50376
1990
263

USTL
FLANDRES ARTOIS

LIFL U.A 369 du C.N.R.S

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

N° d'ordre : 646

THESE

Nouveau Régime
présentée à
l'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS
pour obtenir le titre de

DOCTEUR EN INFORMATIQUE

par

Jean-Luc COQUIDE

CONTROLES ET PREUVES DANS LES SYSTEMES CLOS AUTOMATES à PILES D'ARBRES ET CALCUL DE FORMES NORMALES

Soutenue le 12 Décembre 1990 devant la commission d'Examen

PRESIDENT:	M. NIVAT
RAPPORTEURS:	H. KIRCHNER
	J. BERSTEL
DIRECTEUR DE THESE:	M. DAUCHET
EXAMINATEURS:	H. COMON
	M. LATTEUX
	S. TISON

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE
FLANDRES ARTOIS

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. PAREAU, J. LOMBARD, M. MIGEON, J. CORTOIS.

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

M. A. DUBRULLE.

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CONSTANT Eugène	Electronique
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
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	Probabilitéss
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BREZINSKI Claude	Analyse Numérique

M. BRIDOUX Michel	Chimie-Physique
M. CELET Paul	Géologie Générale
M. CHAMLEY Hervé	Géotechnique
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. DAUCHET Max	Informatique
M. DEBOURSE Jean-Pierre	Gestion des Entreprises
M. DHAINAUT André	Biologie Animale
M. DOUKHAN Jean-Claude	Physique du Solide
M. DYMENT Arthur	Mécanique
M. ESCAIG Bertrand	Physique du Solide
M. FAURE Robert	Mécanique
M. FOCT Jacques	Métallurgie
M. FRONTIER Serge	Ecologie Numérique
M. GRANELLE Jean-Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. LABLACHE-COMBIER Alain	Chimie Organique
M. LACOSTE Louis	Biologie Végétale
M. LAVEINE Jean-Pierre	Paléontologie
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique Atomique et Moléculaire
M. LEROY Jean-Marie	Spectrochimie
M. LHOMME Jean	Chimie Organique Biologique
M. LOMBARD Jacques	Sociologie
M. LOUCHEUX Claude	Chimie Physique
M. LUCQUIN Michel	Chimie Physique
M. MACKE Bruno	Physique Moléculaire et Rayonnements Atmosph.
M. MIGEON Michel	E.U.D.I.L.
M. PAQUET Jacques	Géologie Générale
M. PETIT Francis	Chimie Organique
M. POUZET Pierre	Modélisation - calcul Scientifique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. SALMER Georges	Electronique
M. SCHAMPS Joel	Spectroscopie Moléculaire
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. TILLIEU Jacques	Physique Théorique
M. TOULOTTE Jean-Marc	Automatique
M. VIDAL Pierre	Automatique
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ALLAMANDO Etienne	Composants Electroniques
M. ANDRIES Jean-Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BART André	Biologie animale
M. BASSERY Louis	Génie des Procédés et Réactions Chimiques

Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique Atomique et Moléculaire
M. BERTRAND Hugues	Sciences Economiques et Sociales
M. BERZIN Robert	Analyse
M.BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie Végétale
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie Civil
M. BOIVIN Jean-Claude	Spectroscopie
M. BOUQUELET Stéphane	Biologie Appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BRASSELET Jean-Paul	Géométrie et Topologie
M. BRUYELLE Pierre	Géographie
M. CAPURON Alfred	Biologie Animale
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 Electroniques Optiques
M. COMYN Gérard	Informatique Théorique
M. COQUERY Jean-Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques et Sociales
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique Nucléaire et Corpusculaire
M. COUTURIER Daniel	Chimie Organique
M. CRAMPON Norbert	Tectolique Géodynamique
M. CROSNIER Yves	Electronique
M. CURGY Jean-Jacques	Biologie
Melle DACHARRY Monique	Géographie
M. DEBRABANT Pierre	Géologie Appliquée
M. DEGAUQUE Pierre	Electronique
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELAHAYE Jean-Paul	Informatique
M. DELORME Pierre	Physioliogie Animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
M. DENEL Jacques	Informatique
M. DE PARIS Jean Claude	Analyse
M. DEPREZ Gilbert	Physique du Solide - Cristallographie
M. DERIEUX Jean-Claude	Microbiologie
Melle DESSAUX Odile	Spectroscopie de la réactivité Chimique
M. DEVRAINNE Pierre	Chimie Minérale
Mme DHAINAUT Nicole	Biologie Animale
M. DHAMELINCOURT Paul	Chimie Physique
M. DORMARD Serge	Sciences Economiques
M. DUBOIS Henri	Spectroscopie Hertzienne
M. DUBRULLE Alain	Spectroscopie Hertzienne
M. DUBUS Jean-Paul	Spectrométrie des Solides
M. DUPONT Christophe	Vie de la firme (I.A.E.)
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FAUQUAMBERGUE Renaud	Composants électroniques

M. FONTAINE Hubert	Dynamique des cristaux
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie Sturcturale
M. GAMBLIN André	Géographie urbaine, industrielle et démog.
M. GLORIEUX Pierre	Physique moléculaire et rayonnements Atmos.
M. GOBLOT Rémi	Algèbre
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie Physique
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. GUILBAULT Pierre	Physiologie animale
M. HENRY Jean-Pierre	Génie Mécanique
M. HERMAN Maurice	Physique spatiale
M. HOUDART René	Physique atomique
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et Statistiques
M. Jean Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme (I.A.E.)
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KREMBEL Jean	Biochimie
M. LANGRAND Claude	Probabilités et statistiques
M. LATTEUX Michel	Informatique
Mme LECLERCQ Ginette	Catalyse
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
Melle LEGRAND Solange	Algèbre
M. LEGRAND Pierre	Chimie
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne
M. LE MAROIS Henri	Vie de la firme (I.A.E.)
M. LEROY Yves	Composants électroniques
M. LESENNE Jacques	Systèmes électroniques
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique théorique
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU Jean-Marie	Optique-Physique atomique
M. MAIZIERES Christian	Automatique
M. MAURISSON Patrick	Sciences Economiques et Sociales
M. MESMACQUE Gérard	Génie Mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie Organique
M. MORTREUX André	Chimie Organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
Mme MOUYART-TASSIN Annie Françoise	Informatique
M. NICOLE Jacques	Spectrochimie
M. NOTELET Françis	Systèmes électroniques
M. PARSY Fernand	Mécanique

M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée
M. STEEN Jean-Pierre	Informatique

Je remercie Maurice NIVAT de l'honneur qu'il me fait en présidant ce jury.

Je remercie Jean BERSTEL et Hélène KIRCHNER d'avoir aimablement accepté d'être rapporteurs de cette thèse. Je les remercie de l'intérêt qu'ils ont montré pour ce travail et des commentaires et suggestions qu'ils ont pu faire.

Je remercie Hubert COMON d'avoir accepté de participer à ce jury.

Je suis reconnaissant à Michel LATTEUX de me faire l'amitié de participer à ce jury. Il m'a initié à la recherche et m'a incité à poursuivre dans cette voie.

Sophie TISON a toujours été disponible pour participer activement à ma recherche. Je la remercie à la fois de son amabilité et de tous ses conseils judicieux prouvant sa grande compétence.

Je voudrais exprimer toute ma gratitude à Max DAUCHET qui a été l'âme de cette recherche. Grâce à sa gentillesse et à son savoir, il a toujours su m'apporter le soutien efficace dont j'avais besoin tant du point de vue humain que du point de vue de mes travaux. Effectuer ma recherche sous sa direction fut enrichissant à plus d'un titre.

Je voudrais aussi remercier Rémi GILLERON avec qui j'ai effectué toutes mes études en Informatique. Sa présence et son soutien m'ont aidé à vaincre certaines difficultés qui ont pu se présenter.

Je tiens à remercier Henri GLANC qui a assuré la réalisation matérielle de cette thèse avec la compétence et la gentillesse bien connues et appréciées de tous.

TABLE DES MATIERES

Avant Propos

Une partie de ce travail a été réalisée en commun avec R.Gilleron et l'approche globale est la même dans les deux thèses. La structure de l'introduction de cette thèse reflète cette collaboration. En effet, la partie générale de l'introduction et la partie relative au travail commun sur les automates à piles sont les mêmes dans les deux thèses. Par contre, les parties relatives à la simulation de systèmes par des systèmes clos à garde rationnelle et à l'obtention de preuves de dérivation sont propres à cette thèse.

INTRODUCTION

La réécriture est, à plusieurs titres, un paradigme bien connu de la programmation. En programmation fonctionnelle (comme LISP), un interprète peut être vu naturellement comme un système de réécriture; en programmation équationnelle (comme OBJ), la réécriture permet de calculer des représentants canoniques de types spécifiés par des équations; en programmation logico-fonctionnelle, elle intervient dans l'unification modulo par surréduction. Notre approche se situe en amont de la programmation. Notre but est de contribuer à mieux comprendre les structures algébriques complexes manipulées. Nous participons, en celà, à un courant récent de recherche (citons, parmi d'autres, R.V.Book et J.H.Gallier aux USA, M.Oyamaguchi au Japon, F.Otto et R.Treinen en Allemagne, K.Salomaa en Finlande, H.Comon en France,...).

Nous commencerons, néanmoins, par introduire la notion de réécriture à travers la problématique classique des théories équationnelles avant de passer à l'aspect algébrique et de présenter nos propres résultats.

****

L'idée de simplifier des égalités est partout présente en algèbre. La réécriture formalise cette idée en remplaçant des égaux par des égaux plus simples.

Dans le cadre des *théories équationnelles*, le résultat le plus célèbre, qui est une des bases du développement actuel de la réécriture, est le *théorème de Knuth et Bendix* ([KNBE70]). Il s'agit de vérifier si une égalité u=v donnée est conséquence d'un ensemble E d'équations données (axiomes). Pour celà, on oriente, selon un certain critère, les équations de E en règles de réécriture. On calcule alors les paires critiques, obtenues en réécrivant de deux façons différentes, qui se chevauchent, un membre gauche de règle. Si une paire critique n'est pas convergente, c'est-à-dire si les deux termes se réduisent en deux termes irréductibles distincts, une nouvelle règle est ajoutée, en orientant si possible la paire critique selon le critère préalablement choisi. Cette construction est récursive. Le succès de cet algorithme nécessite la propriété de terminaison du système de réécriture.

On obtient par cette procédure, lorsqu'elle converge, un système de réécriture convergent S (tout terme possède une forme normale unique pour S) qui *fournit un algorithme de décision pour la théorie équationnelle d'ensemble d'axiomes E.*

Il suffit, en effet, pour prouver qu'une égalité u=v est conséquence des équations de E, de vérifier que les formes normales de u et v pour S sont égales.

Un exemple algébrique classique est:
à partir du système d'équations (E) suivant,

$0 + x = x$,
$(-x) + x = 0$,
$(x + y) + z = x + (y + z)$,

on oriente les règles de gauche à droite (en utilisant une notion de poids d'un arbre que nous ne détaillons pas ici) et on obtient le système de réécriture convergent constitué des trois règles précédentes et des règles,

$(-x) + (x + y) \rightarrow y$;
$x + 0 \rightarrow x$;
$-0 \rightarrow 0$;
$-(-x) \rightarrow x$;
$x + (-x) \rightarrow 0$;
$x + ((-x) + y) \rightarrow y$;
$-(x + y) \rightarrow (-y) + (-x)$.

Cet algorithme fait émerger l'importance des propriétés de terminaison et de confluence en réécriture.

Un système de réécriture est *noethérien* (termine) s'il n'existe pas de calcul infini. La terminaison est indécidable même dans le cas où les termes apparaissant dans les règles sont des arbres filiformes ([HULA78]) ou dans le cas où le système est composé d'une seule règle linéaire à gauche ([DAUC89]). Le problème de l'arrêt d'un système de réécriture de mots composé d'une seule règle reste ouvert. Une étude de synthèse recouvrant les problèmes de terminaison se trouve dans [DERS87].

Un système de réécriture est *confluent* si, pour toute réécriture d'un terme de deux façons différentes, on peut réécrire ces deux termes en un même terme. La confluence est en général indécidable, même dans le cas où les termes apparaissant dans les règles sont des arbres filiformes ([BJW81]); elle est décidable pour les systèmes de réécriture noethériens ([KNBE70], grâce au théorème de Newman [NEWM42])), et aussi pour les systèmes de réécriture clos (sans variables) même sans l'hypothèse de terminaison ([DATI85]).

Les algorithmes de complétion ont été largement étudiés (L.Bachmair et N.Dershowitz [BADE86], N.Dershowitz [DERS89] pour des

articles de synthèse) et implantés (REVE [LESC83], FORMEL [FAGE84], KADS [STIC86], RRL [KAZH88]). Il existe cependant des systèmes de réécriture noethériens et non confluents qui ne peuvent être complétés en un nombre fini d'étapes ([KANA85]).

Les notions de confluence et de terminaison ont été étendues à des sytèmes dont certaines égalités ne peuvent être orientées, comme la commutativité, des algorithmes de complétion, essentiellement dans le cas Associatif et Commutatif, ont été étudiés et implantés. Citons, entre autres, D.S.Lankford et A.M.Ballantyne [LABA77], G.Huet [HUET80], G.E.Peterson et M.E.Stickel [PEST81], J.P.Jouannaud et H.Kirchner [JOKI86], L.Bachmair et N.Dershowitz [BADE87], H.Kirchner [KIRC85], C.Kirchner [KIRC85].

L'exemple précédent était tiré de la tradition algébrique. En programmation, on retrouve souvent des spécifications du type suivant:

Soit Σ={0, succ, +, *, exp}, C={0, succ} et F={+, *, exp} et soit (E') le système d'équations défini par:

+(0, x) = x;
+(succ(x), y) = succ(+(x, y));
*(0, x) = 0
*(succ(x), y) = +(*(x, y), y);
exp(0, y) = 1;
exp (succ(x), y) = *(exp(x, y), y).

Soit le système de réécriture S' obtenu en orientant les équations de la gauche vers la droite. Ce système de réécriture est noethérien et *régulier* (il est linéaire à gauche et sans paires critiques). En terme de spécification, on est dans le cas type entier avec les constructeurs 0 et succ (ensemble C) et la *spécification E' est suffisamment complète par rapport à C*, en ce sens que, tout terme clos peut être prouvé égal à un terme clos s'écrivant uniquement à l'aide des éléments de C. Ce caractère régulier est assez naturel en programmation, un cas particulier évident de cette situation étant la programmation fonctionnelle.

Le système S' étant régulier et noethérien, on sait que dans ce cas, il est aussi confluent et, par conséquent, on dispose d'un calcul de formes normales permettant de vérifier la validité de toute égalité. Mais, on peut plus largement chercher à vérifier des formules dites du *premier ordre* comme:

$\forall n>2, \nexists (x,y,z)\neq(0,0,0)\ x^n + y^n = z^n$.(théorème de Fermat)

Cette formule amène à résoudre un problème ouvert, célèbre s'il en est, et illustre toute la différence de problématique entre la décidabilité d'une théorie équationnelle et la décidabilité d'une formule du premier ordre.

Une formule du premier ordre est une formule qui s'écrit à l'aide des quantificateurs, des connecteurs logiques, d'opérateurs et de prédicats définis. *Une théorie du premier ordre est dite décidable* si on peut décider de la validité de toute formule du premier ordre pour cette théorie.

Par exemple, l'arithmétique de Presburger (opérateurs 0, succ et + avec leurs axiomes de définition et pour prédicat l'égalité) est décidable. Lorsqu'une théorie est décidable, le point de vue pessimiste est de dire "la théorie est pauvre"; le point de vue optimiste est de dire "on sait tout faire", mais, en général, même si on sait tout faire, on a souvent des hiérarchies de complexité irréalistes avec des tours d'exponentielles.

Mais, on sait également qu'une théorie est vite indécidable. Il suffit par exemple d'ajouter l'opérateur * et ses axiomes de définition à l'arithmétique de Presburger pour obtenir une théorie indécidable. Lorsqu'une théorie est indécidable, on peut se demander ce qu'il en est pour un problème donné ou pour des fragments de la théorie (expressions d'une certaine forme).

Nous avons, jusqu'à présent, toujours considéré des termes avec variables, nous allons maintenant montrer l'intérêt de considérer des *termes clos*, des *substitutions closes*.

D'un point de vue sémantique, l'avantage est que l'on dispose alors d'une algèbre initiale dans le cas des théories équationnelles. On parle alors de la *théorie inductive* d'ensemble d'axiomes E.

On définit la notion de *confluence close* à partir de la notion de confluence en se limitant aux termes clos. Il est évident qu'un système de réécriture confluent est confluent sur les termes clos. La confluence des systèmes de réécriture noethériens est décidable alors que la confluence sur les termes clos des systèmes de réécriture noethériens est indécidable.

Précisons sur l'exemple suivant ce qu'est un théorème inductif:
$\Sigma=\{0, succ\}$ et $E"=\{succ(succ(0)) = 0\}$.

Le théorème succ(succ(x)) = x n'est pas un théorème équationnel mais est un théorème inductif (on peut le prouver par une induction structurelle 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 u=v à démontrer à l'ensemble E et à vérifier que les relations de congruence sur les termes clos définies par E, et, E auquel on a ajouté le théorème sont les mêmes. On peut prouver ce résultat en prouvant que E est confluent sur les termes clos et que les termes u et v peuvent se réécrire en un même terme pour toute substitution close ([MUSS80], [HUHU82], [GOGU80], [BACH88], [JOKO86]).

Les substitutions closes ont donc des applications importantes. Dans ce domaine, on voit encore la nécessité d'apporter un éclairage algébrique à ces notions: par exemple, l'étude du point de vue algébrique de l'algorithme de décision de l'inductive réductibilité ([PLAI85]).

Un autre domaine de recherche important de la théorie de la réécriture concerne les *problèmes d'unification*. Une application en est la réécriture équationnelle et la programmation dans des langages de type clausal auxquels on ajoute des équations. L'unification modulo AC est décidable, modulo AD, elle est indécidable et modulo D, le problème est ouvert.

****

Nous venons d'esquisser quelques champs d'étude de la réécriture du point de vue de la programmation. Nous avons rencontré au passage quelques problèmes ponctuels non résolus (terminaison d'une seule règle linéaire, unification modulo D) ou mal élucidés (réductibilité inductive, égalité d'un ensemble de formes normales à une sorte). Notre but est, rappelons le, de contribuer à une meilleure connaissance structurelle des arbres et de la réécriture dans les arbres, et de contribuer ainsi à élaborer des outils pour la réécriture en programmation.

Pour conforter notre démarche, rappelons deux outils structurels célèbres sous-jacents à la structure d'arbre: le théorème de Kruskal ([KRUS60]) et les automates de Rabin ([RABI69], [RABI77]) et citons quelques résultats récents, autres que les nôtres, obtenus à partir de cette approche "automates et langages".

Le théorème de Kruskal, qui dit que de toute suite d'arbres on peut extraire une sous-suite croissante pour une certaine notion d'ordre sur les arbres, est à la base de nombreux résultats et concepts sur la terminaison des systèmes de réécriture ([DERS87],[DEJO90]).

Un deuxième résultat fondamental est le théorème de Rabin qui est une mine de résultats de décisions. On trouvera dans [THOM90] une présentation de ce théorème ainsi que de nombreuses applications.

On comprend donc pourquoi l'approche structurelle des problèmes liés à la théorie de la réécriture s'est récemment développée et a amené de nombreux résultats. Parmi ces travaux citons les études de:

A.I.Mal'cev ([MALC71]), M.J.Maher ([MAHE88]), H.Comon ([COMO88, 89,90]), sur une axiomatisation des algèbres d'arbres et les résultats de décidabilité associés

R.V.Book & all, D.Kapur, P.Narendran, F.Otto sur les systèmes de Thue avec de nombreux résultats de décidabilité induits sur les systèmes de réécriture ([KNO90], [NAOT90]),

R.V.Book et J.H.Gallier ([BOGA85]), K.Salomaa ([SALO88]) sur les systèmes de réécriture et les automates à piles d'arbres

M.Oyamaguchi [OYAM87, 90] sur différentes classes de systèmes de réécriture (en particulier clos et clos à droite).

Pour les résultats de l'équipe de Lille, citons, en ce qui concerne la réécriture, la décidabilité de la confluence close ([DATI85]), la décidabilité de la réécriture close (résultat généralisant le précédent [DATI90]), la décidabilité de la terminaison de la surréduction itérée en tête ([LEBE88], [DEVI90]), la décidabilité de la terminaison équitable des systèmes de réécriture clos [TISO89], qui font le pendant à l'indécidabilité de savoir si un système d'équations fini préserve la reconnaissabilité ([CDT89]) et la simulation d'une machine de Turing par une seule règle linéaire à gauche ([DAUC89]).

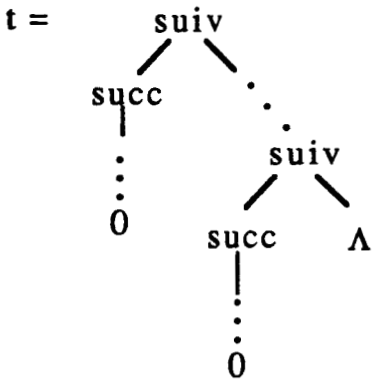
La notion d'*automate fini d'arbres et de forêt reconnaissable* est centrale dans nos travaux.

Nous rappelons que les automates finis d'arbres (J.W.Thatcher [THAT67], W.S.Brainerd[BRAI69]) sont, exprimés en un autre vocabulaire, des définitions de sortes. Nous illustrons cette remarque sur l'exemple suivant:

Soit la sorte ListeEntier définie par:

$0 : \rightarrow$ Nat
succ : Nat $\rightarrow$ Nat
$\Lambda : \rightarrow$ ListeEntier
suiv : Nat x ListeEntier $\rightarrow$ ListeEntier

Un terme t de la sorte ListeEntier a pour configuration :



A la sorte ListeEntier est associée la forêt reconnaissable F reconnue par l'automate $A = (\Sigma, Q, Q_f, R)$ avec $Q = \{q_{Nat}, q_{ListE}\}$, $Q_f = \{q_{ListE}\}$ et

$$R = \{\ 0 \rightarrow q_{Nat}\ ;\ \underset{\mid}{succ}\ \rightarrow q_{Nat}\ ;\ \Lambda \rightarrow q_{ListE}\ ;\ \underset{/\ \backslash}{suiv}\ \rightarrow q_{ListE}\ \}.$$

q_{Nat} ; q_{Nat} q_{ListE}

Cette remarque, trop souvent méconnue, a des applications importantes. Par exemple, H.Comon dans [COMO90] remarque qu'une signature avec sortes ordonnées peut être considérée comme un automate ascendant d'arbres. L'auteur propose un ensemble de règles de transformation correct, complet, qui termine pour les formules du premier ordre dont les atomes sont, soit des équations entre termes, soit de la forme $t \in s$ ou s est une sorte (c'est-à-dire une forêt reconnaissable). Dans la preuve, sont utilisées les propriétés de clôture booléenne de la classe des forêts reconnaissables et la décidabilité du vide.

Les automates d'arbres ont donc un intérêt, tant par des applications éventuelles d'un point de vue programmation, que par les algorithmes connus permettant de résoudre des problèmes de décision en réécriture.

****

Nous présentons maintenant le travail propre à cette thèse. Nous rappelons que la seconde partie de la thèse est commune avec la thèse de R.Gilleron.

Nos principaux résultats sont:

1) Les systèmes clos, munis d'un contrôle reconnaissable, ont la même puissance que les systèmes de réécriture généraux (le contrôle consiste ici à associer une forêt reconnaissable à chaque règle, la règle ne pouvant être appliquée que sur un terme de cette forêt). Plus encore, on peut sans difficulté simuler une machine de Turing par un système de réécriture clos à contrôle clos, seul le codage d'un arbre en peigne pose alors problème.

2) Les ensembles de "preuves" de dérivation dans les problèmes d'accessibilité pour les systèmes clos peuvent être vus comme des forêts reconnaissables.

3) Nous introduisons une notion qui permet d'éclairer et de généraliser les études antérieures sur les liens entre systèmes de réécriture et automates à piles (R.V.Book et J.H. Gallier dans [BOGA85] et K. Salomaa dans [SALO88]).

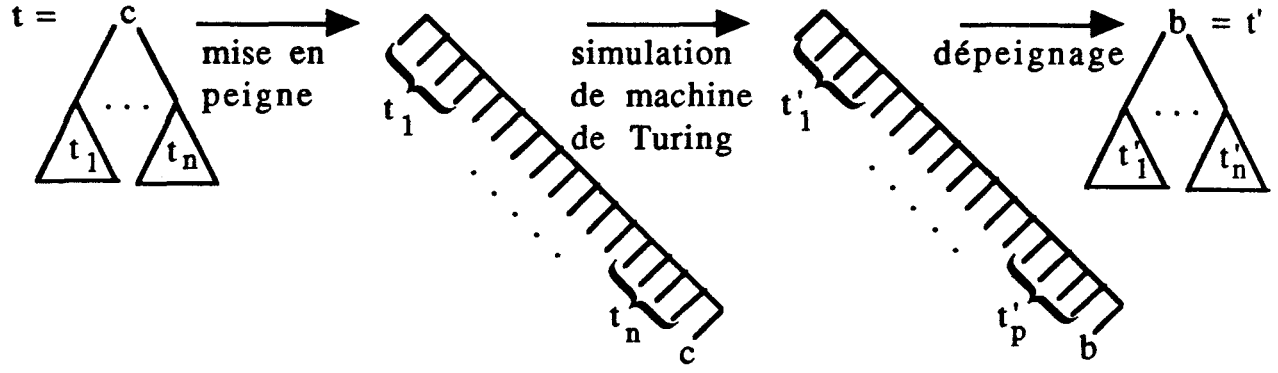
Reprenons avec plus de détails ces trois points.

1) Systèmes de réécriture clos à contrôle reconnaissable.

Afin de correspondre à une étude naturelle d'un problème donné (par exemple de nombreux problèmes arithmétiques utilisent des tests de comparaison), nous pouvons estimer qu'il est naturel que certains systèmes de réécriture soient contrôlés c'est à dire que l'application d'une règle ne soit autorisée que si certaines conditions sont vérifiées. Aussi, après le chapitre I des préliminaires consacré à des rappels sur la théorie des arbres, nous plaçant résolument dans l'optique de systèmes clos, nous étudions dans le chapitre II des systèmes clos contrôlés par l'appartenance de l'arbre à réécrire à une forêt reconnaissable. Par exemple, si nous désirons marquer la première feuille d'un arbre par un symbole spécial δ avant de transformer cet arbre, l'application de la règle de transformation ne sera autorisée que si c'est la constante située à la bonne place qui a été réécrite. Cette condition est vérifiée par l'appartenance de l'arbre à une forêt du genre $T_{\Sigma-\Sigma_0}(\delta,\Sigma_0,\ldots,\Sigma_0)$.

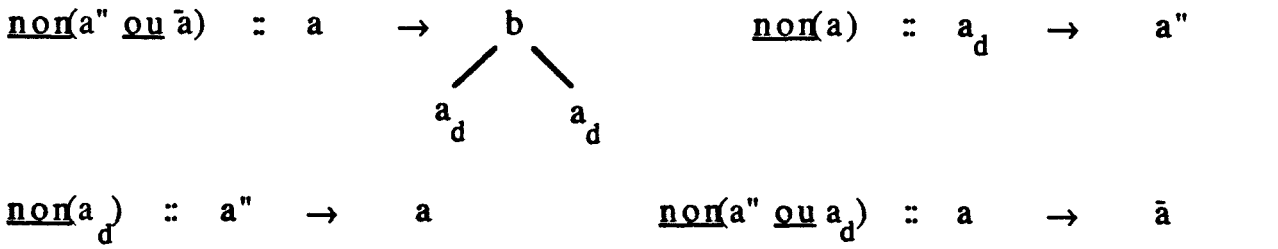
Pour simuler un système quelconque, nous procédons de la façon suivante: dans la section A, nous donnons les règles permettant de réécrire l'arbre sous forme d'un peigne en lisant cet arbre de gauche à droite et de bas en haut (notation post-fixée sous forme de liste), puis nous montrons dans la section B que tout mouvement de machine de Turing correspond à des réécritures sur le peigne; dans la section C nous

donnons les règles permettant de réécrire le peigne obtenu sous sa forme d'arbre en recopiant de haut en bas et de droite à gauche; après avoir donné le résultat principal dans la section D, nous terminons ce chapitre en précisant certains problèmes ouverts dans la section E.

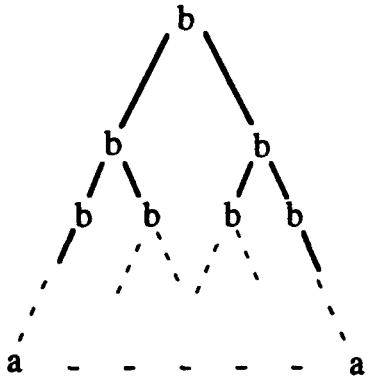


Les systèmes clos avec contrôles reconnaissables ayant la même puissance que les systèmes généraux, nous nous demandons ce qu'il en est si nous nous restreignons à des systèmes clos avec des contrôles clos.

Lors de la simulation des mouvements de la machine de Turing, nous utilisons uniquement des règles closes avec contrôle clos positifs (nous contrôlons simplement qu'un terme clos donné apparait dans l'arbre à réécrire) alors que pour la mise en peigne et le dépeignage, nous utilisons des contrôles reconnaissables qui ne sont pas clos.
Aussi, nous nous demandons si nous pouvons simuler tout système par un système clos à contrôle clos.
Nous ne pouvons pas mettre sous forme de peigne en n'utilisant que des contrôles clos même en utilisant des contrôles clos "positifs" et "négatifs" (non(t) signifie que l'on peut appliquer une règle seulement si le terme clos t n'apparait pas dans le terme où l'on veut appliquer la règle).
Cependant, l'usage de contrôle clos a une capacité (sinon une efficacité) surprenante qui rend la question plus difficile qu'il n'y parait. L'exemple suivant illustre la puissance des contrôles clos.



à partir de a et en considérant l'alphabet $\Sigma=\{a,b \mid ar(a)=0, ar(b)=2\}$, nous obtenons l'ensemble des arbres équilibrés:



2) Preuves d'accessibilité.

Dans le chapitre III, nous étudions certains problèmes liés à l'accessibilité. Le problème d'accessibilité pour des systèmes de réécriture est de décider si, étant donné un système de réécriture S et deux termes clos t et t', t peut être réduit en t' en utilisant des règles de S.

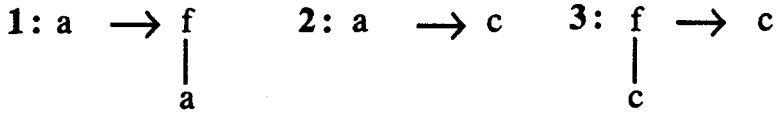
On sait que ce problème est indécidable pour des systèmes de réécriture généraux. Nous étudions ce problème pour des systèmes de réécriture clos, c'est à dire pour des systèmes de réécriture finis pour lesquels les membres gauches et droits de règles sont des termes clos (sans variable). Oyamaguchi [OY86] a prouvé que le problème d'accessibilité pour des systèmes de réécriture clos était décidable et Dauchet & Tison [DATI90] ont montré que c'est une conséquence de la décidabilité de la théorie de la réécriture close. Deruyver & Gilleron [DEGI89] ont aussi étudié ce problème et ont donné un algorithme de décision en temps réel, après compilation du système de réécriture clos en transducteur particulier d'arbres clos (Valeriann est une implémentation de ces algorithmes[DADE89]).

Etant donné que ces problèmes d'accessibilité sont décidables, nous nous posons naturellement les problèmes suivants: si t peut être réduit en t' quelles règles de S peuvent être utilisées pour le faire? Nous introduisons la notion d'arbres de dérivation comme étant des termes sur un nouvel alphabet gradué afin d'obtenir "l'histoire" d'une réduction de t en t' par S. La frontière d'un arbre de dérivation donnera une séquence de règles utilisées pour réduire t en t'; de plus nous saurons à quelle occurence dans l'arbre une règle a été appliquée. Nous prouvons que l'ensemble des arbres de dérivation est un langage d'arbres reconnaissable et nous donnons un algorithme en temps linéaire fournissant un arbre de dérivation pour le problème d'accessibilité.

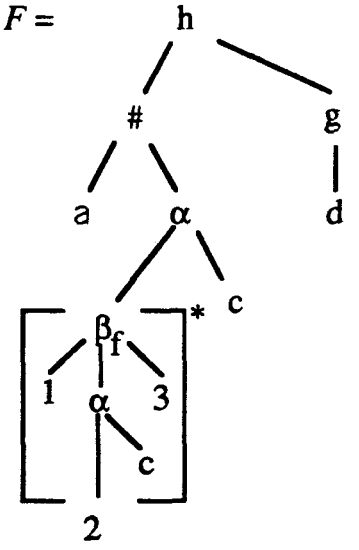
Nous étudions dans la section A le cas standard où les règles $l \rightarrow r$ peuvent être soit effaçantes ($h(l)=1, h(r)=0$) soit génératrices ($h(l)=0, h(r)=1$) soit des ε-règles ($h(l)=h(r)=0$).

Pour obtenir l'ensemble des preuves permettant de réduire t en t', nous définissons dans la section A.1 une grammaire régulière qui génére ce langage d'arbres. Cette grammaire est définie en plusieurs étapes; dans la section A.1.1 nous donnons les règles permettant d'obtenir les arbres de dérivation d'un symbole a en un symbole b par une séquence génération-effacement puis dans la section A.1.2 nous généralisons à une dérivation quelconque de a en b; dans les sections A.1.3 et A.1.4 nous étudions les arbres de dérivation pour passer d'un arbre t à un symbole a (A.1.3) et inversement (A.1.4); enfin dans la section A.1.5 nous obtenons l'ensemble des arbres de dérivation d'un arbre t en un arbre t'.

Exemple:



Soit S = {1,2,3}. Le langage reconnaissable d'arbres des preuves de dérivation de t = h(a, g(d)) en t' = h(c, g(d)) est F =

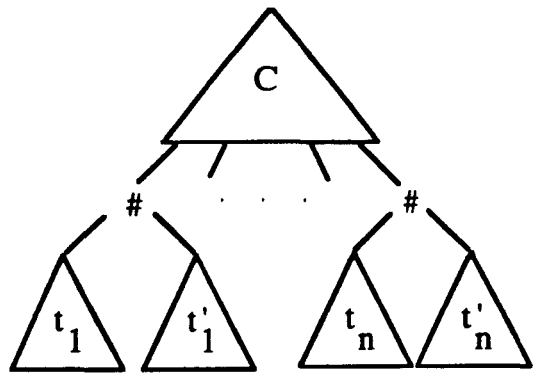


Si nous considérons le feuillage des arbres obtenus, en faisant abstraction des constantes, nous obtenons la séquence de dérivation $1^n.2.3^n$, $n \in N$. En effet, les seules dérivations possibles sont des générations de f à partir de a (règles 1) puis l'effacement de ces f (règles 3) une fois que a a été réécrit en c (règle 2). Cet exemple est donc assez simple car il n'y a pas la possibilité d'obtenir plusieurs séquences génération-effacement consécutives. Ce n'est pas toujours aussi simple comme le montre l'exemple situé au chapitre III, page 66.

Dans la section A.2, nous étendons ce résultat à la recherche des arbres de dérivation d'une forêt reconnaissable F en une autre F' en effectuant en parallèle à la construction du langage le contrôle de l'appartenance des arbres utilisés à F et F'. Lorsque nous avons un arbre de dérivation, afin de retrouver les arbres t de F et t' de F' correspondants à cette dérivation, nous utilisons un transducteur d'arbres qui permet de transformer l'arbre de preuve. Les transducteurs d'arbres sont une généralisation des transducteurs de mots [BERS79] au cas des arbres.

Les résultats obtenus n'étant pas utilisables en pratique car ils ont une complexité avec des tours d'exponentielle, nous donnons alors dans la section A.3 un algorithme qui permet d'obtenir un arbre de dérivation de t en t' en temps linéaire.

Lors de la construction du transducteur d'arbres clos V=(A,B) nous associons à chaque ε-règle créée un arbre "court" de dérivation, puis, C étant le plus grand contexte commun aux arbres t et t', nous considérons l'arbre



Nous définissons un transducteur ascendant déterministe qui associe à chaque noeud de cet arbre l'ensemble des états accessibles en ce noeud par les automates A et B du transducteur V et qui spécifie en même temps s'il y a un interface commun à A et B au niveau du noeud. Nous terminons la construction en utilisant un transducteur descendant qui détermine un arbre de dérivation grâce aux états marqués au niveau de chaque noeud et à la table d'histoire définie pour chaque ε-règle.

Dans la section B, nous nous plaçons dans le cas général où le système ne contient pas que des règles standards; nous nous ramenons alors au cas standard en décomposant chaque règle en une suite de règles standards. Il reste ensuite à transducter le langage obtenu pour obtenir l'ensemble des arbres souhaité.

Dans la section B.1, nous donnons un exemple de détermination de l'ensemble des arbres de dérivation d'un arbre t en un arbre t'.

Dans la section B.2, nous explicitons la méthode permettant d'obtenir un arbre de dérivation en temps linéaire.

****

La seconde partie de la thèse est la présentation d'un travail qui a été réalisé à l'occasion de la venue de S.Vàgvölgyi à Lille. Cette partie est commune à cette thèse et à la thèse de R.Gilleron.

Nous étudions dans cette partie les liens entre les automates à piles d'arbres et les systèmes de réécriture.

Nous poursuivons, en cela, une étude commencée par R.V.Book et J.H. Gallier dans [BOGA85] et K. Salomaa dans [SALO88]. Les automates à piles d'arbres peuvent être considérés comme le point de vue algorithmique de problèmes spécifiés à l'aide de certains systèmes de réécriture. On peut alors parler de compilation d'un système de réécriture à l'aide d'automates à piles d'arbres.

Plus précisément, R.V. Book, J.H. Gallier et K. Salomaa ont montré que l'on pouvait associer, à tout système de réécriture monadique linéaire et convergent S, un automate à piles déterministe d'arbres qui calcule les formes normales pour S. Un système de réécriture est dit monadique quand tout membre gauche de règle est de hauteur supérieure ou égale à 1 et tout membre droit est soit une variable, soit une constante, soit un terme du type $b(y_1, ..., y_n)$ où y_i est une variable pour tout i, $1 \leq i \leq n$.

L'originalité du travail présenté ici est que nous mettons en évidence l'intérêt d'une nouvelle propriété des systèmes de réécriture: la propriété trf (pour tail reduction free). Un système de réécriture S possède la propriété trf si et seulement si, pour tout membre droit $r=l(t_1,...,t_n)$ de règle de S avec l lettre d'arité n, pour toute substitution close et irréductible σ (pour S), alors, pour tout i, $1 \leq i \leq n$, $\sigma(t_i)$ est irréductible, c'est à dire, pour tout membre droit de règle de S, pour toute substitution σ close et irréductible pour S, le terme $\sigma(r)$ n'est réductible par S qu'en sa racine.

Nous démontrons que, à tout système de réécriture linéaire et convergent S possédant la propriété trf, on peut associer un automate à piles déterministe qui calcule les formes normales pour S. Les systèmes

de réécriture monadiques possédent, de façon évidente, la propriété trf; les résultats précédemment démontrés sont alors des cas particuliers des résultats que nous obtenons.

Dans la section A , nous comparons les différentes définitions possibles d'automates à piles ascendants d'arbres selon l'arité des états, la profondeur des termes dépilés et la forme des règles. Nous définissons également la notion de déterminisme associé de la façon suivante: un automate à piles est déterministe si le système de réécriture (définissant les règles) est linéaire à gauche et sans paire critique. Nous montrons, dans le cas général et dans le cas déterministe, l'équivalence des différentes définitions.

Dans la section B, nous introduisons la propriété trf. Nous montrons que la classe des systèmes de réécriture possédant cette propriété est large (contient, par exemple, la classe des systèmes de réécriture monadiques). Nous prouvons que l'on peut simuler une machine de Türing à l'aide d'un système de réécriture ayant cette propriété. Nous démontrons la décidabilité de la propriété trf en utilisant la décidabilité de l'inductive réductibilité ([PLAI85]). Nous associons à tout système de réécriture possédant la propriété trf un automate à piles d'arbres qui calcule les formes normales des termes clos pour S et précisons les relations entre la linéarité du système de réécriture et la nécessité d'un contrôle pour l'automate à piles. En effet, dans le cas linéaire à gauche, l'automate à piles associé est déterministe et, dans le cas non linéaire à gauche, il est nécessaire d'introduire un contrôle (de priorité à la réduction) dans l'automate à piles.

Dans la section C, nous étudions les systèmes de réécriture possédant la propriété trf et les automates à piles d'arbres du point de vue théorie des langages (d'arbres). Un système de réécriture est semi-monadique si tout membre droit de règle a pour configuration $r=l(y_1,\ldots,y_n)$ avec l lettre d'arité n et, pour tout i, $1\leq i\leq n$, y_i est une variable ou un terme clos. Nous étendons un résultat de K. Salomaa ([SALO88]) sur la préservation de la reconnaissabilité des systèmes de réécriture monadiques aux systèmes de réécriture semi-monadiques.

****

Citons quelques prolongements et perspectives à ce travail.

Concernant les systèmes contrôlés, nous avons montré que l'adjonction de contrôles reconnaissables aux systèmes de réécriture clos faisait "exploser" du reconnaissable au récursivement énumérable le

comportement décrit en termes de langages. Il serait intéressant de déterminer des classes de problèmes qui restent décidables par contrôle reconnaissable.

Les contrôles reconnaissables introduits, même si nous nous restreignons à des contrôles clos, expriment des priorités sur la réécriture. En ce sens, le problème de mise en peigne d'un terme quelconque peut s'exprimer sous la forme: peut-on, avec des réécritures closes à priorité, passer d'une structure quelconque à une structure de liste? Aussi il serait éclairant d'un point de vue structurel de mieux comprendre le rôle des contraintes closes.

Nous pouvons associer à tout système de réécriture un système de réécriture clos en substituant à chaque variable sa sorte, qui est un langage reconnaissable d'arbres. Par exemple, si nous considérons les sortes *Var* (variables), *Const* (constantes), *Mon* (monômes) et *Pol* (polynômes), au système comprenant les règles:
+(*(x,X),*(z,X))→*(+(x,z),X) où sort(x)=*Pol*, sort(z)=*Mon* et
+(*(x,X),*(C,X))→*(+(x,C),X) où sort(x)=*Pol*,
nous associons le système clos:
C→*Const*, X→*Var*,
Const→*Mon*, *(*Mon*,*Var*)→*Mon*,
Mon→*Pol*, +(*Pol*,*Mon*)→*Pol*,
+(*(*Pol*,X),*(*Mon*,X))→*(+(*Pol*,*Mon*),X) et +(*(*Pol*,X),*(C,X))→*(+(*Pol*,C),X).
Nous pouvons alors définir un évaluateur symbolique qui approxime le programme. Les arbres de dérivation deviennent des preuves pour l'évaluation symbolique. Aussi, il serait intéressant de développer cette notion d'arbre de dérivation dans le cadre du logiciel Valeriann.

En considérant les feuillages de nos arbres de dérivation, on voit que les ensembles de séquences de règles attachées aux problèmes d'accessibilité sont des langages algébriques quand les systèmes sont clos. Il serait intéressant de connaître de même la structure des arbres de dérivation pour des systèmes plus généraux, comme ceux associés aux automates à piles.

D'une façon plus générale, nous comptons développer nos efforts dans une double perspective: mieux connaître la structure de classes de systèmes de réécriture et de classes de problèmes; trouver des algorithmes de transformation (de "compilation") de ces systèmes. Nous allons développer avec nos méthodes l'étude de fragments de théories décidables (éventuellement sous hypothèses indécidables), la combinaison modulaire de celles-ci, une généralisation et un éclaircissement des problèmes liés à l'inductive réductibilité.

