_l}(x_l^i)$
  avec $\forall j,\ x_j^i \in \{x_1, \ldots, x_n\}$
- On regroupe ensemble dans chaque A_i les atomes ou négations d'atomes qui ont la même variable:
  $A_i = (\text{facteur}_{t_1}(x_1^i) \wedge \ldots \wedge \neg\text{facteur}_{t_{n_1}}(x_1^i)) \wedge \ldots \wedge (\text{facteur}_{t_1}(x_{l_i}^i) \wedge \ldots \wedge \neg\text{facteur}_{t_{n_{l_i}}}(x_{l_i}^i))$
  avec $\forall j,\ x_j^i \in \{x_1, \cdots, x_n\}$ et $j \neq j' \Rightarrow x_j^i \neq x_{j'}^i$
  On notera cela $A_i = B_1^i(x_1^i) \wedge \cdots \wedge B_{l_i}^i(x_{l_i}^i) = \bigwedge_{j=1}^{l_i} B_j^i(x_j^i)$

2. Résolution de $\varphi = Q_1 x_1 \ldots Q_n x_n \bigvee_{i=1}^{m} (\bigwedge_{j=1}^{l_i} B_j^i(x_j^i))$

   La résolution est assez simple et utilise le fait que les prédicats sont unaires.

   - A chaque formule $B_j^i(x_j^i)$ on peut associer un automate de AFA M_j^i tel que $B_j^i(x_j^i)$ est satisfiable si et seulement si $L(M_j^i)$ est différent du vide. M_j^i reconnaît l'ensemble des termes clos qui satisfont la formule:
     $B_j^i(x) = \text{facteur}_{t_1}(x) \wedge \ldots \wedge \text{facteur}_{t_k}(x) \wedge \neg\text{facteur}_{t_{k+1}}(x) \wedge \ldots \wedge \neg\text{facteur}_{t_l}(x)$
     En effet,
     - On construit un automate M_t qui reconnaît l'ensemble des termes qui entourent t grâce à un algorithme de pattern-matching du type "Knuth, Morris et Pratt" [KMP77] dans les arbres, présenté par Hoffmann et O'Donnell [HO82] dans le cas de termes linéaires.
       Donc à tout atome $\text{facteur}_t(x)$ on peut associer un automate M_t tel que pour tout terme clos s, $\text{facteur}_t(s)$ est vrai si et seulement si il existe un état final q de M_t pour lequel $s \overset{*}{\vdash}_{M_t} q$.
     - Grâce à la clôture booléenne de la classe AFA, on peut construire M_j^i à partir de la formule $B_j^i(x)$:
       Pour tout terme clos s, $\text{facteur}_t(s) \wedge \text{facteur}_{t'}(s)$ est vrai si et seulement si il existe un état final q de $M_t \cap M_{t'}$ tel que $s \overset{*}{\vdash}_{M_t \cap M_{t'}} q$.
       De plus, pour tout terme clos s, $\neg\text{facteur}_t(s)$ est vraie si et seulement si il existe un état final q de $\overline{M}_t$ tel que $s \overset{*}{\vdash}_{\overline{M}_t}$.

- grâce aux automates associés aux B_j^i, on peut décider de la validité de la formule $\varphi = Q_1 x_1 \dots Q_n x_n \bigvee_{i=1}^{m} (\bigwedge_{j=1}^{l_i} B_j^i(x_j^i))$
  On utilise l'algorithme suivant, qui transforme φ en ψ :

```
Pour r de 1 à n Faire
   Si Q_r = "∃" Alors
      Pour i de 1 à m Faire
         Si A_i contient un B_j^i(x_j^i) avec x_j^i = x_r alors
            Si B_j^i satisfiable alors on remplace B_j^i par VRAI
            Sinon on remplace A_i par FAUX
            Fin de Si
         Fin de Si
      Fin de Pour
   Fin de Si
Fin de Pour
```

Bien-sûr, B_j^i est satisfiable si et seulement si le langage reconnu par l'automate M_j^i associé n'est pas vide (ce qui est décidable).

Cet algorithme permet de supprimer les quantificateurs existentiels de φ. On obtient une formule ψ équivalente à φ, sans quantificateurs existentiels.

– Si φ ne contenait que des quantificateurs existentiels alors ψ est une formule sans quantificateurs, écrite avec VRAI, FAUX, $\neg$, $\wedge$, $\vee$. On peut donc dire si elle est valide ou non.

– Sinon, ψ ne contient plus que des quantificateurs universels. On va donc chercher à résoudre la formule $\neg\psi$ qui ne contient que des quantificateurs existentiels. Il faut transformer $\neg\psi$ afin d'obtenir une forme normale prenexe disjonctive et appliquer l'algorithme ci-dessus permettant de savoir si $\neg\psi$ est valide. On en déduit alors immédiatement la valeur de vérité de ϕ.