CHAPITRE I

PRELIMINAIRES

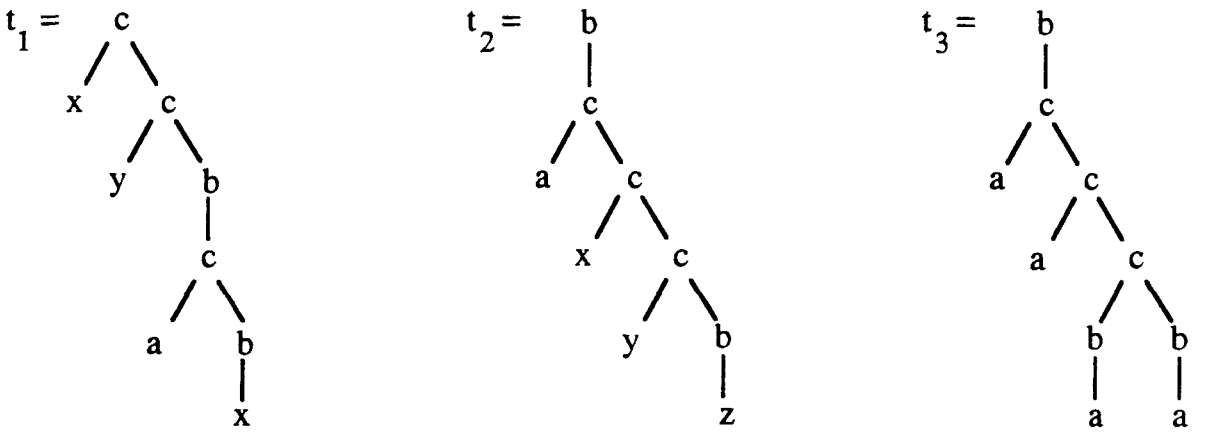
1) TERMES , SUBSTITUTIONS

1.1) Termes

- Σ est un *alphabet fini gradué* si et seulement si chaque élément b de Σ a une arité unique dans l'ensemble des entiers naturels notée ar(b). Les éléments d'arité 0 sont appelés les *constantes*. Nous supposons que Σ contient toujours une constante. Pour tout entier positif n, Σ_n est le sous-ensemble de Σ qui contient tous les éléments d'arité n.
  Exemple: $\Sigma_0 = \{a\}$, $\Sigma_1 = \{b\}$, $\Sigma_2 = \{c\}$, $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$.

- Soit X un ensemble dénombrable de variables.
  L'ensemble $T_\Sigma(X)$ des arbres (ou *termes*) sur $\Sigma \cup X$ est défini comme étant le plus petit ensemble pour lequel
  i) $X \subseteq T_\Sigma(X)$
  ii) $b(t_1,\ldots,t_n) \in T_\Sigma(X)$ pour tout $b \in \Sigma_n$ avec $n \geq 0$ et $t_1,\ldots, t_n \in T_\Sigma(X)$.
  Posons $X_n = \{x_1,\ldots,x_n\}$ pour tout entier positif n. $T_\Sigma(X_n)$ est l'ensemble des arbres sur $\Sigma \cup X_n$.

- L'ensemble $T_\Sigma(\varnothing)$, noté T_Σ, est l'ensemble des arbres clos sur Σ.
  Exemple:



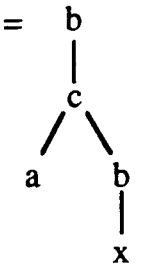
t_1 est un élément de $T_\Sigma(X_2)$, t_2 de $T_\Sigma(X_3)$ et t_3 de T_Σ.

- *L'ensemble des variables* apparaissant dans t est noté V(t).
  Si nous reprenons l'exemple, nous avons $V(t_1) = \{x,y\}$, $V(t_2) = \{x,y,z\}$ et $V(t_3) = \varnothing$.
  Un arbre t est *linéaire* si et seulement si aucune variable n'apparait deux fois dans t.
  Dans l'exemple, t_2 est linéaire mais t_1 ne l'est pas.

- La *racine* d'un arbre t est le symbole c situé en son sommet ($t=c(t_1,\ldots,t_n)$), une *feuille* est un symbole d'arité 0 apparaissant dans t (c'est une constante ou une variable).

- Une *occurrence* dans un arbre est une position dans l'arbre. Elle peut être déterminée par une suite d'entiers positifs (notation de Dewey) donnant le chemin permettant d'aller de la racine au symbole situé à la position considérée. *L'ensemble des occurrences* d'un arbre est notée O(t). Nous notons $t_{/p}$ le *terme situé à l'occurrence* p.

Dans l'exemple, nous avons $t_{1/2.2}$ =



- L'ensemble de toutes les chaînes de symboles de Σ qui apparaissent le long d'un chemin allant de la racine de l'arbre t jusqu'à une feuille, noté *Chemin(t)*, est défini de la façon suivante:

i) si $t \in \Sigma_0 \cup X$ alors Chemin(t) = {t}

ii) si $t = b(t_1,\ldots,t_n)$ avec $n \geq 1$, $b \in \Sigma_n$ et $t_1,\ldots t_n \in T_\Sigma(X)$ alors

$$\text{Chemin}(t) = \{b.ch \;/\; ch \in (\bigcup_{i=1}^{n} \text{Chemin}(t_i))\}.$$

Dans l'exemple, $\text{Chemin}(t_1)$ = {c.x, c.c.y, c.c.b.c.a, c.c.b.c.b.x}.

- La *hauteur* (ou *profondeur*) d'un arbre t correspond à la longueur de sa plus grande branche dans t (à la feuille près). Elle est notée h(t).
La *taille* d'un arbre est le nombre de noeuds que contient l'arbre. Elle est notée ||t||.
La *frontière* d'un arbre est le mot constitué de ses feuilles. Elle est notée fr(t).
Elles sont définies de la façon suivante:

i) si $t \in \Sigma_0 \cup X$ alors $h(t) = 0$ $\quad ||t|| = 1$ $\quad fr(t) = t$,

ii) si $t = b(t_1,\ldots,t_n)$ avec $n \geq 1$, $b \in \Sigma_n$ et $t_1,\ldots t_n \in T_\Sigma(X)$ alors

$$h(t) = 1 + \max(\{h(t_i) \;/\; 1 \leq i \leq n\}),\quad ||t|| = 1 + \sum_{i=1}^{n} ||t_i||,\quad fr(t) = \prod_{i=1}^{n} fr(t_i).$$

Dans l'exemple, $h(t_1) = 5$, $||t_1|| = 9$, $fr(t_1)$ = x.y.a.x.

- La *largeur* d'un arbre est le nombre de feuilles de l'arbre. Elle est notée largeur(t). Dans l'exemple, $\text{largeur}(t_1) = 4$.

- L'*ensemble des sous-arbres* d'un arbre t, noté Sub(t), est défini de la façon suivante:

  i) si $t \in \Sigma_0 \cup X$ alors $Sub(t) = \{t\}$

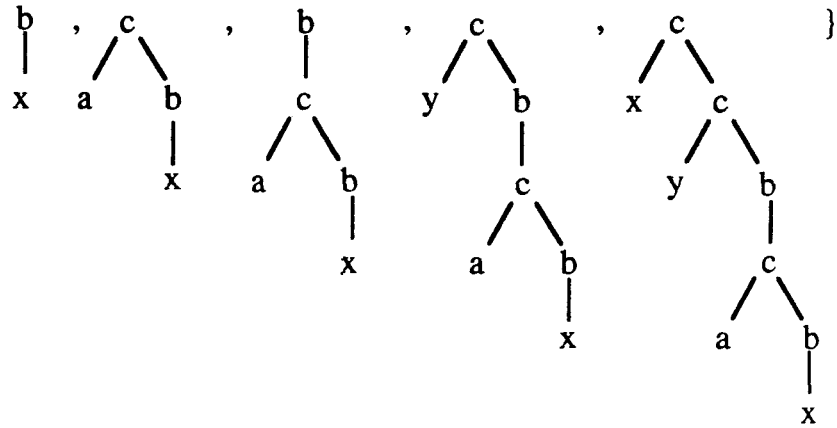
  ii) si $t = b(t_1,\ldots,t_n)$ avec $n \geq 1$, $b \in \Sigma_n$ et $t_1,\ldots t_n \in T_\Sigma(X)$

  $$\text{alors } Sub(t) = \{t\} \cup (\bigcup_{i=1}^{n} Sub(t_i))$$

  Un *sous-arbre propre* est un sous-arbre distinct de t.

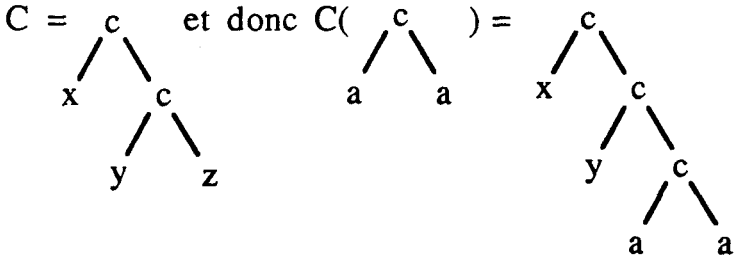
  Dans l'exemple, nous avons:

$Sub(t_1) = \{ x, y, a, b, c, b, c, c, \}$



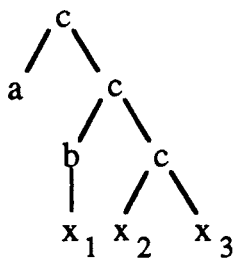
- Un *contexte* C est un terme de $T_\Sigma(X_n)$ tel que chaque variable apparait exactement une fois dans C et nous notons $C(t_1,\ldots,t_n)$ le résultat de la substitution de chaque x_i par un terme t_i. Soit t un terme et p une occurrence de t, le contexte associé à p dans t est le terme C défini par $t=C(t/p)$. C(u) désigne alors le sous-terme obtenu en substituant u à t/p à l'occurrence p de t. Dans l'exemple, en considérant l'arbre t_1 et l'occurrence 2.2, le contexte associé à 2.2 dans t_1 est:

C = et donc C( ) =



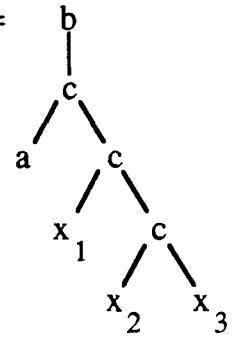
- Un *arbre k-normalisé* (k entier positif) est un arbre linéaire de $T_\Sigma(X_n)$ de hauteur k tel que pour toute occurrence p de O(t) soit $|p| = k$ et t/p est un élément de X soit $|p| < k$ et la racine de t/p est un élément de Σ et de plus $fr(t) \in \Sigma^* x_1 \Sigma^* \ldots x_n \Sigma^*$ où les variables sont ordonnées et toutes distinctes.

Exemple: arbre 3-normalisé:



- La *troncature* d'un arbre à la profondeur k est notée tronc(t,k). Elle est définie de la façon suivante: pour toute occurrence p de O(tronc(t,k)) soit |p| = k et $tronc(t,k)_{/p}$ est élément de X soit |p| < k et la racine de $tronc(t,k)_{/p}$ est égale à la racine de $t_{/p}$ et de plus si tronc(t,k) est un élément de $T_\Sigma(X_n)$ alors $fr(tronc(t,k)) \in \Sigma^* x_1 \Sigma^* \ldots x_n \Sigma^*$.

Dans l'exemple, nous avons tronc(t_2, 4) =

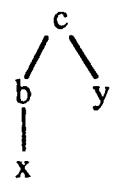


1.2) Substitutions

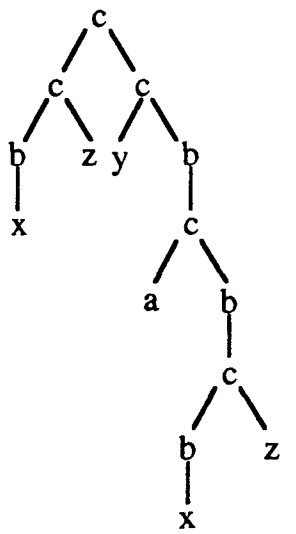
- Une *substitution* σ est une application de X dans $T_\Sigma(X)$ distincte de l'identité pour un nombre fini de variables. Nous notons D(σ) le *domaine* de σ. Nous définissons σ(t) par extension sur $T_\Sigma(X_n)$ de la façon suivante: pour tout f de X_n, pour tous les termes $t_1,\ldots,t_n$ appartenant à $T_\Sigma(X)$, $\sigma(f(t_1,\ldots,t_n))=f(\sigma(t_1),\ldots,\sigma(t_n))$. Nous confondons dans les notations la substitution et son extension.

Dans l'exemple 1, considérons la substitution σ telle que σ(y) = y,

σ(x) =



nous obtenons $\sigma(t_1)$ =

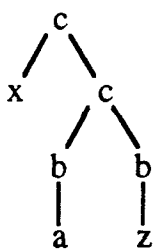


- Une *substitution close* est une substitution de X dans T_Σ.
  Exemple: Reprenons les arbres t_2 et t_3. Si nous définissons la sustitution close $\sigma'(x) = \sigma'(z) = a$, $\sigma'(y) = b$, nous obtenons $\sigma'(t_2) = t_3$.

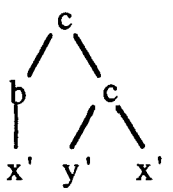
- la substitution σ est *plus générale* que τ si et seulement s'il existe une substitution ρ telle que $\rho o \sigma = \tau$.

- Deux arbres t et t' sont *unifiables* s'il existe une substitution σ telle que $\sigma(t) = \sigma(t')$.
  Exemple: considérons les arbres suivants

t =

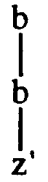


t' =



Si nous prenons la substitution la plus générale σ définie par

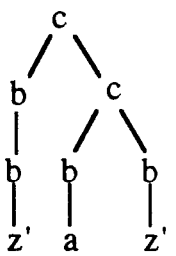
$\sigma(x)$ =



$\sigma(z) = z'$ $\sigma(x') =$

$\sigma(y') =$

nous voyons que t et t' sont unifiables et que leur unifié est:



- Etant donnés deux arbres unifiables t et t', il existe un unificateur le plus général de t et t' [PLOT72].

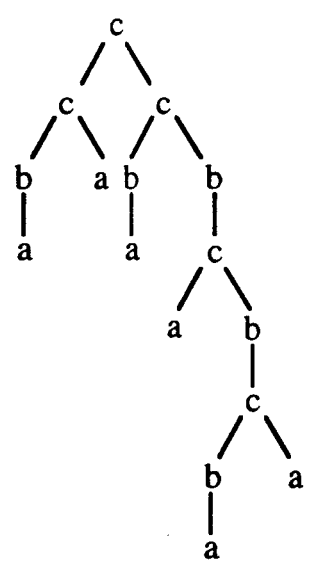
- Un arbre t *filtre* un arbre t' s'il existe une substitution σ telle que $\sigma(t)=t'$. La relation ainsi définie est un préordre et la relation d'équivalence associée correspond à l'égalité des arbres modulo un renommage des variables.

  Exemple: Si nous reprenons les arbres t_2 et t_3, nous avons montré qu'il existait σ' telle que $\sigma'(t_2) = t_3$ donc t_2 filtre t_3.

- La *composition* de deux substitutions est la composition des applications.

  Exemple: avec les substitutions définies çi-dessus, nous obtenons

$\sigma'(\sigma(t_1))$ =



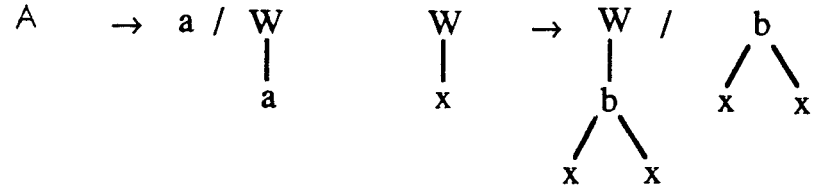
2) Grammaires d'arbres

2.1) Grammaire algébrique

- Une *grammaire algébrique d'arbres* (ou à contexte libre) est un quadruplet $G=(A,V,\Sigma,R)$ où V est un ensemble de symboles non

terminaux, V et Σ sont disjoints, A appartenant à V_0 est l'axiome et R est un ensemble fini de règles, $R=\{l \rightarrow r \ / \ l=b(x_1,\ldots,x_n), b \in V, r \in T_{\Sigma \cup V}(X_n)\}$

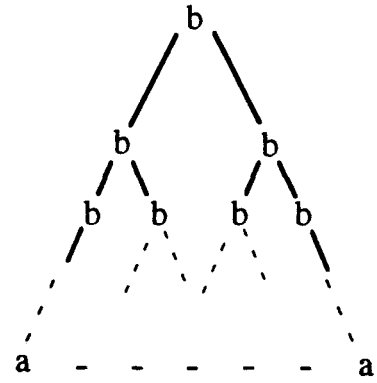
Exemple: Considérons la grammaire algébrique suivante: $G=(A,V,\Sigma,R)$ avec $V = \{A, W \ / \ ar(A) = 0 \text{ et } ar(W) = 1\}$, $\Sigma = \{a, b \ / \ ar(a) = 0 \text{ et } ar(b) = 2\}$ et R contient les deux règles suivantes:



- Un arbre t *se dérive* en un arbre t' par application d'une règle $l \rightarrow r$ si et seulement s'il existe une occurrence de l dans t, une substitution σ telle que $t_{/p}=\sigma(l)$ et un terme u de $T_\Sigma(X)$ tels que $t=u.\sigma(l)$ et $t'=u.\sigma(r)$. L'arbre t se dérive en t' par une dérivation de longueur n , notée $t \xrightarrow{n} t'$, s'il existe des arbres $t_0,\ldots,t_n$ tels que $t_0 = t$, $t_n = t'$ et $t_i \rightarrow t_{i+1}$ pour tout i, $0 \leq i \leq n-1$.

- G génère le *langage d'arbres* $L(G) = \{t \in T_\Sigma \ / \ A \xrightarrow{*}_R t\}$. L(G) est un langage algébrique d'arbres.

Exemple: Reprenons l'exemple précédent. A partir de A nous obtenons l'ensemble de tous les arbres équilibrés:

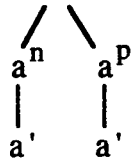


2.2) Grammaire régulière

- Une grammaire est *régulière* si tous les symboles non terminaux sont d'arité 0. L(G) est un langage régulier d'arbres.

Exemple: Considérons la grammaire régulière suivante: $G=(A,V,\Sigma,R)$ avec $V = \{A, W \ / \ ar(A) = ar(W) = 0\}$, $\Sigma = \{a, b \ / \ ar(a) = 0 \text{ et } ar(b) = 2\}$ et R contient les deux règles suivantes:

Nous obtenons L(G) = { $b^{n}\,b\,a^{p}$ / n,p∈ N }



- Une forêt F est régulière si et seulement s'il existe une grammaire régulière d'arbres telle que F soit générée par cette grammaire.
  Exemple: La forêt définie ci-dessus est régulière.

Pour de plus amples développements, voir [ARDA76,78], [LEGU80], [GEST84].

3) Automates d'arbres et langages reconnaissables d'arbres

3.1) Automates d'arbres

- Un *automate ascendant d'arbres* (aaa) est un quadruplet $AA=(\Sigma,Q,Q_f,R)$ où Σ est un alphabet fini gradué, Q un ensemble fini d'états d'arité 0, Q_f un sous ensemble de Q, R un ensemble fini de règles de la configuration suivante:
  (i) $f(q_1,...,q_n)\rightarrow q$ avec $f\in\Sigma_n$, $n\geq 0$, $q_1,...,q_n$ dans Q
  (ii) $q\rightarrow q'$ avec q,q' dans Q (ε-règles).

  Notons que si l'on considère R comme un système de réécriture sur $T_{\Sigma\cup Q}$, $\rightarrow_{AA}$ est la relation de réécriture $\rightarrow_R$.

  Exemple:
  Soit la signature avec sortes ordonnées définie par:

  les constantes O et Λ, les opérateurs succ d'arité 1 et suiv d'arité 2,
  les trois sortes Nat, ListeEntier et ListeEntierNonVide,
  O : → Nat.
  succ : Nat → Nat
  Λ : → ListeEntier.
  suiv : Nat x ListeEntier → ListeEntierNonVide.
  ListeEntierNonVide < ListeEntier.

A cette signature est associée l'automate A = (S,Q,Q_f,R) avec
Q = {q_{Nat}, q_{ListE}, q_{LENV}} = Q_f et

$$R = \{0 \to q_{Nat};\ succ(q_{Nat}) \to q_{Nat};\ \Lambda \to q_{ListE};\ suiv(q_{Nat}, q_{ListE}) \to q_{LENV};\ q_{LENV} \to q_{ListE}$$

- Un automate est *déterministe* s'il n'existe pas deux règles de même partie gauche.
  Dans l'exemple, l'automate est déterministe.

- Le langage d'arbres reconnu par A est L(A)={t∈ T_Σ / t $\xrightarrow{*}_A$ q, q∈ Q_f}.
  exemple:

Soit Σ={a,f,g / ar(a)=0, ar(g)=1, ar(f)=2}.
Soit A=(Σ,Q,Q_f,R) l'automate ascendant d'arbres défini par:

Q = {q, q_{pair}, q_{imp}} ; Q_f = {q} et

$$R = \{ a \to q_{pair};\ g(q_{pair}) \to q_{imp};\ g(q_{imp}) \to q_{pair};\ f(q_{pair}, q_{pair}) \to q;\ f(q_{imp}, q_{pair}) \to q \}$$

On obtient:

L(A)={ $f(g^n(a), g^{2p}(a))$ / n ≥ 0, p≥0}

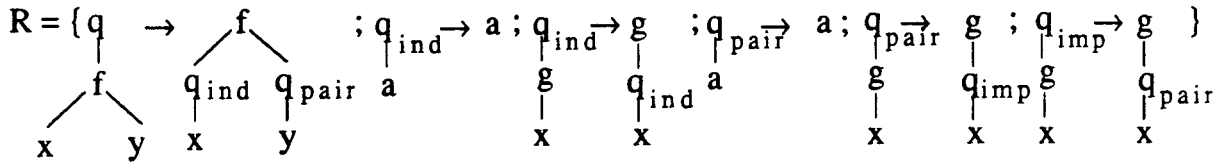
- Soit A un automate ascendant d'arbres, il existe un automate ascendant sans ε-règles tel que L(B)=L(A) et il existe un automate ascendant deterministe (i.e sans ε-règles et n'ayant pas deux règles avec le même membre gauche) C tel que L(C)=L(A).

- Un *automate descendant d'arbres* (ada) est un quadruplet A=(Σ,Q,Q_i,R) où Σ est un alphabet fini gradué, Q un ensemble fini d'états d'arité 1, Q_i un sous ensemble de Q, R un ensemble fini de règles de la configuration suivante:
  (i) $q[f(x_1,...,x_n)] \to f(qi_1[x_1],...,qi_n[x_n])$ avec n≥0, $qi_1,...,qi_n$ dans Q
  (ii) q[x]→q'[x] avec q,q' dans Q (ε-règles).

  Exemple:

Soit Σ={a,f,g / ar(a)=0, ar(g)=1, ar(f)=2}.
Soit A=(Σ,Q,Q_i,R) l'automate descendant d'arbres défini par:

$Q = \{q, q_{ind}, q_{pair}, q_{imp}\}$; $Q_i = \{q\}$ et

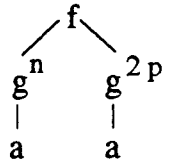
$R = \{$  $\}$

- Un automate est *déterministe* s'il n'existe pas deux règles de même partie gauche.

  Remarque: on ne peut pas toujours réduire le non-déterminisme pour un automate descendant.

- Le langage d'arbres reconnu par A est $L(A)=\{t \in T_\Sigma / q[t] \xrightarrow{*}_A t, q \in Q_i\}$.

  Dans l'exemple, nous obtenons:

  $L(A)=\{$  $/ n \geq 0, p \geq 0\}$

  3.2) Forêts reconnaissables d'arbres

- Un langage d'arbres F est *reconnaissable* s'il existe un automate ascendant A tel que L(A)=F.

- La classe Rec des langages d'arbres reconnaissables est close par union, intersection et complémentation. On peut décider si un langage d'arbres reconnaissable est vide, l'inclusion, l'égalité de langages d'arbres reconnaissables.

- Si on considère deux forêts F et F' de T_Σ et un symbole α de Σ_0, la forêt obtenue par *substitution* dans F de F' à α, notée $F[F'/\alpha]$, est l'ensemble de tous les arbres obtenus en remplaçant dans tout arbre de F chaque occurrence de α par un arbre de F'. La classe Rec est close par substitution.

  Pour de plus amples développements, voir [GEST84].

4) Transducteurs d'arbres

4.1) Transducteurs d'arbres

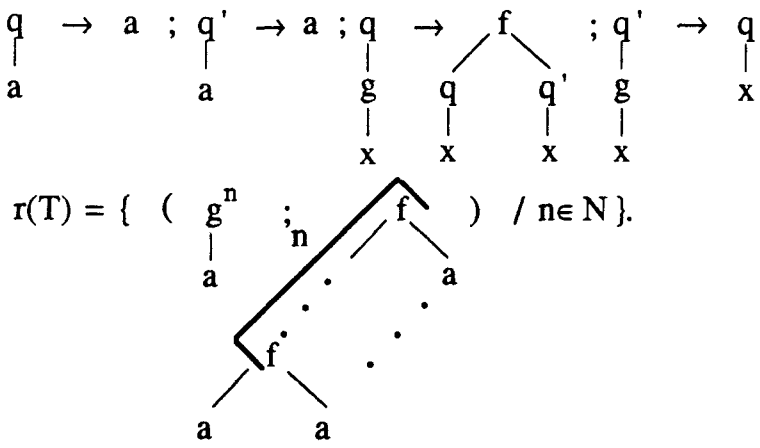
- Un *transducteur descendant d'arbres* (tda) est un quintuplet $T=(\Sigma,\Delta,Q,I,R)$ avec
  - Σ et Δ alphabets gradués
  - Q ensemble fini d'états (alphabet distingué, lettres d'arité 1)
  - I ensemble des états initiaux (I est inclus dans Q)
  - R ensemble de règles de réécriture de la forme
    $q[c(x_1,...,x_n)]\rightarrow t(q_{i1}[x_{i1}],...,q_{ip}[x_{ip}])$, $c\in\Sigma_n, t(x_{i1},...,x_{ip})\in T_\Delta(X_n)$

- On note $\rightarrow_T$ la relation de réécriture associée à R et $\overset{*}{\rightarrow}_T$ sa cloture réflexive et transitive.

- La transformation associée à T est $r(T)=\{(t,t')/t\in T_\Sigma,\ t'\in T_\Delta, \exists q\in I$ tel que $q(t)\overset{*}{\rightarrow}_T t'\}$.

Exemple: Soit $T=(\Sigma,\Delta,Q,I,R)$ avec $\Delta=\Sigma=\{a,f,g$ / ar(a)=0, ar(g)=1, ar(f)=2$\}$.

Q={q, q'}, I={q} et R est défini par

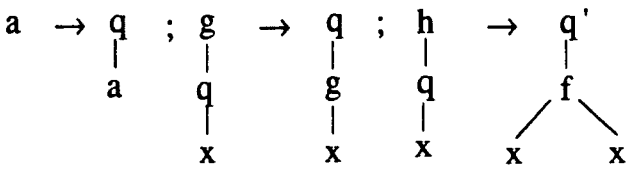


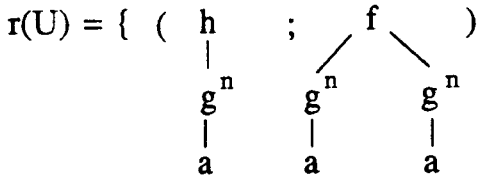
- Un *transducteur ascendant d'arbres* (taa) est un quintuplet $U=(\Sigma,\Delta,Q,J,R)$ avec
  - Σ et Δ alphabets gradués
  - Q ensemble fini d'états (alphabet distingué, lettres d'arité 1)
  - J ensemble des états finaux (J est inclus dans Q)
  - R ensemble de règles de réécriture de la forme
    $c(q_1[x_1],...,q_n[x_n])\rightarrow q[t(x_{j1},...,x_{jp})]$, $c\in\Sigma_n$, $t\in T_\Delta(X_n)$.

- On note $\rightarrow_U$ la relation de réécriture associée à R et $\overset{*}{\rightarrow}_U$ sa cloture réflexive et transitive.La transformation associée à U est $r(U)=\{(t,t')/t\in T_\Sigma,\ t'\in T_\Delta,\ \exists q\in J$ tel que $t\overset{*}{\rightarrow}_U q(t')\}$.

Exemple: Soit $U=(\Sigma,\Delta,Q,J,R)$ avec $\Delta=\Sigma=\{a,g,h,f$ / $ar(a)=0$, $ar(g)=ar(h)=1$ et $ar(f)=2\}$.

$Q=\{q, q'\}$, $J=\{q'\}$ et R est défini par





On peut remarquer que U n'est pas linéaire, que si l'on prend $F=\{h(g^n(a))$ / $n\geq 0\}$ forêt reconnaissable alors $U[F]=\{f(g^n(a),g^n(a)))$ / $n\geq 0\}$ n'est pas une forêt reconnaissable.

- T (respectivement U) est *linéaire* si et seulement si pour toute règle $t(x_{i1},...,x_{ip})$ (respectivement $t(x_{j1},...,x_{jp})$) est linéaire.

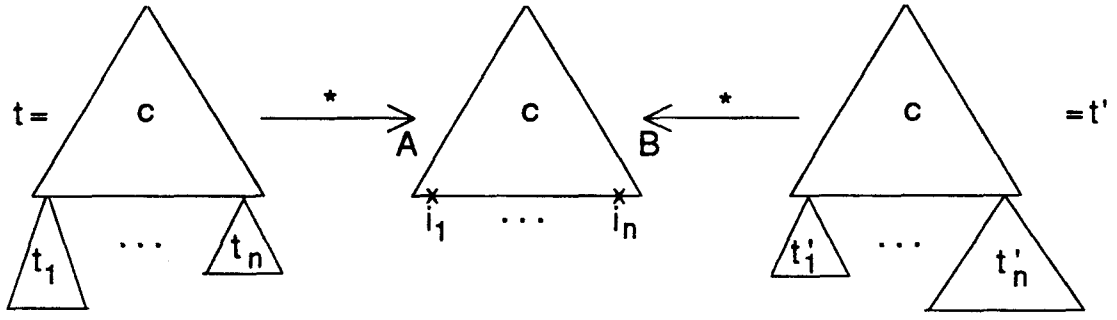
- La classe des forêts reconnaissables est close par transducteurs linéaires et inverses de transducteurs c'est à dire si F est une forêt reconnaissable alors $T[F]=\{t'/t\in F$ et $(t,t')\in r(T)\}$ avec T tda linéaire, $U[F]=\{t'/t\in F$ et $(t,t')\in r(U)\}$ avec U taa linéaire, $T[F]^{-1}=\{t'/t\in F$ et $(t',t)\in r(T)\}$ avec T quelconque sont reconnaissables.

Pour de plus amples développements voir [GEST84], [ENGE75].

4.2) Transducteurs d'arbres clos

- Un *transducteur d'arbres clos* (gtt pour ground tree transducer) est un couple V=(A,B) d'automates ascendants d'arbres . Soit $A=(\Sigma,Q_A,Q_{Af},R_A)$ et $B=(\Sigma,Q_B,Q_{Bf},R_B)$, la transformation associée à V est l'ensemble $r(V)=\{(t,t')$ / $t\in T_\Sigma,\ t'\in T_\Sigma,\ \exists s\in T_{\Sigma\cup(Q_A\cap Q_B)}$ tel que $t\overset{*}{\rightarrow}_A s\ {}_B\overset{*}{\leftarrow} t'\}$ c'est à dire encore:

r(V) est l'ensemble des couples (t,t') de termes de T_Σ tels que



avec c, contexte commun, appartient à $T_\Sigma(X_n)$ et, pour tout j, t_j, t'_j appartiennent à T_Σ, i_j appartient à $Q_A \cap Q_B$.
On peut remarquer que dans cette définition les ensembles Q_{Af} et Q_{Bf} ne jouent aucun rôle et donc on pourra les choisir égaux à l'ensemble vide. Par contre on peut noter l'importance des états de $Q_A \cap Q_B$, ces états sont appelés états interfaces.

Exemple: Soit Σ={a,b,c,f,g / ar(a)=ar(b)=ar(c)=0 et ar(f)=ar(g)=1}.
Considérons le système de réécriture clos
R={1: a → f(b); 2: b→ g(c); 3: f(g(c)) → g(a) }.

G_1: a → i_1 $\qquad$ D_1: b → q'_1 , f(q'_1)→ i_1
G_2: b → i_2 $\qquad$ D_2: c → q'_2 , g(q'_2)→i_2
G_3: c →q_1 , g(q_1)→ q_2 , f(q_2)→ i_3 $\qquad$ D_3: a → q'_3 , g(q'_3)→i_3
On a ainsi construit U=(G,D) avec $R_G=R_{G1}\cup R_{G2}\cup R_{G3}$ et $R_D=R_{D1}\cup R_{D2}\cup R_{D3}$.

On obtient E={ q'_3→ i_1 ($a\to_G i_1$, $a\to_D q'_3$)
q'_1→ i_2 ($b\to_G i_2$, $b\to_D q'_1$)
q'_2→ q_1 ($c\to_G q_1$, $c\to_D q'_2$)
i_2→ q_2 ($g(q_1)\to_G q_2$, $g(q'_2)\to_D i_2$, $q'_2 \to_E q_1$)
i_1→ i_3 ($f(q_2)\to_G i_3$, $f(q'_1)\to_D i_1$, $q'_1\to_E i_2 \to_E q_2$) }.

et on définit V=(A,B) en posant:
$R_A=R_G \cup$ { $i_2\to q_2$, $i_1\to i_3$} et $R_B=R_D \cup$ { $i_1\to q'_3$, $i_2\to q'_1$, $i_3\to i_1$}.
Avec R , a $\xrightarrow{*}_R$ g(a).
Avec U, $a \xrightarrow{*}_G i_1$ $_D\xleftarrow{*}$ f(b) $\xrightarrow{*}_G$ f(i_2) $_D\xleftarrow{*}$ f(g(c)) $\xrightarrow{*}_G i_3$ $_D\xleftarrow{*}$ g(a).
Avec V , a $\xrightarrow{*}_A i_3$ $_B\xleftarrow{*}$ g(a).
Les nouvelles règles créées permettent de supprimer les phases de génération-effacement.

Exemple: Soit $\Sigma=\{a,b,f,g$ / $ar(a)=ar(b)=0$, $ar(f)=1$ et $ar(g)=2\}$.
Considérons le système de réécriture clos
$R=\{1: g(a,b)\rightarrow g(b,a);\ 2: a\rightarrow f(a);\ 3: f(a)\rightarrow a\}$.

G_1: $a\rightarrow q_1$, $b\rightarrow q_2$, $g(q_1,q_2)\rightarrow i_1$ $\quad$ D_1: $b\rightarrow q'_1$, $a\rightarrow q'_2$, $g(q'_1,q'_2)\rightarrow i_1$
G_2: $a\rightarrow i_2$ $\quad$ D_2: $a\rightarrow q'_3$, $f(q'_3)\rightarrow i_2$
G_3: $a\rightarrow q_3$, $f(q_3)\rightarrow i_3$ $\quad$ D_3: $a\rightarrow i_3$

On obtient alors $R_A=R_{G1}\cup R_{G2}\cup R_{G3}\cup\{ i_3\rightarrow q_1, i_3\rightarrow q_3, i_3\rightarrow i_2, i_2\rightarrow i_3\}$ et $R_B=R_{D1}\cup R_{D2}\cup R_{D3}\cup\{ i_2\rightarrow q'_2 , i_2\rightarrow q'_3 , i_3\rightarrow i_2 , i_2\rightarrow i_3\}$.

Avec R , $g(f^3(a),b)\xrightarrow{*}_R g(b,f^2(a))$
Avec V, $g(f^3(a),b)\xrightarrow{*}_A i_1\ {}_B\xleftarrow{*} g(b,f^2(a))$.

- La classe des transformations associées aux gtt est close par inverse, composition, itération. Pour l'union, on a la propriété suivante: soit $V_1=(A_1,B_1)$ et $V_2=(A_2,B_2)$ deux gtt tels que les ensembles d'états utilisés pour V_1 et V_2 soient disjoints et V le gtt obtenu en prenant l'union des états et l'union des règles, on a $r(V)^*=(r(V_1)\cup r(V_2))^*$. Les preuves de ces résultats peuvent être trouvées dans [DATI85].

5) Systèmes de réécriture

5.1) Définition

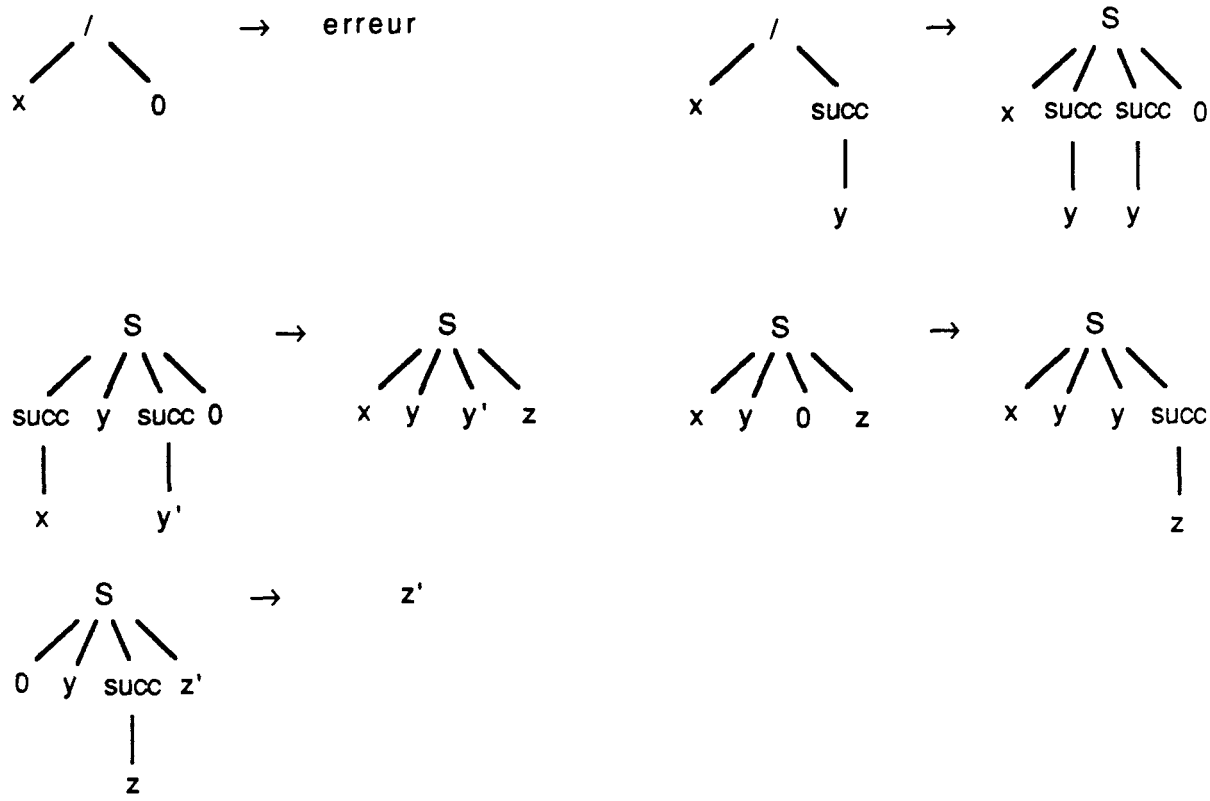
- Un *système de réécriture* $S=\{l_i\rightarrow r_i$ / $l_i,r_i\in T_\Sigma(X)$ et $V(r_i)\subset V(l_i)$, $i\in I\}$ sur T_Σ est un ensemble de couples de termes sur $T_\Sigma(X)$. Nous ne considérons que le cas I fini. $\rightarrow_S$ est la relation de réécriture induite par S c'est à dire $t\rightarrow_S t'$ si et seulement si t se dérive en t' par application d'une règle de S et $\xrightarrow{*}_S$ la clôture reflexive et transitive de $\rightarrow_S$.

5.2) Types de systèmes de réécritures

On peut considérer trois grandes classes dans les systèmes de réécriture:

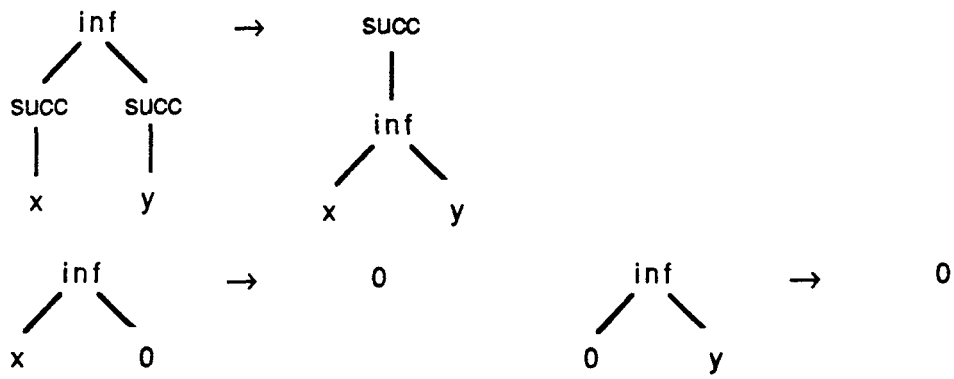
5.2.1) *Les systèmes généraux*: ensembles finis de règles de réécriture du type: $l\rightarrow r$.

exemple: nous donnons les règles qui permettent le calcul dans N du quotient entier de x par y.



5.2.2) *Les systèmes linéaires*: dans chaque membre de règle, les variables apparaissent au plus une fois.

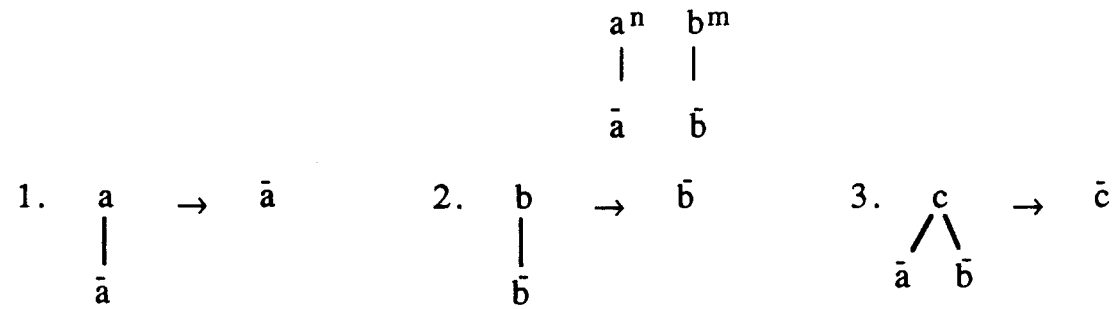
exemple: nous donnons les règles de calcul dans N de l'inf de x et y.



5.2.3) *Les systèmes clos*: chaque membre de règle est un terme clos (sans variable). Pour plus de développements voir [ROSE73], [HUOP80] et [DEJO90].

exemple: règles d'effacement de c (m,n) ∈ N^2

/ \



- Dauchet et Decomité ont montré qu'il y avait un fossé des systèmes généraux aux systèmes linéaires[DADE87]. Decomité a prouvé que l'on pouvait passer d'un système quelconque à un système linéaire mais que l'on perdait des propriétés très importantes (confluence ou terminaison finie)[DECO84].

5.3) Propriétés

- Un système de réécriture est *noethérien* si et seulement s'il n'existe pas de chaîne de dérivation infinie.
Cette propriété est indécidable pour un système quelconque même dans le cas d'une seule règle [DAUC89] mais est décidable pour un système clos ([GuKaMu82], [HuLa78]).

- Si $l \rightarrow r$ et $s \rightarrow t$ sont deux règles, p une occurrence de s telle que $s=c(s/p)$ et s/p ne soit pas réduit à une variable, et σ un unificateur le plus général de s/p et de l, alors le couple $(\sigma(t),\sigma(c)(\sigma(r)))$ est une *paire critique* obtenue à partir des deux règles.

- Un système est *confluent* si et seulement si:

$$\forall x, \forall y, \exists z \in T_\Sigma(X) \quad \begin{matrix} z \xrightarrow{*} x \\ z \xrightarrow{*} y \end{matrix} \quad \Rightarrow \quad \exists t,\ \begin{matrix} x \xrightarrow{*} t \\ y \xrightarrow{*} t \end{matrix}$$

Cette propriété est indécidable pour un système quelconque mais est décidable pour un système noethérien ([HuOp80]). Lorsque le système est linéaire à gauche, une condition suffisante sur les paires critiques a été donnée par Huet [HUET80] et étudiée par Jouannaud [JOUA83].
La décidabilité de la confluence pour des systèmes clos a été prouvée par Dauchet et Tison[DATI85].

- Un système est *Church-Rosser* si pour tout x et y, $x \overset{*}{\leftrightarrow}_S y$ entraine l'existence d'un z tel que $x \overset{*}{\rightarrow}_S z$ et $y \overset{*}{\rightarrow}_S z$ [CHRO36]. Les deux propriétés (confluence et Church-Rosser) sont équivalentes [NEWM42].

- Un système est *convergent* s'il est confluent et noethérien.

- Un système est *confluent sur les termes clos* si la propiété de confluence restreinte aux termes clos est vérifiée. Un système confluent est confluent sur les termes clos.

- Etant donnés un système S et un ordre partiel < sur S, nous définissons la *relation de réécriture IO* (par valeur) induite par S en respectant <, notée $\rightarrow_{i,S,<}$, de la façon suivante: $t \rightarrow_{i,S,<} t'$ si et seulement si $t \rightarrow_S t'$ par une règle appliquée à une occurrence o telle qu'aucune autre règle n'est applicable en dessous de o et qu'il n'y a pas de règle plus grande applicable à l'occurrence o. S'il n'y a pas d'ordre alors $\rightarrow_{i,S}$ est la relation de réécriture IO induite par S.

- Un terme est *irréductible* pour un système donné si on ne peut plus lui appliquer de règle du système. Nous notons IRR(S) l'ensemble de tous les termes irréductibles pour S. Une substitution σ est *close irréductible* pour un système S si pour toute variable x appartenant au domaine de σ, $\sigma(x)$ est close et irréductible.

- Nous notons S*(t) l'ensemble des transformés de t par le système S.

5.4 Problèmes d'accessibilité

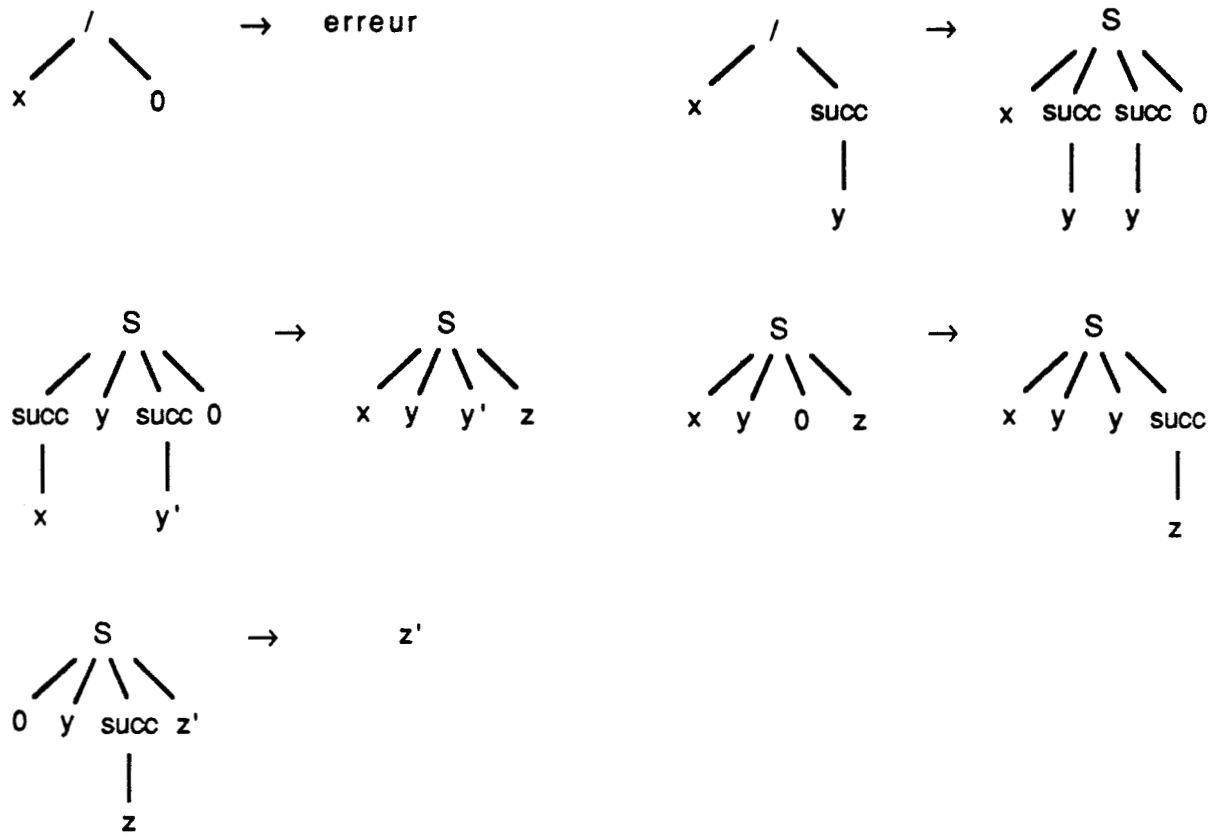
- Soit S un système de réécriture, le problème *d'accessibilité du premier ordre* est : Etant donnés deux termes t et t', peut-on réduire t en t' en utilisant S?
- Différents problèmes *d'accessibilité du second ordre* sont: Etant donnés deux langages d'arbres reconnaissables F et F', l'ensemble des réductions de termes de F par S est-il inclus dans F'? ou existe-t-il un terme de F qui se réduit en un terme de F' par S?,...

CHAPITRE II

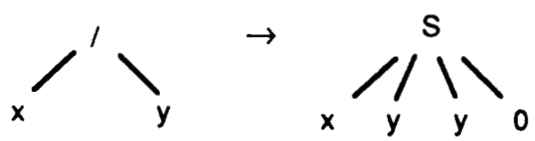
SYSTEMES DE REECRITURE CLOS AVEC CONTROLE RECONNAISSABLE

INTRODUCTION

Pour certaines applications, les systèmes de réécriture généraux ne correspondent pas à une modélisation naturelle du problème.
Si on reprend l'exemple:



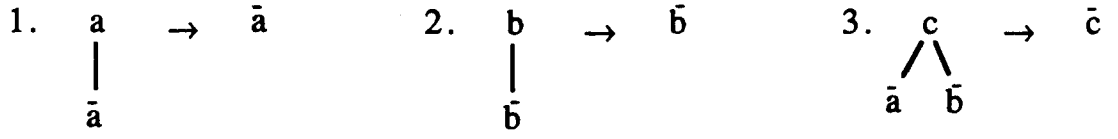
il semble plus naturel de remplacer les deux premières règles par la règle conditionelle suivante:

<u>non</u>(y = 0) :: 

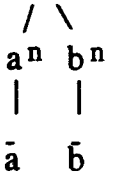
On peut définir les systèmes de réécriture contrôlés de différentes façons:

1) En contrôlant le système par un ensemble de chaînes de règles de réécriture (Control sets: Ginsburg-Spanier [GISP68], Greibach [GREI77]).

Reprenons l'exemple:

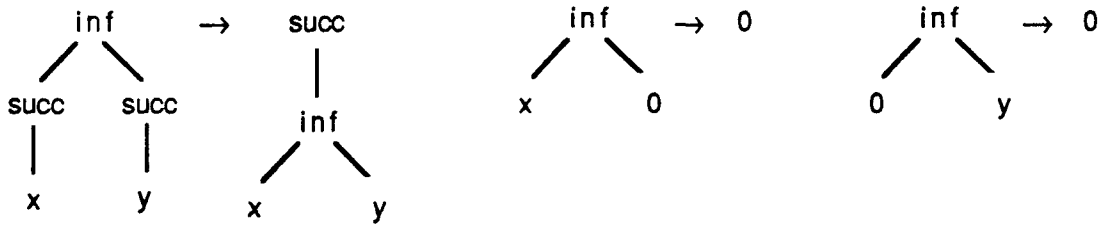


Si nous considérons le contrôle rationnel (1.2)*.3, nous obligeons le système à utiliser le même nombre de fois la règle 1 et la règle 2, donc l'effacement d'autant de b que de a. Ainsi, par application de la règle 3 en final,on contrôle l'effacement de c $n \in N$



2) En effectuant un contrôle sur les arguments, les spécifications étant construites hiérachiquement (Zhang et Rémy [ZHRE85]).

Si on reprend l'exemple:



< étant défini à un niveau inférieur de la hiérarchie, on peut remplacer le système par:

x<y : : inf → x
/ \
x y

non(x<y) : : inf → y
/ \
x y

Pour notre part, nous désirons simuler un système de réécriture par un système clos contrôlé; aussi le contrôle sur les arguments ne peut être utilisé ici, les règles devant être closes.

De plus, le contrôle par un ensemble de chaînes de règles détermine la succession de règles à appliquer mais ne correspond pas à la notion de garde que nous désirons étudier.

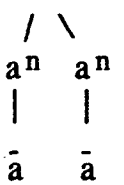
Aussi, nous allons considérer des systèmes clos gardés par précondition.

Une règle sera du type: $(F_i) :: \alpha_i \rightarrow \beta_i$ où F_i est une forêt reconnaissable

α_i et β_i sont des termes clos.

L'application d'une telle règle dans un arbre t se fera lorsque α_i sera un sous-arbre de t et que l'arbre t appartiendra à la forêt F_i (en fait, pour des simplifications évidentes d'écriture, nous ne spécifierons au niveau des préconditions que les sous-arbres utiles des arbres de F_i).

Exemple: processus d'effacement de



$n \in N$

Pour effectuer cet effacement, nous allons d'abord marquer les feuilles $\bar{a}$ à l'aide de deux symboles distincts.

$\underline{non}(a_1 \underline{ou} a_1') :: \bar{a} \rightarrow a_1$

$\underline{non}(a_2 \underline{ou} a_2') :: \bar{a} \rightarrow a_2$

Pour effacer le même nombre de a sur les deux branches, nous utilisons une bascule qui marque en alternance et de façon synchronisée a_1 et a_1' sur l'une des branches et a_2 et a_2' sur l'autre.

L'effacement d'un a sur la branche marquée à l'aide de l'indice 1 sera subordonnée à l'existence d'un a sur l'autre branche. Cet effacement entrainera alors l'effacement du a correspondant sur la branche indicée par 2 pour respecter la concordance des symboles. Grâce aux contrôles indiqués, au plus une des règles précédentes est applicable lors de la

réécriture. En effet, l'existence d'une lettre a sur les deux branches donne soit le séquencement $(a_1,a_2)\rightarrow(a_1',a_2)\rightarrow(a_1',a_2')$ soit le séquencement $(a_1',a_2')\rightarrow(a_1',a_2)\rightarrow(a_1,a_2)$.

La réécriture en $\bar{b}$ force la terminaison de l'un des deux séquencements indiqués ci-dessus et donc l'effacement de la dernière lettre a.

Malgré la contrainte très forte de règles closes, nous allons montrer que *l'adjonction de gardes rationnelles permet de simuler tout système en utilisant les systèmes particuliers obtenus.*

Pour simuler un système quelconque par des règles closes, nous allons:

1) coder l'arbre sous forme d'un peigne;

2) montrer que l'on peut simuler tout mouvement d'une machine de Turing par un système clos à garde close ce qui permet de simuler l'application de règles quelconques;

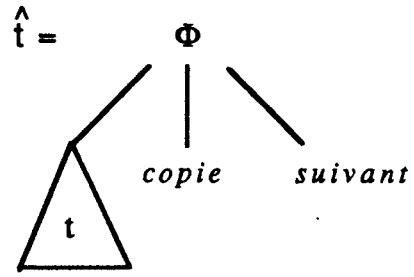
3) décoder le peigne afin de retrouver l'arbre transformé.

Nous allons utiliser dans les contrôles la notion de *produit tensoriel* défini comme suit:

si t est un p-uple d'arbres de $T_\Sigma(X_q)$ et t' un p'-uple d'arbres de $T_\Sigma(X_{q'})$ alors $t\otimes t'$ est le seul (p+p')-uple d'arbres de $T_\Sigma(X_{q+q'})$ tel que la i-ème composante de $t\otimes t'$ est égale à la i-ème composante de t si $1\leq i\leq p$, àla (i-p)-ème composante de t' si $i>p$ (en substituant x_{k+q} à x_k).
$t^{\otimes p}$ est défini par: $t^{\otimes 1}=t$ et $t^{\otimes p}=t^{\otimes(p-1)}\otimes t$.
Pour de plus amples développements, voir [ARDA79].

La mise sous forme de peigne doit être effectuée à un endroit précis; de plus, il faut préparer la simulation des mouvements de la machine de Turing.

Pour cela, nous allons utiliser les symboles spéciaux Φ d'arité 3, *copie* et *suivant* d'arité 0.
Plus précisement, nous allons associer à tout système R un système $\hat{R}$ clos gardé rationnellement qui va simuler le système R. A tout arbre t, nous associons l'arbre suivant:



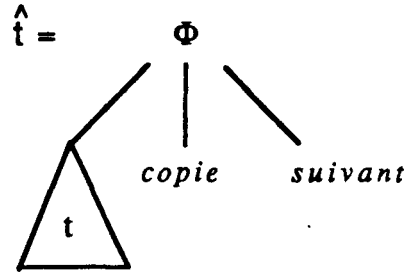
Posons $\hat{T}_\Sigma = \{\hat{t} / t \in T_\Sigma\}$ et Δ est l'ensemble des symboles particuliers utilisés lors de la simulation.

Nous allons montrer le théorème suivant:
$\forall (t,u) \in (T_\Sigma)^2$
$[\, t \xrightarrow{*}_R u] \Leftrightarrow [\, \hat{t} \rightarrow_{\hat{R}} t_1 \xrightarrow{*}_{\hat{R}} t_2 \rightarrow_{\hat{R}} \hat{u}$ / $(t_1,t_2) \in (T_{\Sigma\cup\Delta})^2$, $(\hat{t},\hat{u}) \in \hat{T}_\Sigma^2$ et
propriété (Q): chaque règle autre que la première est gardée par une forêt reconnaissable dont chaque arbre contient au moins un élément de Δ].

A partir de $\hat{t}$, l'application de la première règle amène l'apparition d'un symbole de Δ et lance la mise en peigne de l'arbre. La dernière règle appliquée termine le dépeignage de l'arbre en donnant l'arbre $\hat{u}$ et fait disparaître les derniers symboles de Δ apparaissant dans l'arbre.

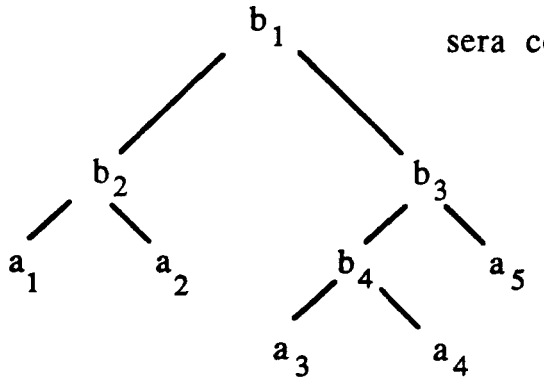
A. CODAGE D'UN ARBRE SOUS FORME DE PEIGNE

Nous partons de l'arbre :

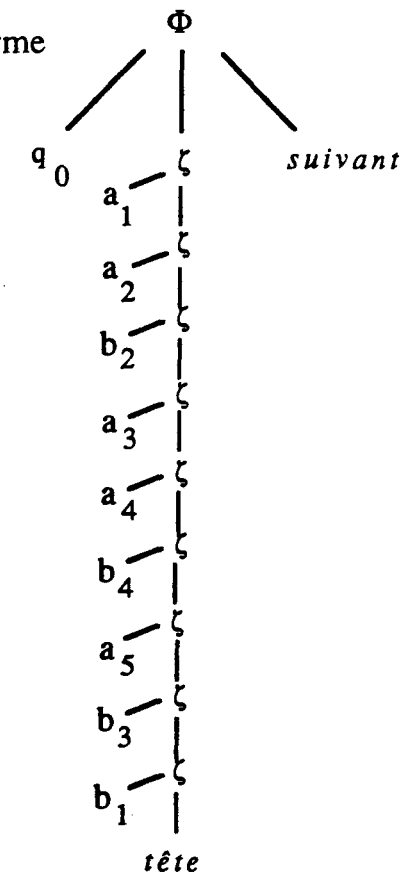


Nous allons copier l'arbre à la place *copie* . Cette copie sous forme de peigne est réalisée en lisant t de gauche à droite et en remontant vers la racine de t, ce qui correspond à la notation post-fixée sous forme de liste ("en peigne").

exemple. (q_0 servira pour indiquer la fin de la mise en peigne et permettra de lancer la simulation d'une machine de Turing)



sera copié sous la forme



Pour copier un symbole a, nous allons le marquer par un symbole spécial θ_a puis il nous faut copier ce symbole au niveau du peigne; il faut ensuite remplacer θ_a par le symbole θ afin de poursuivre la copie de l'arbre.
Lorsque tous les fils d'un noeud ont été recopiés, il faut recopier ce noeud avant tout autre symbole.

COPIE D'ARBRE

Début
  % copie de la première feuille %
  marquer une feuille a par δ_a;
  Vérifier le marquage;
  copier la 1ère feuille dans le peigne;
  effacer les contre-marques des mauvaises feuilles;
  remplacer δ_a par δ;
  % fin de copie de la 1ère feuille %
  % copie de l'arbre %
Tant que arbre non copié faire
    Si $b(\theta^{\otimes k})$ ou $b(\delta\otimes\theta^{\otimes(k-1)})$ alors
      le remplacer par θ_b ou δ_b;
      copier le noeud b dans le peigne;
      remplacer θ_b par θ ou δ_b par δ;
    sinon marquer une feuille a par θ_a;
      Vérifier le marquage;
      copier la feuille dans le peigne;
      effacer les contre-marques des mauvaises feuilles;
      remplacer θ_a par θ;
    finsi
fait
  % fin de copie de l'arbre %
Fin

avec:

Vérifier le marquage;
début
  tant que ce n'est pas la bonne feuille faire
    mettre une contre-marque sur la feuille non valable;
    marquer une autre feuille a;
  fait
Fin;

Nous allons décrire chacune de ces primitives .

Posons $\Sigma'_0 = \Sigma_0 \cup \{ \tilde{a} \ / \ a \in \Sigma_0 \}$, $\Sigma' = \Sigma \cup \Sigma'_0$ et $\Sigma'' = \{\zeta, \textit{tête}\} \cup \{\bar{a} \ / \ a \in \Sigma\}$
Les symboles $\tilde{a}$ serviront à marquer les feuilles a non valables lors de la procédure "Vérifier le marquage".

1) Marquer une feuille a par δ_a;

$a \in \Sigma_0$

$\Phi[T_{\Sigma\text{-}\Sigma_0}.(a \otimes \Sigma_0'^{\otimes}), \textit{copie}, \textit{suivant}\ ]$ et $[\text{non}(\delta_{a'})]a' \in \Sigma_0 \quad :: \quad a \to \delta_a$

Le contrôle $[\text{non}(\delta_{a'})]a' \in \Sigma_0$ permet de ne générer qu'un seul symbole de ce type.

2) Vérifier le marquage;

Il faut vérifier que c'est la bonne feuille qui a été marquée.

$a \in \Sigma_0$

$\text{non}[\Phi(T_{\Sigma\text{-}\Sigma_0}.(\delta_a \otimes \Sigma_0'^{\otimes}), \textit{copie}, \textit{suivant}\ )] \quad :: \quad \delta_a \to \tilde{a}$

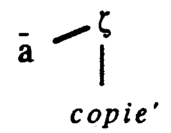
Le contrôle permet de constater que ce n'est pas la feuille la plus à gauche de l'arbre qui a été marquée.
Grâce au marquage $\tilde{a}$, la règle précédente peut à nouveau être appliquée.

3) Copier la première feuille dans le peigne;

Nous allons copier la première feuille au niveau de *copie*.

$a \in \Sigma_0$

$\Phi[T_{\Sigma\text{-}\Sigma_0}.(\delta_a \otimes \Sigma_0'^{\otimes}), \textit{copie}, \textit{suivant}] \quad :: \quad \textit{copie} \quad \to$



copie' est une constante.
Tous les symboles devant être réécrits sous forme de constantes, nous allons les "barrer" afin de ne plus faire intervenir leur arité et de les distinguer des éléments de Σ_0.

4) Effacer les contre-marques des mauvaises feuille;

$a \in \Sigma_0$

$\textit{copie'} \quad :: \quad \tilde{a} \to a$

5) Remplacer δ_a par δ;

Pour que la règle précédente ne soit plus appliquée à mauvais escient lors d'une copie ultérieure de feuille , nous devons aussi remplacer *copie'* par *tête*.

$a \in \Sigma_0$

$\underline{non}(\tilde{a})$ $\underline{et}$ *copie'* :: $\delta_a \rightarrow \beta$

$\underline{non}(\tilde{a})$ force l'effacement des contre-marques et *copie'* l'écriture effective de $\tilde{a}$ au niveau du peigne.

β :: *copie'* $\rightarrow$ *tête*

tête :: $\beta \rightarrow \delta$

L'utilisation d'une "bascule" (passage par β) empêche la réécriture de *copie'* lors d'une copie de feuille avant que les $\tilde{a}$ ne soient tous effacés.

6) Copie d'une feuille.

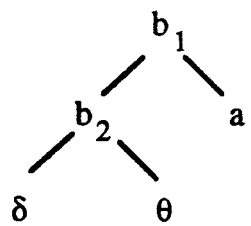
6.1) marquer une feuille a par θ_a;

$a \in \Sigma_0$

$\Phi[T_{\Sigma-\Sigma_0}.(\delta \otimes \theta^{\otimes} \otimes a \otimes \Sigma_0'^{\otimes}),\ T_{\Sigma''},\ suivant\ ]$ $\underline{et}$ $[\underline{non}(b[\delta \otimes \theta^{\otimes(k-1)}]\ \underline{ou}\ b[\theta^{\otimes k}])]_{b \in \Sigma-\Sigma_0}$ $\underline{et}$ $[\underline{non}(\theta_{a'})]_{a' \in \Sigma_0}$ :: $a \rightarrow \theta_a$

Les contrôles sur b permettent de vérifier que tous les noeuds situés à gauche de a ont été recopiés.

exemple:



a ne peut être réécrit avant que b_2 ne le soit.

6.2) Vérifier le marquage;

$a \in \Sigma_0$

$\underline{non}\ [\Phi(T_{\Sigma-\Sigma_0}.(\delta \otimes \theta^{\otimes} \otimes \theta_a \otimes \Sigma_0'^{\otimes}),\ T_{\Sigma''},\ suivant\ )]$ :: $\theta_a \rightarrow \tilde{a}$

6.3) copier la feuille dans le peigne;

$a \in \Sigma_0$

$\Phi[T_{\Sigma-\Sigma_0}.(\delta \otimes \theta^{\otimes} \otimes \theta_a \otimes \Sigma_0'^{\otimes}), T_{\Sigma''}, suivant]$ :: *tête* $\rightarrow$ $\tilde{a}$ — ζ | *copie'*

6.4) effacer les contre-marques des mauvaises feuilles;

$a \in \Sigma_0$

copie' :: $\tilde{a} \rightarrow a$

6.5) remplacer θ_a par θ;

$a \in \Sigma_0$

$\underline{non}(\tilde{a})$ $\underline{et}$ *copie'* :: $\theta_a \rightarrow \gamma$

γ :: *copie'* $\rightarrow$ *tête*

tête :: $\gamma \rightarrow \theta$

7) Copie d'un noeud.

Lorsque tous les fils d'un noeud ont été recopiés, il faut recopier ce noeud en priorité.

7.1) le remplacer par θ_b ou δ_b;

$b \in \Sigma - \Sigma_0$

$b(\theta^{\otimes k}) \rightarrow \theta_b$

$b[\delta \otimes \theta^{\otimes (k-1)}] \rightarrow \delta_b$

7.2) copier le noeud b dans le peigne;

$b \in \Sigma - \Sigma_0$

θ_b $\underline{ou}$ δ_b :: *tête* $\rightarrow$ $\bar{b}$ — ζ | *copie'*

7.3) remplacer θ_b par θ ou δ_b par δ;

$b \in \Sigma - \Sigma_0$

copie' :: $\theta_b \rightarrow \gamma$

γ :: *copie'* $\rightarrow$ *tête*

tête :: $\gamma \rightarrow \theta$

copie' :: $\delta_b \rightarrow \beta$

β :: *copie'* $\rightarrow$ *tête*

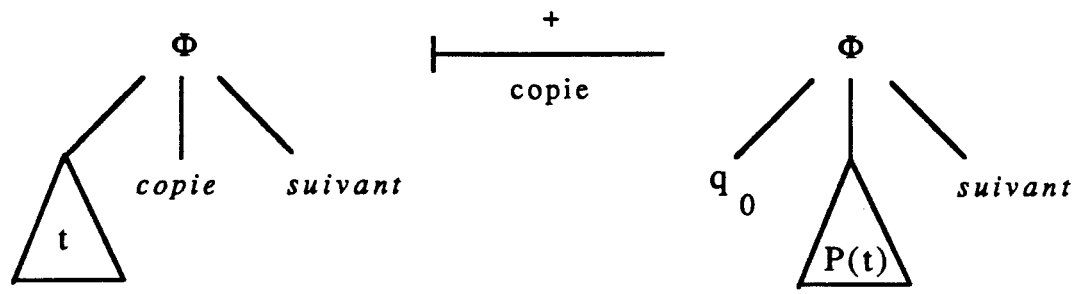
tête :: $\beta \rightarrow \delta$

8) Marquer la fin de la copie;

Lorsque la copie de l'arbre est terminée, nous remplaçons δ par un symbole spécial q_0 qui permet de lancer la simulation de la machine de Turing.

$\Phi(\ \delta,\ T_{\Sigma''},\ \textit{suivant}\ )\quad ::\quad \delta \rightarrow q_0$

LEMME A. *Le système contrôlé défini ci-dessus permet de réécrire un arbre du type $\hat{t}$ en codant t sous forme d'un peigne.*



en posant : *P(t) est l'écriture de l'arbre t en notation post-fixée sous forme de peigne.*

Preuve.

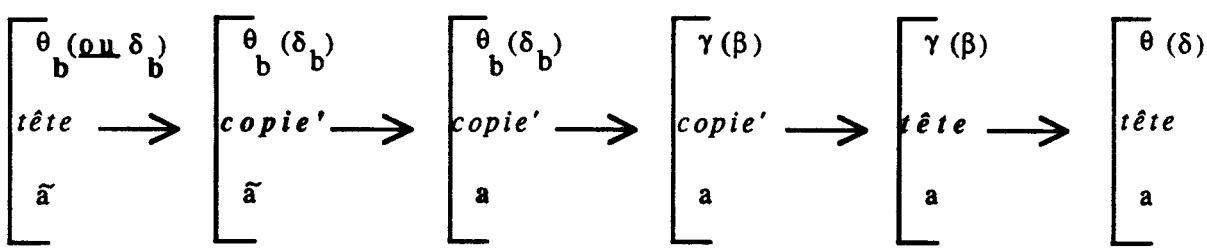
On vérifie aisément :

1) que tout noeud ne peut être marqué par un symbole spécial que si toute réécriture précédente est terminée (les seuls symboles spéciaux autorisés dans l'arbre sont des $\tilde{a}$ en cas de mauvais choix et δ et θ correspondants aux noeuds déjà réécrits),

2) que l'on ne peut marquer qu'un seul noeud à la fois (contrôle par $\underline{non}(\theta_{a'})$ ou $\underline{non}(\delta_{a'})$),

3) que c'est le bon noeud qui a été marqué en respectant la copie en notation post-fixée (l'écriture dans la peigne ne peut se faire que si cette condition est vérifiée).

Nous avons alors le cheminement suivant:

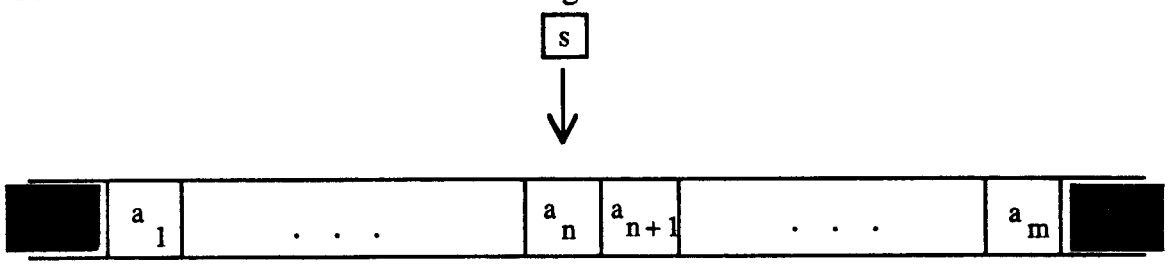


qui permet de contrôler la copie du symbole et la restauration des symboles permettant le marquage d'un autre noeud (en gras sont indiqués les symboles utiles au contrôle permettant la réécriture). ❐

B. SIMULATION D'UNE MACHINE DE TURING PAR SYSTEME CLOS à GARDE CLOSE

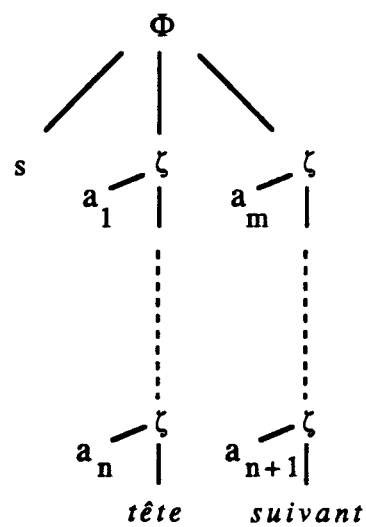
Il est bien connu que toute règle de réécriture peut être simulée par des mouvements de machine de Turing; aussi nous allons montrer comment nous pouvons simuler tout mouvement d'une machine de Turing par des règles closes à garde close.

Considérons une machine de Turing T de la forme :



s correspond à l'état, $a_1,\ldots,a_n$ aux contenus de la bande.

T est simulée par un terme T' de la forme:



Nous utilisons les symboles spéciaux suivants:

1) s_a qui correspond au fait que T est sur le point d'éxécuter la transition d'état labellée par s,a;

2) les symboles *tête'*, *suivant'* et $suivant_a$ pour marquer les états intermédiaires et pour éxécuter les mouvements de la tête lecture/écriture;

La simulation fonctionne de la manière suivante:

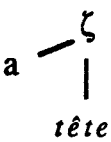
1) entrer une nouvelle transition;

1. s et  :: s → s_a

2) éxécuter l'instruction;

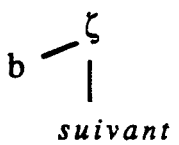
2.1) effectuer un mouvement gauche:

2.1.1 s_a et *tête* :: *suivant* → 

2.1.2 s_a et *suivant'* ::  → *tête'*

2.1.3 s_a et *tête'* :: *suivant'* → *suivant*

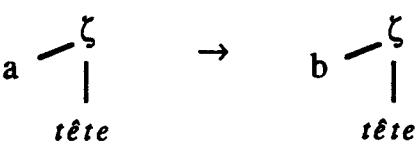
2.2) effectuer un mouvement droit:

2.2.1 s_a et *tête* ::  → *suivant* $_b$

2.2.2 s_a et *suivant* $_b$ :: *tête* → 

2.2.3 s_a et *tête'* :: *suivant* $_b$ → *suivant*

2.3) écriture de "b" sur le ruban:

2.3 s_a ::  →

3) prendre l'état suivant (noté t):

3.1 *suivant* et *tête'* :: s_a → t

3.2 t :: *tête'* → *tête*

LEMME B. *Tout mouvement d'une machine de Turing peut être simulé grâce aux règles précédentes.*

Preuve.
Tout mouvement est préparé grâce à la règle 1 qui correspond à la partie gauche de la transition à effectuer.

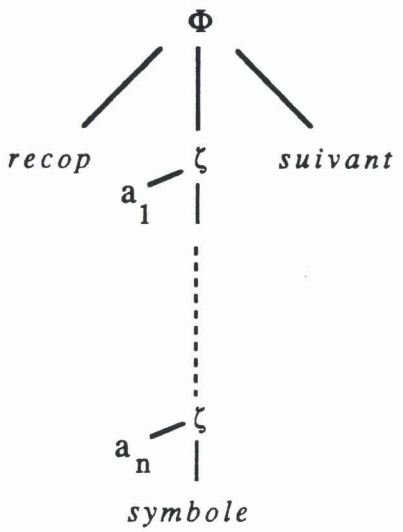
1) La machine effectue un mouvement de déplacement à gauche.
La règle 2.1.1 permet d'effectuer le décalage à gauche en réécrivant le symbole a au niveau du peigne droit, 2.1.2 efface le symbole du peigne gauche, 2.1.3, 3.1 et 3.2 remettent T' sous sa forme normale permettant de poursuivre la simulation.

2) La machine effectue un mouvement de déplacement à droite.
La règle 2.2.1 efface le symbole b du peigne droit, 2.2.2 écrit le symbole au niveau du peigne gauche, 2.2.3, 3.1 et 3.2 remettent T' sous sa forme normale permettant de poursuivre la simulation.

3) La machine effectue une écriture.
La règle 2.3 effectue la même écriture au niveau du peigne gauche, 3.1 et 3.2 remettent T' sous sa forme normale permettant de poursuivre la simulation.□

C. DECODAGE DU PEIGNE EN ARBRE

Nous allons pour terminer réécrire l'arbre sous sa forme classique en respectant les arités des noeuds. Cette recopie se fera de droite à gauche et en descendant en correspondance avec la méthode employée au B.
Pour lancer la recopie de l'arbre à la fin de la simulation de la machine de Turing, nous repositionnons la tête de lecture à l'extrémité droite de la bande, nous remplaçons s par *recop* et *tête* par *symbole*.
Nous avons alors l'arbre:



Nous recopions la feuille la plus profonde du peigne à la place de *recop* puis nous effaçons cette feuille du peigne.
Deux cas peuvent se présenter lorsque nous recopions une feuille du peigne:

1) la feuille correspond à un élément de Σ-Σ_0.

Nous recopions le symbole correspondant à cette feuille en marquant par un symbole spécial son fils le plus à droite qui sera l'emplacement où réécrire le prochain symbole.

2) la feuille correspond à une constante.

Si elle est la dernière feuille du peigne, il suffit de la recopier sinon il faut la recopier puis marquer par un symbole spécial la feuille située juste à sa gauche qui sera l'emplacement où réécrire le prochain symbole.

RECOPIE D'ARBRE

Début

% recopie de la première feuille %

recopier la première feuille;

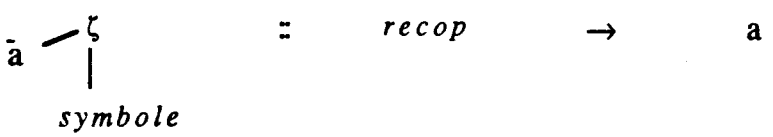
effacer la première feuille dans le peigne;

```
    % fin de recopie de la première feuille %
    % recopie de l'arbre %
  Tant que arbre non recopié faire
      Si la feuille correspond à un élément de Σ-Σ0 alors
          recopier le symbole;
          effacer le symbole dans le peigne;
      sinon recopier la feuille;
          Si non[dernière feuille du peigne] alors
            marquer un parent gauche non connu;
            Vérification de marquage;
            effacer les contre-marques des mauvais parents;
          finsi
          effacer la feuille dans le peigne;
      finsi
  fait
    % fin de recopie de l'arbre %
  Fin
```

Nous allons décrire chacune de ces primitives .

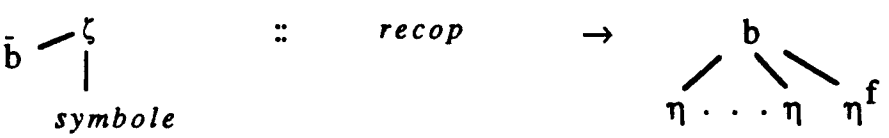
1) recopier la première feuille;

1.1) la feuille correspond à un élément de Σ_0:



1.2) la feuille correspond à un élément b de Σ-Σ_0:

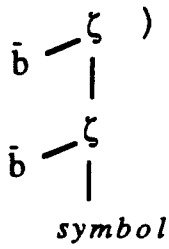
Les fils du symbole à recopier sont marqués à l'aide du symbole η, le dernier fils à droite indiquant la place de la prochaine recopie étant marqué de façon spéciale par le symbole η^f.



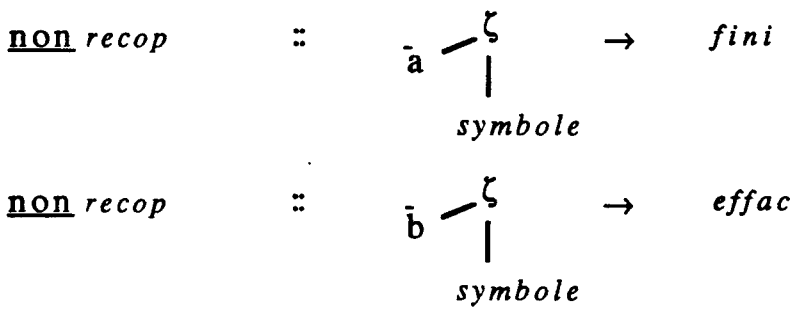
2) effacer la première feuille dans le peigne;

Le contrôle non *recop* empêche l'effacement de la feuille avant qu'elle ne soit réécrite.

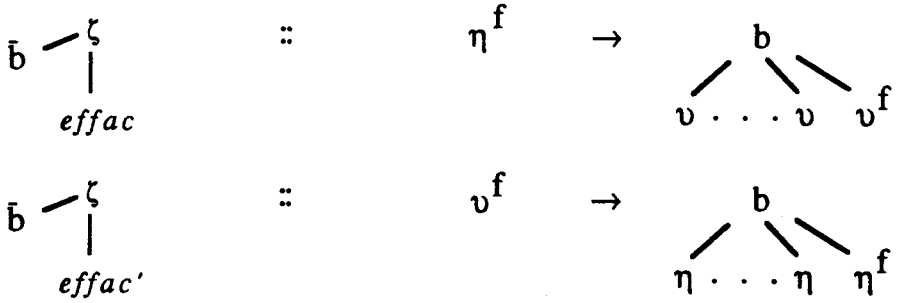
Nous utilisons le symbole *effac* afin d'éviter les effacements intempestifs,

(dans le cas 

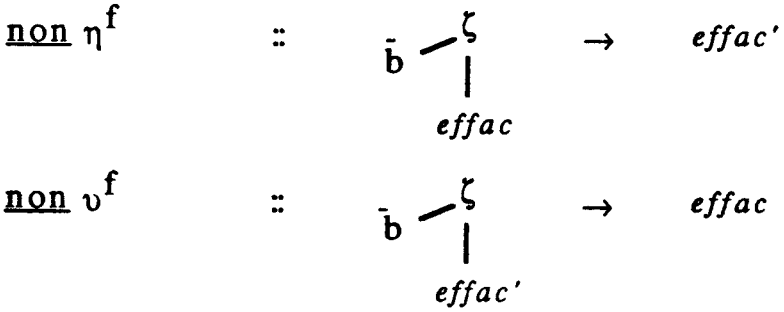
)



3) recopier le symbole;
Pour empêcher des réécritures intempestives, nous allons utiliser pour les fils du noeud les symboles υ, υ^f et η, η^f en alternance;
$b \in \Sigma\text{-}\Sigma_0$



4) effacer le symbole dans le peigne;
Pour empêcher des effacements successifs dans le peigne, nous utilisons en alternance *effac* et *effac'*.
$b \in \Sigma\text{-}\Sigma_0$



5) recopier la feuille a ($a \in \Sigma_0$);

$\bar{a}\,/\,\zeta$ (*effac*) :: $\eta^{f} \rightarrow a$

$\bar{a}\,/\,\zeta$ (*effac'*) :: $\upsilon^{f} \rightarrow a$

6) marquer un parent gauche non connu;
Nous allons d'abord commencer l'effacement de la feuille afin d'empêcher des réécritures successives en utilisant le symbole intermédiaire *effac"*.

$\underline{\text{non}}\ \eta^{f}\ \underline{\text{et}}\ (\upsilon\ \underline{\text{ou}}\ \eta)$ :: $\bar{a}\,/\,\zeta$ (*effac*) $\rightarrow$ *effac"*

$\underline{\text{non}}\ \upsilon^{f}\ \underline{\text{et}}\ (\upsilon\ \underline{\text{ou}}\ \eta)$ :: $\bar{a}\,/\,\zeta$ (*effac'*) $\rightarrow$ *effac"*

Nous marquons un parent et un seul à l'aide du symbole intermédiaire η^{f_1}. Si le symbole marqué n'est pas le bon, nous mettons une contre-marque $\bar{\eta}$

Posons $H = \{\eta, \bar{\eta}\}$ et $N = \{\upsilon, \bar{\upsilon}\}$.

effac" $\underline{\text{et}}\ (\underline{\text{non}}\ \eta^{f_1})\ \underline{\text{et}}\ b(H^{\otimes} \otimes \eta \otimes T_{\Sigma}^{\otimes})$ :: $\eta \rightarrow \eta^{f_1}$

effac" $\underline{\text{et}}\ (\underline{\text{non}}\ \upsilon^{f_1})\ \underline{\text{et}}\ b(N^{\otimes} \otimes \upsilon \otimes T_{\Sigma}^{\otimes})$ :: $\upsilon \rightarrow \upsilon^{f_1}$

7) Vérification de marquage;

$\underline{\text{non}}\ b(H^{\otimes} \otimes \eta^{f_1} \otimes T_{\Sigma}^{\otimes})_{b \in \Sigma - \Sigma_0}$ :: $\eta^{f_1} \rightarrow \bar{\eta}$

$b(H^{\otimes} \otimes \eta^{f_1} \otimes T_{\Sigma}^{\otimes})$ :: $\eta^{f_1} \rightarrow \eta^{f}$

Nous procédons de la même façon avec les symboles υ :

$$\underline{\text{non}}\ b(N^{\otimes} \otimes \upsilon^{f_1} \otimes T_{\Sigma}^{\otimes})_{b \in \Sigma - \Sigma_0} \quad :: \quad \upsilon^{f_1} \rightarrow \bar{\upsilon}$$

$$b(N^{\otimes} \otimes \upsilon^{f_1} \otimes T_{\Sigma}^{\otimes}) \quad :: \quad \upsilon^{f_1} \rightarrow \upsilon^{f}$$

8) effacer les contre-marques des mauvais parents;

$$\eta^{f} \quad :: \quad \bar{\eta} \rightarrow \eta$$

$$\upsilon^{f} \quad :: \quad \bar{\upsilon} \rightarrow \upsilon$$

9) effacer la feuille dans le peigne;

9.1) ce n'est pas la dernière feuille.