Index

Bibliographie

[Bog90] B. Bogaert. *Automates d'arbres avec tests d'égalités.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, 1990.

[BT92] B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In *Lecture Notes in Computer Science Vol 577*, pages 161–171. Symposium on theoretical aspects of computer science, 1992.

[Car91] A.C. Caron. Linear bounded automata and rewrite systems: Influence of initial configurations on decisions properties. In *Lecture Notes in Computer Science Vol 493*, pages 74–89. Colloquium on Trees in Algebra and Programming, 1991.

[Car92] A.C. Caron. Decidability of reachability and disjoint union of term rewriting systems. In *Lecture Notes in Computer Science Vol 581*, pages 86–101. Colloquium on Trees in Algebra and Programming, 1992.

[CCD93] 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.

[CG90] J.L. Coquidé and R. Gilleron. Proofs and reachability problems for rewrite systems. In *Lecture Notes in Computer Science.* IMYCS 90, 1990.

[CN71] Y. Cochet and M. Nivat. Une généralisation des ensembles de Dyck. *J. Math*, 9:389–395, 1971.

[Com88] H. Comon. *Unification et disunification. Théorie et applications.* PhD thesis, I.N.P.Grenoble, 1988.

[Com90] H. Comon. Equational formulas on order-sorted algebras. In *Proceedings of ICALP'90*, pages 674–688, 1990.

[Com92] H. Comon. Résolution de contraintes dans les algèbres de termes. Master's thesis, L.R.I. Université de Paris-Sud, 1992. Rapport d'Habilitation.

[Dau89] M. Dauchet. Simulation of Turing machines by a left-linear rewrite rule. In *Lecture Notes in Computer Science Vol 355*, pages 109–120. Rewriting Techniques and Applications, 1989.

[DD89] M. Dauchet and A. Deruyver. VALERIAN: Compilation of ground term rewriting systems and applications. In *Lecture Notes in Computer Science Vol 355*, pages 556–558. Rewriting Techniques and Applications, 1989.

[Der85] N. Dershowitz. Computing with rewrite systems. *Information and Control*, 65:122–157, 1985.

[Der87] N. Dershowitz. Termination of rewriting. *J. Symbolic Computation*, 3:69–116, 1987.

[DG89] A. Deruyver and R. Gilleron. Compilation of term rewriting systems. In Diaz and Orejas, editors, *Lecture Notes in Computer Sciences Vol 351*, pages 227–243. CAAP'89, 1989.

[DHLT87] M. Dauchet, T. Heuillard, P. Lescanne, and S. Tison. Decidability of the confluence of ground term rewriting systems. In *Proceeding of LICS*, pages 353–360. IEEE Computer Society Press, 1987.

[DJ90] N. Dershowitz and J.P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter Rewrite Systems, pages 243–320. Elsevier, 1990.

[Dro89] K. Drosten. Termersetzungssysteme. Informatik-Fachberichte 210, Springer, Germany, 1989.

[DT85] M. Dauchet and S. Tison. Tree automata and decidability in ground term rewriting systems. In *Lecture Notes in Computer Sciences Vol 199*, pages 80–84. Fundamentals of Computation Theory, 1985.

[DT90] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. Rapport interne I.T. 182, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, 1990.

[FS92] C. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. Litp 92.26, Laboratoire d'Informatique Théorique et Programmation, Paris, France, 1992.

[Gra91] B. Gramlich. A structural analysis of modular termination of term rewriting systems. Seki-Report SR-91-15, Universitat Kaiserslautern, Germany, 1991.

[GS84] F. Gecseg and M. Steinby. *Tree Automata.* Akademiai Kiado, 1984.

[HL78] G. Huet and D.S. Lankford. On the uniform halting problem for term rewriting systems. Rapport Laboria 283, Institut de Recherche en Informatique et en Automatique, Le Chesnay, France, 1978.

[HO82] C. Hoffmann and M. O'Donnell. Pattern matching in trees. *J.A.C.M.*, 29(1):68–95, January 1982.

[Hoo66] P. K. Hooper. The undecidability of the Turing machine immortality problem. *J. Symbolic Logic*, 31(2), 1966.

[HU67] J.E. Hopcroft and J.D. Ullman. Some results on tape-bounded Turing machines. *J.A.C.M.*, 16(1):168–177, 1967.

[Hue80] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting system. *J.A.C.M.*, 27(4):797–821, 1980.

[JK89] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82(1):1–33, July 1989.

[KB70] D. Knuth and P. Bendix. *Computational Problems in Abstract Algebra*, chapter Simple Word Problems in Universal Algebras, pages 263–297. Pergamon Press, 1970.

[Klo91] J.W. Klop. *Handbook of Logic in Computer Science*, volume I, chapter Term Rewriting Systems. Oxford University Press, 1991. To appears.