Dans ce cas, il y a eu marquage du symbole *effac"* au niveau du peigne. Nous contrôlons qu'il y a eu marquage de la feuille située juste à gauche du symbole et nous remplaçons *effac"*.

$$\underline{\text{non}}\ \bar{\eta}\ \underline{\text{et}}\ \eta^{f} \quad :: \quad effac'' \rightarrow effac$$

$$\underline{\text{non}}\ \bar{\upsilon}\ \underline{\text{et}}\ \upsilon^{f} \quad :: \quad effac'' \rightarrow effac'$$

9.2) c'est la dernière feuille du peigne.

Dans ce cas, nous mettons le symbole terminal *fini* au niveau du peigne.

$$\underline{\text{non}}\ \eta^{f}\ \underline{\text{ou}}\ \eta\ \underline{\text{ou}}\ \upsilon \quad :: \quad \bar{a} \swarrow \zeta \ (\text{effac}) \rightarrow fini$$

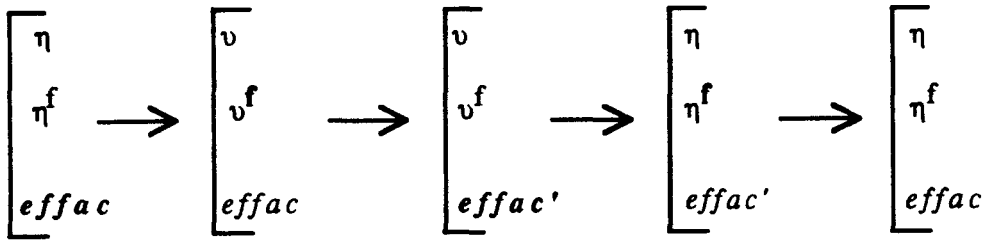
$$\underline{\text{non}}\ \upsilon^{f}\ \underline{\text{ou}}\ \eta\ \underline{\text{ou}}\ \upsilon \quad :: \quad \bar{a} \swarrow \zeta \ (\text{effac}') \rightarrow fini$$

LEMME C. *Le sytème contrôlé défini ci-dessus permet de réécrire un arbre donné sous forme de peigne en son écriture normale respectant l'arité des symboles.*

Preuve.

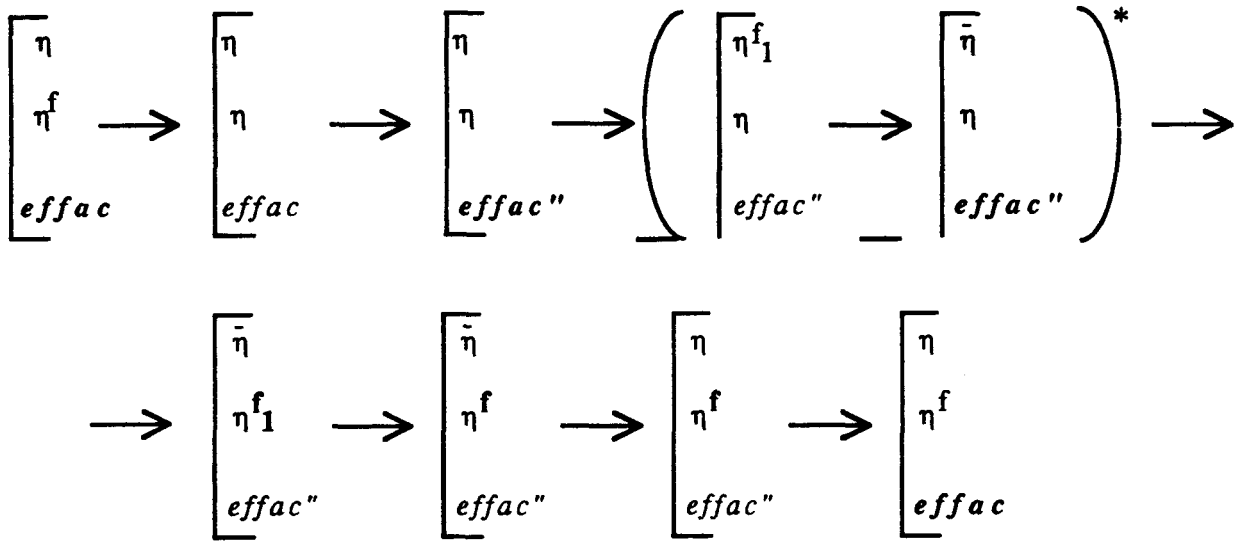
La remise en forme de l'arbre est lancée grâce à l'apparition des symboles *recop* et *symbole*(règle 1.1 ou 1.2).

Nous avons ensuite les enchainements suivants:



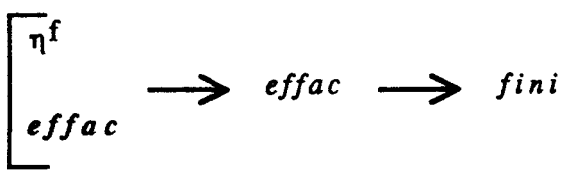
Le seul problème qui se pose est: lorsque nous recopions un élément de Σ_0, il n'y a plus de symbole η^f ou υ^f indiquant l'emplacement du prochain symbole à réécrire.

1) si ce n'est pas la dernière feuille, il faut marquer la feuille située juste à sa gauche. Ceci est réalisé de la façon suivante:



L'enchaînement est réalisé de la même façon si la feuille à marquer est υ; dans ce cas, à la fin de la séquence, nous avons les symboles υ^f et *effac'*.
Nous procédons de la même façon en partant de (υ, *effac'*).

2) c'est la dernière feuille à recopier.
Nous avons la séquence finale:



□

D. THEOREME DE SIMULATION.

THEOREME. *Tout système de réécriture peut être simulé par un système clos à contrôle reconnaissable.*

$\forall (t,u) \in (T_\Sigma)^2$

$[\, t \xrightarrow{*}_R u] \Leftrightarrow [\, \hat{t} \rightarrow_{\hat{R}} t_1 \xrightarrow{*}_{\hat{R}} t_2 \rightarrow_{\hat{R}} \hat{u}$ / $(t_1,t_2) \in (T_{\Sigma\cup\Delta})^2$, $(\hat{t},\hat{u}) \in \hat{T}^2_\Sigma$ et propriété (Q) : chaque règle autre que la première est gardée par une forêt reconnaissable dont chaque arbre contient au moins un élément de Δ].

Preuve.

1) $\Rightarrow$

Grâce au lemme A, nous savons qu'en partant de $\hat{t}$, t peut être mis sous forme de peigne avec des règles gardées par une forêt reconnaissable dont chaque arbre contient un élément de Δ (excepté pour la première règle).

Sachant que toute règle de réécriture peut être simulée par des mouvements de machine de Turing, grâce au lemme B, nous pouvons simuler l'application de règles de R.

Enfin grâce au lemme C, nous savons que nous obtenons l'arbre $\hat{u}$.

2) $\Leftarrow$

Si $\hat{t}$ a été réécrit en $\hat{u}$, il y a eu simulation de mouvements de machine de Turing qui correspondent à des applications de règles de R donc à une réécriture de t en u. ❐

E. CONCLUSION

Nos résultats prouvent que l'approche "contrôle reconnaissable" est inadéquate du point de vue de la programmation équationnelle. En effet, si nous analysons la signification de ces résultats, nous voyons que si nous considérons la classe très faible des systèmes clos dont la théorie est décidable ([DATI90]), en ajoutant la notion de contrôle reconnaissable, nous obtenons tout (par des simulations d'une complexité rédhibitoire). Si nous considérons des opérateurs surchargés et si, à chaque pas de calcul, nous imposons aux opérateurs d'appartenir à une certaine sous-sorte (c'est le sens du contrôle reconnaissable), nous voyons que l'application de ces contraintes de sorte fait "exploser" du domaine des reconnaissables au domaine des récursivement énumérables le comportement décrit en termes de langages.

Même si nous nous restreignons aux contrôles clos, les contrôles reconnaissables introduits expriment des priorités sur la réécriture. Par exemple, si nous considérons deux processus (P): $\alpha \rightarrow \beta$ et (P'): $\alpha' \rightarrow \beta'$, pour exprimer que (P') est prioritaire sur (P), nous utiliserons le contrôle clos négatif $\underline{non}(\alpha')$ pour pouvoir appliquer $\alpha \rightarrow \beta$. En ce sens, le problème de mise sous forme de peigne peut s'exprimer sous la forme: peut-on, avec des réécritures closes à priorité, passer d'une structure quelconque à une structure de liste?

Du point de vue structurel de la théorie des arbres, nous pouvons donc nous poser les questions suivantes:

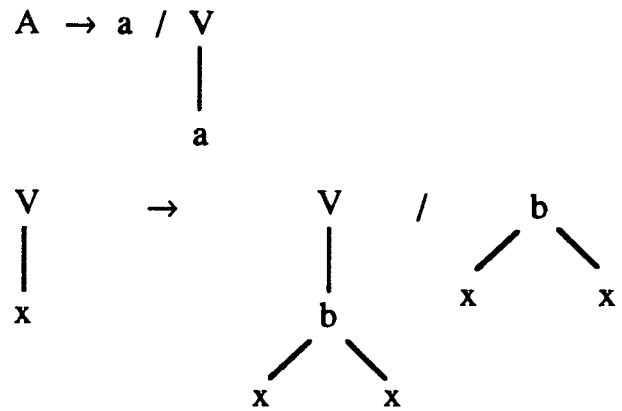
1) Pouvons nous simuler un système par un système clos avec des contrôles clos positifs uniquement?

2) Pouvons nous simuler un système par un système clos avec des contrôles clos positifs ou négatifs?

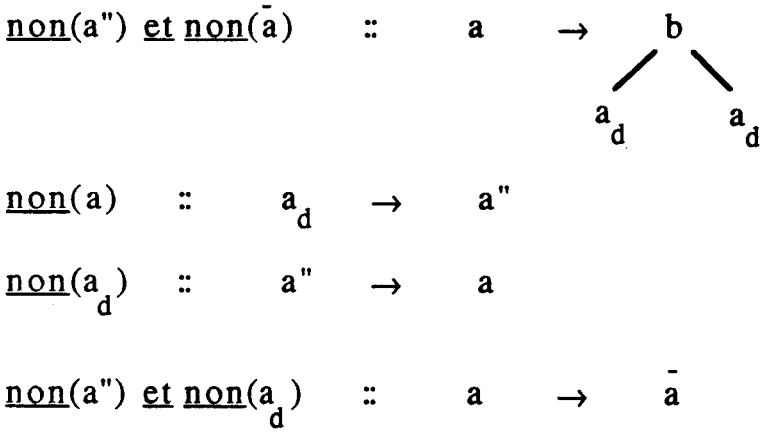
L'exemple ci-dessous montre que l'usage de contrôle clos positif ou négatif a une capacité surprenante qui rend ces questions plus difficiles qu'il n'y parait.

Exemple de système simulable par un système clos à contrôle positif ou négatif.

Nous considérons l'ensemble Σ = {b, a, ar(b)=2, ar(a)=0}
Nous obtenons l'ensemble des arbres équilibrés à partir de l'unique symbole A grâce au système suivant:



Nous pouvons simuler ce système grâce au système clos à contrôle clos suivant:



D'un point de vue plus pragmatique (par exemple si on voulait étudier le comportement de processus décrits par des systèmes clos avec priorité), nos résultats montrent qu'il n'y a pas de cadre général garantissant quoi que ce soit puisque nous avons établi que, même si la classe était très faible, nous pouvions tout faire.
Des sous-classes particulières réalistes seraient alors à étudier mais cela n'était pas notre objectif, qui était seulement de cerner globalement le comportement de ce type de contrôle.

CHAPITRE III

ENSEMBLES D'ARBRES DE DERIVATION PAR SDR CLOS

INTRODUCTION

Le problème d'accessibilité du premier ordre est de décider si, étant donné un système de réécriture S et deux termes t et t', t peut être réduit en t' en utilisant des règles de S.
Un des problèmes d'accessibilité du second ordre est de décider si, étant donné un système de réécriture S et deux forêts reconnaissables F et F', il existe un terme de F qui se réécrit en un terme de F' en utilisant des règles de S.
Etant donné que ces problèmes d'accessibilité sont décidables pour des systèmes clos, nous nous posons naturellement le problème suivant: si t peut être réduit en t', quelles règles de S peuvent être utilisées pour le faire?
Nous introduisons la notion d'arbre de dérivation comme étant un terme sur un nouveau alphabet gradué afin d'obtenir "l'histoire" d'une réduction de t en t' par S. La frontière d'un arbre de dérivation donnera la séquence des règles utilisées pour réduire t en t'; de plus nous saurons à quelle occurence dans l'arbre une règle a été appliquée.
Nous prouvons que l'ensemble des arbres de dérivation est un langage d'arbres reconnaissable. Nous donnons un algorithme en temps linéaire fournissant un arbre de dérivation "courte" pour le problème d'accessibilité.

A-ETUDE DE L'ENSEMBLE DES PREUVES DANS LE CAS STANDARD

Le problème d'accessibilité du premier ordre est de décider si, étant donné un système de réécriture S et deux termes clos t et t', t peut être réduit en t' en utilisant des règles de S. Ce problème est décidable pour S, aussi, étant donnés t et t', nous pouvons décider si t se réduit en t' en utilisant S.
Notre but est de trouver une ou toutes les dérivations possibles de t en t' par S. Nous définissons des arbres de dérivation comme étant des termes sur T_Δ où Δ est un nouveau alphabet gradué tel que la frontière d'un terme donne la séquence des règles utilisées pour réduire t en t' par S.
Nous ne considérons que des systèmes de réécriture clos standards que nous définirons plus loin. Dans la Section A.0, nous définissons la notion d'arbre de dérivation, dans la Section A.1, nous déterminons les langages reconnaissables d'arbres de dérivation pour le problème d'accessibilité du premier ordre pour (t,t',S), dans la Section A.2, nous donnons la construction pour le problème d'accessibilité du second ordre pour (F,F',S) et dans la Section A.3, la construction d'un arbre de dérivation pour (t,t',S) en temps linéaire.

$S=\{l_i \rightarrow r_i \ / i\in I\}$ est un système de réécriture clos de *règles standards*, c'est à dire que nous avons trois types de règles:

(i) Règles effaçantes: $f(a_1,...,a_n)\rightarrow a$, $f\in\Sigma_n$, $a_1,...,a_n,a\in\Sigma_0$

(ii) Règles génératrices: $a\rightarrow f(a_1,...,a_n)$, $f\in\Sigma_n$, $a_1,...,a_n,a\in\Sigma_0$

(iii) ε-règles: $a\rightarrow b$, $a,b\in\Sigma_0$

Nous utilisons un nouvel alphabet fini gradué Δ:

$\Delta=\Sigma\cup I\cup\{\beta_c/c\in\Sigma,\ c\notin\Sigma_0,\ ar(\beta_c)=ar(c)+2\}\cup\{\alpha,\#/\ ar(\alpha)=ar(\#)=2\}$.

Les symboles β_c seront utilisés lorsque nous appliquerons une séquence génération-effacement avec c comme racine. Nous indiquerons au niveau des fils de β_c les numéros des règles appliquées et les dérivations éventuelles concernant les fils de c.
α sera utilisé lorsque nous appliquerons des ε-règles et # pour séparer sous un contexte commun non transformé les sous-arbres de t et t' qui se correspondent par application de règles de S.

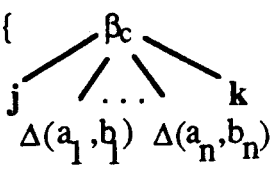
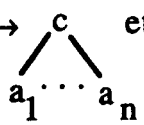
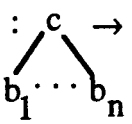
A-0 Notion d'arbre de dérivation

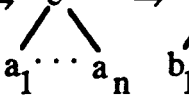
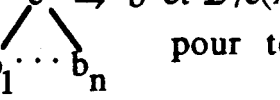
Nous allons donner en parallèle la définition syntaxique et la définition sémantique des arbres de preuve. Auparavant, nous explicitons certaines notations utilisées lors de ces définitions.

$\mathfrak{l}(\Delta)$ sera employé pour donner l'interprétation de l'ensemble Δ, c'est à dire l'ensemble des dérivations correspondant à Δ,
le symbole "." correspond à la concaténation des dérivations,
$\mathfrak{D}$ est l'abréviation pour désigner une séquence de dérivations où l'on indique à chaque pas de dérivation le numéro de la règle utilisée et l'endroit de son utilisation (par l'indice de Dewey par exemple), ainsi $t \xrightarrow{m,i} u$ signifie qu'on a appliqué la règle i au noeud d'indice de Dewey m,
$\mathfrak{D}/u(x_i)$ correspond à la restriction de $\mathfrak{D}$ à la branche située au niveau de x_i dans le contexte u qui n'est pas modifié lors de la séquence de dérivation, sans s'occuper de ce qui se passe au niveau des autres branches,
δ désigne un arbre de dérivation,
$\Delta(t,t')$ désigne l'ensemble des arbres de dérivations de t en t',
$\Delta^1(t,a)$ l'ensemble des arbres de dérivations premières de t en a ($a \in \Sigma_0$) c'est à dire pour les dérivations où l'on n'obtient jamais d'arbre de hauteur 0 avant d'arriver à a,
$\Delta^<(a,b)$ l'ensemble des arbres de dérivations strictes de a en b ($a,b \in \Sigma_0$) c'est à dire pour les dérivations effectives où l'on n'obtient jamais d'arbre de hauteur 0 entre a et b.

Définitions syntaxiques et sémantiques:

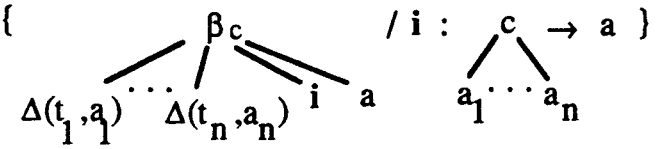
- pour tout a et tout b appartenant à Σ_0,

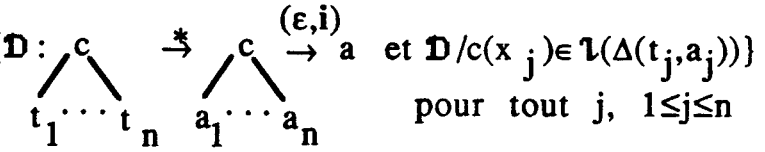
$\Delta^<(a,b) = \{i \,/\, i : a \to b\} \cup \{$  $/\, j : a \to$  et $k :$  $\}$

$\mathfrak{l}(\Delta^<(a,b)) = \{\mathfrak{D} : a \xrightarrow{(\varepsilon,i)} b\} \cup \{\mathfrak{D} : a \xrightarrow{(\varepsilon,j)}$  $\xrightarrow{*}$  $\xrightarrow{(\varepsilon,k)} b$ et $\mathfrak{D}/c(x_i) \in \mathfrak{l}(\Delta(a_i,b_i))\}$ pour tout i, $1 \le i \le n$

(on dérive a en b soit par l'application d'une ε-règle soit par génération d'une racine c, puis dérivation quelconque des feuilles de c et finalement effacement de c, ce qui amène l'écriture de b).

- pour tout a appartenant à Σ_0 et tout arbre $t=c(t_1,\ldots,t_n)$ de hauteur supérieure ou égale à 1

$\Delta^1(t,a) = \{$  $/\ i : c \to a \}$

$\mathfrak{L}(\Delta^1(t,a)) = \{\mathfrak{D}:$  $\overset{*}{\to}$ $\overset{(\varepsilon,i)}{\to} a$ et $\mathfrak{D}/c(x_j) \in \mathfrak{L}(\Delta(t_j,a_j))\}$ pour tout j, $1 \le j \le n$

(on dérive t en a uniquement en réécrivant au niveau des branches de c, seule la dernière dérivation amène l'effacement de c pour obtenir a).

- pour tout a appartenant à Σ_0,

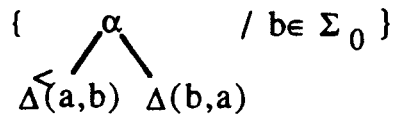
$\Delta^1(a,a)=\{a\}$ et $\mathfrak{L}(\Delta^1(a,a))=\{\mathfrak{D}: a \overset{\varepsilon,\varnothing}{\to} a\}$ (dérivation de longueur 0)

- pour tout a et tout b appartenant à Σ_0, $a \neq b$

$\Delta^1(a,b)=\varnothing$

(on ne peut passer de a à b sans effectuer de dérivation).

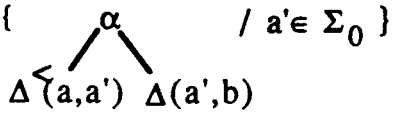
- pour tout a appartenant à Σ_0,

$\Delta(a,a) = \{a\} \cup \{$  $/\ b \in \Sigma_0 \}$

$\mathfrak{L}(\Delta(a,a))=\{\mathfrak{D}: a \overset{\varepsilon,\varnothing}{\to} a\} \cup \{\mathfrak{D}: a \overset{+}{\to} b \overset{*}{\to} a\}$

(on dérive a en a soit par dérivation de longueur 0 soit par dérivation stricte de a en b puis par dérivation de b en a).

- pour tout a et tout b appartenant à Σ_0, $a \neq b$

$\Delta(a,b) = \{$  $/\ a' \in \Sigma_0 \}$

$\mathfrak{L}(\Delta(a,b))=\cup\{\mathfrak{D}: a \overset{+}{\to} a' \overset{*}{\to} b\}$

(on dérive a en b par dérivation stricte de a en a' puis par dérivation de a' en b).

- pour tout a appartenant à Σ_0 et tout arbre $t=c(t_1,\ldots,t_n)$ de hauteur supérieure ou égale à 1

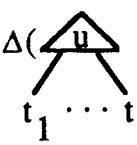
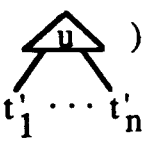
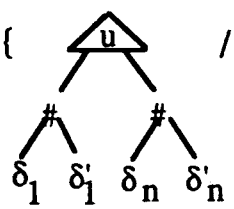
$\Delta(t,a) = \{ \quad \alpha \quad / \ a' \in \Sigma_0 \}$

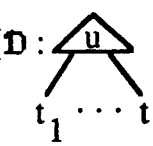
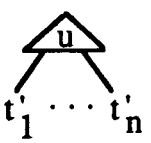
$\Delta^1(t,a') \quad \Delta(a',a)$

$\mathfrak{l}(\Delta(t,a)) = \mathfrak{l}(\Delta^1(t,a')).\mathfrak{l}(\Delta(a',a))$

(on dérive t en a par dérivation première de t en a' puis par dérivation de a' en a).

- $\delta_i \in \Delta^1(t_i,a_i)$, $\delta'_i \in \Delta(a_i,t'_i)$, $a_i \in \Sigma_0$, $1 \le i \le n$,

$\Delta = \Delta($  ,  $) = \{$  $/ \ \delta_i \in \Delta^1(t_i,a_i), \delta'_i \in \Delta(a_i,t'_i), 1 \le i \le n \}$

$\mathfrak{l}(\Delta) = \{ \mathfrak{D} :$  $\xrightarrow{*}$  $/ \ \mathfrak{D}/u(x_i) \in \mathfrak{l}(\delta_i).\mathfrak{l}(\delta'_i), \quad 1 \le i \le n \}$

(après avoir isolé un contexte commun à t et t' qui ne sera pas transformé lors de la séquence de dérivations, on dérive t en t' par dérivation première des t_i en a_i puis par dérivation des a_i en t'_i).

A-1 Reconnaissabilité de l'ensemble des arbres de dérivation pour le problème d'accessibilité du premier ordre

Le but de cette section est de prouver le théorème suivant:

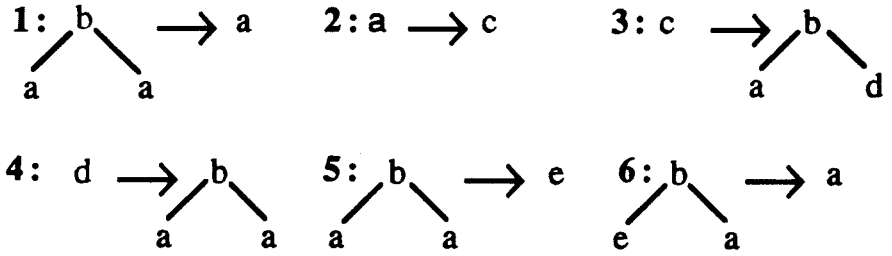
Théorème. *Soient t, t' deux termes dans T_Σ et S un système de réécriture clos standard, l'ensemble de tous les arbres de dérivation pour (t,t',S) est un langage d'arbres reconnaissable.*

Preuve.

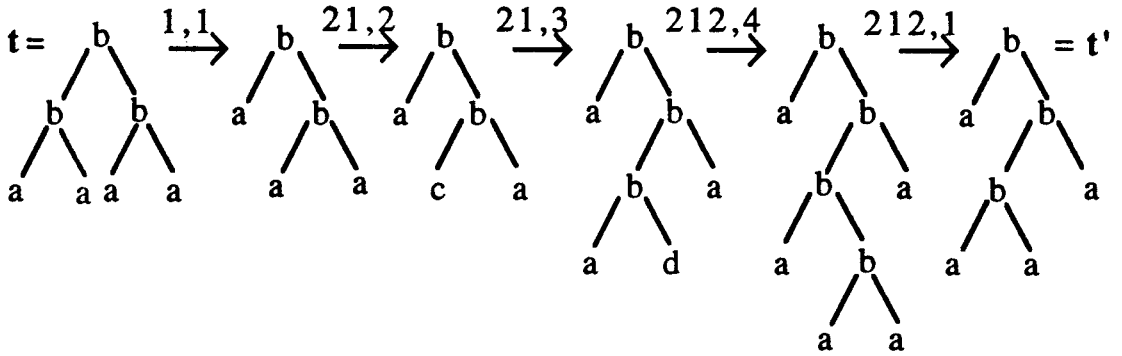
Nous définissons dans les sections A.1.1 à A.1.4 une grammaire régulière afin d' engendrer ce langage d'arbres.

Nous allons prouver que cette construction est cohérente dans la mesure où tous les arbres obtenus par cette grammaire correspondront à des dérivations de t en t' et qu'elle est complète puisqu'à toute dérivation de t en t' correspondra un arbre de dérivation.

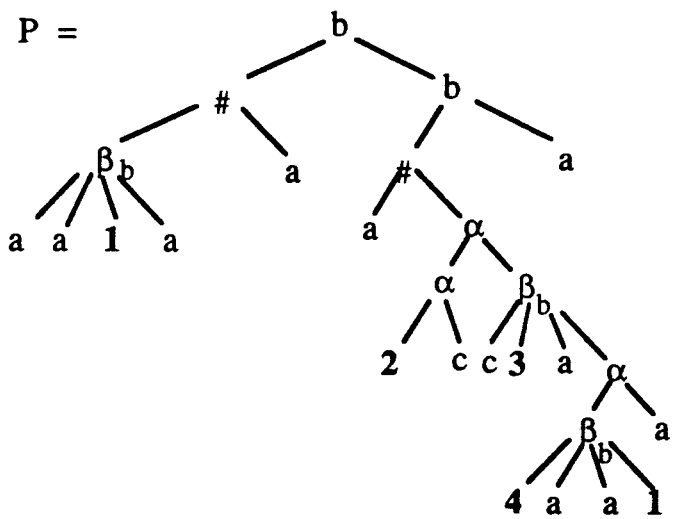
Exemple:



Soit S={1,2,3,4,5,6}. Considérons la dérivation suivante:



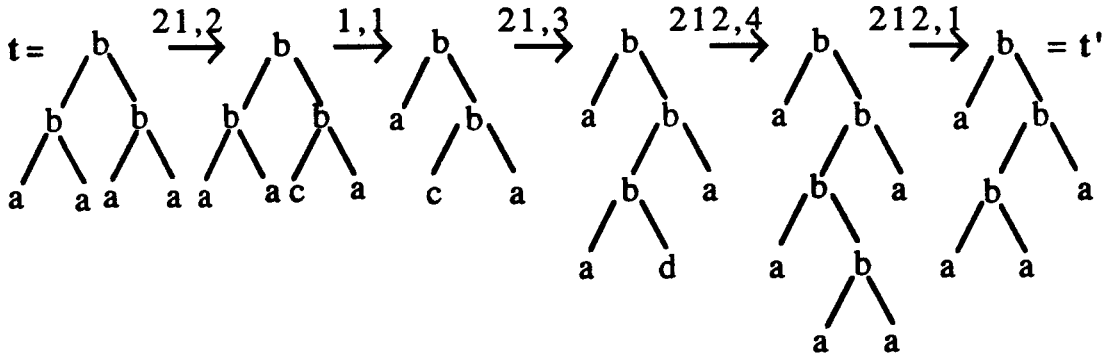
Un arbre de dérivation est P =

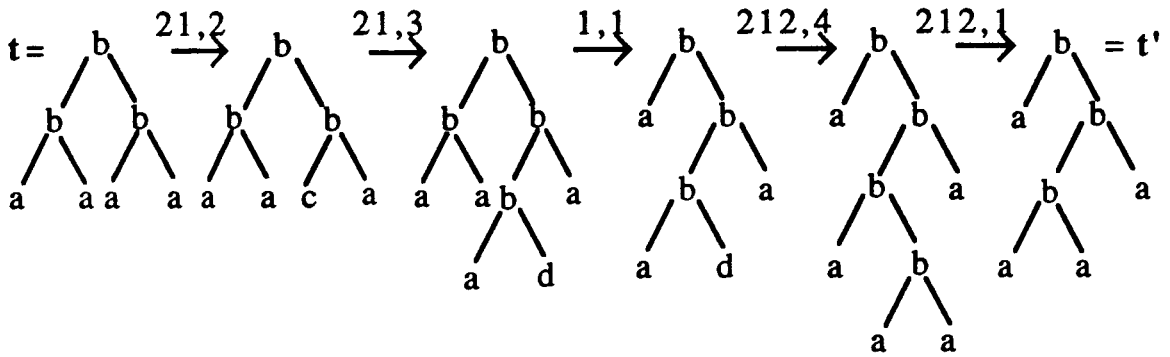


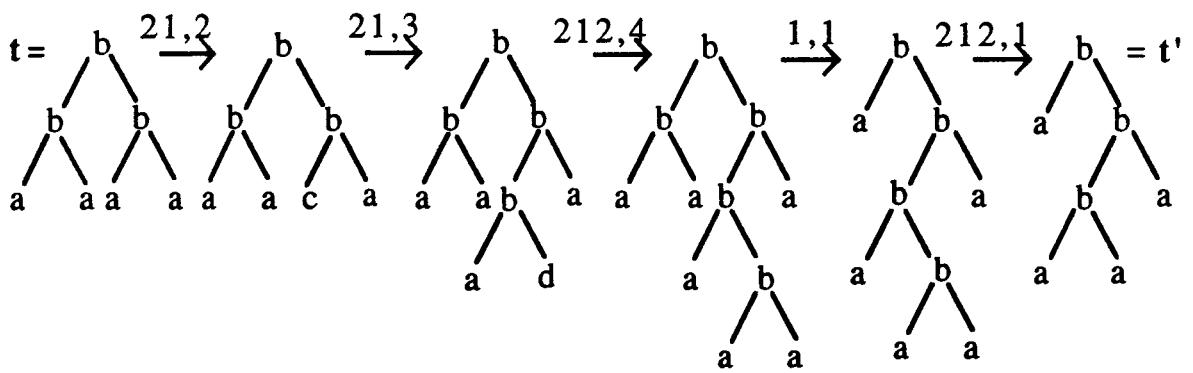
La projection de la frontière de P sur I* est 12341.

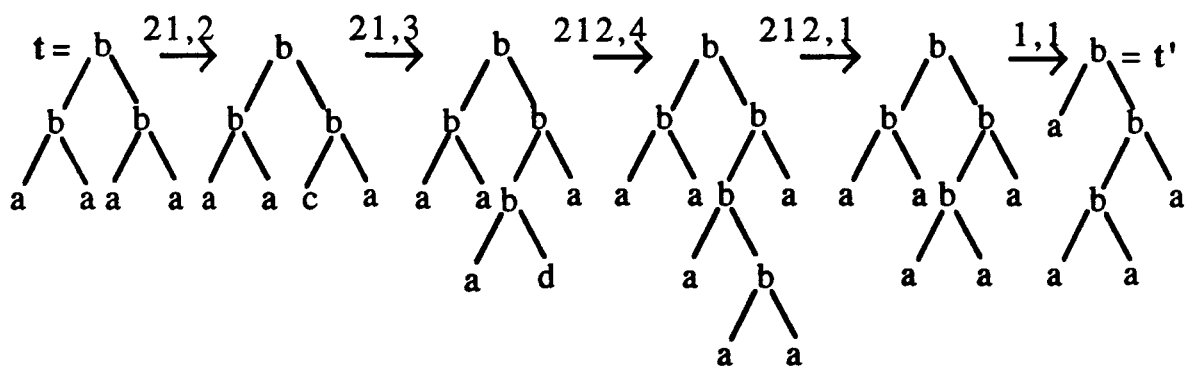
Remarques: α est utilisé pour des réductions conservant la hauteur de l'arbre; les β sont utilisés pour des réductions à hauteur croissante ou décroissante.

Cet arbre code aussi les dérivations suivantes:









A.1.1. Etude des arbres de dérivation pour (a,b,S) où a, b sont des symboles de Σ_0.

On définit la grammaire régulière $\mathcal{H}_{a,b}=(H_{a,b},V_1,\Delta,R_1)$ où l'on a:

$V_1= \{H_{a',b'}, H^{<}_{a',b'} / (a',b') \in \Sigma_0 * \Sigma_0\}$

R_1 est l'ensemble des règles définies comme suit:

$$H_{a,a} \rightarrow a + \sum_{b \in \Sigma_0} \alpha \left(H^{<}_{a,b}, H_{b,a} \right)$$

$a \neq b$

$$H_{a,b} \rightarrow \sum_{c \in \Sigma_0} \alpha \left(H^{<}_{a,c}, H_{c,b} \right)$$

$$H^{<}_{a,b} \rightarrow \sum_{i:a \rightarrow b} i + \sum_{j:a \rightarrow c} \beta_c \left(j, H_{a_1,b_1}, \ldots, H_{a_n,b_n}, k \right) \quad (a_1 \cdots a_n) \quad (k:c \rightarrow b)\ (b_1 \cdots b_n)$$

LEMME. *$\mathcal{L}(\mathcal{H}_{a,b})$ = {arbres de dérivation pour (a,b,S) où a et b sont des symboles de Σ_0}.*

Preuve.

Nous appliquons d'abord des ε-règles, puis, si nous appliquons une règle génératrice, sa racine sera effacée durant la dérivation et à ce moment-là nous retrouverons un arbre de hauteur nulle.

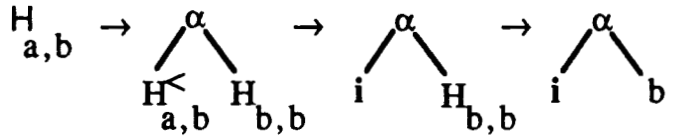
Nous prouvons le lemme par induction sur la longueur de la dérivation et la hauteur de l'arbre.

1) $\supset$: considérons une dérivation de a en b.

* si la dérivation est de longueur 0, la seule règle possible est: $H_{a,a} \rightarrow a$

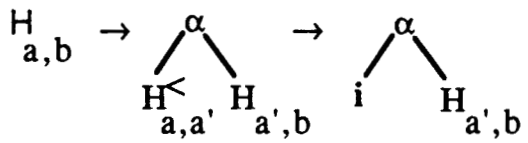
* si la dérivation est de longueur 1, nous avons utilisé forcément une règle stable i: a→b.

La seule séquence possible est, à l'ordre des deux dernières règles près,



*Considérons une dérivation de longueur m (m>1): $a \overset{+}{\rightarrow} b$.
On a donc $a \rightarrow u \overset{+}{\rightarrow} b$

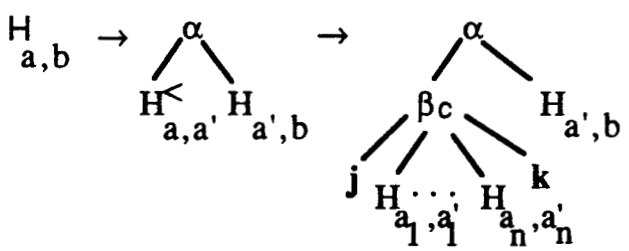
1er cas: u = a'.
Nous appliquons la séquence de règles suivante:



$a' \overset{+}{\rightarrow} b$ est une dérivation de longueur m-1 donc par récurrence, sa preuve est obtenue à partir de $H_{a',b}$.

2ème cas: $u = c(a_1, \ldots, a_n)$
$a \rightarrow u \overset{*}{\rightarrow} v \rightarrow a' \overset{*}{\rightarrow} b$ avec $v = c(a'_1, \ldots, a'_n)$ et c n'est jamais effacé lors de la dérivation $u \overset{*}{\rightarrow} v$.

Nous appliquons la séquence de règles suivante:

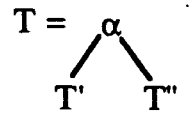


La dérivation $u \overset{*}{\rightarrow} v$ étant de longueur inférieure ou égale à m-2, les dérivations $a_i \overset{*}{\rightarrow} a'_i$ sont de longueur inférieure ou égale à m-2 donc par récurrence, leur preuve est obtenue à partir de H_{a_i,a'_i}. L'arbre de la dérivation de a' en b est obtenu à partir de $H_{a',b}$, la dérivation étant de longueur inférieure ou égale à m-2.

2) $\subset$: montrons que tout arbre obtenu à partir de $H_{a,b}$ correspond à une dérivation $a \xrightarrow{*} b$ par récurrence sur la hauteur de l'arbre.

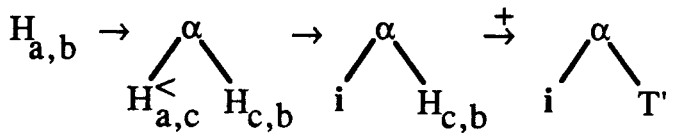
* arbre de hauteur 0:
T = a est obtenu en utilisant la règle $H_{a,a} \rightarrow a$ donc il n'y a pas de dérivation.

* Supposons que tout arbre de hauteur inférieure ou égale à m ($m \geq 0$) obtenu à partir de $H_{a,b}$ correspond à une dérivation $a \xrightarrow{*} b$.
Considérons un arbre de hauteur m+1.

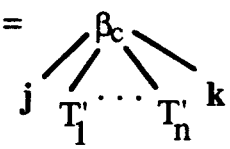


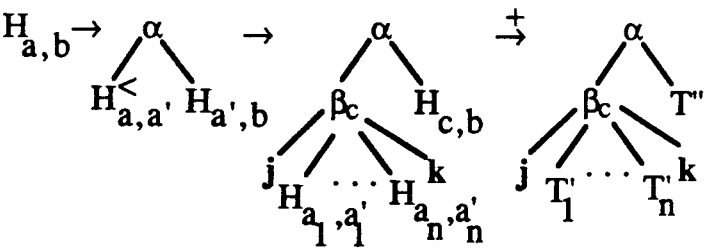
où sup(h(T'),h(T'')) = m

i) T'=i

T est obtenu par la séquence 

La première règle $a \xrightarrow{i} c$ est une ε-règle et, par hypothèse de récurrence, l'arbre T'',de hauteur égale à m, obtenu à partir de $H_{c,b}$ correspond à une dérivation $c \xrightarrow{*} b$. Par conséquent T correspond à la séquence de dérivations: $a \xrightarrow{i} c \xrightarrow{*} b$

ii) T' = 

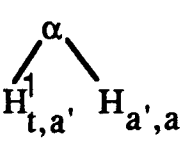
T est obtenu par la séquence 

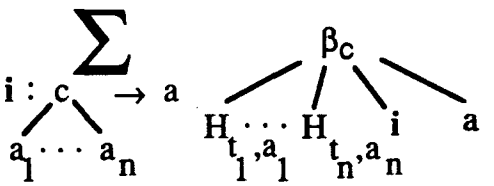
Dans ce cas, m est supérieur ou égal à 1. Par récurrence, les arbres T'_i, de hauteur inférieure ou égale à m-1, obtenus à partir de H_{a_i,a'_i} correspondent à des dérivations $a_i \xrightarrow{*} a'_i$; de même l'arbre T'', de hauteur inférieure ou égale à m, obtenu à partir de $H_{a',b}$ correspond à la dérivation $a' \xrightarrow{*} b$. Par conséquent T correspond à la séquence de dérivations: $a \xrightarrow{i} c(a_1, ..., a_n) \xrightarrow{*} c(a'_1, ..., a'_n) \xrightarrow{k} a' \xrightarrow{*} b$. □

A.1.2. Etude des arbres de dérivation pour (t,a,S) où t est un élément de T_Σ ($t = c(t_1,...,t_n)$ si $h(t) \geq 2$) et a un élément de Σ_0.

Nous définissons la grammaire régulière $\mathcal{H}_{t,a} = (H_{t,a}, V_2, \Delta, R_2)$:
$V_2 = V_1 \cup \{H_{u,a'}, H^1_{u,a'} / a' \in \Sigma_0$, u est un sous-terme clos de t$\}$

$R_2 = R_1 \cup \{$ règles définies comme suit:

$t \notin \Sigma_0$ $\qquad H_{t,a} \longrightarrow \sum_{a' \in \Sigma_0}$ 

$t \notin \Sigma_0$ $\qquad H^1_{t,a} \longrightarrow \sum_{i : c \to a}$ 

LEMME. *$L(\mathcal{H}_{t,a}) = \{$ arbres de dérivation pour (t,a,S) / $t \in T_\Sigma$, $a \in \Sigma_0 \}$.*

Preuve.

Si la racine de t est c , nous devons l'effacer durant la dérivation; aussi nous appliquons une règle effaçante dont le sommet est c. Lorsque nous n'avons plus qu'un élément de Σ_0, nous utilisons les règles de R_1. Nous prouvons le lemme par induction sur la longueur des dérivations et la hauteur de l'arbre.

1) $\supset$: considérons une dérivation de t en a.

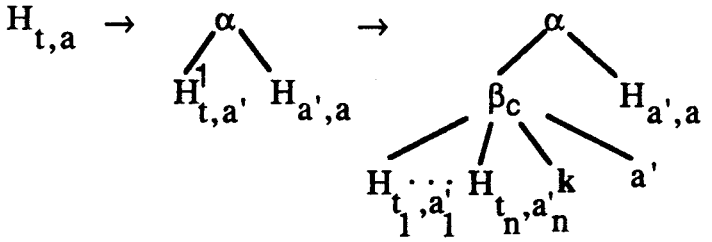
* si la dérivation est de longueur 0, nous appliquons la règle de R_1: $H_{a,a} \to a$

*Considérons une dérivation de longueur m ($m \geq 1$): $t \overset{+}{\to} a$.

i) $t = a'$. On a donc $a' \overset{+}{\to} a$. Nous appliquons les règles de R_1; nous avons donc un arbre de dérivation à partir de $H_{t,a}$.

ii) $t = c(t_1,...,t_n) \overset{*}{\to} c(a_1,...,a_n) \overset{k}{\to} a' \overset{*}{\to} a$ et c n'est pas effacé avant l'application de la règle **k**.

Nous appliquons la séquence de règles de R_2



Les dérivations $t_i \xrightarrow{*} a_i$ étant de longueur inférieure ou égale à m-1, par récurrence les preuves sont obtenues à partir de H_{t_i,a_i} ; de plus $H_{a',a}$ donne une preuve de $a' \xrightarrow{*} a$.

Donc nous avons donc un arbre de dérivation de $t \xrightarrow{*} a$ à partir de $H_{t,a}$.

2) $\subset$: montrons que tout arbre obtenu à partir de $H_{t,a}$ correspond à une dérivation $t \xrightarrow{*} a$ par récurrence sur la hauteur de l'arbre.

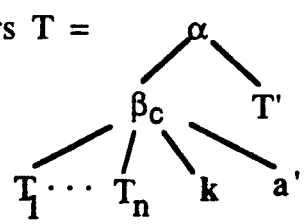
* arbre de hauteur 0:
L'arbre T = a ne peut être obtenu qu'à partir de $H_{a,a}$ donc la preuve est déjà faite.

* Supposons que tout arbre obtenu à partir de $H_{t,a}$ de hauteur inférieure ou égale à m ($m \geq 0$) correspond à une dérivation $t \xrightarrow{*} a$.
Considérons un arbre de hauteur m+1 ($m \geq 0$).

i) t = a' ; nous avons déjà prouvé que l'arbre obtenu correspond à une dérivation $a' \xrightarrow{+} a$.

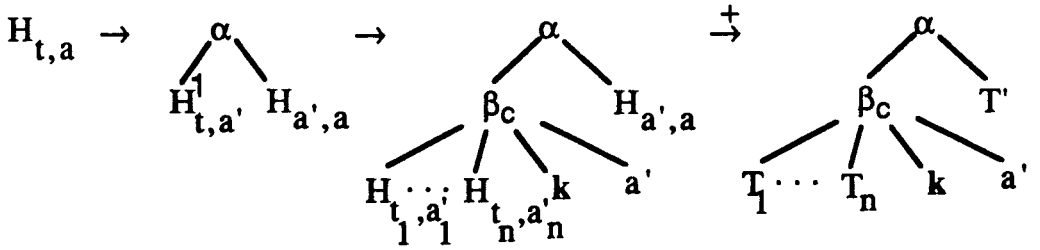
ii) $t = c(t_1,\ldots,t_n)$; par construction de la grammaire, on a forcément $m \geq 1$.

On a alors T =



avec $h(T_i) \leq m-1$, $1 \leq i \leq n$, et $h(T') \leq m$.

Nous avons appliqué la séquence de règles de R_2 suivante:

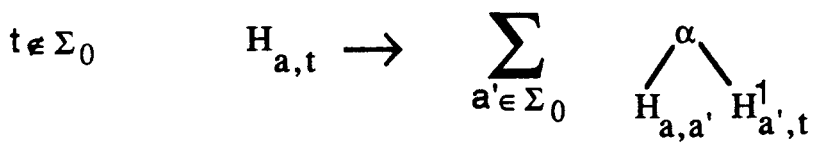


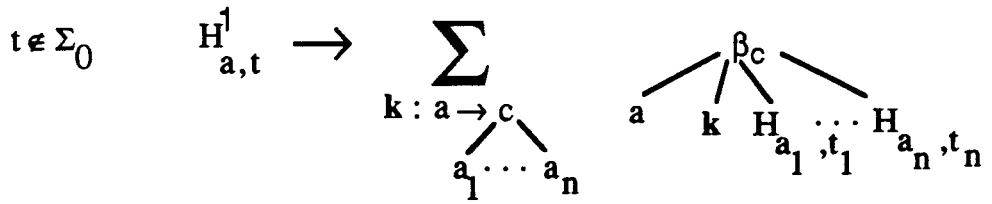
Par hypothèse de récurrence, les T_i, obtenus à partir des $H_{t_i,a_i'}$, correspondent à une dérivation $t_i \xrightarrow{*} a_i'$, $1 \le i \le n$, et T', obtenu à partir de $H_{a',a}$, à une dérivation $a' \xrightarrow{*} a$.

T correspond donc à une dérivation $t \xrightarrow{*} c(a_1', ..., a_n') \xrightarrow{k} a' \xrightarrow{*} a$. ❐

A.1.3. Etude des arbres de dérivation pour (a,t,S) où t est un élément de T_Σ , a est un symbole de Σ_0.

Nous définissons une grammaire régulière $\mathcal{H}_{a,t} = (H_{a,t}, V_3, \Delta, R_3)$:
$V_3 = V_1 \cup \{H_{a',u}, H^1_{a',u} / \ a' \in \Sigma_0$, u est un sous-terme clos de t$\}$
$R_3 = R_1 \cup \{$ règles définies comme suit:





LEMME. *$L(\mathcal{H}_{a,t}) = \{$ arbres de dérivation pour (a,t,S) / $a \in \Sigma_0$, $t \in T_\Sigma\}$.*

Preuve.

Identique à celle du lemme précédent. ❐

A.1.4. Etude des preuves de dérivation pour (t,t',S), t et t' étant des termes de T_Σ.

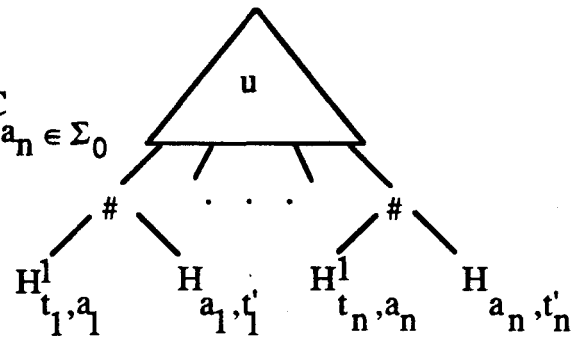
Nous définissons une grammaire régulière $\mathcal{H}_{t,t'} = (H_{t,t'}, V_4, \Delta, R_4)$:

$V_4 = V_2 \cup V_3 \cup \{H^1_{a,a}\}$

$R_4 = R_2 \cup R_3 \cup$ { règles définies comme suit:

(C est l'ensemble des contextes communs à t et t')

$$H_{t,t'} \longrightarrow \sum_{\substack{u \in C \\ a_1,\ldots,a_n \in \Sigma_0}}$$



$H^1_{a,a} \rightarrow a$

LEMME. *$\mathcal{L}(\mathcal{H}_{t,t'}) = \{$ arbres de dérivation pour (t,t',S) / $t, t' \in \Sigma \}$*

Preuve.

Nous prouvons d'abord

$(t \xrightarrow{*}_S t') \Leftrightarrow (\exists u \in T_\Sigma(X_n), \exists t_1,\ldots,t_n,t'_1,\ldots,t'_n \in T_\Sigma, \exists a_1,\ldots,a_n \in \Sigma_0$
$t = u(t_1,\ldots,t_n) \xrightarrow{*}_S u(a_1,\ldots,a_n) \xrightarrow{*}_S u(t'_1,\ldots,t'_n) = t')$

$\Leftarrow$) évident.

$\Rightarrow$)

1) la racine de t est effacée lors de la réécriture; cet effacement se réalise à l'aide d'une règle $c(a_1,\ldots,a_n) \rightarrow a$; il suffit de considérer pour u l'arbre vide. On a alors $t \xrightarrow{*}_S a \xrightarrow{*}_S t'$.

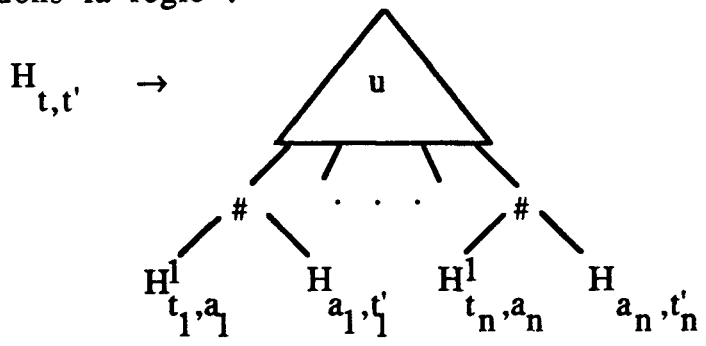
2) la racine de t n'est pas effacée ; ce noeud devient la racine de u et on itére le procédé pour chacun de ses fils.

Prouvons maintenant le lemme.

1) $\supset$: considérons une dérivation de t en t'.

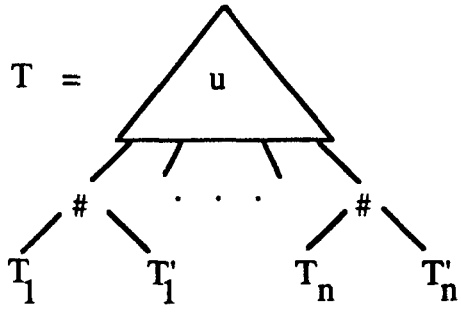
* t = t'. Nous appliquons la règle $H_{t,t} \rightarrow t$, t étant un élément de C.

* $t \overset{+}{\rightarrow}_S t'$, $t' \neq t$. En utilisant la décomposition précédente, nous appliquons la règle :



Pour tout i, $1 \leq i \leq n$, nous avons $t_i \overset{+}{\rightarrow}_S a_i \overset{*}{\rightarrow}_S t'_i$. Grâce aux lemmes précédents, nous savons que l'arbre de dérivation première de $t_i \overset{*}{\rightarrow}_S a_i$ est obtenu à partir de $H^1_{t_i,a_i}$ et que l'arbre de dérivation de $a_i \overset{*}{\rightarrow}_S t'_i$ est obtenu à partir de H_{a_i,t'_i}. Nous avons donc l'arbre de dérivation de $t \overset{+}{\rightarrow}_S t'$ à partir de $H_{t,t'}$.

2) $\subset$: montrons que tout arbre T obtenu à partir de $H_{t,t'}$ correspond à une dérivation $t \overset{*}{\rightarrow} t'$. Nous avons



Par construction de la grammaire, nous savons que pour tout i, $1 \leq i \leq n$, il existe a_i tel que T_i est obtenu à partir de $H^1_{t_i,a_i}$ et T'_i à partir de H_{a_i,t'_i}. D'après les lemmes précédents, les T_i correspondent à des dérivations premières $t_i \overset{*}{\rightarrow}_S a_i$ et les T'_i à des dérivations $a_i \overset{*}{\rightarrow}_S t'_i$. Par conséquent, T correspond à une dérivation $t=u(t_1,...,t_n) \overset{*}{\rightarrow}_S u(a_1,...,a_n) \overset{*}{\rightarrow}_S u(t'_1,...,t'_n)=t'$. □

De plus, la preuve du théorème est terminée car $L(\mathcal{H}_{t,t'})$ est un langage d'arbres reconnaissable. □

A.2 Reconnaissabilité de l'ensemble des arbres de dérivation pour le problème d'accessibilité du second ordre.

Soient F,F' deux langages reconnaissables d'arbres et S un système de réécriture clos standard. Nous considérons le problème d'accessibilité du second ordre: A-t-on $[F]_S \cap F' \neq \emptyset$, c'est à dire existe-t-il un terme t de F qui se réduit en un terme t' de F'?

THEOREME. *Soient F, F' deux langages reconnaissables d'arbres et S un système de réécriture clos standard; l'ensemble des arbres de dérivation pour (F,F',S) est un langage reconnaissable d'arbres.*

Preuve.

Comme pour le théorème précédent, nous définissons une grammaire régulière qui engendre ce langage reconnaissable d'arbres. La construction est similaire, nous utilisons les automates ascendants $M = (\Sigma,Q,Q_f,R)$ et $M' = (\Sigma,Q',Q'_f,R')$ tels que $L(M)=F$ et $L(M')=F'$. L'ensemble des arbres de dérivation pour (F,F',S) est l'ensemble de tous les arbres de dérivation pour (t,t',S) tels que $t \in F$, $t' \in F'$ et $t \xrightarrow{*}_S t'$.

Nous définissons la grammaire régulière suivante: $G = (H_{ist}, V_g, \Sigma_g, R_g)$ où l'on a:

$V_g = V_1 \cup \{H_{ist}\} \cup \{H_{q,q'}, H_{a,q'}, H_{q,a} \ / \ q \in Q, q' \in Q', a \in \Sigma_0\}$

$\Sigma_g = \Delta \cup \{\gamma_c / c \in \Sigma, c \notin \Sigma_0, ar(\gamma_c)=ar(c)+2\}$ (les γ_c ont un rôle identique aux β_c).

$R_g = R_1 \cup \{$ règles définies comme suit:

$$H_{ist} \rightarrow \sum_{(q,q') \in Q_f \times Q'_f} H_{q,q'}$$

$\forall (q,q') \in Q \times Q'$:

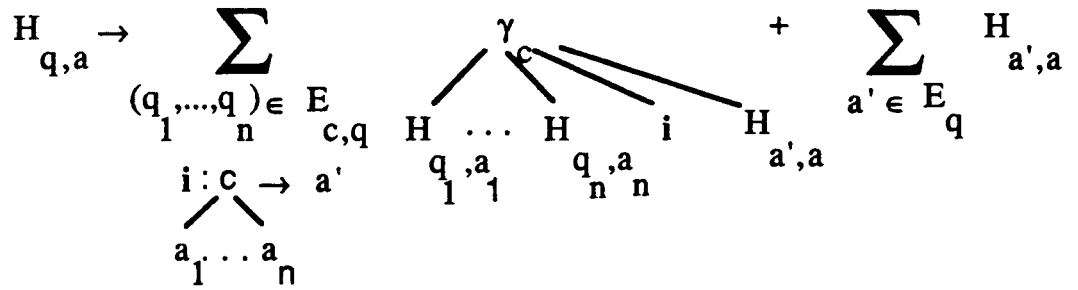
$$H_{q,q'} \rightarrow \sum_{a \in \Sigma} \sum_{\substack{(q_1,\ldots,q_n) \in E_{a,q} \\ (q'_1,\ldots,q'_n) \in E'_{a,q'}}} a\left(H_{q_1,q'_1}, \ldots, H_{q_n,q'_n}\right) + \sum_{a \in \Sigma_0} \#\left(H_{q,a}, H_{a,q'}\right).$$

en posant: $E_{a,q} = \{(q_1,\ldots,q_n) \in Q^n \ / \ a(q_1[x_1],\ldots,q_n[x_n]) \rightarrow q[a(x_1,\ldots,x_n)] \in R\}$

$E'_{a,q'} = \{(q'_1,\ldots,q'_n) \in Q'^n \ / \ a(q'_1[x_1],\ldots,q'_n[x_n]) \rightarrow q'[a(x_1,\ldots,x_n)] \in R'\}$

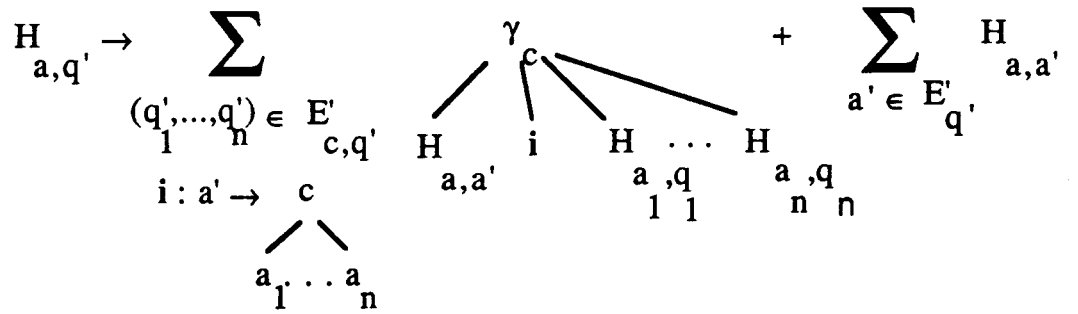
(les règles du premier type engendrent un contexte commun à deux arbres t de F et t' de F'; celles du deuxième type permettent de lancer la construction d'arbres de dérivation d'un arbre t de F en un arbre t' de F')

$\forall(q,a)\in Q\times\Sigma_0$:



en posant: $E_q = \{a\in\Sigma_0 / a \rightarrow q[a]\in R\}$

$\forall(q',a)\in Q'\times\Sigma_0$:

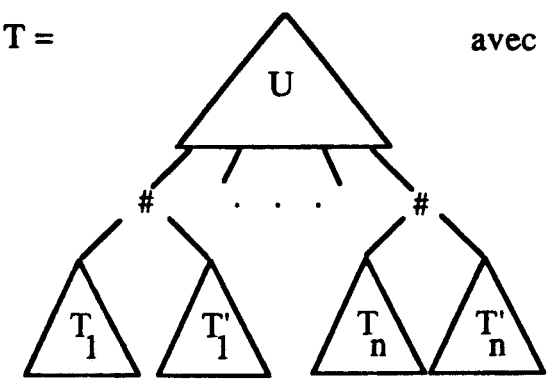


en posant: $E'_{q'} = \{a\in\Sigma_0 / a \rightarrow q'[a]\in R'\}$

LEMME. *$\mathcal{L}(G)$= { arbres de dérivation pour (F,F',S) }*

Preuve.

1) $\subset$: considérons un arbre T appartenant à $\mathcal{L}(G)$.

T = 

avec $U\in T(\Sigma)^1_n$ et $T_i, T'_i \in T(\Sigma_g)$, $1\le i\le n$

En effet, par définition de G, # ne peut plus apparaître en dessous de lui-même.
Il nous faut d'abord retrouver les arbres t et t' tels que T soit un arbre de dérivation de t en t'.
Pour cela, considérons le transducteur descendant déterministe linéaire $V = (\Sigma_g, \Delta_V, Q_V, I_V, R_V)$ où l'on a:

$\Delta_V = \Sigma \cup \{\#\}$

$Q_V = \{q'', q''_1, q''_2\}$

$I_V = \{q''\}$

R_V = { règles définies comme suit:

1) simple lecture du sous-arbre initial commun à t et t' qui n'a pas été transformé.

$$a \in \Sigma : q''[a(x_1,...,x_n)] \rightarrow a(q''[x_1],...,q''[x_n])$$

2) séparation de la recherche de t en fils gauche et de celle de t' en fils droit.

$$q''[\#(x,y)] \rightarrow \#(q''_1[x], q''_2[y])$$

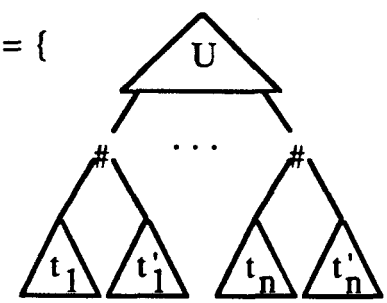
3) reconstruction de l'arbre t.

$$q''_1[\gamma_c(x_1,...,x_n,x_{n+1},x_{n+2})] \rightarrow c(q''_1[x_1],...,q''_1[x_n])$$
$$q''_1[\alpha(x,y)] \rightarrow q''_1[x]$$
$$q''_1[\beta_c(x_1,...,x_{n+2})] \rightarrow q''_1[x_1]$$
$$q''_1[a] \rightarrow a$$
$$q''_1[i] \rightarrow a \quad \text{avec } i: a \rightarrow a' \quad \text{ou} \quad i: a \rightarrow c(a_1,...,a_n)$$

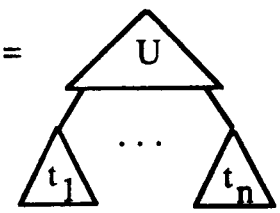
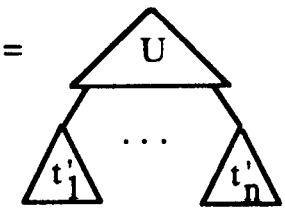
4) reconstruction de l'arbre t'.

$$q''_2[\gamma_c(x_1,...,x_n,x_{n+1},x_{n+2})] \rightarrow c(q''_2[x_3],...,q''_2[x_{n+2}])$$
$$q''_2[\alpha(x,y)] \rightarrow q''_2[y]$$
$$q''_2[\beta_c(x_1,...,x_{n+2})] \rightarrow q''_2[x_{n+2}]$$
$$q''_2[a] \rightarrow a$$
$$q''_2[i] \rightarrow a \quad \text{avec } i: a' \rightarrow a \quad \text{ou} \quad i: c(a_1,...,a_n) \rightarrow a$$

Nous avons alors:

$\Diamond(T) = \{$ 

$/\ t_i, t'_i \in T_\Sigma,\ t_i = \Diamond(T_i),\ t'_i = \Diamond(T'_i),\quad 1 \le i \le n\}$

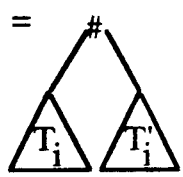
t =  et t' = 

T appartient à $\mathcal{L}(G)$ donc:
$(\exists (q,q') \in Q_f \times Q'_f)\ (H_{ist} \rightarrow H_{q,q'} \overset{*}{\rightarrow} U(H_{q_1,q'_1}, \ldots, H_{q_n,q'_n}) \overset{*}{\rightarrow} T)$.

Par définition de G, on a : $U(q_1[x_1],\ldots,q_n[x_n]) \rightarrow_M q[U(x_1,\ldots,x_n)]$ et $U(q'_1[x_1],\ldots,q'_n[x_n]) \rightarrow_{M'} q'[U(x_1,\ldots,x_n)]$.

Il nous faut donc montrer que pour tout i ($1 \leq i \leq n$):

* il existe une dérivation de t_i en t'_i d'arbre de dérivation:

$U_i =$ 

* $t_i \rightarrow_M q_i$ et $t'_i \rightarrow_{M'} q'_i$

Nous avons $H_{q_i,q'_i} \overset{*}{\rightarrow} U_i = \#(T_i,T'_i)$, c'est à dire par définition de G :

$$(\exists\, a_i \in \Sigma_0)(\ H_{q_i,q'_i} \rightarrow \#(H_{q_i,a_i},H_{a_i,q'_i}) \overset{*}{\rightarrow} U_i)$$

et donc $(\exists\, a_i \in \Sigma_0)(\ H_{q_i,a_i} \overset{*}{\rightarrow} T_i$ et $H_{a_i,q'_i} \overset{*}{\rightarrow} T'_i)$

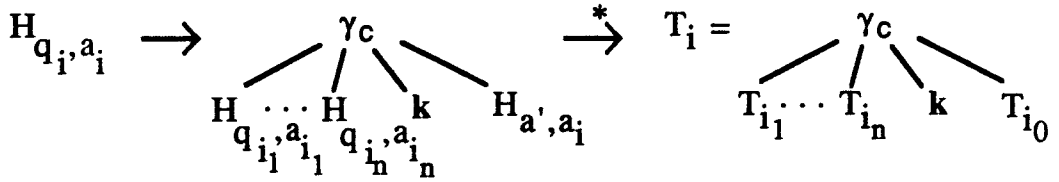
a) Démontrons par récurrence sur la hauteur de T_i qu'il est l'arbre d'une dérivation de t_i en a_i et que $t_i \rightarrow_M q_i$.

*) arbre de hauteur 0:
$T_i = a_i$ donc il n'y a pas eu de dérivation; nous avons appliqué la séquence de règles $H_{q_i,a_i} \rightarrow H_{a_i,a_i} \rightarrow a_i$; de plus $q''_1[T_i] \rightarrow a_i$ donc $t_i = a_i$ et comme $H_{q_i,a_i} \rightarrow H_{a_i,a_i}$, on a $t_i \rightarrow_M q_i$.

*) supposons que tout arbre de hauteur inférieure ou égale à m ($m \geq 0$) vérifie les hypothèses de récurrence. Soit T_i un arbre de hauteur m+1.

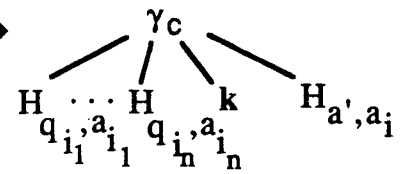
i) Sa racine est un γ_c

Il a été obtenu par application des règles :

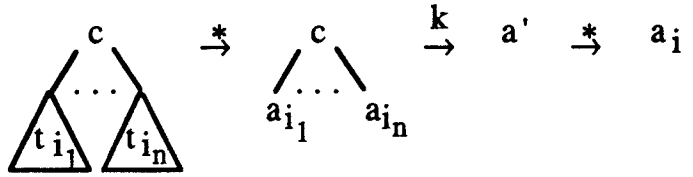


avec k: $c(a_{i_1},\ldots,a_{i_n}) \rightarrow a'$; de plus $q_1''[T_i] \rightarrow c(q_1''[T_{i_1}],\ldots,q_1''[T_{i_n}]) \rightarrow_M t_i$ donc $t_i=c(t_{i_1},\ldots,t_{i_n})$.

Nous avons $H_{q_j,a_j} \overset{*}{\rightarrow} T_{i_j}$ avec $h(T_{i_j}) \leq h(T_i)-1$
Par hypothèse de récurrence, nous savons que $t_{i_j} \overset{*}{\rightarrow} a_{i_j}$ a pour arbre de dérivation T_{i_j} et $t_{i_j} \rightarrow_M q_{i_j}$, $1 \leq j \leq n$.

Par définition de la règle $H_{q_i,a_i} \rightarrow$ 

, nous avons $(q_{i_1},\ldots,q_{i_n}) \in E_{c,q_i}$ donc $c(q_{i_1},\ldots,q_{i_n}) \rightarrow_M q_i$ et $t_i \rightarrow_M q_i$.
Grâce au lemme du A.1.1, nous savons que H_{a',a_i} est un arbre de dérivation de $a' \overset{*}{\rightarrow} a_i$.

Nous avons donc $t_i=$ 

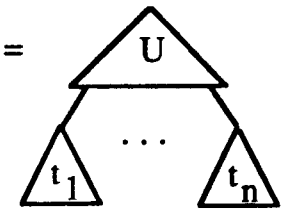
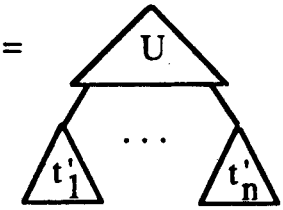
ii) Sa racine n'est pas un γ_c.

Il a été obtenu par application d'une règle : $H_{q_i,a_i} \rightarrow H_{a',a_i}$.
Grâce au lemme du A.1.1, nous savons que H_{a',a_i} est un arbre de dérivation de $a' \overset{*}{\rightarrow} a_i$. De plus $a' \in E_{q_i}$ donc $a' \rightarrow_M q_i$.
$\hat{V}(T_i) = \{t_i = a'\}$ car T_i ne possède que des noeuds du type α ou β_c donc nous avons $t_i \rightarrow_M q_i$ et T_i est un arbre de dérivation de $t_i \overset{*}{\rightarrow} a_i$.

b) Nous montrons de manière identique que T_i' est un arbre de dérivation de a_i en t_i' et que $t_i' \rightarrow_{M'} q_i'$.

Nous avons donc $t_i \rightarrow_M q_i$, $t'_i \rightarrow_{M'} q'_i$ et $U_i = \#(T_i , T'_i)$ est un arbre de dérivation de t_i en t'_i.

2) $\supset$: soit t un élément de F et t' un élément de F' avec $t \overset{*}{\rightarrow} t'$.

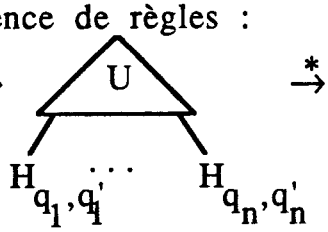
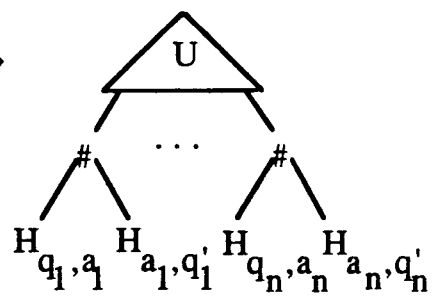
t =  et t' = 

avec $(\forall i \in \{1,\ldots,n\})(\exists a_i \in \Sigma_0)(t_i \overset{*}{\rightarrow} a_i \overset{*}{\rightarrow} t'_i)$.

Comme t est un élément de F , nous avons : $t \overset{*}{\rightarrow}_M U(q_1[t_1],\ldots,q_n[t_n]) \overset{*}{\rightarrow}_M q[t]$ avec $q \in Q_f$.

Comme t' est un élément de F' , nous avons : $t' \overset{*}{\rightarrow}_{M'} U(q'_1[t'_1],\ldots,q'_n[t'_n]) \overset{*}{\rightarrow}_{M'} q'[t']$ avec $q' \in Q'_f$.

Nous appliquons la séquence de règles :

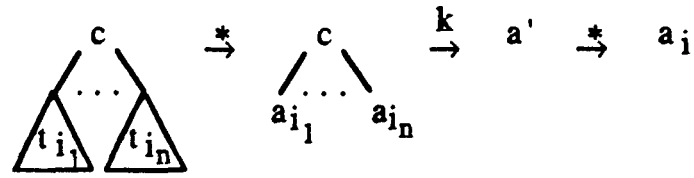
$H_{ist} \rightarrow H_{q,q'} \overset{*}{\rightarrow}$  $\overset{*}{\rightarrow}$ 

a) Démontrons par récurrence sur la longueur de la dérivation qu'un arbre de dérivation de $t_i \overset{*}{\rightarrow} a_i$ est obtenu à partir de H_{q_i,a_i}.

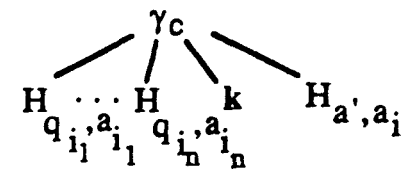
*) longueur 0 : $t_i = a_i$ et $t_i \rightarrow_M q_i$ donc $a_i \in E_{q_i}$; nous appliquons la séquence de règles $H_{q_i,a_i} \rightarrow H_{a_i,a_i} \rightarrow a_i$.

*) supposons que le résultat soit vrai pour une longueur inférieure ou égale à m, $m \geq 0$. Considérons une dérivation $t_i \overset{+}{\rightarrow} a_i$ de longueur m+1.

i) $t_i = a'$. Comme $a' \rightarrow_M q_i$ nous avons $a' \in E_{q_i}$. Nous appliquons la règle $H_{q_i,a_i} \rightarrow H_{a',a_i}$. H_{a',a_i} donne un arbre de dérivation de $t_i \overset{+}{\rightarrow} a_i$ d'après le lemme du A.1.1.

ii) $t_i =$  $\xrightarrow{*}$ $\xrightarrow{k}$ $a' \xrightarrow{*} a_i$

Comme $t_i \to_M c(q_{i_1}[t_{i_1}],\ldots,q_{i_n}[t_{i_n}]) \to q_i[c(t_{i_1},\ldots,t_{i_n})]$ nous avons $(q_{i_1},\ldots,q_{i_n}) \in E_{c,q_i}$.

Nous appliquons la règle $H_{q_i,a_i} \to$ 

H_{a',a_i} donne un arbre de dérivation de $a' \xrightarrow{*} a_i$ d'après le lemme du A.1.1.
Pour tout j, $1 \le j \le n$, par récurrence, $H_{q_{ij},a_{ij}}$ donne un arbre de dérivation de $t_{ij} \xrightarrow{*} a_{ij}$ car la longueur de la dérivation est inférieure ou égale à m d'où H_{q_i,a_i} donne un arbre de dérivation de $t_i \xrightarrow{+} a_i$. □

De plus, la preuve du théorème est terminée car $\mathcal{L}(G)$ est un langage d'arbres reconnaissable. □

A.3 Obtention d'un arbre de dérivation en temps linéaire pour le problème d'accessibilité du premier ordre

La méthode utilisée dans la Section A.1 n'est intéressante que d'un point de vue théorique, notre but maintenant est de trouver un algorithme qui fournit un arbre de dérivation pour (t,t',S) en temps linéaire afin de réaliser une implémentation de cet algorithme dans notre logiciel Valeriann.

PROPOSITION. *Soit S un système de réécriture standard, pour tous t et t', nous pouvons décider, en temps linéaire, si t se réduit en t' par S et construire un arbre de dérivation pour (t,t',S).*

Preuve.

Rappelons qu'il est bien connu que pour un automate ascendant non déterministe, il est possible de reconnaître un terme t en temps linéaire. L'idée est d'associer à chaque noeud l'ensemble des états que peut atteindre ce noeud par M (nous réduisons le non-déterminisme le long des branches du terme t).

1) première étape: durant la compilation du système de réécriture clos S en un transducteur d'arbres clos V=(A,B), nous associons à chaque ε-règle un arbre "court" de dérivation. La compilation est en temps polynomial (voir [DEGI89]).
On construit d'abord pour chaque règle **j** un gtt (G_j,D_j) et les arbres associés:

a) règle j: $c(a_1,\ldots,a_n) \rightarrow a$

$G_j = (\Sigma,Q_j,\varnothing,R_j)$ avec:

$Q_j = \{q_{j,i} \;/\; 1\leq i\leq ar(c)\}\cup\{q_{j,0}\}$

R_j = {règles définies comme suit:

$\mathbf{j_i} : a_i \rightarrow q_{j,i}$ avec $pr(j_i) = \varnothing$ $\quad 1\leq i\leq n$

$\mathbf{j_{n+1}} : c(q_{j,1},\ldots,q_{j,n}) \rightarrow q_{j,0}$ avec $pr(j_{n+1}) = j$ }

$D_j = (\Sigma,\{q_{j,0}\},\varnothing,R'_j)$ avec:

R'_j = {règle : $\mathbf{j_{n+2}} : a \rightarrow q_{j,0}$ avec $pr(j_{n+2}) = \varnothing$}

b) règle j: $a \rightarrow c(a_1,\ldots,a_n)$

$G_j = (\Sigma,\{q_{j,0}\},\varnothing,R_j)$ avec:

R_j = {règle : $\mathbf{j_1} : a \rightarrow q_{j,0}$ avec $pr(j_1) = \varnothing$}

$D_j = (\Sigma,Q'_j,\varnothing,R'_j)$ avec:

$Q'_j = \{q'_{j,i} \;/\; 2\leq i\leq ar(c)+1\}\cup\{q_{j,0}\}$,

R'_j = {règles définies comme suit:

$\mathbf{j_i} : a_{i-1} \rightarrow q'_{j,i}$ avec $pr(j_i) = \varnothing$ $\quad 2\leq i\leq n+1$

$\mathbf{j_{n+2}} : c(q'_{j,2},\ldots,q'_{j,n+1}) \rightarrow q_{j,0}$ avec $pr(j_{n+2}) = j$ }

c) règle j: $a \rightarrow a'$

$G_j = (\Sigma,\{q_{j,0}\},\varnothing,R_j)$ avec:

R_j = {règle: $\mathbf{j_1} : a \rightarrow q_{j,0}$ avec $pr(j_1) = \varnothing$}

$D_j = (\Sigma,\{q_{j,0}\},\varnothing,R'_j)$ avec:

R'_j = {règle: $\mathbf{j_2} : a' \rightarrow q_{j,0}$ avec $pr(j_2) = \alpha(j,a')$}

Nous considèrons maintenant le gtt (G',D') obtenu par union des gtt (G_j,D_j).

Nous créons ensuite l'ensemble E' des ε-transitions:

nous ajoutons la règle $\mathbf{m} : q_{1,0} \rightarrow q_{n,0}$ dans E' tant que nous pouvons trouver des états $q_{n,0}$, $q_{1,0}$, $q_{n,1}$,..., $q_{n,p}$, $q'_{1,2}$,..., $q'_{1,p+1}$ tels que:

$$\mathbf{l_{p+2}} : c(q'_{1,2},\ldots,q'_{1,p+1}) \rightarrow_{D'} q_{1,0}$$

$$\mathbf{m_i} : q'_{1,i+1} \overset{*}{\rightarrow}_E q_{n,i} \qquad 1 \leq i \leq p$$

$$\mathbf{n_{p+1}} : c(q_{n,1},\ldots,q_{n,p}) \rightarrow_{G'} q_{n,0}$$

Nous définissons en même temps l'arbre de dérivation pr(**m**) comme suit:

*) p = 0.

$$\mathbf{n_1} : a \rightarrow_{G'} q$$

$$\mathbf{l_2} : a \rightarrow_{D'} q'$$

Comme nous ajoutons la règle $\mathbf{m} : q' \rightarrow q$ nous définissons $pr(\mathbf{m}) = pr(\mathbf{l_2})$ (dans le cas où $pr(\mathbf{l_2}) \neq \varnothing$, nous avions la règle l: a' → a; la partie droite de cette règle disparait donc nous devons stocker l'arbre de dérivation de l).

Afin d'éviter les problèmes d'ε-transitions consécutives (ce qui pourrait entraîner des séquences infinies), nous ajoutons la nouvelle règle $\mathbf{m} : q' \rightarrow q$ $(q' \neq q)$ telle que :

$q' \rightarrow q \notin E'$

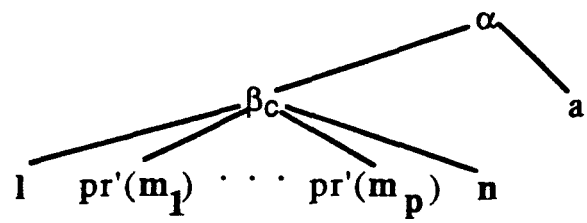
$\exists\, q_1 : q' \xrightarrow{m1} q_1 \xrightarrow{m2} q$ est une séquence de règles de E'.

Nous définissons l'arbre de dérivation

$$pr(\mathbf{m}) = \begin{cases} pr(\mathbf{m1})[a \leftarrow pr(\mathbf{m2})] & \text{si } pr(\mathbf{m1}) \neq \varnothing \text{ et } pr(\mathbf{m2}) \neq \varnothing \\ pr(\mathbf{m2}) & \text{si } pr(\mathbf{m1}) = \varnothing \\ pr(\mathbf{m1}) & \text{si } pr(\mathbf{m2}) = \varnothing \end{cases}$$

*) p≥1. Nous définissons l'arbre de dérivation :

$pr(\mathbf{m}) =$



avec $pr'(\mathbf{m_i}) = pr(\mathbf{m_i})$ si $pr(\mathbf{m_i}) \neq \varnothing$

a_i si $pr(\mathbf{m_i}) = \varnothing$ et $a_i \rightarrow q'_{m,i+1}$

et $a \rightarrow q_{n,0}$

Exemple.

Considérons les règles suivantes:

1. $a \rightarrow f(b)$
2. $b \rightarrow c$
3. $f(c) \rightarrow d$
4. $b \rightarrow f(c)$

a) Construction des gtt (G_i,D_i) , $1 \leq i \leq 4$.

-1.1. $a \rightarrow q_{1,0}$ avec pr(1.1) = $\varnothing$

-1.2. $b \rightarrow q'_{1,2}$ avec pr(1.2) = $\varnothing$

1.3. $f(q'_{1,2}) \rightarrow q_{1,0}$ avec pr(1.3) = 1

-2.1. $b \rightarrow q_{2,0}$ avec pr(2.1) = $\varnothing$

-2.2. $c \rightarrow q_{2,0}$ avec pr(2.2) = $\alpha(2,c)$

-3.1. $c \rightarrow q_{3,1}$ avec pr(3.1) = $\varnothing$

3.2. $f(q_{3,1}) \rightarrow q_{3,0}$ avec pr(3.2) = 3

-3.3. $d \rightarrow q_{3,0}$ avec pr(3.3) = $\varnothing$

-4.1. $b \rightarrow q_{4,0}$ avec pr(4.1) = $\varnothing$

-4.2. $c \rightarrow q'_{4,2}$ avec pr(4.2) = $\varnothing$

4.3. $f(q'_{4,2}) \rightarrow q_{4,0}$ avec pr(4.3) = 4

Construction de E':

5. $q'_{1,2} \rightarrow q_{2,0}$ avec pr(5) = $\varnothing$
6. $q_{2,0} \rightarrow q_{3,1}$ avec pr(6) = $\alpha(2,c)$
7. $q_{1,0} \rightarrow q_{3,0}$ avec pr(7) = $\alpha[\ \beta f(1,\alpha(2,c),3),\ d]$
8. $q'_{1,2} \rightarrow q_{3,1}$ avec pr(8) = $\alpha(2,c)$
9. $q'_{1,2} \rightarrow q_{4,0}$ avec pr(9) = $\varnothing$
10. $q'_{4,2} \rightarrow q_{3,1}$ avec pr(10) = $\varnothing$
11. $q_{4,0} \rightarrow q_{3,0}$ avec pr(11) = $\alpha[\ \beta f(4,c,3),\ d]$
12. $q'_{1,2} \rightarrow q_{3,0}$ avec pr(12) = $\alpha[\ \beta f(4,c,3),\ d]$

Cette construction se termine car les ensembles $R_{G'}$, $R_{D'}$, $Q_{D'} \times Q_{G'}$ sont finis et l'ensemble E' est inclus dans $Q_{D'} \times Q_{G'}$.

LEMME. *Pour toute règle* m *de* $R_G \cup R_D$ *, l'arbre* $pr(m)$ *donne un arbre de dérivation de* l_m *en* r_m.

Preuve.

Considérons une règle m de $R_G \cup R_D$.

1er cas: la règle appartient à $R_{G'} \cup R_{D'}$; par construction $pr(m) = \varnothing$ sauf si la règle initiale j est effectivement appliquée auquel cas $pr(m) = j$ ou $\alpha(j,a)$.

2ème cas: la règle m (ou son inverse) appartient à E';

$m: q' \rightarrow q \in E'$ ou $m: q \rightarrow q'$ tel que $q' \rightarrow q \in E'$

avec $q' \in Q_G \cap Q_D$ et $q \in Q_G \cap Q_D$

Démontrons par récurrence que $pr(m)$ donne un arbre de dérivation.

i) Considérons une séquence de longueur 2.

$m_1: a \rightarrow q' \in R_{D'}$

$m_2: a \rightarrow q \in R_{G'}$

aucune règle initiale n'est appliquée sauf si $a \rightarrow q'$ correspond à une règle j stable auquel cas $pr(m_1) = \alpha(j,a)$. Nous avons alors $pr(m) = \alpha(j,a)$.

ii) supposons que cela soit vrai pour toute séquence de longueur inférieure ou égale à n ($n \geq 2$). Considérons une séquence de longueur n+1.

a) $m_1: q' \rightarrow q_1 \in E'$ et $m_2: q_1 \rightarrow q \in E'$

Comme $pr(m) = pr(m_1)[a \leftarrow pr(m_2)]$ si $pr(m_1) \neq \varnothing$ et $pr(m_2) \neq \varnothing$, l'arbre correspond à la séquence de dérivation $q' \rightarrow q_1$ suivie de $q_1 \rightarrow q$.

Si $pr(m) = pr(m_1)$, alors $pr(m_2) = \varnothing$ et $q_1 \rightarrow q$ ne correspond pas à une dérivation effective donc l'arbre $pr(m_1)$ est aussi l'arbre de dérivation de $q' \rightarrow q$

Si $pr(m) = pr(m_2)$, alors $pr(m_1) = \varnothing$ et $q' \rightarrow q_1$ ne correspond pas à une dérivation effective donc l'arbre $pr(m_2)$ est aussi l'arbre de dérivation de $q' \rightarrow q$

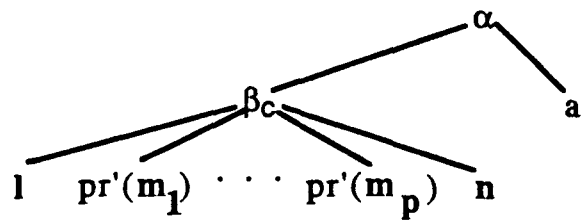
b) $l_{p+2} : c(q'_{1,2},\ldots,q'_{1,p+1}) \rightarrow_{D'} q_{1,0}$

$m_i : q'_{1,i+1} \overset{*}{\rightarrow}_E q_{n,i}$ $\qquad 1 \leq i \leq p$

$$\mathbf{n_{p+1}} : c(q_{n,1},\ldots,q_{n,p}) \rightarrow_{G'} q_{n,0}$$

$q' \rightarrow q \in R_{G'}$ simule la séquence $q' \rightarrow f(q'_1,\ldots,q'_p) \overset{*}{\rightarrow} f(q_1,\ldots,q_p) \rightarrow q$ (et inversement pour $q \rightarrow q' \in R_{D'}$).