[KMP77] D. Knuth, J. Morris, and V. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

[KMTdV91] J.W. Klop, A. Middeldorp, Y. Toyama, and R. de Vrijer. A simplified proof of Toyama's theorem. technical report CS-R9156, CWI, Amsterdam, 1991.

[KNZ87] D. Kapur, P. Narendran, and H. Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24:395–415, 1987.

[KO90a] M. Kurihara and A. Ohuchi. Modularity of simple termination of term rewriting systems. *Journal of IPS Japan*, 31(5):633–642, 1990.

[KO90b] M. Kurihara and A. Ohuchi. Modularity of simple termination of term rewriting systems with shared constructors. Technical Report SF-36, Hokkaido University, Sapporo, 1990.

[Koz77] D. Kozen. Complexity of finitely presented algebras. In *9th ACM Symposium on Theory of Computing*, pages 164–177, 1977.

[Mah88] M.J. Maher. Complete axomatizations of the algebras of finite, rational and infinite trees. In *Proc. 3rd IEEE Symposium of Logic in Computer Science*, pages 348–357, July 1988.

[Mid89a] A. Middeldorp. Modular aspects of properties of term rewriting systems related to normal forms. In *Lecture Notes in Computer Science Vol 355*, pages 263–277, Chapel Hill, 1989. 3rd International Conference on Rewriting Techniques and Applications. Full version: Report IR-164, Vrije Universiteit, Amsterdam, 1988.

[Mid89b] A. Middeldorp. A sufficient condition for the termination of the direct sum of term rewriting systems. In *Proceeding of the 4th IEEE Symposium on Logic in Computer Science*, pages 396–401, Pacific Grove, 1989.

[Mid90] A. Middeldorp. *Modular Properties of Term Rewriting Systems.* PhD thesis, Vrije Universiteit, Amsterdam, 1990.

[Mid91] A. Middeldorp. Modular properties of conditional term rewriting systems. Report CS R9105, Centre for Mathematics and Computer Science, Amsterdam, 1991.

[Mon81] J. Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG.* PhD thesis, Laboratoire d'Informatique Fondamentale

de Lille, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, 1981.

[MT91] A. Middeldorp and Y. Toyama. Completeness of combinations of constructor systems. In *Lecture Notes in Computer Science Vol 488*, pages 188–199. Conference on Rewriting Techniques and Applications, 1991.

[Myh60] J. Myhill. Linear-bounded automata. WADD Tech. Note 60-165, Wright-Patterson Air Force Base, Ohio, 1960.

[NB70] M. Nivat and M. Benois. Congruences parfaites et quasi-parfaites. In *Proc. 2nd ACM Symposium on Theory and Computing*, pages 221–225, 1970.

[New42] M.H.A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223–243, 1942.

[Ott86] F. Otto. The undecidability of self-embedding for finite semi-Thue and Thue systems. *Theoretical Computer Science*, 47:225–232, 1986.

[Ott87] F. Otto. On deciding the confluence of a finite string-rewriting system on a given congruence class. *J. Comput. System Sciences*, 35:285–310, 1987.

[Oya86] M. Oyamaguchi. The reachability problem for quasi-ground term rewriting systems. *Journal of Information Processing*, page 9, 1986.

[Oya87] M. Oyamaguchi. The Church-Rosser property for ground term rewriting systems is decidable. *TCS*, 49:43–79, 1987.

[Pla85] D.A. Plaisted. Semantic confluence tests and completion method. *Information and Control*, 65:182–215, 1985.

[Plo72] G. Plotkin. Building in equational theories. *Machine Intelligence*, 7:73–90, 1972.

[Pos46] E.L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52:264–268, 1946.

[Rab69] M.O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[Rab77] M.O. Rabin. *Handbook of Mathematical Logic*, chapter Decidable theories, pages 595–627. North Holland, 1977.

[Rus87] M. Rusinowitch. On termination of the direct sum of term rewriting systems. *Information Processing Letters*, 26:65–70, 1987.

[Tho90] W. Thomas. *Handbook of Theoretical Computer Science*, volume B, chapter Automata on Infinite Objects, pages 134–191. Elsevier, 1990.

[TKB89] Y. Toyama, J.W. Klop, and H.P. Barendregt. Termination for the direct sum of left-linear term rewriting systems. In *Lecture Notes in Computer Science Vol 355*, pages 477–491, Chapel Hill, 1989. 3rd International Conference on Rewriting Techniques and Applications.

[Toy87a] Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.

[Toy87b] Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.

[JK86] J.P. Jouannaud and E. Kounalis. Automatic progs by induction in equational theories without constructors. In Proc. 1st IEEE Symp. Logic in Computer Science, June 1986.

[KO92] M. Kurihara and A. Ohuchi. Non-Copying Term Rewriting and Modularity of Termination. Technical Report. Hokkaido University. 1992.