Nous avons alors pr(m) =



pr(**m**) est un arbre de dérivation car:
la règle **l** est appliquée,
puis les règles $\mathbf{m_i}$ sont appliquées (les pr'($\mathbf{m_i}$) en sont des arbres de dérivation)
enfin la règle **n** est appliquée.

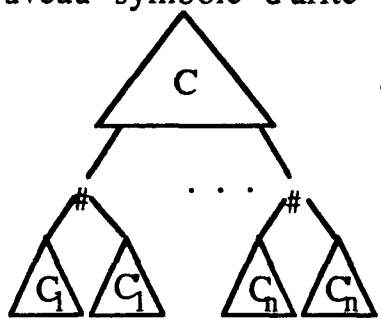
Nous construisons aussi les ensembles G_q et D_q définis comme suit:
$G_q = \{q\} \cup \{q' \in Q / q \rightarrow q' \in E'\}$ et $D_q = \{q\} \cup \{q' \in Q / q' \rightarrow q \in E'\}$

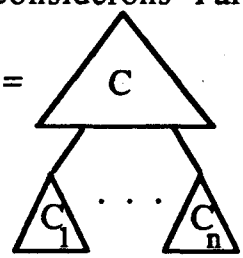
suite de l'exemple.

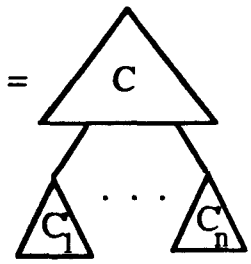
$G_{q1,0} = \{q_{1,0}, q_{3,0}\}$
$G_{q2,0} = \{q_{2,0}, q_{3,1}\}$
$G_{q3,1} = \{q_{3,1}\}$ $G_{q3,0} = \{q_{3,0}\}$
$G_{q4,0} = \{q_{3,0}, q_{4,0}\}$

$D_{q'1,2} = \{q'_{1,2}\}$ $D_{q1,0} = \{q_{1,0}\}$
$D_{q2,0} = \{q_{1,0}, q'_{1,2}\}$
$D_{q3,0} = \{q_{1,0}, q_{3,0}, q_{4,0}, q'_{1,2}\}$
$D_{q'4,2} = \{q'_{4,2}\}$ $D_{q4,0} = \{q_{4,0}, q'_{1,2}\}$

2) deuxième étape : soit C le plus grand contexte commun à t et t'. # est un nouveau symbole d'arité 2; nous considérons l'arbre

T = 

avec t = 

et t' = 

Nous allons définir un transducteur d'arbres ascendant déterministe U_1 afin d'associer à chaque noeud de T l'ensemble des états que peut atteindre ce noeud par A et B. Nous obtenons $U_1(T)$.

$U_1 = (\Sigma', \Delta_1, Q_1, J_1, R_1)$ avec:
$\Sigma' = \Sigma \cup \{\#\}$
$Q_1 = P(Q) \times P(Q)$
$\Delta_1 = \Sigma' \times Q_1$
$J_1 = Q_1$
R_1 = {règles définies comme suit:

1) $\#(\$^{(1)}[x_1], \$^{(2)}[x_2]) \rightarrow \$[(\#,\$)(x_1,x_2)]$ avec $\$_0 = (\$^{(1)})_0$, $\$_1 = (\$^{(2)})_1$

2) $c \in \Sigma$
$c(\$^{(1)}[x_1],\ldots,\$^{(n)}[x_n]) \rightarrow \$[(c,\$)(x_1,\ldots,x_n)]$ avec

* $\$_0 = \{q \in Q \,/\, \exists r \in R_G : c(\alpha_1[x_1],\ldots,\alpha_n[x_n]) \rightarrow q[c(x_1,\ldots,x_n)]$

$$\text{et} \quad \forall i \in \{1,\ldots,n\}\ \alpha_i \in \bigcup_{\beta \in (\$^{(i)})_0} G\beta \quad \}$$

* $\$_1 = \cup D_\alpha$ où $\alpha \in \{q \in Q \,/\, \exists r \in R_D : c(\alpha_1[x_1],\ldots,\alpha_n[x_n]) \rightarrow q[c(x_1,\ldots,x_n)]$

$$\text{et} \quad \forall i \in \{1,\ldots,n\}\ \alpha_i \in \bigcup_{\beta \in (\$^{(i)})_1} D\beta \quad \}$$

$\$_0$ ($\$_1$) permet de mémoriser les états atteints en lisant c par application d'une règle de G' (D') après application d'ε-règles éventuelles au niveau de ses fils, ce que nous avons avec $\cup G\beta$ ($\cup D\beta$).

Nous définissons aussi un automate d'arbres ascendant U_2 qui va marquer chaque noeud avec un symbole i ,$0 \leq i \leq 3$, défini comme suit:

0 signifie que nous n'avons pas rencontré de #; lorsque le symbole # a été rencontré,

1 signifie qu'il n'y a pas d'interface commun à A et B ni en dessous ni au niveau du noeud, c'est à dire que le sous-arbre ne code pas une transformation,

2 que le sous-arbre code une transformation la plus courte possible, c'est à dire qu'il n'y a pas de dérivation possible en dessous, et

3 que le sous-arbre code une transformation mais que ce n'est pas la meilleure possible, c'est à dire qu'il y a une dérivation possible en dessous.

$U_2 = (\Delta_1,\Delta_2,Q_2,J_2,R_2)$ avec:
$Q_2 = \{0,1,2,3\}$,
$\Delta_2 = \Delta_1 x Q_2$,
$J_2 = Q_2$,
R_2 = {règles définies comme suit:

1) $(\#,\$)(q_1[x_1],q_2[x_2]) \rightarrow q[((\#,\$),q)(x_1,x_2)]$ avec

$$q = 1 \text{ si } \$(0) \cap (\bigcup_{\beta \in \$_1} D\beta) \cap QG \cap QD = \varnothing$$

$$2 \text{ si } \$(0) \cap (\bigcup_{\beta \in \$_1} D\beta) \cap QG \cap QD \neq \varnothing$$

2) $c \in \Sigma$.
$(c,\$)(q_1[x_1],\ldots,q_n[x_n]) \rightarrow q[((c,\$),q)(x_1,\ldots,x_n)]$ avec
$q = 0$ si pour tout i, $1 \leq i \leq n$, $q_i = 0$

1 s'il existe i, $1 \leq i \leq n$, tel que $q_i = 1$ et $\$(0) \cap (\bigcup_{\beta \in \$_1} D\beta) \cap QG \cap QD = \varnothing$

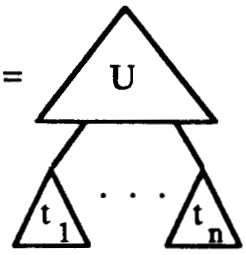
2 s'il existe i, $1 \leq i \leq n$, tel que $q_i = 1$ et $\$(0) \cap (\bigcup_{\beta \in \$_1} D\beta) \cap QG \cap QD \neq \varnothing$

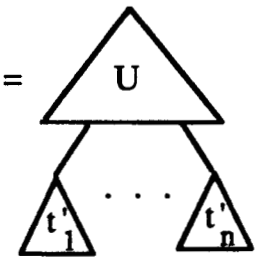
3 si pour tout i, $1 \leq i \leq n$, $q_i = 0$, 2 ou 3 et il existe j, $1 \leq j \leq n$, $q_j \neq 0$

LEMME. *Il existe une dérivation de t en t' si et seulement si* $U_1(T) \overset{*}{\rightarrow} q[U_2(U_1(T))]$ *avec* $q = 0, 2$ *ou* 3.

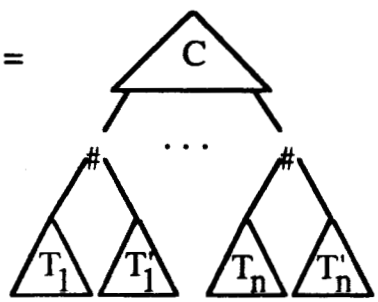
Preuve.
1) $\Rightarrow$: t se dérive en t'.

* t = t' donc T = t et q = 0

* $t \xrightarrow{+} t' \neq t$ avec $t =$ 

, $t' =$ 

avec $t_i \rightarrow_G a_i\ D \leftarrow t'_i$, $a_i \in Q_G \cap Q_D, 1 \leq i \leq n$.

$T =$ 

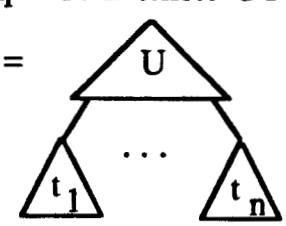
On a alors $U_1(T) \xrightarrow{*} q[U_2(U_1(T))]$ avec q = 1, 2 ou 3.
Comme il y a une dérivation de t en t', on ne peut avoir q = 1.

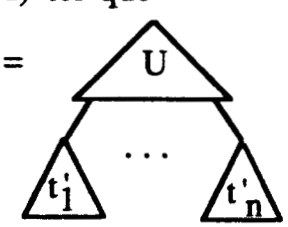
2) $\Leftarrow$: $U_1(T) \xrightarrow{*} q[U_2(U_1(T))]$ avec q = 0, 2 ou 3.

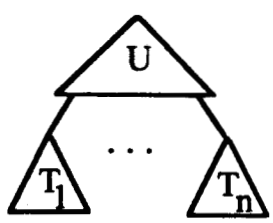
* q = 0: nous n'avons pas rencontré de (#,$) par conséquent T = t = t' et il y a dérivation de t en t'.

* q = 2: il existe un élément α dans $\$_0 \cap (\bigcup_{\beta \in \$_1} D\beta) \cap Q_G \cap Q_D$ donc nous avons la dérivation $t \rightarrow_G \alpha\ D \leftarrow t'$.

* q = 3: il existe $U \in T_\Sigma(X)$ tel que

$t =$ 

, $t' =$ 

, $T =$ 

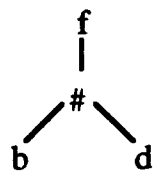
avec $T_i \xrightarrow{*} q_i[U_2(U_1(T_i))]$ et $q_i = 2$, $1 \leq i \leq n$, et tous les noeuds de U sont étiquetés par 3 ou 0; nous avons alors:
$t = U(t_1,\ldots,t_n) \overset{G}{\rightarrow} U(\alpha_1,\ldots,\alpha_n) \overset{D}{\leftarrow} U(t'_1,\ldots,t'_n) = t'$.

Ce marquage pouvait être effectué lors de la transduction U_1 mais pour des raisons évidentes de compréhension, nous avons défini séparement les deux transducteurs.

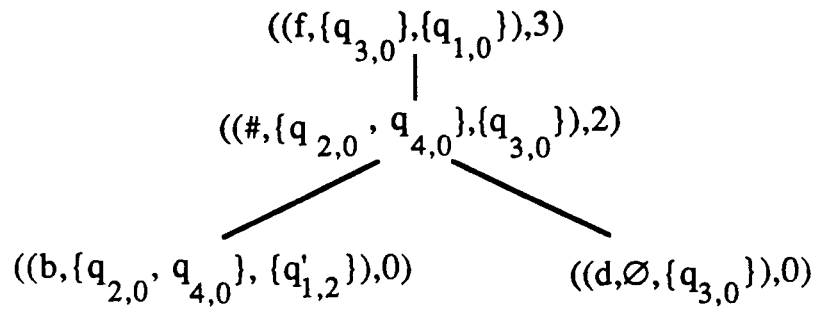
suite de l'exemple.

Prenons t = f(b) et t' = f(d).

Nous avons alors T =

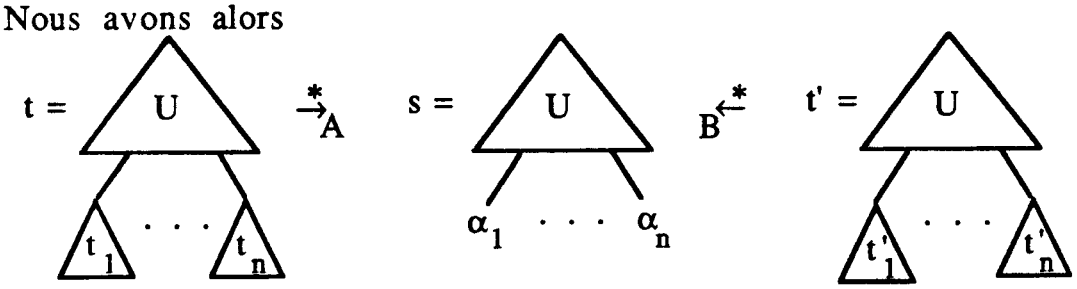


$U_2(U_1(T))$ =



3) troisième étape.

$t \rightarrow\!\!\!\!\!\!*_S\ t'$ si et seulement si, pour toute branche partant du sommet de $U_2(U_1(T))$ juqu'à un symbole #, il existe un noeud tel que les ensembles correspondant d'états pour A et B contiennent un même état interface. Nous avons alors



donc $t \rightarrow\!\!\!\!\!\!*_S\ t'$ et par conséquent nous pouvons décider en temps linéaire si $t \rightarrow\!\!\!\!\!\!*_S\ t'$.

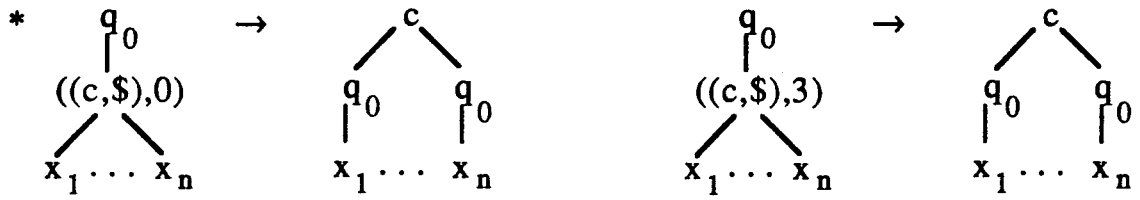
De plus, nous construisons en temps linéaire un arbre de dérivation. Nous utilisons la décomposition précédente et la table qui associe à chaque ε-règle un arbre de dérivation. Cette dernière construction peut être implémentée par un transducteur descendant d'arbres à partir de $U_2(U_1(T))$.

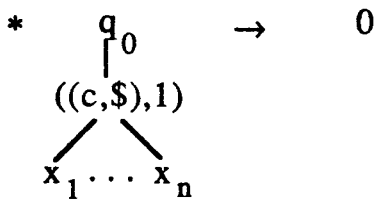
$D_1 = (\Delta_1, \Delta_3, Q_3, I_3, R_3)$ avec:

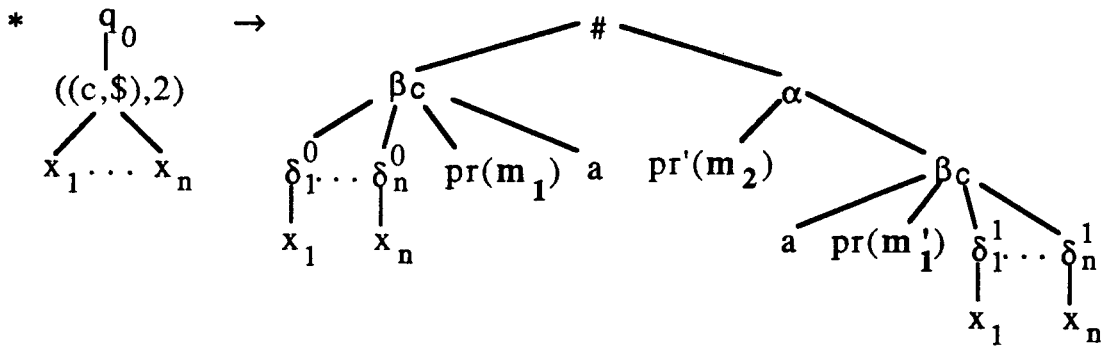
$\Delta_3 = \Delta \cup \{0\}$

$Q_3 = \{q_0\} \cup \{\delta^0, \delta^1, \delta^{00} / \delta \in Q\}$
$I_3 = \{q_0\}$
R_3 est l'ensemble des règles définies comme suit:

1) $c \in \Sigma - \Sigma_0$





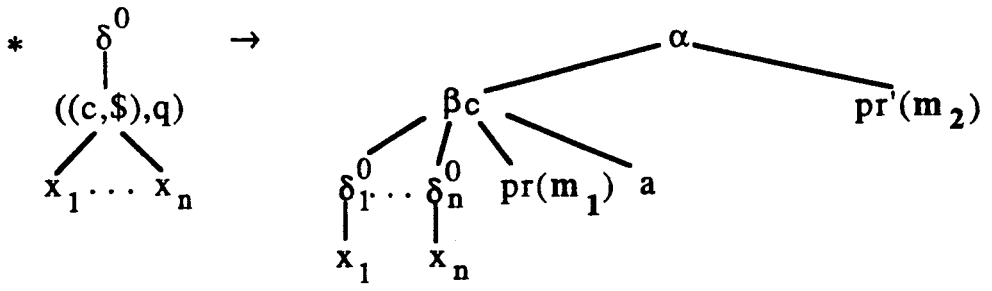


tel qu'il existe $\delta \in \$_0 \cap (\bigcup_{\beta \in \$_1} D\beta) \cap Q_G \cap Q_D$ avec

*) règle $\mathbf{m1}$: $c(\delta^0_1,...,\delta^0_n) \to \delta \in R_{G'}$ $(pr(\mathbf{m1}) = \mathbf{m})$ et $a \to \delta \in R_{D'}$

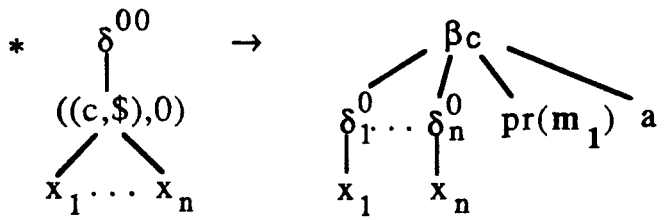
*) règle $\mathbf{m'_1}$: $c(\delta'^1_1,...,\delta'^1_n) \to \delta' \in R_{D'}$ $(pr(\mathbf{m'_1}) = \mathbf{m'})$, $\delta' \in \$_1$, $\delta \in D_{\delta'}$ et $a' \to \delta \in R_{G'}$

*) règle $\mathbf{m2}$: $\delta' \to \delta \in R_D$ et $pr'(\mathbf{m2}) =$ a' si $pr(\mathbf{m2}) = \varnothing$
$pr(\mathbf{m2})$ sinon

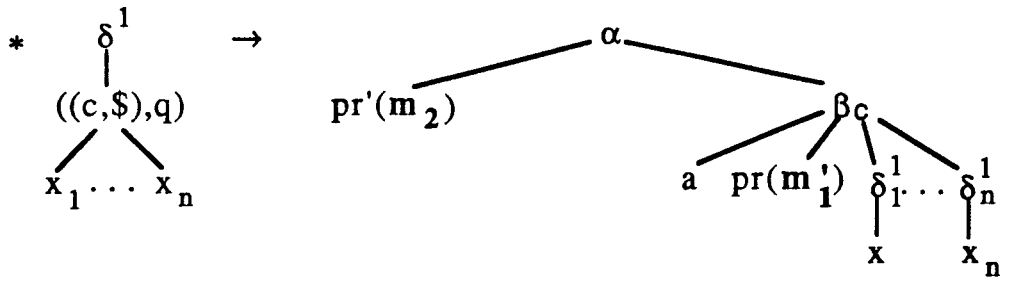


avec règle **m1** : $c(\delta^0_1,...,\delta^0_n) \rightarrow \delta' \in R_{G'}$, $a \rightarrow \delta' \in R_{G'}$, $\delta' \in \$_0$ et $\delta \in G_{\delta'}$

et règle **m2** : $\delta' \rightarrow \delta \in R_G$ et pr'(**m2**) = a si pr(**m2**) = ∅
pr(**m2**) sinon



avec règle **m1** : $c(\delta^0_1,...,\delta^0_n) \rightarrow \delta' \in R_{G'}$, $a \rightarrow \delta' \in R_{G'}$, $\delta' \in \$_0$ et $\delta \in G_{\delta'}$



avec règle **m1** : $c(\delta^1_1,...,\delta^1_n) \rightarrow \delta' \in R_{D'}$, $a \rightarrow \delta' \in R_{G'}$, $\delta' \in \$_1$ et $\delta \in D_{\delta'}$

2) $a \in \Sigma_0$

* $q_0[((a,\$),q)] \rightarrow a$ ($q = 0$ et $t = t'$)

* $\delta[((a,\$),q)] \rightarrow$ pr'(**m**)
avec pr'(**m**) = a si $\delta \in \$_0$
pr(**m**) sinon (règle **m** : $\delta' \rightarrow \delta \in R_G$, $\delta' \in \$_0$ et $\delta \in G_{\delta'}$

$*$ δ^1 (sous $((a,\$),q)$) $\rightarrow$ $pr'(m)$

avec règle $m1 : a \rightarrow \delta' \in R_{D'}$

règle $m2 : \delta' \rightarrow \delta \in R_{D'}$

et $pr'(m) =$ $pr'(m1)[a \leftarrow pr(m2)]$ si $pr(m2) \neq \varnothing$

$pr'(m1)$ si $pr(m2) = \varnothing$

3) a = #

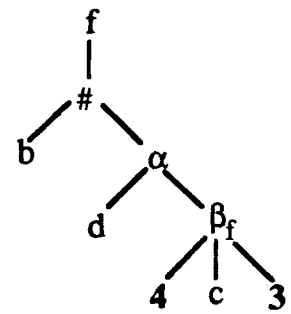
$*$ q_0 (sous $((\#,\$),2)$: x_1, x_2) $\rightarrow$ $\#$: δ_1^{00} (sur x_1), δ_1^{1} (sur x_2)

avec $\delta_1 \in \$_0 \cap \$_1 \cap QG \cap QD$ ou sinon $\delta_1 \in \$_0 \cap (\bigcup_{\beta \in \$_1} D\beta) \cap QG \cap QD$

$* \delta^0[((\#,\$),2)(x_1,x_2)] \rightarrow \delta^0[x_1]$ et $\delta^1[((\#,\$),2)(x_1,x_2)] \rightarrow \delta^1[x_2]$

$* \delta^0[((\#,\$),1)(x_1,x_2)] \rightarrow \delta^0[x_1]$ et $\delta^1[((\#,\$),1)(x_1,x_2)] \rightarrow \delta^1[x_2]$

fin de l'exemple (on prend l'état $q_{4,0}$ commun à $\$_0$ et $Dq_{3,0}$)

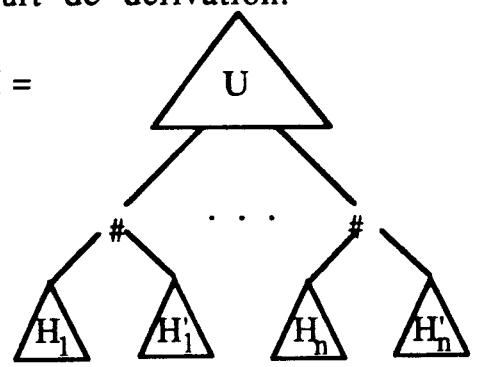
H = 

LEMME. *Si H appartient à $D(U_2(U_1(T)))$ alors H est un arbre de dérivation de t en t' en temps linéaire.*

Preuve.

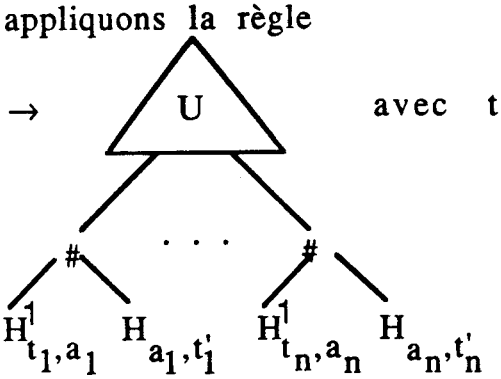
1) Si H = 0 alors nous avons $T \not\rightarrow q[U_2(U_1(T))]$ avec q = 1 donc il n'y a pas de dérivation de t en t'.

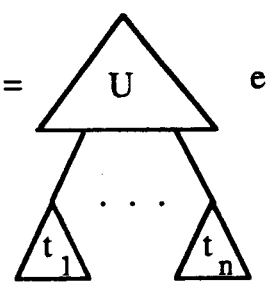
2) $H \in T_\Sigma$. De part les règles de R_3, nous n'avons donc jamais rencontré de q égal à 2 donc q est toujours égal à 0 et t = t'. H est bien un arbre court de dérivation.

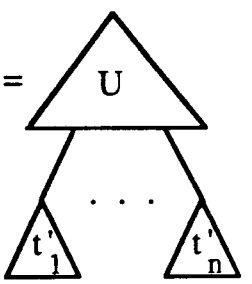
3) H = 

avec $u \in T_\Sigma(X)$.

Nous appliquons la règle

$H_{t,t'} \rightarrow$ 

avec t = 

et t' = 

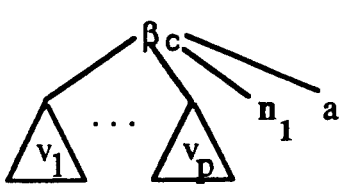
Par conséquent, il nous faut montrer que pour tout i ,$1 \leq i \leq n$, il existe un élément a_i de Σ_0 tel que :

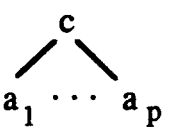
(1) $H^1_{t_i,a_i} \overset{*}{\rightarrow} H_i$ et (2) $H_{a_i,t'_i} \overset{*}{\rightarrow} H'_i$

Nous allons montrer (1) par récurrence sur la hauteur de H_i.

*) H_i est de hauteur 0, donc H_i = a. De part la construction de H_i, nous avions t_i = a. Nous appliquons la règle $H^1_{a,a} \rightarrow a$.

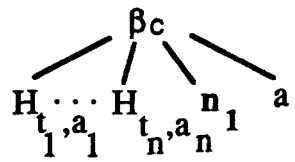
*) Supposons que (1) soit vérifié pour tout arbre de hauteur inférieure ou égale à m ($m \geq 0$). Considérons un arbre H de hauteur m+1.

H = 

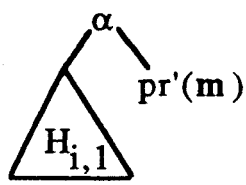
avec n_1 : 

$\rightarrow$ a

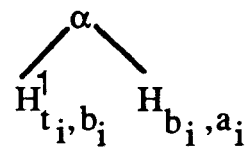
Nous appliquons la règle :

$H^1_{t,a} \rightarrow$ 

Si $v_j = a_j$, nous appliquons la règle $H_{a_j,a_j} \rightarrow a_j$.
Considérons les v_j tels que $h(v_j) \geq 1$. Nous avons deux cas possibles.

1) $v_i =$ 

et la racine de $H_{i,1}$ est du type β

Nous appliquons la règle $H_{t_i,a_i} \rightarrow$ 

Par récurrence, $H^1_{t_i,b_i} \overset{*}{\rightarrow} H_{i,1}$.

2) $v_i = pr'(m)$.
Il reste à prouver que pr'(m) est un arbre de dérivation obtenu à partir de H_{b_i,a_i}.
D'après le lemme de la première étape, pr'(m) est un arbre de dérivation de l_n en r_n.
Nous avons alors :

soit $b_i \rightarrow l_n \overset{*}{\rightarrow} a_i \rightarrow r_n$. La règle $b_i \rightarrow l_n$ est d'arbre de dérivation vide comme $a_i \rightarrow r_n$.

soit $c'(c_1,\ldots,c_n) \rightarrow l_n \rightarrow b_i \overset{*}{\rightarrow} a_i \rightarrow r_n$. La règle $l_n \rightarrow b_i$ est d'arbre de dérivation vide comme $a_i \rightarrow r_n$.

soit $b_i \rightarrow l_n \overset{+}{\rightarrow} c'(c_1,\ldots,c_n) \rightarrow r_n \rightarrow a_i$. La règle $b_i \rightarrow l_n$ est d'arbre de dérivation vide comme $r_n \rightarrow a_i$.

soit $c'(c_1,\ldots,c_n) \rightarrow l_n \rightarrow b_i \overset{+}{\rightarrow} c''(c'_1,\ldots c'_n) \rightarrow r_n \rightarrow a_i$. La règle $l_n \rightarrow b_i$ est d'arbre de dérivation vide comme $r_n \rightarrow a_i$.

Donc, dans tous les cas, pr'(**m**) est un arbre de dérivation obtenu à partir de H_{b_i,a_i}.

De la même façon, nous montrons (2) par récurrence sur la hauteur de H'_i.

Donc H correspond à une séquence de dérivation obtenue à partir de $H_{t,t'}$.
H est donc un arbre de dérivation de t en t'.□

Exemple complet.

Soit le sdr suivant:

1. a(a') → b' **2.** a(b') → a' **3.** b(a',b') → c
4. c → b(d,e) **5.** d → a(e) **6.** e → a(d) **7.** e → a'

a) Construction des gtt (G_i,D_i) , $1 \leq i \leq 7$.

-1.1. $a' \rightarrow q_{1,1}$ pr(**1.1**) = ∅ **1.2.** $a(q_{1,1}) \rightarrow q_{1,0}$ pr(**1.2**) = **1**
-1.3. $b' \rightarrow q_{1,0}$ pr(**1.3**) = ∅

-2.1. $b' \rightarrow q_{2,1}$ pr(**2.1**) = ∅ **2.2.** $a(q_{2,1}) \rightarrow q_{2,0}$ pr(**2.2**) = **2**
-2.3. $a' \rightarrow q_{2,0}$ pr(**2.3**) = ∅

-3.1. $a' \rightarrow q_{3,1}$ pr(**3.1**) = ∅ **3.2.** $b' \rightarrow q_{3,2}$ pr(**3.2**) = ∅
3.3. $b(q_{3,1},q_{3,2}) \rightarrow q_{3,0}$ pr(**3.3**) = **3**
-3.4. $c \rightarrow q_{3,0}$ pr(**3.4**) = ∅

-4.1. $c \rightarrow q_{4,0}$ pr(**4.1**) = ∅
-4.2. $d \rightarrow q'_{4,2}$ pr(**4.2**) = ∅ **4.3.** $e \rightarrow q'_{4,3}$ pr(**4.3**) = ∅
4.4. $b(q'_{4,2},q'_{4,3}) \rightarrow q_{4,0}$ pr(**4.4**) = **4**

-5.1. $d \rightarrow q_{5,0}$ pr(**5.1**) = ∅
-5.2. $e \rightarrow q'_{5,2}$ pr(**5.2**) = ∅ **5.3.** $a(q'_{5,2}) \rightarrow q_{5,0}$ pr(**5.3**) = **5**

-6.1. $e \rightarrow q_{6,0}$ pr(**6.1**) = ∅

-6.2. $d \rightarrow q'_{6,2}$ $pr(6.2) = \varnothing$ 6.3. $a(q'_{6,2}) \rightarrow q_{6,0}$ $pr(6.3) = 6$

-7.1. $e \rightarrow q_{7,0}$ $pr(7.1) = \varnothing$
-7.2. $a' \rightarrow q_{7,0}$ $pr(7.2) = \alpha(7,a')$

b) Construction de E':

8. $q_{2,0} \rightarrow q_{1,1}$ $pr(8) = \varnothing$ 9. $q_{7,0} \rightarrow q_{1,1}$ $pr(9) = \alpha(7,a')$
10. $q_{1,0} \rightarrow q_{2,1}$ $pr(10) = \varnothing$ 11. $q_{2,0} \rightarrow q_{3,1}$ $pr(11) = \varnothing$
12. $q_{7,0} \rightarrow q_{3,1}$ $pr(12) = \alpha(7,a')$ 13. $q_{1,0} \rightarrow q_{3,2}$ $pr(13) = \varnothing$
14. $q_{3,0} \rightarrow q_{4,0}$ $pr(14) = \varnothing$ 15. $q'_{4,2} \rightarrow q_{5,0}$ $pr(15) = \varnothing$
16. $q'_{6,2} \rightarrow q_{5,0}$ $pr(16) = \varnothing$ 17. $q'_{4,3} \rightarrow q_{6,0}$ $pr(17) = \varnothing$
18. $q'_{5,2} \rightarrow q_{6,0}$ $pr(18) = \varnothing$ 19. $q'_{4,3} \rightarrow q_{7,0}$ $pr(19) = \varnothing$
20. $q'_{5,2} \rightarrow q_{7,0}$ $pr(20) = \varnothing$ 21. $q'_{4,3} \rightarrow q_{1,1}$ $pr(21) = \alpha(7,a')$
22. $q'_{4,3} \rightarrow q_{3,1}$ $pr(22) = \alpha(7,a')$ 23. $q'_{5,2} \rightarrow q_{1,1}$ $pr(23) = \alpha(7,a')$
24. $q'_{5,2} \rightarrow q_{3,1}$ $pr(24) = \alpha(7,a')$
25. $q_{5,0} \rightarrow q_{1,0}$ $pr(25) = \alpha(\beta_a[5,\alpha(7,a'),1], b')$
26. $q_{5,0} \rightarrow q_{2,1}$ $pr(26) = \alpha(\beta_a[5,\alpha(7,a'),1], b')$
27. $q_{5,0} \rightarrow q_{3,2}$ $pr(27) = \alpha(\beta_a[5,\alpha(7,a'),1], b')$
28. $q'_{4,2} \rightarrow q_{1,0}$ $pr(28) = \alpha(\beta_a[5,\alpha(7,a'),1], b')$
29. $q'_{6,2} \rightarrow q_{1,0}$ $pr(29) = \alpha(\beta_a[5,\alpha(7,a'),1], b')$
30. $q'_{4,2} \rightarrow q_{2,1}$ $pr(30) = \alpha(\beta_a[5,\alpha(7,a'),1], b')$
31. $q'_{6,2} \rightarrow q_{2,1}$ $pr(31) = \alpha(\beta_a[5,\alpha(7,a'),1], b')$
32. $q'_{4,2} \rightarrow q_{3,2}$ $pr(32) = \alpha(\beta_a[5,\alpha(7,a'),1], b')$
33. $q'_{6,2} \rightarrow q_{3,2}$ $pr(33) = \alpha(\beta_a[5,\alpha(7,a'),1], b')$
34. $q_{6,0} \rightarrow q_{2,0}$ $pr(34) = \alpha(\beta_a[6,\alpha(\beta_a[5,\alpha(7,a'),1], b'),2], a')$
35. $q'_{4,3} \rightarrow q_{2,0}$ $pr(35) = \alpha(\beta_a[6,\alpha(\beta_a[5,\alpha(7,a'),1], b'),2], a')$
36. $q'_{5,2} \rightarrow q_{2,0}$ $pr(36) = \alpha(\beta_a[6,\alpha(\beta_a[5,\alpha(7,a'),1], b'),2], a')$
37. $q_{6,0} \rightarrow q_{1,1}$ $pr(37) = \alpha(\beta_a[6,\alpha(\beta_a[5,\alpha(7,a'),1], b'),2], a')$
38. $q_{6,0} \rightarrow q_{3,1}$ $pr(38) = \alpha(\beta_a[6,\alpha(\beta_a[5,\alpha(7,a'),1], b'),2], a')$

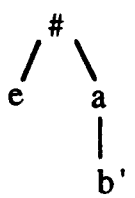
c) Construction des ensembles G_q et $D_{q'}$:

$G_{q_{1,1}} = \{q_{1,1}\}$ $G_{q_{2,1}} = \{q_{2,1}\}$ $G_{q_{3,1}} = \{q_{3,1}\}$ $G_{q_{3,2}} = \{q_{3,2}\}$
$G_{q_{1,0}} = \{q_{1,0}, q_{2,1}, q_{3,2}\}$ $G_{q_{2,0}} = \{q_{2,0}, q_{1,1}, q_{3,1}\}$
$G_{q_{3,0}} = \{q_{3,0}, q_{4,0}\}$ $G_{q_{4,0}} = \{q_{4,0}\}$
$G_{q_{5,0}} = \{q_{5,0}, q_{1,0}, q_{2,1}, q_{3,2}\}$ $G_{q_{6,0}} = \{q_{6,0}, q_{2,0}, q_{1,1}, q_{3,1}\}$
$G_{q_{7,0}} = \{q_{7,0}, q_{1,1}, q_{3,1}\}$

$D_{q'_{4,2}} = \{q'_{4,2}\}$ $D_{q'_{4,3}} = \{q'_{4,3}\}$ $D_{q'_{5,2}} = \{q'_{5,2}\}$ $D_{q'_{6,2}} = \{q'_{6,2}\}$
$D_{q_{1,0}} = \{q_{1,0}, q_{5,0}, q'_{4,2}, q'_{6,2}\}$ $D_{q_{2,0}} = \{q_{2,0}, q_{6,0}, q'_{4,3}, q'_{5,2}\}$
$D_{q_{3,0}} = \{q_{3,0}\}$ $D_{q_{4,0}} = \{q_{3,0}, q_{4,0}\}$
$D_{q_{5,0}} = \{q_{5,0}, q'_{4,2}, q'_{6,2}\}$ $D_{q_{6,0}} = \{q_{6,0}, q'_{4,3}, q'_{5,2}\}$
$D_{q_{7,0}} = \{q_{7,0}, q'_{4,3}, q'_{5,2}\}$

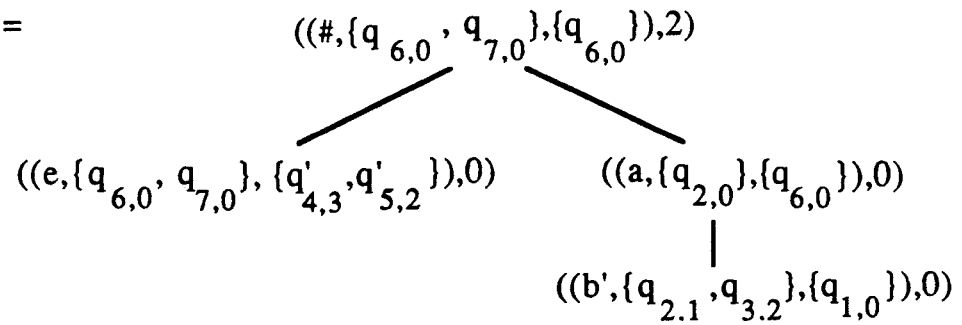
d) Considérons les arbres t = e et t' = a'.

Nous avons alors T =



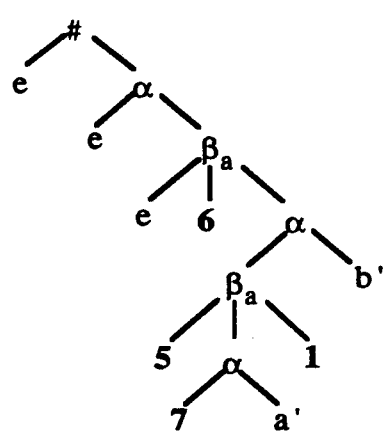
Nous obtenons

$U_2(U_1(T)) =$

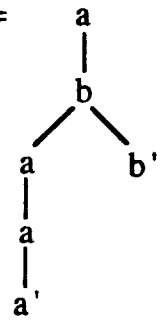


en prenant $q_{6,0}$ comme interface commun, nous obtenons:

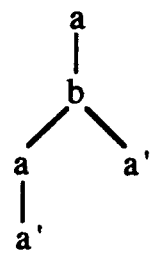
H =

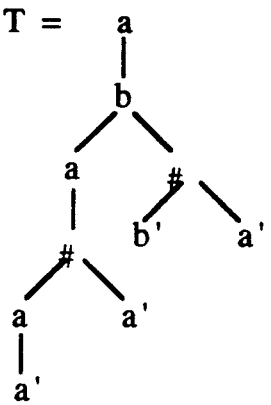


e) Considérons les arbres

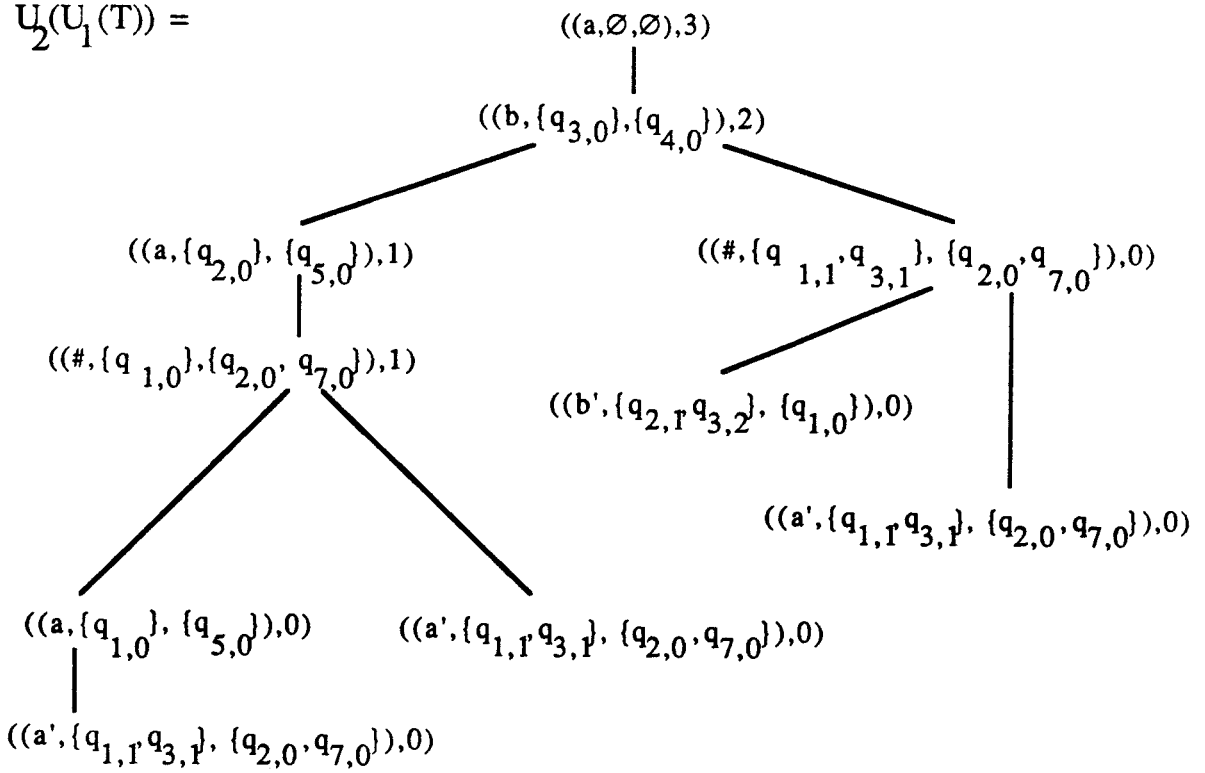
t = 

et

t' = 

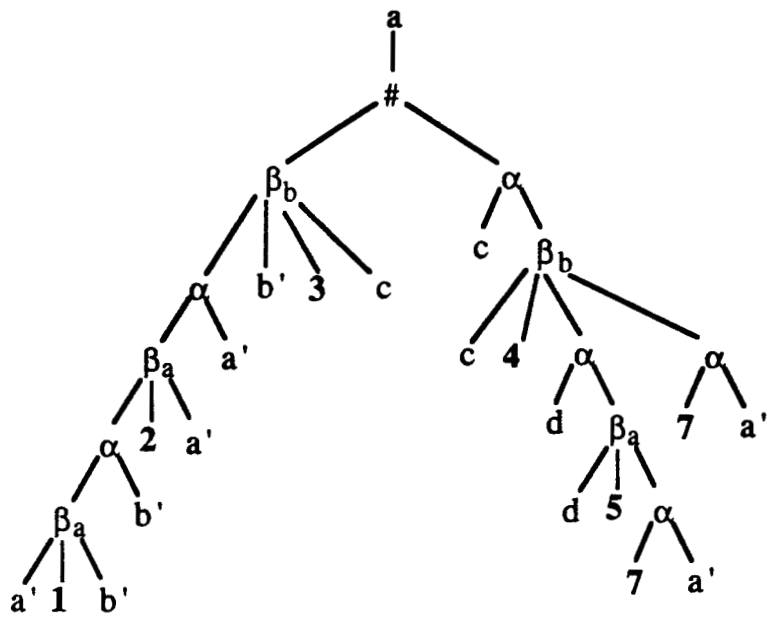
Nous avons alors T = 

Nous obtenons

$U_2(U_1(T))$ = 

en prenant $q_{3,0}$ comme interface commun, nous obtenons:

H =



B. Etude de l'ensemble des arbres de dérivation dans le cas général.

Si nous ne supposons pas S standard, c'est à dire que S est un système de réécriture clos général, en utilisant de nouveaux symboles, nous pouvons décomposer chaque règle de S sous forme de règles standards, puis nous pouvons associer à S un système de réécriture clos standard S'. Nous pouvons définir une grammaire régulière pour S' afin d'obtenir les arbres de dérivation pour S' et ensuite grâce à un transducteur d'arbres, nous pouvons obtenir les arbres de dérivation pour S.

B.1. Reconnaissabilité de l'ensemble des arbres de dérivation pour le problème d'accessibilité du premier ordre.

1) Chaque règle peut être décomposée en une séquence de règles normalisées:

$u \xrightarrow{i} v$ se décompose sous la forme

$$u = u_n \overset{i,1}{\rightarrow} u_{n-1} \overset{i,2}{\rightarrow} \dots \overset{i,n-1}{\rightarrow} u_1 \overset{i,n}{\rightarrow} v_1 \overset{i,n+1}{\rightarrow} v_2 \overset{i,n+2}{\rightarrow} \dots \overset{i,n+p-1}{\rightarrow} v_p = v$$

avec $h(u) = n$

$h(u_j) = n-j \qquad 0 \leq j \leq n-1$

$h(v) = p+1$

$h(v_j) = j \qquad 1 \leq j \leq p$

exemple.

1 : $b(a(a'),a') \rightarrow d(a(a(a')))$ se décompose en:

1.1 : $a(a') \rightarrow \lambda \qquad \lambda \notin \Sigma_0$

1.2 : $b(\lambda,a') \rightarrow \mu \qquad \mu \notin \Sigma_0$

1.3 : $\mu \rightarrow d(\gamma) \qquad \gamma \notin \Sigma_0$

1.4 : $\gamma \rightarrow a(\delta) \qquad \delta \notin \Sigma_0$

1.5 : $\delta \rightarrow a(a')$

2) Au système de réécriture S nous pouvons donc associer un système de réécriture clos standard S'. Nous pouvons alors définir une grammaire régulière pour S' afin d'obtenir l'ensemble des arbres de dérivation pour (t,t',S').

3) Il reste à transducter l'ensemble des arbres de dérivation pour (t,t',S') pour obtenir l'ensemble des arbres pour (t,t',S).
Si en théorie la définiton de ce transducteur ne présente pas de difficulté, dans la pratique l'écriture des règles de transduction apparait

comme étant très astreignante et ne présentant pas d'intérêt majeur. Aussi nous nous contenterons de donner un exemple.

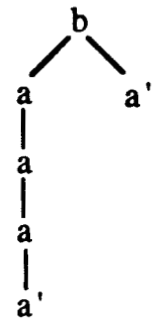
exemple.

Nous reprenons la règle de l'exemple précédent et nous ajoutons la règle 2 : $a(a(a')) \rightarrow a'$ qui se décompose en:

2.1 : $a(a') \rightarrow \phi \qquad \phi \notin \Sigma_0$

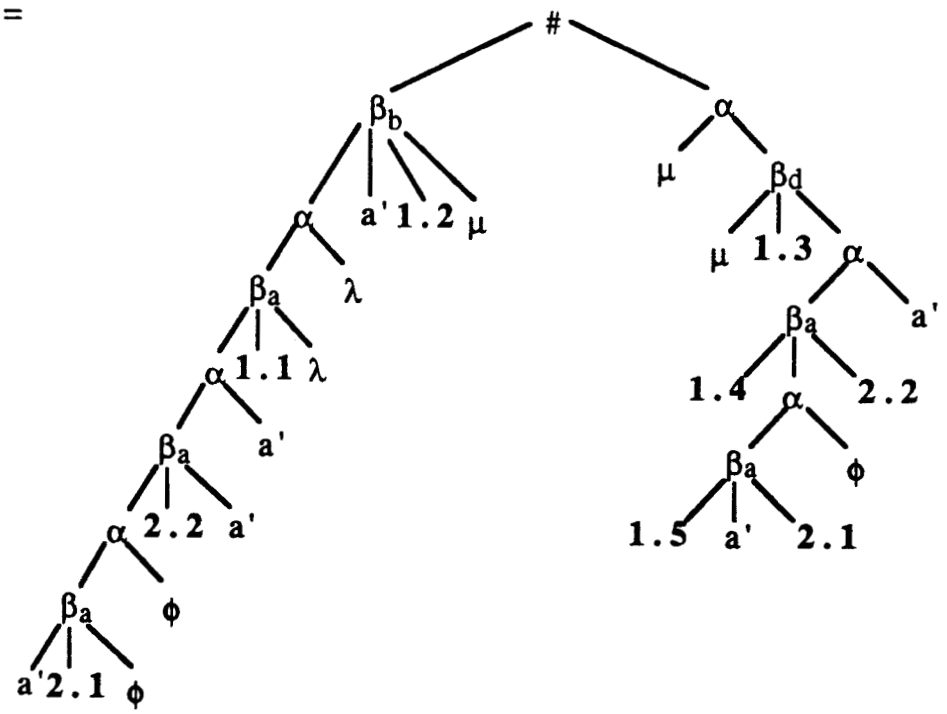
2.2 : $a(\phi) \rightarrow a'$

Si nous prenons

t = 

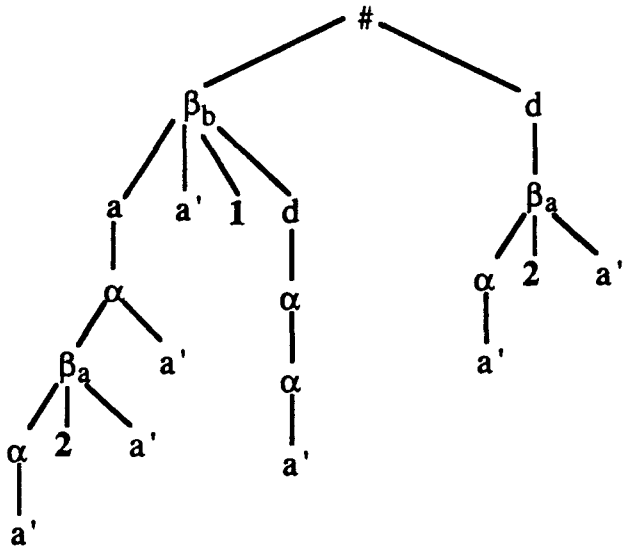
et t' = 

une preuve de dérivation pour (t,t',S') est:

H' = 

Par la transduction , nous obtenons finalement:

H =



B.2. Obtention d'un arbre de dérivation en temps linéaire pour le problème d'accessibilité du premier ordre.

La méthode d'obtention d'un arbre dans le cas standard peut encore s'appliquer au cas général.
La première étape est identique à la seule différence près que, pour certains gtt (G_j,D_j), le nombre de règles créées sera beaucoup plus important; nous utiliserons la même méthode de construction pour les arbres de dérivation associés.
La deuxième étape sera totalement identique.
Seule la dernière étape varie légèrement étant donné qu'il faudra éventuellement descendre de plusieurs niveaux pour obtenir soit la règle appliquée soit les branchements d'ε-règles effectués alors que, dans le cas standard, l'étude se fait à chaque niveau. La génération de l'arbre de dérivation se réalise alors comme dans le cas standard.

CHAPITRE IV

AUTOMATES A PILES ET CALCUL DE FORMES NORMALES

A. Automates à piles d'arbres

Les automates à piles d'arbres sont une généralisation, au cas des arbres, des automates à piles de mots. Une classe d'automates à piles descendants a été définie et étudiée par I. Guessarian dans [GUES83] et ascendants par K.M.Schimpf et J.H. Gallier dans [GASC85] et [SCHI82].

Dans les deux cas, les auteurs ont défini des classes d'automates à piles d'arbres telles que la classe des langages reconnus soit exactement la classe des langages algébriques d'arbres. Nous ne nous intéressons ici qu'aux automates à piles ascendants d'arbres.

Nous comparons dans cette partie les différentes définitions possibles pour les automates ascendants à piles d'arbres.

Une définition d'automate à piles d'arbres doit correspondre, dans le cas des arbres filiformes, à la définition des automates à piles dans les mots. Lorsqu'on veut généraliser la définition utilisée dans les mots au cas des arbres, deux problèmes se posent: le premier concerne la hauteur des termes dépilés, le second concerne l'arité des états.

Une règle de transition, dans le cas des mots, est de la forme: $(q,a,\alpha)\rightarrow(q',\beta)$ où q, q' sont des états, a est une lettre de l'alphabet d'entrée ou le mot vide, α est une lettre de l'alphabet de pile et β un mot sur l'alphabet de pile. Dans le mouvement associé à cette transition, a est lu, le sommet de pile α est dépilé et on empile le mot β. Il est facile de voir que l'on peut généraliser cette définition en prenant pour α un mot de l'alphabet de pile (c'est-à-dire encore que l'on peut lire dans la pile à une profondeur bornée) sans modifier la puissance de l'automate à pile ainsi défini. L'idée de la construction est de dépiler lettre par lettre à l'aide d' ε-règles en mémorisant, dans les états, le mot dépilé.

La situation est toute différente dans le cas des arbres comme l'illustre l'exemple suivant:

Soit l'ε-règle $q(b(b(x,y),z))\rightarrow q'(c(x,y,z)$ associée à un automate à piles d'arbres où q, q' sont des états, b, c des lettres de l'alphabet (gradué) de pile et x, y et z des variables. Si on impose que les états soient d'arité 1, cette règle ne pourra être simulée par des règles d'un automate à piles pour lequel on ne peut dépiler que le sommet de pile (c'est à dire à la profondeur 1). En effet, lorsque l'on dépile le premier b, les états ayant pour arité 1, on perd l'information sur l'un de ses deux sous-arbres. L'idée est donc d'introduire des états avec arité. En effet, reprenons l'exemple précédent et introduisons un nouvel état q" d'arité

2, la règle pourra alors être simulée à l'aide des deux règles suivantes: $q(b(x,y))\rightarrow q''(x,y)$ et $q''(b(x,y),z)\rightarrow q'(c(x,y,z))$.

Nous donnons donc ici les différentes définitions possibles en fonction de la profondeur des termes dépilés et de l'arité des états.

Nous montrons l'équivalence des différentes définitions: automates à piles d'arbres avec arité quelconque pour les états et profondeur quelconque pour les termes dépilés, avec arité 1 et profondeur quelconque, arité quelconque et profondeur 1 pour les termes dépilés.

Une autre option est d'utiliser la définition de R.V. Book et J.H. Gallier, J.H. Gallier et K.M. Schimpf, K.Salomaa (voir [BOGA85], [GASC85], [SALO88]).

Cette définition utilise deux types de règles: des règles de lecture (read-rules) qui permettent de lire une lettre de l'alphabet d'entrée en empilant une lettre de l'alphabet de pile au sommet de l'arbre et des règles de réécriture de la pile (reduce-rules) plutôt que des règles qui lisent et réduisent en même temps.

Nous prouvons que cette définition est équivalente aux précédentes comme l'illustre l'exemple suivant:

la règle $a(q_1(u_1),\ldots,q_n(u_n))\rightarrow q(u)$ avec a d'arité n, pour tout i, u_i terme de $T_\Gamma(X)$ où Γ est l'alphabet de pile sera simulée par la règle de lecture (read-rule) $a(q_1(x_1),\ldots,q_n(x_n))\rightarrow q'(a'(x_1,\ldots,x_n))$ avec q' nouvel état et a' nouvelle lettre de Γ et la règle de réduction (reduce-rule) $q'(a'(u_1,\ldots,u_n))\rightarrow q(u)$.

Nous définissons également la notion de déterminisme associé à un automate à piles. Un automate à piles est déterministe si l'ensemble des règles de réécriture associé est linéaire à gauche et sans paires critiques. En effet, le système de réécriture associé est alors confluent ([HUET80]), ceci n'étant vrai que dans le cas linéaire à gauche. Nous démontrons que les différentes définitions sont encore équivalentes dans le cas déterministe.

A.1 Définitions des automates à piles

Définition 1: Un *automate à piles ascendant d'arbres* (tpda pour tree pushdown automaton) est un quintuplet $T=(\Sigma, \Gamma, Q, Q_f, R)$ où:

Σ est un alphabet gradué fini d'entrée.

Γ est un alphabet gradué fini de pile.

Q est un ensemble gradué fini d'états.

Q_f, inclus dans Q_1 (états d'arité 1), ensemble d'états finaux.

R est un système de réécriture fini sur $T_{\Sigma\cup\Gamma\cup Q}$ dont les règles ont l'une des deux configurations suivantes:

règles standard:

$u= \alpha(q_1(u_{11},...,u_{1n_1}),...,q_m(u_{m1},...,u_{mn_m})) \rightarrow q(u_1,...,u_n)$ avec $\alpha\in\Sigma_m$, $q_i\in Q$, n_i est l'arité de q_i, n est l'arité de q, $u_{ij},u_k\in T_\Gamma(X)$ pour $j=1,...,n_i$, $i=1,...,m$, $k=1,...,n$, les ensembles de variables $V(u_{ij})$ sont disjoints.

ε-règles:

$q(u_1,...,u_n) \rightarrow q'(u'_1,...,u'_{n'})$ avec q, q' dans Q d'arité n et n', $u_i,u'_j\in T_\Gamma(X)$ pour $i=1,...,n$, $j=1,...,n'$.

Soit T un automate à piles d'arbres, on définit:

Une *configuration* est un terme c de $T_{\Sigma\cup\Gamma\cup Q}$ tel que chemin(c) est inclus dans $\Sigma^*Q\Gamma^+\cup\Sigma^*$.

La *relation* $\rightarrow_T$ est la relation de réécriture $\rightarrow_R$ restreinte à l'ensemble des configurations de T. Un mouvement élémentaire de l'automate à pile T est donc défini par $c\rightarrow_T c'$ si c, c' sont des configurations de T et $c\rightarrow_R c'$.

La *relation* $\overset{*}{\rightarrow}_T$ est la clôture réflexive et transitive de $\rightarrow_T$.

Une *configuration initiale* est un terme c=t de T_Σ.

Soit $c=t_0(q_1(u_{11},...,u_{1n_1}),...,q_m(u_{m1},...,u_{mn_m}))$ une configuration de T obtenue à partir de la configuration initiale $t=t_0(t_1,...,t_m)$, alors $t_1,...,t_m$ ont été lus, la $i^{ème}$ tête de lecture est dans l'état q_i d'arité n_i avec pour arbres de pile associés $u_{i1},...,u_{in_i}$.

Une *configuration finale* est un terme c=q(v) où q est un état final et v un arbre de pile dans T_Γ.

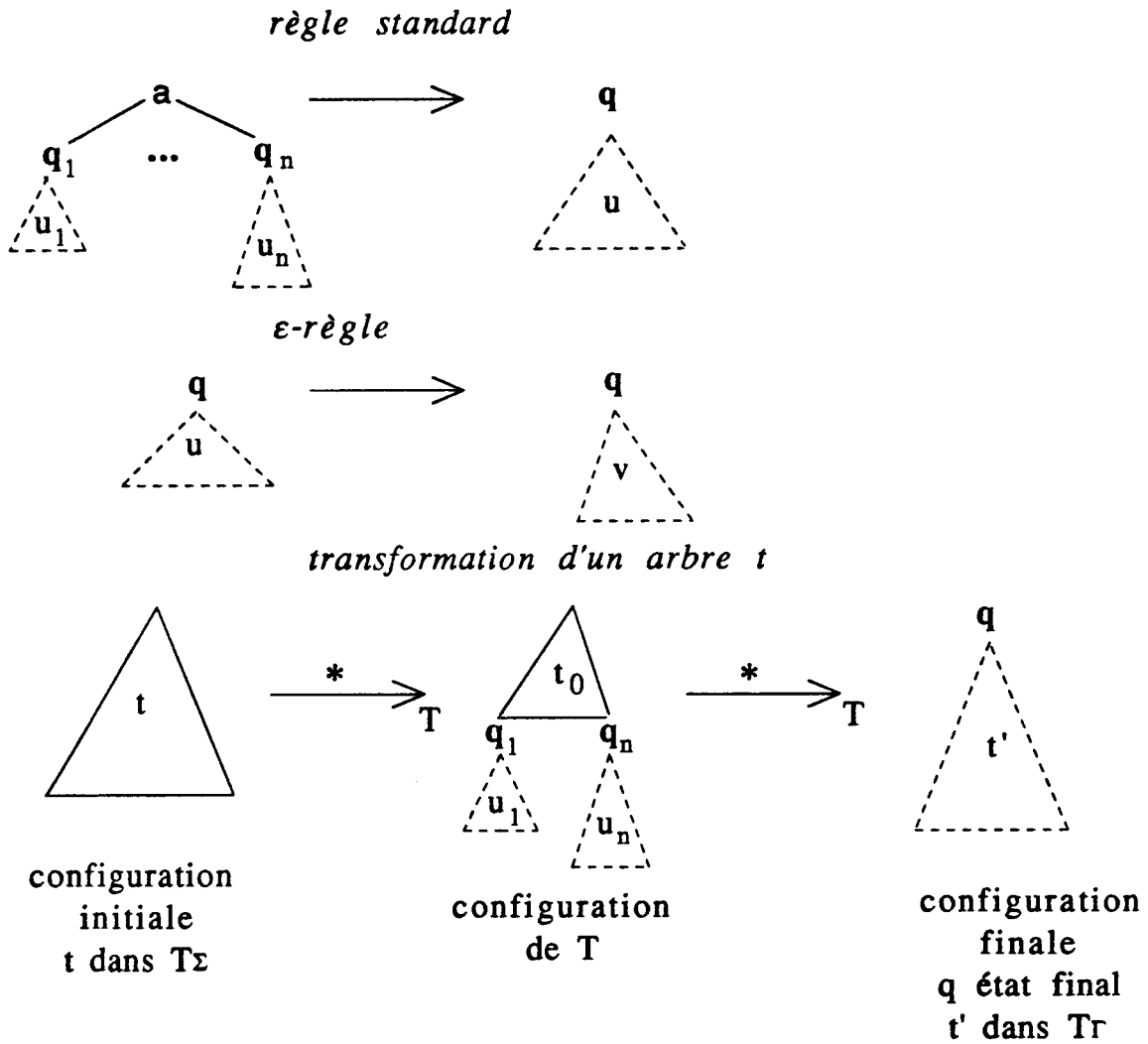
La *tranformation associée à T* est l'ensemble $\tau(T)$ défini par:
$\tau(T)=\{t,t')\in T_\Sigma x T_\Gamma / \ t \overset{*}{\rightarrow}_T q(t'), q\in Q_f \}$.

T et T' sont *équivalents* (noté T≡T') si et seulement si $\tau(T) = \tau(T')$.

L'arité maximale des états de Q est appelée le rang de T et notée rang(T). La hauteur maximale des arbres de piles (appartenant à $T_\Gamma(X)$) apparaissant dans les membres gauches de règles de R est appelée la profondeur de T et notée prof(T).

Nous définissons et notons TPDA**, TPDA$_n$*, TPDA*$_k$, TPDA$_{nk}$ les classes d'automates à piles ascendants d'arbres vérifiant respectivement: rang et profondeur quelconques, rang n et profondeur quelconque, rang quelconque et profondeur k, rang n et profondeur k.

Exemple: Nous illustrons les définitions précédentes dans le cas d'un automate à piles d'arbres appartenant à la classe TPDA$_1$* définie ci dessus.



Proposition 1: *Pour tout automate à piles T dans TPDA**, il existe un automate T' dans TPDA$_1$* tel que T≡T'.*

Preuve.

Soit T=(Σ,Γ,Q,Q$_f$,R) un automate à piles ascendant d'arbres appartenant à la classe TPDA**. Nous construisons un automate à piles T' équivalent appartenant à la classe TPDA$_1$* comme suit:

A tout état q d'arité n supérieure ou égale à 2, on associe un nouvel état qn d'arité 1. On obtient ainsi un ensemble d'états Q'_1. Nous posons $Q'=Q_0 \cup Q_1 \cup Q'_1$.

On ajoute à l'alphabet de pile Γ de nouvelles lettres d'arité k, pour tout k supérieur ou égal à 2 et inférieur ou égal au rang de T. Formellement, $\Gamma'=\Gamma \cup \{<\#,k> / 2 \leq k \leq rang(T)\}$.

Pour tout terme $q(u_1,...,u_n)$ avec n supérieur ou égal à 2 apparaissant dans un membre gauche ou droit de règle de R, on lui substitue $qn(<\#,n>(u_1,...,u_n))$. On obtient ainsi un nouvel ensemble de règles de réécriture R'.

Soit $T'=(\Sigma,\Gamma',Q',Q_f,R')$. Par construction, T' est un automate qui appartient à la classe $TPDA_1*$ (tous les états sont maintenant d'arité maximale 1). La preuve de l'équivalence des automates T et T', c'est-à-dire de l'égalité $\tau(T)=\tau(T')$, est sans difficultés:

A toute configuration $c=t_0(q_1(u_{11},...,u_{1n_1}),...,q_m(u_{m1},...,u_{mn_m}))$, on associe la configuration c' de T' obtenue en substituant à tout terme $q_i(u_{i1},...,u_{in_i})$ le terme $q_in_i(<\#,n_i>(u_{i1},...,u_{in_i}))$ si n_i est supérieur ou égal à 2.

On démontre alors que, pour toutes configurations c_1 et c_2 de T et leurs configurations associées c'_1 et c'_2 de T', on a l'équivalence suivante: $c_1 \rightarrow_T c_2$ si et seulement si $c'_1 \rightarrow_{T'} c'_2$.

De plus, à toute configuration initiale t de T, on associe la configuration initiale t de T' et les configurations finales de T et T' sont les mêmes car $Q_f=Q'_f$ et, dans la construction, les états d'arité 1 ont été laissés inchangés.❑

On peut remarquer, dans la construction utilisée dans la preuve de la proposition 1, que si T appartient à la classe $TPDA_{nk}$, alors T' appartient à la classe $TPDA_{1k+1}$.

Proposition 2: *Pour tout automate à piles T dans $TPDA*_1$, il existe un automate T' dans $TPDA_{12}$ tel que $T \equiv T'$.*

Preuve. Ce résultat est une conséquence de la proposition 1.

Proposition 3: *Pour tout automate à piles T dans $TPDA_1*$, il existe un automate T' dans $TPDA*_1$ tel que $T \equiv T'$.*

Preuve.

Soit $T=(\Sigma,\Gamma,Q,Q_f,R)$ un automate à piles ascendant d'arbres appartenant à la classe $TPDA_1*$. Nous construisons un automate à piles $T'=(\Sigma,\Gamma,Q',Q'_f,R')$ équivalent de la classe $TPDA*_1$ comme suit:

L'ensemble des états Q' de T' contient Q. L'ensemble des états finaux Q'_f de T' est égal à Q_f.

Nous construisons de nouveaux états dans l'ensemble Q' et le système de réécriture R' de la façon suivante:

Soit un terme q(u) apparaissant dans un membre gauche de règle de R tel que la profondeur de u soit supérieure ou égale à 2. Illustrons, tout d'abord, sur un exemple, la construction faite. L'idée intuitive étant de descendre niveau par niveau, en vérifiant l'égalité du terme avec u et en mémorisant tous les sous-arbres utiles.

Exemple: Soit le terme q(u)=q(c(x,b(y,b(a,z)),t) apparaissant dans un membre gauche de règle de R, on définit les nouveaux états q_1, q_2, q_3 et q_4 et les règles suivantes:

$q(c(x_1,x_2,x_3)) \rightarrow q_1(x_1,x_2,x_3)$

$q_1(x_1,b(x_2,x_3),x_4) \rightarrow q_2(x_1,x_2,x_3,x_4)$

$q_2(x_1,x_2,b(x_3,x_4),x_5) \rightarrow q_3(x_1,x_2,x_3,x_4,x_5)$

$q_3(x_1,x_2,a,x_3,x_4) \rightarrow q_4(x_1,x_2,x_3,x_4)$

Formellement, pour q(u) vérifiant les conditions précédentes, on introduit les états et les règles suivantes:

Si $u=l(u_1,...,u_m)$, on ajoute à R' la règle $q(l(x_1,...,x_m))\rightarrow q_1(x_1,...,x_m)$.

Soit k un entier supérieur ou égal à 1 et soit $tronc(u,k)=u_k$, la troncature à la profondeur k de u dont nous rappelons la définition: Pour toute occurrence p dans $O(u_k)$, soit $|p|<k$ et les symboles apparaissant à l'occurrence p de u_k et u sont égaux, soit $|p|=k$ et $u_{k/p}\in X$, de plus si u_k appartient à $T_\Sigma(X_m)$, $fr(u_k)\in \Gamma^* x_1 \Gamma^* ... x_m \Gamma^*$ et les variables sont ordonnées et toutes distinctes.

Pour tout entier k supérieur ou égal à 1 et inférieur à la profondeur de u, considérons les termes u_k et u_{k+1} avec $u_{k+1}=u_k(l_1(x_{11},...,x_{1n_1}),...,l_p(x_{p1},...,x_{pn_p})$ dans $T_\Sigma(X_m)$ (remarquons que l_j peut être une lettre d'arité 0). On définit alors la règle $q_k(y_1,...,y_q)\rightarrow q_{k+1}(x_1,...,x_m)$ telle que $y_i=l_j(x_{j1},...,x_{jn_j})$ ou $y_i=x_n$ (x_n est une variable apparaissant dans u_k). Une borne supérieure pour l'arité des nouveaux états créés est la largeur de u.

Cette construction est faite pour tous les termes q(u) ant dans un membre gauche avec la profondeur de u supérieure ou égale à 2. Pour deux termes distincts, on introduit de nouveaux ensembles d'états disjoints.

Pour chaque ε-règle $q(u)\rightarrow q'(u')$ de R, si la profondeur de u est inférieure ou égale à 1, on ajoute la règle $q(u)\rightarrow q'(u')$ dans R', sinon, on ajoute la règle $q_{h(u)}(x_{i_1},\ldots,x_{i_n})\rightarrow q'(u')$ dans R' , où $(x_{i_1},\ldots,x_{i_n})$ désigne le n-uplet de variables obtenu en renommant les variables de u dans l'ordre du feuillage et en respectant les égalités éventuelles. Une telle règle sera dite de type (*).

Exemple: Soit l'ε-règle $q(\ c(x,\ b(y,x),\ a(y))\ )\rightarrow q'(u')$, on définit les règles:

$q(c(x_1,x_2,x_3)) \rightarrow q_1(x_1,x_2,x_3)$

$q_1(x_1,b(x_2,x_3),a(x_4)) \rightarrow q_2(x_1,x_2,x_3,x_4)$

et l'ε-règle $q_2(x_1,x_2,x_1,x_2)\rightarrow q'(u')$ dans R'.

De plus, soit $\alpha(q_1(u_1),\ldots,q_m(u_m))\rightarrow q'(u')$ une règle standard de T, on substitue à chaque sous-terme $q_i(u_i)$ tel que la profondeur $h(u_i)$ de u_i est supérieure ou égale à 2 le sous-terme $q_{h(ui)}(x_{i_1},\ldots,x_{i_{ni}})$ défini comme dans le cas précédent. La règle ainsi définie est ajoutée à R'. Une telle règle sera dite de type (**).

La construction de T' étant faite, il nous reste à prouver que les automates obtenus sont équivalents, soit encore l'égalité $\tau(T)=\tau(T')$.

Si c et c' sont deux configurations de T telles que $c\rightarrow_T c'$, c et c' sont deux configurations de T' et on montre, sans difficultés, que $c\overset{*}{\rightarrow}_{T'}c'$, par construction de R'. On peut donc en déduire, par récurrence, que $\tau(T)$ est inclus dans $\tau(T')$.

Réciproquement, montrons que $\tau(T')$ est inclus dans $\tau(T)$. Considérons la réduction par T' d'un terme t de T_Σ (configuration initiale) en un terme q(t') avec q état final de T' et t' terme de $T_{\Gamma'}$ (configuration finale). t est, de façon évidente, une configuration initiale de T, q(t') est une configuration finale de T, en effet, les alphabets de piles de T et T' sont les mêmes, et, par construction de R', les membres droits de règles de R' de type (*) ou (**) ne contiennent que des états de Q, donc lors de l'application de la dernière règle, q est bien un état de T.

Nous allons, pour terminer la preuve, prouver le lemme technique suivant:

Lemme: Soit c et c' deux configurations de T, si $c \overset{*}{\rightarrow}_{T'} c'$ en utilisant n règles de type (*) ou (**) alors $c \overset{*}{\rightarrow}_T c'$ en utilisant n règles de R.

Preuve.

La preuve se fait par récurrence sur n.

Pour n=1, comme l'ensemble des états créés est disjoint de l'ensemble Q et comme c' est une configuration de T, on ne peut que simuler une règle de R et donc, $c \rightarrow_T c'$.

Soit $n \geq 2$ et supposons la propriété vraie jusqu'à n-1. Considérons deux configurations c et c' de T telles que $c \rightarrow^*_{T'} c'$ avec n utilisations de règles de type (*) ou (**). Considérons la première application d'une règle r de type (*) ou (**), nous avons donc: $c \rightarrow^*_{T'} c_1 \rightarrow^r_{T'} c_2 \rightarrow^*_{T'} c'$.

Cas 1: Si c_2 est une configuration de T, alors $c \rightarrow_T c_2$ (cas n=1) et il suffit d'appliquer l'hypothèse de récurrence à la réduction de c_2 en c'.

Cas 2: Si c_2 n'est pas une configuration de T, les applications des règles de T' permettant d'appliquer la règle r sont indépendantes des applications des autres règles, ceci par construction de T', en effet, les ensembles d'états créés pour deux termes distincts sont disjoints. Donc, il existe une configuration c'_3 de T' et une configuration c_3 de T telles que: $c \rightarrow^*_{T'} c'_3 \rightarrow^r_{T'} c_3 \rightarrow^*_{T'} c_2$. Il suffit alors d'utiliser le même raisonnement que dans le cas 1. Finpreuvelemme.❑
Finpreuveproposition❑

Nous avons donc prouvé l'équivalence entre les classes TPDA**, $TPDA_{1*}$, $TPDA_{*1}$ et $TPDA_{12}$. La classe la plus utilisée est la classe $TPDA_{1*}$. Nous allons maintenant rappeler la définition des automates à piles ascendants d'arbres de R.V.Book, J.H.Gallier et K.M.Schimpf. Dans cette définition, les règles de lecture ("read-rules") n'autorisent pas de modification des arbres de pile.

Définition 2: *Un automate à piles ascendant d'arbres read-reduce* (noté $tpda^{rr}$) est un quintuplet $A=(\Sigma,\Gamma,Q,Q_f,R)$ où:

Σ est un alphabet gradué fini d'entrée.

Γ est un alphabet gradué fini de pile.

Q est un ensemble fini d'états d'arité 1.

Q_f, inclus dans Q, ensemble d'états finaux.

R est un système de réécriture fini sur $T_{\Sigma \cup \Gamma \cup Q}$ dont les règles ont l'une des deux configurations suivantes:

Règles de lecture ("read rules"):

$\alpha(q_1(x_1),...,q_k(x_k)) \rightarrow q(\beta(x_1,...,x_k))$ avec $\alpha \in \Sigma_k$, $k \geq 0$, $q_i \in Q$, $x_i \in X$, $i=1,...,k$, $\beta \in \Gamma_k$.

Règles de réduction ("reduce rules", "tree stack update rules"):

$q(l) \rightarrow q'(r)$ avec $q,q' \in Q$, $l,r \in T_\Gamma(X)$.

Nous notons $TPDA^{rr}$ la classe des automates à piles ascendants read-reduce. On peut remarquer que cette classe est une sous-classe de la classe $TPDA_1*$.

Les définitions précédentes de l'ensemble des configurations, de configuration initiale, de configuration finale, de mouvement élémentaire, de transformation associée à A s'appliquent donc à tout automate A de la classe $TPDA^{rr}$.

Proposition 4: *Pour tout automate à piles T de la classe $TPDA_1*$, il existe un automate à piles A de la classe $TPDA^{rr}$ tel que $T \equiv A$.*

Preuve.

Soit $T=(\Sigma,\Gamma,Q,Q_f,R)$ un automate de la classe $TPDA_1*$. Nous construisons l'automate équivalent $A=(\Sigma,\Gamma',Q',Q'_f,R')$ de la classe $TPDA^{rr}$ comme suit:

Remarquons, tout d'abord, que Q peut contenir des états d'arité 0. On considère donc un nouveau symbole $\perp$ (correspondant au symbole de pile vide), et, à tout état q d'arité 0, on associe un nouvel état q^1 d'arité 1. On remplace alors toute occurrence d'un état q d'arité 0 par le terme $q^1(\perp)$.

Pour toute règle standard r: $\alpha(q_1(u_1),\ldots,q_m(u_m)) \rightarrow q(u)$ de T, on définit un nouvel état qr, une nouvelle lettre $\bar{\alpha}$ et les nouvelles règles suivantes: $\alpha(q_1(x_1),\ldots,q_m(x_m)) \rightarrow qr(\bar{\alpha}(x_1,\ldots,x_m))$ (règle de lecture, "read-rule") et $qr(\bar{\alpha}(u_1,\ldots,u_m)) \rightarrow q(u)$ (règle de réduction, "reduce-rule").

On définit Γ' comme l'union de Γ et des nouvelles lettres créées, Q' comme l'union de Q et de l'ensemble des nouveaux états, Q'_f est égal à Q_f et R' est la réunion de l'ensemble des ε-règles de R et de l'ensemble des nouvelles règles. A appartient bien à la classe $TPDA^{rr}$ et on vérifie facilement l'égalité entre $\tau(T)$ et $\tau(A)$. □

A.2 Automates à piles déterministes

Nous allons maintenant définir la notion de déterminisme associé à un automate à piles ascendant d'arbres.

Nous restreignons notre définition au cas d'automates à piles dont l'ensemble R de règles de réécriture est linéaire à gauche. En effet, nous dirons qu'un automate est déterministe si l'ensemble R est sans paires critiques et, dans le cas où R est linéaire à gauche, on peut en déduire que la relation $\rightarrow_R$ est confluente (voir [HUET80] pour le résultat et pour un exemple de système de réécriture sans paires critiques et non confluent).

Définition 3: Un automate à piles $T=(\Sigma,\Gamma,Q,Q_f,R)$ ascendant d'arbres est *déterministe* si le système de réécriture R est linéaire à gauche et sans paires critiques.

Nous notons DTPDA**, $DTPDA_1*$, $DTPDA*_1$ et $DTPDA^{rr}$ les classes d'automates à piles ascendants déterministes d'arbres associées aux classes définies précédemment. Nous prouvons que l'équivalence des différentes classes est encore vraie pour les classes déterministes correspondantes.

Proposition1': *Pour tout automate à piles T dans DTPDA**, il existe un automate T' dans* $DTPDA_1*$ *tel que* $T\equiv T'$.

Preuve.

Il est facile de vérifier que, dans la construction de T' faite dans la preuve de la proposition 1, si R est linéaire à gauche et sans paires critiques, R' vérifie les mêmes propriétés. ❑

Proposition 2': *Pour tout automate à piles T dans* $DTPDA*_1$, *il existe un automate T' dans* $DTPDA_{12}$ *tel que* $T\equiv T'$.

Preuve. Ce résultat est une conséquence de la proposition 1'.❑

Pour les preuves des deux dernières propositions, nous démontrons tout d'abord un lemme technique.

Lemme: *Soit T un automate à piles déterministe ascendant d'arbres de la classe* $DTPDA_1*$, *il existe un automate T' de* $DTPDA_1*$ *équivalent à T et vérifiant la propriété (P) suivante:*

(P): pour tous états q et q', pour tous arbres de piles u et v de $T_\Gamma(X)$, *si q(u) apparaît dans un membre gauche d'une règle standard et q'(v) apparaît dans un membre gauche de règle (standard ou ε-), alors on a l'équivalence* $(q=q')\Leftrightarrow(u=v)$.

Preuve.

Remarquons, tout d'abord, que la propriété (P) n'est pas en général vérifiée par un automate déterministe.

En effet, considérons deux règles standard dont les membres gauches respectifs sont $b(q(u),q_1(u_1))$ et $b(q_2(u_2),q(v))$. Ces deux règles ne définissent pas de paire critique pour le système de réécriture même dans le cas où u et v sont unifiables.

Un autre exemple est de considérer une règle standard et une ε-règle de membres gauches respectifs $b(q(u),q_1(u_1))$ et $q(v)$, si u et v sont distincts et non unifiables, ces deux règles n'induisent pas de paire critique.

Nous verrons dans les démonstrations des propositions 3' et 4' que nous avons besoin de cette propriété (P) pour que les constructions des automates équivalents préservent le déterminisme.

Soit un état q et soit L(q) l'ensemble des arbres de piles u de $T_\Gamma(X)$ tels que q(u) apparait dans un membre gauche de règle standard ou d'ε-règle. Soit $k(q) = \max\{ h(u) / u \in L(q) \} + 1$. Soit $K = \max\{k(q) / q \in Q\}$.

Pour tout arbre K-normalisé t, on définit un nouvel état q_t et on définit l'ensemble de règles $q(t) \rightarrow q_t(t)$, pour tout q de Q.

Pour toute ε-règle $q(u) \rightarrow q'(u')$, pour tout arbre K-normalisé t tel qu'il existe une substitution σ vérifiant $\sigma(u)=t$, on définit la règle $q_t(t) \rightarrow q'(t')$, avec $t'=\sigma(u')$.

Pour toute règle standard $\alpha(q_1(u_1),\ldots,q_m(u_m)) \rightarrow q(u)$ de R, pour tous arbres K-normalisés $t_1,\ldots,t_m$ tels qu'il existe une substitution σ vérifiant $\sigma(u_i)=t_i$, pour tout i, on définit la règle $\alpha(q_{t1}(t_1),\ldots,q_{tm}(t_m)) \rightarrow q(t)$, avec $t=\sigma(u)$.

Q' est la réunion de Q et de l'ensemble des états q_t associés aux arbres de $T_\Gamma(X,K)$. Q'_f est égal à Q_f. R' est la réunion des ensembles de règles ainsi construites. On vérifie aisément que l'automate T' appartient à la classe $TPDA_1*$ et que le déterminisme de T est préservé dans la construction de T'. De plus, T' satisfait la propriété (P), en effet, considérant la propriété (P), le seul cas à considérer est $q(u)=q_t(t)$ et $q'(v)=q_{t'}(t')$ et l'équivalence demandée est bien vérifiée. En effet, $q_t=q_{t'}$ si et seulement si t=t'. L'équivalence des automates T et T' se vérifie par double inclusion.❑

Proposition 3': *Pour tout automate à piles T dans $DTPDA_1*$, il existe un automate T' dans $DTPDA*_1$ tel que $T \equiv T'$.*

Preuve.

Soit $T=(\Sigma,\Gamma,Q,Q_f,R)$ un automate à piles ascendant d'arbres appartenant à la classe $DTPDA_1*$. Nous pouvons supposer, d'après le lemme 2, que T satisfait la propriété (P).

Nous construisons un automate à piles $T'=(\Sigma,\Gamma,Q',Q'_f,R')$ équivalent de la classe $DTPDA*_1$ par la même construction que celle utilisée dans la preuve de la proposition 3.

On peut vérifier que cette construction n'induit pas de nouvelles paires critiques à l'exception du cas des règles de la forme $q(l(x_1,...,x_m))\rightarrow q_1(x_1,...,x_m)$ avec $u=l(u_1,...,u_m)$, q(u) apparait dans un membre gauche de règle de R et la profondeur h(u) de u est supérieure ou égale à 2.

Soit q un état et soit L(q) l'ensemble des arbres de pile u tel que q(u) apparaît en membre gauche d'une règle de R. En utilisant la propriété (P), on ne peut avoir deux termes q(u) et q(v), avec u et v distincts, que dans le cas où ils apparaissent tous les deux en membre gauche d'ε-règles de R. Mais, dans ce cas, l'automate T étant déterministe, deux termes distincts de L(q) ne sont pas unifiables (on aurait en effet une paire critique entre deux ϵ-règles de T).

On simule les règles de la même façon que dans la preuve de la proposition 3, mais, on introduit de plus un ordre total sur les éléments de L(q). D'après les remarques précédentes, une et une seule de ces simulations peut mener à un succès. On ajoute donc de nouvelles règles de façon à ce que les simulations se fassent en respectant l'ordre total imposé à l'ensemble L(q) et, de plus, que lorsqu'une simulation échoue, on réinitialise l'arbre en cours de traitement pour essayer de reconnaître l'arbre suivant en respectant l'ordre sur L(q). Nous ne détaillons pas ici les règles correspondant à cette partie de la construction.

Par cette construction, nous obtenons un automate déterministe de la classe $DTPDA*_1$. La preuve de l'équivalence entre T et T' est identique à la preuve de la proposition 3. ❑

Proposition 4': *Pour tout automate à piles T de la classe $DTPDA_1*$, il existe un automate à piles A de la classe $DTPDA^{rr}$ tel que $T \equiv A$.*

Preuve.

Soit $T=(\Sigma,\Gamma,Q,Q_f,R)$ un automate de la classe $DTPDA_1*$ possédant la propriété (P) (lemme). Nous construisons l'automate $A=(\Sigma,\Gamma',Q',Q'_f,R')$ équivalent à T de la classe $TPDA^{rr}$ comme dans la proposition 4. Montrons que A est déterministe.

Il est clair que R' est linéaire à gauche car R est linéaire à gauche. Montrons donc que R' est sans paire critique.

Cas 1: Il est évident que la construction de A n'induit pas de nouvelles paires critiques entre deux règles de réduction ("reduce-rules").

Cas 2: Considérons maintenant le cas d'un règle de réduction ("reduce-rule") et d'une règle de lecture ("read-rule").

Cas 2.1: Soit $\alpha(q_1(x_1),...,q_m(x_m)) \rightarrow qr(\bar{\alpha}(x_1,...,x_m))$ une règle de lecture de A (déduite de la règle standard $\alpha(q_1(u_1),...,q_m(u_m)) \rightarrow q'(u)$ de T) et $q(v) \rightarrow q''(v')$ une règle de réduction de A. T est déterministe donc, pour tout i, q(v) et $q_i(u_i)$ ne sont pas unifiables, et donc, pour tout i, on a soit $q \neq q_i$, soit $q=q_i$ avec v et u_i non unifiables. En utilisant la propriété (P), le deuxième cas ne peut se produire donc, pour tout i, q est différent de q_i, donc on n'a pas de paires critiques entre deux règles de ce type.

Cas 2.2: Si on considère maintenant une règle de lecture et une règle de réduction de la forme $qr(\bar{\alpha}(u_1,...,u_m)) \rightarrow q(u)$. Ces deux règles ne peuvent induire de paire critique car la nouvelle lettre $\bar{\alpha}$ ne peut, par construction de A, apparaître en membre gauche d'une règle de lecture.

Cas 3: Le cas de deux règles de lecture se résoud exactement comme le cas 2.1 en utilisant le déterminisme de T et la propriété (P).❑

La définition du déterminisme pour le cas des automates de la classe $TPDA^{rr}$ correspond exactement à la définition du déterminisme donné par R.V.Book et J.H.Gallier dans [BOGA85]. Dans la suite de cette section , sauf mention contraire, nous considérerons toujours des automates à piles appartenant aux classes $TPDA^{rr}$ ou $DTPDA^{rr}$.

B. Calcul de formes normales

Nous définissons dans cette partie la notion de système de réécriture trf (pour tail reduction free).

Dans la section B.1, nous donnons la définition de la propriété trf et montrons que la classe des systèmes de réécriture trf est large. Elle contient, en particulier, les systèmes de réécriture monadiques.

Nous prouvons que pour tout système de réécriture semi-monadique convergent, on peut trouver un système de réécriture semi-monadique équivalent ayant la propriété trf et démontrons que l'on peut simuler une machine de Türing par un système de réécriture trf.

Dans la section B.2, nous donnons quelques exemples de systèmes ayant la propriété trf.

Dans la section B.3, nous démontrons que cette propriété est décidable. Pour prouver ce résultat, nous utilisons la décidabilité de l'inductive réductibilité (D.A.Plaisted).

Dans la section B.4, le théorème 1 permet d'associer, à tout système de réécriture trf et convergent, un automate à piles ascendant d'arbres avec priorité à la réduction (on ne peut appliquer une règle de lecture que lorsqu'on ne peut utiliser aucune règle de réduction) qui calcule les formes normales. Dans le cas où le système de réécriture, vérifiant les mêmes hypothèses, est, de plus, linéaire à gauche, la notion de priorité devient inutile et le théorème 2 permet d'associer à tout système de réécriture trf, convergent et linéaire à gauche un automate à piles ascendant d'arbres et déterministe qui calcule les formes normales.

Nous rappelons que $\rightarrow_{i,S,<}$ est la relation de réécriture IO induite par S respectant <. Par cette relation, on réécrit un terme à une occurence o si aucune règle n'est applicable en dessous de cette occurence et s'il n'existe pas de règle plus grande selon < applicable à cette occurence o.

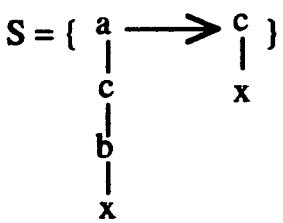
On étend cette définition aux automates à piles d'arbres et nous notons $\rightarrow_{<A}$ *la relation de réécriture IO induite par R selon un ordre < sur R*, où R désigne l'ensemble des règles de A.

Cette définition nous permet de décrire formellement la transformation respectant la stratégie *"priorité à la réduction"* pour un automate à piles ascendant d'arbres.

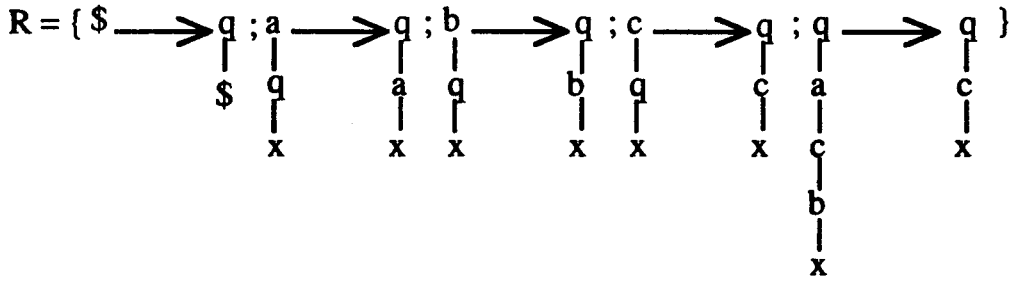
B.1 La propriété trf

Nous introduisons la propriété trf à l'aide des deux exemples suivants.

Exemple 1: Soit $\Sigma=\{a,b,c,\$ \ / \ ar(a)=ar(b)=ar(c)=1, \ ar(\$)=0\}$.
Soit le système de réécriture S réduit à une seule règle défini par:



Soit l'automate à piles $A=(\Sigma,\Gamma,Q,Q_f,R)$ tel que $\Sigma=\Gamma$, $Q=Q_f=\{q\}$ et



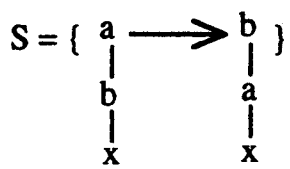
A appartient à la classe $TPDA^{rr}$ et A est non déterministe. Considérons le terme t=aaacbb$ de T_Σ (les parenthèses ont été omises). Les différentes transformations possibles du terme t par l'automate A ainsi défini sont: $t \twoheadrightarrow_A q(t)$ ou $t \twoheadrightarrow_A q(aacb\$)$ ou $t \twoheadrightarrow_A q(ac\$)$.

Les arbres de piles ainsi obtenus sont donc les différentes réductions du terme t par le système de réécriture S.

Si on impose, de plus, que la transformation par A se fasse en respectant la priorité à la réduction (on ne peut appliquer une règle de lecture que lorsqu'aucune règle de réduction n'est applicable, c'est-à-dire que les quatre premières règles sont inférieures à la cinquième pour <), on obtient alors $t \twoheadrightarrow_{<A} q(ac\$)$ et donc l'arbre de pile obtenu dans ce cas est la forme normale de t pour le système de réécriture S.

Exemple 2: Soit $\Sigma=\{a,b,\$ \ / \ ar(a)=ar(b)=1, \ ar(\$)=0\}$.

Soit le système de réécriture S défini par:



Soit l'automate à piles $A=(\Sigma,\Gamma,Q,Q_f,R)$ tel que $\Sigma=\Gamma$, $Q=Q_f=\{q\}$ et R est défini comme précédemment par les règles de lecture de $, a et b et la règle de type "reduce": $q(a(b(x))) \rightarrow q(b(a(x)))$.

On impose, en outre, la stratégie priorité à la réduction, si on considère le terme t=aabb$, on obtient $t \twoheadrightarrow_{<A} aq(abb\$)$ par les règles de lecture, $aq(abb\$) \rightarrow_{<A} aq(bab\$)$ par la règle de type "reduce". On peut alors remarquer que l'on ne pourra pas calculer la forme normale du terme t pour le système de réécriture S.

En effet, on obtient $t \twoheadrightarrow_{<A} q(baab\$)$ et l'arbre de pile baab$ n'est pas la forme normale de t, cette forme normale étant baba$.

La différence entre les systèmes de réécriture considérés dans ces deux exemples est que, si l'on considère le membre droit b(a(x)) de la règle de l'exemple 2, il existe une substitution close irréductible σ telle

que $\sigma(a(x))$ soit réductible par S. Il suffit, en effet, de considérer la substitution $\sigma=\{x\rightarrow b\}$.

Définition: Soit S un système de réécriture, *S possède la propriété trf* (pour tail reduction free) si et seulement si, pour toute règle $l\rightarrow r$ de S avec $r=b(r_1,\ldots,r_n)$ et b lettre d'arité n, pour toute substitution close irréductible σ, alors, pour tout i, $1\leq i\leq n$, on a $\sigma(r_i)$ irréductible pour S.

Remarque: Nous ne considérons que des systèmes de réécriture pour lesquels il existe au moins une substitution close irréductible. Par exemple, nous interdisons que le membre gauche d'une règle de S soit réduit à une variable.

Exemples: Le système de réécriture défini dans l'exemple 1 de la section B.1 possède de façon évidente la propriété trf. Par contre, celui de l'exemple 2 de la même section ne possède pas la propriété trf en considérant la substitution close irréductible $\sigma=\{x\rightarrow b\}$.

B.2 Exemples de systèmes trf

Exemple 1: Systèmes de réécriture monadiques.

Définition: Un système de réécriture est appelé *monadique* si, pour toute règle $l\rightarrow r$ de S, $h(l)\geq 1$ et r est soit une variable, soit une constante, soit un terme du type $b(y_1,\ldots,y_n)$ où les y_i sont des variables pour tout i, $1\leq i\leq n$.

Proposition 1: *tout système de réécriture monadique a la propriété trf.*

Preuve. La preuve est évidente car $h(r)\leq 1$.❑

Exemple 2: Systèmes de réécriture semi-monadiques.

Définition: Un système de réécriture est appelé *semi-monadique* si, pour toute règle $l\rightarrow r$ de S, $h(l)\geq 1$ et r est soit une variable ($r\in X$) soit du type $b(y_1,\ldots,y_k)$ où b est un élément de Σ_k et, pour tout i de [1,k], y_i est une variable ($y_i\in X$) ou un terme clos ($y_i\in T_\Sigma$).

On remarquera que les systèmes de réécriture monadiques et les systèmes clos sont des cas particuliers de systèmes de réécriture semi-monadiques.

Proposition 2: *Considérons un système de réécriture S semi-monadique et convergent. Il existe un système de réécriture semi-monadique et convergent S' équivalent à S tel que S' posséde la propriété trf.*

Preuve.

Nous remplaçons chaque terme clos apparaissant dans une partie droite de règle par sa forme normale pour S. ❑

Exemple 3: Simulation d'une machine de Turing.

Nous pouvons identifier toute machine de Turing à un automate à piles de la classe $TPDA_{1,2}$, que l'on peut également considérer comme un système de réécriture trf, de la façon suivante:

Nous mettons en correspondance les chaînes de caractères et les arbres monadiques et utilisons les nouveaux symboles $ et #(x) pour délimiter les réécritures.

Le mot d'entrée t de Σ^* est identifié à l'arbre #t'$ sur un alphabet Σ'. L'alphabet de pile Σ est distinct de Σ', Σ' est une copie de Σ (nous notons a' la copie de a). Le mot de sortie u est identifié à #u$. Nous notons ^t le miroir de t; s, s' sont des états spéciaux, s_f est l'état final.

"règles de lecture" :

$\$ \rightarrow s(\$)$;

$a's(x) \rightarrow s(ax)$ pour toute lettre a' de Σ';

$\#s(x) \rightarrow q_i(T(\#(x),\$))$ (q_i est l'état initial de la machine de Turing).

"règles de réécriture" :

A chaque état q de la machine de Turing on associe l'état q d'arité 1 du tpda.

A chaque règle r de la machine de Turing (q,a) → (q',b,droite) nous associons les règles du tpda:

$q(T(a(x),y)) \rightarrow q(u_r(b(x),y))$,

$q(u_r(x,c(y))) \rightarrow q'(T(c(x),y))$, pour tout c de Σ;

A chaque règle de la machine de Turing (q,*) → (q',b*) (extension de la bande), nous associons les règles du tpda $q(T(\#(x), y)) \rightarrow q'(T(b(x),\#(y)))$;

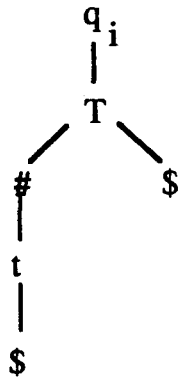
A chaque règle de la machine de Turing (q,a) → (q',b,gauche), nous associons la règle du tpda $q(T(a(x),y)) \rightarrow q'(T(x,b(y)))$;

A chaque état final q_f de la machine de Turing, nous associons un mouvement final du tpda qui remonte l'état final en utilisant les règles suivantes:

$q_f(T(a(x),y) \rightarrow s'(T(x,a(y)))$, pour tout a de Σ,
$s'(T(a(x),y) \rightarrow s'(T(x,a(y)))$, pour tout a de Σ,
$s'(T(\$,y) \rightarrow s_f(\#(y))$.

Théorème 1: *Nous pouvons simuler toute machine de Turing par un système de réécriture trf associé à un* $TPDA_{1,2}$.

Preuve.
Grâce aux règles de lecture, l'arbre #t'$ est transformé en l'arbre



La simulation ne peut commencer qu'après l'apparition de l'état q_i au sommet de l'arbre; les règles de réécriture données ci-dessus simulent les différents mouvements d'une machine de Turing; à tout état final de la machine correspondent des réécritures amenant dans l'état final s_f. Aussi on vérifie qu'à partir d'une donnée ^t , nous obtenons ^u par la machine de Turing si et seulement si $\#t'\$ \xrightarrow{*} s_f(\#u\$)$ pour le tpda.
De plus, l'ensemble des règles que nous avons défini est un système de réécriture trf. ❑

B.3 La propriété trf est décidable

Théorème: *La propriété trf est décidable.*

Preuve.
Nous rappelons, tout d'abord, quelques définitions.

Un *terme t* de $T_\Sigma(X)$ est *inductivement réductible* pour S si et seulement si toutes ses instances closes $\sigma(t)$ dans T_Σ sont réductibles pour S.

Une *substitution* σ est *inductivement réductible* pour S si et seulement si $\$(\sigma(x_1),...,\sigma(x_n))$ est inductivement réductible pour S avec $D(\sigma)=\{x_1,...,x_n\}$ et $ comme nouveau symbole (c'est à dire si et seulement

si pour toute substitution close δ, il existe x dans $D(\sigma)$ tel que $\delta(\sigma(x))$ soit réductible pour S).

Soit W un sous-ensemble de V, nous notons $\sigma_{/W}$ la substitution dont le domaine est l'intersection du domaine $D(\sigma)$ et de W, elle sera appelée restriction de σ à W.

La décidabilité de la propriété trf pour un système de réécriture fini S correspond à la décidabilité de la *propriété P(t)*: "pour toute substitution close irréductible σ, $\sigma(t)$ est irréductible", étant donné que le nombre de règles et donc de termes r_i est fini. Pour prouver la décidabilité de P(t), il est équivalent de prouver la décidabilité de P'(t), la négation de P(t). *Propriété P'(t):* "il existe une substitution close irréductible σ telle que $\sigma(t)$ est réductible".

Pour cela, nous allons prouver le résultat suivant:

Lemme: *Il existe une substitution close irréductible σ pour laquelle $\sigma(t)$ est réductible si et seulement s'il existe une partie gauche l de règle de R et un sous-terme t' (différent d'une variable) de t tels que l et t' soient unifiables par un unificateur le plus général σ', $\sigma'/V(t')$ n'étant pas inductivement réductible.*

Preuve.

1) $\Rightarrow$:

Supposons que $\sigma(t)$ soit réductible pour la substitution close irréductible σ. Il existe donc une décomposition t = t".t' avec t' non réduit à une variable telle que $\sigma(t') = \sigma_2(l)$ pour une partie gauche l de règle de R et pour une substitution σ_2. t' et l étant unifiables, considérons l'unificateur le plus général σ' de t' et l vérifiant donc $\sigma'(t')=\sigma'(l)$.
Nous pouvons alors distinguer deux cas:

Premier cas: σ' est une substitution close.
Dans ce cas $\sigma_{/V(t')}=\sigma'_{/V(t')}$ (i.e. $\sigma(t')=\sigma'(t')$). Si $\sigma'_{/V(t')}$ était réductible alors σ le serait aussi ce qui contredit l'affirmation que σ est irréductible.

Deuxième cas: σ' est une substitution non close.
Considérons la substitution $\sigma"$ telle que $\sigma(t') = \sigma"(\sigma'(t'))$ et $\sigma_{/V(t')}= \sigma"(\sigma'_{/V(t')})$.
Si $\sigma'_{/V(t')}$ était inductivement réductible, alors pour la substitution close $\sigma"$, $\sigma"(\sigma'_{/V(t')})$ serait réductible ce qui contredit l'affirmation que σ est irréductible. Donc $\sigma'/V(t')$ n'est pas inductivement réductible.

2) $\Leftarrow$:

Inversement, supposons qu'il existe une partie gauche l de règle de R et un sous-terme t' (différent d'une variable) de t tels que l et t' soient unifiables par un unificateur le plus général σ', $\sigma'/_{V(t')}$ n'étant pas inductivement réductible.

Nous distinguons à nouveau deux cas:

Premier cas: σ' est une substitution close.

Comme $\sigma'/_{V(t')}$ est irréductible, pour toute variable x_i de V(t'), $\sigma'(x_i)$ est irréductible.

Deuxième cas: σ' est une substitution non close.

Il existe une substitution close σ'' telle que $\sigma''(\sigma'/_{V(t')})$ soit irréductible. Pour toute variable x_i de V(t'), $\sigma''(\sigma'(x_i))$ est irréductible.

Dans les deux cas, nous définissons la substitution σ de la façon suivante: $\sigma(x_i) = \sigma''(\sigma'(x_i))$ ($\sigma'(x_i)$ dans le premier cas) pour tout $x_i \in V(t')$ et $\sigma(y) = t_y$ pour tout y de V(t)-V(t') où t_y est un terme clos irréductible arbitraire. Il est évident que σ est une substitution close et irréductible et que $\sigma(t)$ est réductible. ❑

En utilisant ce lemme et la décidabilité de l'inductive réductibilité ([PLAI85]), nous en déduisons la décidabilité de P'(t), d'où la décidabilité de P(t) et donc la décidabilité de la propriété trf. ❑

B.4 Calcul des formes normales d'un système trf

Dans cette partie, nous allons associer, à tout système de réécriture possédant la propriété trf, un automate à piles ascendant d'arbres.

Dans les propositions 1 et 2, nous démontrons qu'il existe un automate à piles avec priorité (cas général), déterministe (cas linéaire à gauche) qui calcule les formes normales pour le système de réécriture pour la relation de réécriture IO induite par S.

Avec l'hypothèse supplémentaire que S est convergent, on déduit de ces résultats les deux théorèmes principaux de cette section relatifs au calcul des formes normales d'un système de réécriture trf convergent.

Proposition 1: *Pour tout système de réécriture S possédant la propriété trf, il existe un automate à piles ascendant d'arbres* $A=(\Sigma,\Gamma,Q,Q_f,R)$ *et un ordre* $<$ *sur R tels que l'équivalence (*) suivante soit vérifiée:* (*): $(\ t \xrightarrow{*}_{i,S} t' \text{ et } t' \in IRR(S)\ ) \Leftrightarrow (\ t \xrightarrow{*}_{<A} q(t') \text{ et } q \in Q_f\ )$.

Preuve.

Soit S un système de réécriture sur $T_\Sigma(X)$ possédant la propriété trf. Nous définissons, tout d'abord, A et $<$.

$A=(\Sigma,\Gamma,Q,Q_f,R)$ avec $\Gamma=\Sigma$, $Q=\{q, q'\}$, $Q_f=\{q\}$ et R constitué des règles des trois types suivants:

(i): $q(l) \rightarrow q(r)$, pour toute règle $l \rightarrow r$ de S.

(ii): $q(x) \rightarrow q'(x)$.

(iii): $b(q'(x_1),...,q'(x_k)) \rightarrow q(b(x_1,...,x_k))$, pour tout b de Σ_k.

Les règles de type (i) et (ii) sont les règles de réduction ("reduce-rules") de A, les règles de type (iii) sont les règles de lecture ("read-rules") de A.

La relation d'ordre $<$ sur R est une relation d'ordre partiel définie par: $(q(x) \rightarrow q'\langle x \rangle) < (q(l) \rightarrow q(r))$, pour toute règle $q(l) \rightarrow q(r)$ de R.

Intuitivement, l'ordre $<$ est défini pour que, dans la transformation par A, on n'applique une règle de lecture (de type (iii)) que lorsque les arbres de piles correspondants sont irréductibles pour les règles de réduction (de type (i) et (ii)), c'est-à-dire encore lorsque ces arbres de pile sont irréductibles pour le système de réécriture S.

1) $\Rightarrow$: Nous prouvons pour cela l'implication suivante:

(**): Pour t_0 dans $T_\Sigma(X_n)$, $t_1,..., t_n, u_1,...,u_n$ dans T_Σ,
si (1) $t=t_0(t_1,...,t_n) \rightarrow^*_{i,S} u=t_0(u_1,...,u_n)$, et

(2) Pour tout i de [1,n], $t_i \rightarrow^*_{i,S} u_i$ et la racine de t_i est réécrite,
alors il existe une configuration c de A vérifiant:

(a) $t \rightarrow^*_{<A} c=t_0(q_1(u_1),...,q_n(u_n))$,

(b) si u_i est irréductible pour S, alors $q_i=q'$,

(c) si u_i est réductible pour S, alors $q_i=q$,

(d) si u_i est réductible pour S et $u_i=b(v_1,...,v_k)$ avec b dans Σ_k et $v_1,...,v_k$ dans T_Σ, alors, pour tout j de [1,k], v_j est irréductible pour S.

Preuve de (**): La preuve se fait par induction sur la longueur p de la dérivation $t \rightarrow^*_{i,S} u$.

Pour p=0, c=t est la configuration vérifiant les propriétés.

Soit p un entier supérieur ou égal à 1, et supposons (**) prouvée pour des dérivations de longueur inférieure à p. Considérons une dérivation: $t=t_0(t_1,...,t_n) \rightarrow^{(p-1)}_{i,S} u=t_0(u_1,...,u_n) \rightarrow_{i,S} u'$. Par hypothèse d'induction, il existe une configuration c de A associée à t et u vérifiant les hypothèses (a), (b), (c) et (d). Nous distinguons deux cas:

Premier cas: La réécriture de u en u' se fait à une occurence o n'appartenant pas à $O(t_0)$. Donc, nous réécrivons un sous-terme de l'un des u_i: $u=t_0(u_1,...,u_i,...,u_n) \rightarrow_{i,S} u'=t_0(u_1,...,u'_i,...,u_n)$ avec la règle $l \rightarrow r$ de R.

Soit $c=t_0(q_1(u_1),...,q_i(u_i),...,q_n(u_n))$ la configuration de A associée à t et u. Comme u_i est réductible pour S, $q_i=q$ et, de plus, si $u_i = b(v_1,...,v_k)$, alors, pour tout j de [1,k], v_j est irréductible pour S. Donc la réécriture se fait à une occurence o telle que $u/_o=u_i$. Nous avons donc $u/_o=u_i=\sigma(l)$, $u'/_o=u'_i=\sigma(r)$ avec σ qui est une substitution close et irréductible pour S. Par définition de A, il existe une règle $q(l)\rightarrow q(r)$ dans R (associée à la règle $l\rightarrow r$ de S) et donc $c \rightarrow_{<A} c'=t_0(q_1(u_1),...,q(u'_i),...,q_n(u_n))$ par définition de $\rightarrow_{<A}$.

Si u'_i est réductible pour S, l'état correspondant est bien l'état q et, de plus, si $u'_i = b'(v'_1,...,v'_s)$, alors, pour tout j de [1,s], v'_j est irréductible pour S car σ est une substitution close et irréductible pour S et S possède la propriété trf. Dans ce cas, la configuration c' vérifie les propriétés (a), (b), (c) et (d).

Si u'_i est irréductible pour S, alors, aucune règle de type (i) n'est applicable à l'occurence o pour la configuration c', donc on peut appliquer la règle de type (ii), donc $c' \rightarrow_{<A} c''=t_0(q_1(u_1),...,q'(u'_i),...,q_n(u_n))$ et c" vérifie les conditions (a), (b), (c) et (d).

Deuxième cas: La réécriture de u en u' par S se fait à une occurence o appartenant à $O(t_0)$. Nous avons donc:
$u=t_0(u_1,...,u_n) = u_0(u_1,...,u_{i-1},u',u_{j+1},...,u_n)$ avec $u' = u'_0(u_i,...,u_j)$, $u \rightarrow_{i,S} v$, $v=u_0(u_1,...,u_{i-1},v',u_{j+1},...,u_n)$ et $u' = \sigma(l)$, $v' = \sigma(r)$ pour une règle $l\rightarrow r$ de R.

Par définition de la relation de réécriture IO engendrée par S, tout sous terme propre de u' est irréductible pour S et donc, en particulier, tous les états associés aux arbres de pile $u_i,...,u_j$, pour la configuration c, sont égaux à q'. Soit le sous terme de c dont la racine est à l'occurence o, nous avons $c/_o= u'_o(q'(u_i),..,q'(u_j))$. Pour tout symbole dans u'_0 distinct de la racine de u'_0, on peut appliquer une règle de lecture, on obtient alors q(w) avec w sous-terme propre de u', et donc, w est irréductible pour S, donc, on ne peut, à chaque étape, qu'appliquer la règle de type (ii). Donc, par induction sur la structure du terme, on peut montrer que:
$c/_o \rightarrow^*_{<A} q(u')$ et $c \rightarrow^*_{<A} c'=u_0(q_1(u_1),...,q_{i-1}(u_{i-1}),q(v'),q_{j+1}(u_{j+1}),...,q_n(u_n))$.

Par le même raisonnement que dans le premier cas, la configuration c' ou la configuration c" (selon que v' soit réductible par S ou pas) vérifie les propriétés (a), (b), (c) et (d).

❑finpreuve(**)

Supposons maintenant que $t \rightarrow^*_{i,S} t'$ and $t' \in IRR(S)$. Il existe une décomposition $t=t_0(t_1,...,t_n)$ avec t_0 dans $T_\Sigma(X_n)$ telle que, pour tout i de [1,n], $t_i \rightarrow^*_{i,S} u_i$ et $t'=t_0(u_1,...,u_n)$. En utilisant l'implication (**), il existe une configuration $c=t_0(q'(u_1),...,q'(u_n))$ de A telle que $t \rightarrow^*_{<A} c$ et $c \rightarrow^*_{<A} q'(t')$.
❑finpreuve de $\Rightarrow$.

2) $\Leftarrow$: Pour cela, nous prouvons l'implication suivante:

(***): Pour t_0 dans $T_\Sigma(X_n)$, $t_1,...,t_n$, $\pi_1,...,\pi_n$ dans T_Σ, si
$t=t_0(t_1,...,t_n) \overset{*}{\rightarrow}_{<A} c=t_0(q_1(\pi_1),...,q_n(\pi_n))$, alors
(a) si $q_i=q'$, alors π_i est irréductible pour S,
(b) si $q_i=q$ et $\pi_i=b(\upsilon_1,...,\upsilon_k)$ avec b dans Σ_k et $v_1,...,v_k$ dans T_Σ, alors, pour tout j de [1,k], υ_j est irréductible pour S,
(c) $t \overset{*}{\rightarrow}_{i,S} t_0(\pi_1,...,\pi_n)$.

Preuve de (***): La preuve se fait par induction sur la longueur p du calcul.

Pour p=0, la propriété est vérifiée.

Soit p un entier supérieur ou égal à 1, et supposons (***) vérifiée pour tout calcul de longueur inférieure à p. Considérons un calcul:
$t \overset{(p-1)}{\rightarrow}_{<A} c=t_0(q_1(\pi_1),...,q_n(\pi_n)) \rightarrow_{<A} c'$. Nous distinguons trois cas:

Premier cas: La règle appliquée à l'étape p est de type (i). Donc un sous-terme $q_i(\pi_i)$ est réécrit en $q_i(\pi'_i)$ avec $q_i=q$ pour un certain i et une règle $q(l)\rightarrow q(r)$ (associée à la règle $l\rightarrow r$ de S) de R. Donc, dans cette étape du calcul, nous ne modifions que le $i^{ème}$ arbre de pile et n'introduisons pas de nouvelle occurence de l'état q'. Il suffit donc de prouver que si $\pi'_i=b'(\upsilon'_1,...,\upsilon'_m)$ alors, pour tout j de [1,m], υ'_j est irréductible pour S. Soit σ la substitution utilisée pour réécrire $q_i(\pi_i)$ en $q_i(\pi'_i)$, nous avons $\sigma(l) = \pi_i$ et donc, par hypothèse d'induction, σ est une substitution close irréductible pour S (tout sous-terme propre de π_i est irréductible pour S). De plus $\sigma(r)= \pi'_i$ et donc, S possédant la propriété trf, nous en déduisons que pour tout j de [1,m], υ'_j est irréductible pour S.

Deuxième cas: La règle appliquée à l'étape p est de type (ii). Il existe donc i tel que $q(\pi_i)$ est transformé en $q'(\pi_i)$. Par définition de $\rightarrow_{<A}$ et de <, aucune règle de type (i) ne peut être appliquée au sous-terme $q(\pi_i)$, ce qui signifie que π_i est irréductible pour S.

Troisième cas: La règle appliquée à l'étape p est de type (iii). Donc, un sous-terme de la forme $b(q'(\pi_i),...,q'(\pi_{i+k}))$ avec b dans Σ_k est réécrit en $q(b(\pi_i,...,\pi_{i+k}))$. Par hypothèse d'induction, tous les sous-termes $\pi_i,...,\pi_{i+k}$ sont irréductibles pour S.

Nous avons donc prouvé, par induction, que la configuration c' vérifie (a) et (b). La démonstration de (c) par induction ne présente aucune difficulté.❑finpreuve⇐.
❑finpreuveproposition1.

Nous allons maintenant considérer le cas d'un système de réécriture S linéaire à gauche possédant la propriété trf tel que S soit muni d'une relation d'ordre total et prouver que, dans ce cas, on peut calculer les formes normales pour la relation de réécriture IO induite par S en respectant < à l'aide d'un automate à piles ascendant et déterministe.

Proposition 2: *Pour tout système de réécriture S linéaire à gauche possédant la propriété trf et pour tout ordre total < sur S, il existe un automate à piles déterministe ascendant d'arbres* $B=(\Sigma,\Gamma,Q,Q_f,R)$ *tel que:* $(t \xrightarrow{*}_{i,S,<} t' \text{ and } t' \in IRR(S)) \Leftrightarrow (t \xrightarrow{*}_B q(t') \text{ and } q \in Q_f)$.

Preuve.

Soit S un système de réécriture linéaire à gauche possédant la propriété trf. Soit < un ordre total sur S. Soit $k = \max\{h(l)/\ l \to r \in S\}+1$ et $T_\Sigma(X,k)$ l'ensemble des arbres k-normalisés.

Nous définissons $B=(\Sigma,\Gamma,Q,Q_f,R)$ comme suit:

$\Gamma=\Sigma$,

$Q=\{q\} \cup \{q_t / t \in T_\Sigma(X,k)\}$,

$Q_f=\{q_t$ / t n'est pas unifiable avec un membre gauche de règle de S$\}$,

R est l'ensemble des règles des trois types suivants:

(i): $q(t) \to q_t(t)$, pour tout t de $T_\Sigma(X,k)$.

(ii): $q_t(l) \to q(r)$, pour tout q_t n'appartenant pas à $Q_f \cup \{q\}$, avec $l \to r$ la plus grande règle pour < telle que l filtre t.

(iii): $b(q_1(x_1),...,q_n(x_n)) \to q(b(x_1,...,x_n))$ avec b dans Σ_k et, pour tout i de [1,n], q_i appartient à Q_f.

Les règles de type (i) et (ii) sont les règles de réduction (reduce-rules") de B, les règles de type (iii) sont les règles de lecture ("read-rules") de B.

B est déterministe car R est linéaire à gauche et sans paires critiques. La preuve de l'équivalence annoncée se fait exactement de la même façon que pour la proposition 1.

Intuitivement,

les règles de type (i) permettent de mémoriser dans l'état le sous-arbre de pile à une profondeur k;

les règles de type (ii) permettent de réécrire les arbres de pile par S en respectant l'ordre < sur S. Les états de Q_f correspondent à

l'état q' de la proposition 1, c'est-à-dire que les sous-arbres de piles correspondants sont irréductibles pour S;

les règles de type (iii) permettent de lire un nouveau symbole de l'alphabet d'entrée tout en vérifiant que les sous arbres de piles correspondants sont irréductibles pour S.

Remarque importante: La construction faite n'est valable que pour un système de réécriture linéaire à gauche. En effet, dans le cas non linéaire à gauche, on ne peut pas exprimer avec un nombre fini d'états ou un nombre fini d'arbres (les états q_t ou les arbres t de $T_\Sigma(X,k)$) qu'un terme est unifiable ou pas avec un membre gauche de règle, en effet, on doit tester une égalité de sous-termes à une profondeur non bornée.

Nous allons conclure cette partie avec les deux théorèmes principaux. Nous ajoutons maintenant l'hypothèse convergent (noethérien et confluent) et donc la stratégie de réécriture utilisée pour le système de réécriture S n'importe pas sur le résultat, la forme normale d'un terme étant unique.

Théorème 1: *Pour tout système de réécriture S convergent possédant la propriété trf, il existe un automate à piles ascendant d'arbres* $A=(\Sigma,\Gamma,Q,Q_f,R)$ *et un ordre* $<$ *sur R tel que:*
$(t \xrightarrow{*}_S t' \text{ and } t' \in IRR(S)) \Leftrightarrow (t \xrightarrow{*}_{<A} q(t') \text{ and } q \in Q_f)$.

Théorème 2: *Pour tout système de réécriture S linéaire à gauche convergent possédant la propriété trf, il existe un automate à piles déterministe ascendant d'arbres* $B=(\Sigma,\Gamma,Q,Q_f,R)$ *tel que:*
$(t \xrightarrow{*}_S t' \text{ and } t' \in IRR(S)) \Leftrightarrow (t \xrightarrow{*}_B q(t') \text{ and } q \in Q_f)$.

C. Automates à piles d'arbres et langages d'arbres

C.1 Systèmes de réécriture semi-monadiques et langages reconnaissables

Nous prouvons dans cette partie que la reconnaissabilité est préservée pour les systèmes de réécriture semi-monadiques. Ce résultat généralise le résultat de K. Salomaa ([SALO88]) obtenu pour les systèmes de réécriture monadiques et le même résultat pour les systémes de réécriture clos ([DATI85], [COGI90]).

Nous considérons un système de réécriture S et un langage reconnaissable d'arbres L; nous rappelons que S(L) désigne l'ensemble des réductions des termes de l'ensemble L par le système de réécriture S, i.e $S(L)=\{ t' / t \rightarrow_S^* t' \ \& \ t\in L\}$.

Théorème: *Pour tout système de réécriture semi-monadique linéaire S et tout langage reconnaissable d'arbres L, S(L) est reconnaissable.*

Preuve.

Considérons un automate ascendant d'arbres $A = (\Sigma, Q, Q_f, R)$ complètement spécifié tel que $L(A) = L$.

Soit S un système de réécriture semi-monadique linéaire, c'est à dire tel que pour toute règle $l \rightarrow r$ de S, $h(l)\geq 1$ et r est soit une variable ($r\in X$) soit un arbre du type $b(y_1,...,y_k)$ où b est un élément de Σ_k, y_i est une variable ($y_i\in X$) ou un terme clos ($y_i\in T_\Sigma$), pour tout i de [1,k].

Nous notons Sub l'ensemble de tous les sous-termes clos des parties droites de règles de S et des termes clos y_i.

Nous utilisons le nouveau symbole # lors de la définition des états permettant de reconnaître S(L) lorsque nous rencontrons un arbre n'appartenant pas à l'ensemble Sub.

Nous notons $\bar{Q}$ l'ensemble des états utilisés lors de la reconnaissance de S*(L). Il est défini par:

$\bar{Q}=\{(q,p) \in Q\times(Sub\cup\{\#\}) /$ soit $p=\#$ et $(\exists r\in T_\Sigma)$: $r \rightarrow_A^* q$ avec $r\notin Sub$
soit $p \in Sub$ et $p \rightarrow_A^* q$ $\}$

Nous définissons, par induction, les ensembles R_i de règles de la façon suivante:

$R_0=\{\sigma((q_1,p_1),...,(q_n,p_n))\rightarrow(q,p) / \sigma\in\Sigma_n, (q_1,p_1),...,(q_n,p_n),(q,p) \in \bar{Q},$
$\sigma(q_1,...,q_n)\rightarrow q\in R,$
$p= \sigma(p_1,...,p_n)$ si $\sigma(p_1,...,p_n)\in Sub$
$\#$ sinon $\}$

Pour tout i,i≥1, $R_i=R_{i-1}\cup N_i$ avec

$N_i=\{\sigma((q_1',p_1'),...,(q_n',p_n'))\to(q,p)/\ (q_1',p_1'),...,(q_n',p_n'),(q,p)\in \bar{Q},$

$\sigma\in\Sigma_n,\ l(x_1,...,x_k)\to\sigma(y_1,...,y_n)\in S,$

$l((q_1,p_1),...,(q_k,p_k))\xrightarrow{*}_{R_{i-1}}(q,p),\ (q_1,p_1),...,(q_n,p_n)\in\bar{Q},$

Si $y_j\in X,\ y_j\in V(l)$ alors $(q_j,p_j)=(q_j',p_j')$,

Si $y_j\in T_\Sigma$ alors $(q_j',p_j')=(q_j,y_j)$ avec $y_j\xrightarrow{*}_A q_j$ }

$\cup\{\sigma((q_1',p_1'),...,(q_n',p_n'))\to(q,p)/\ (q_1',p_1'),...,(q_n',p_n'),(q,p)\in\bar{Q},$

$\sigma\in\Sigma_n,\ l(x_1,...,x_k)\to x_i\in S,$

$\sigma((q_1',p_1'),...,(q_n',p_n'))\xrightarrow{*}_{R_{i-1}}(q_i,p_i),$

$l((q_1,p_1),...,(q_k,p_k))\xrightarrow{*}_{R_{i-1}}(q,p),\ (q_1,p_1),...,(q_n,p_n)\in\bar{Q}\}.$

Comme $Qx(Sub\cup\{\#\})$ et R sont des ensembles finis il existe un entier naturel k_0 tel que $R_k = R_{k_0}$ pour tout $k \geq k_0$.

Nous pouvons maintenant définir l'automate $B = (\Sigma, Q_f, \bar{Q}_f, R_{k_0})$ tel que $\bar{Q}_f = \bar{Q}\cap(Q_f x(Sub\cup\{\#\}))$. Nous allons démontrer que L(B) = S(L).

Considérons un terme t de T_Σ et u un élément de dom(t); nous dirons que le noeud u est réécrit par utilisation d'une R_i-règle propre avec $0 \leq i \leq k_0$, si la règle appliquée en u est un élément de R_i-R_{i-1}.

Si j, $0\leq j\leq k_0$, est le maximum des i tels que des R_i-règles propres ont été appliquées lors du calcul C et si w > 0 est le nombre d'applications de R_j-règles propres lors de C alors nous dirons que C a un (j,w)-calcul.

De plus nous dirons que C est un j-calcul, si C est un (j,w)-calcul pour un w>0 donné.

Soit t un élément de L(B) et C un (j,w)-calcul de B sur t, $0 \leq j \leq k_0$, w > 0.

(i) si j = 0 alors $t\in L$

(ii) Prenons j > 0 et w ≥ 2. De par la définition de N_i nous savons qu'il existe un arbre t' tel que $t' \to_S t$ et il existe un (j,w-1)-calcul de B sur t'.

(iii) Prenons j > 0 et w = 1. De par la définition de N_i nous savons qu'il existe un arbre t' tel que $t' \to_S t$ et il existe un (j-1,w')-calcul de B sur t'.

Par conséquent il existe un arbre t_0 et un entier naturel w_0 (≥1) tels que $t_0\xrightarrow{*}_S t$ et il existe un $(0,w_0)$-calcul de B sur t_0. Nous avons $t_0 \in L$ et donc $t\in S(L)$.

Réciproquement, s'il existe un i-calcul sur t et si $t \rightarrow_S t'$ alors il existe un j-calcul sur t' avec $j \leq \sup(i+1,k_0)$. Puisque L est inclus dans L(B), S(L) est inclus dans L(B).

Nous avons donc L(B) = S(L) et, par conséquent, S(L) est un langage reconnaissable. ❑

Remarque: Ce résultat est évidemment faux si S n'est pas linéaire à droite; il suffit de considérer le système de réécriture réduit à la règle $f(x) \rightarrow g(x,x)$ et le langage reconnaissable d'arbres L= fa*$.

Nous conjecturons que le résultat reste valable dans le cas d'un système de réécriture linéaire à droite. La preuve précédente ne s'étend pas de façon immédiate au cas non linéaire à gauche, en effet, il faut dans la construction de B déterminiser les automates définis à chaque étape, ce qui complique singulièrement la preuve de terminaison de la construction.

C.2 Automates à piles d'arbres et langages algébriques.

Soit $A=(\Sigma,\Gamma,Q,Q_f,R)$ un automate à piles, on peut définir le langage reconnu par A par états finaux ou par pile vide.

$L_f(A) = \{ t \in T_\Sigma / t \xrightarrow{*}_A q(t') \text{ et } t \in Q_f \}$ est le langage reconnu par états finaux de A.

$L_e(A)=\{ t \in T_\Sigma / t \xrightarrow{*}_A q(\perp) \}$ où $\perp$ est le symbole de pile vide, symbole distingué de Γ.

On peut montrer que ces deux définitions sont équivalentes pour les classes considérées. Nous avons montré (théorème 1, section B.2) que toute machine de Turing pouvait être simulée par un automate à piles de la classe $TPDA_{1,2}$, on montrerait facilement le même résultat pour la classe $TPDA_{2,1}$.

Considérons maintenant la classe $TPDA^{rr}$. K.M.Schimpf et J.H.Gallier ont défini une sous-classe de $TPDA^{rr}$ associée à la classe des langages algébriques ([GASC85], [SCHI82]). Le principe de la construction est le suivant: l'automate possède un seul état q, le système de réécriture R est obtenu de la façon suivante :

A toute règle $l \rightarrow r$ de la grammaire algébrique G, avec $l=B(x_1,...,x_n)$ où B appartenant à V est d'arité n, et, r est un terme de $T_{\Sigma \cup V}(X_n)$, on associe la règle de réécriture monadique $q(r) \rightarrow q(l)$.

Notons que pour la règle $q(r) \rightarrow q(l)$ ainsi définie, l'ensemble des variables du membre droit n'est pas nécessairement inclus dans l'ensemble des variables du membre gauche. Nous avons toujours supposé cette condition vérifiée, donc, par le même raisonnement, nous obtiendrions le même résultat mais uniquement en nous limitant au cas complet (toute variable apparaissant dans l apparaît dans r) et strict (r n'est pas réduit à une variable). Ce cas correspond à un algorithme non déterministe linéaire d'analyse.

Une classe intéressante à étudier, du point de vue théorie des langages d'arbres, est la classe $TPDA_{11}$ des automates à piles d'arbres dont les états ont pour arité 1 et tels que la profondeur des arbres de piles dépilés est au maximum 1.

Nous conjecturons que la classe des langages reconnus est une sous-classe des langages algébriques et que dans cette classe, un arbre d'entrée peut être analysé en temps linéaire.

BIBLIOGRAPHIE

[ARDA76]: A. Arnold and M. Dauchet, Un théorème de duplication pour les forêts algébriques, *J. Computer Systems Science*, 13, pp 223-244, (1976).

[ARDA78]: A. Arnold and M. Dauchet, Forêts algébriques et homomorphismes inverses, *Inform. and Control*, 37, pp182-196, (1978).

[BACH88]: L.Bachmair, Proof by consistency in equational theories, in *Proc. 3rd IEEE Symp. Logic in Computer Science*, (1988).

[BADE86]: L.Bachmair and N.Dershowitz, Commutation, transformation and termination, *Lec. Notes in Comp. Sci.*, 230, (1986).

[BADE87]: L.Bachmair and N.Dershowitz, Completion for rewriting modulo a congruence, *Lec. Notes in Comp. Sci.*, 256, pp 192-203, (1987).

[BERS79]: J.Berstel, Transductions and context free languages, in *Teubnen Studienbücher Informatik*, (1979).

[BJW81]: R.V. Book, M. Jantzen and C. Wrathall, Monadic Thue Systems, *Theoret. Comput. Sci. 19* , 3, pp 231-252, (1981).

[BOCK87]: A.Bockmayr, A note on a canonical theory with undecidable unification and matching problem, *Journal of Automated reasoning*, 3, pp 379-381, (1987).

[BOGA85]: R.V. Book and J.H. Gallier, Reductions in Tree Replacement Systems, *Theoret. Comput. Sci.*, 37, pp 123-150, (1985).

[BOGA90]: B.Bogaert, Automates à tests d'égalités, Thèse de l'université de Lille, (1990).

[BOOK83]: R.V. Book, Decidable sentences of Church-Rosser congruences, *Theoret. Comput. Sci.*, 24, pp 301-312, (1983).

[BOOK87]: R.V. Book, Thue systems as Rewriting Systems, *J. of Symbolic Computation*, 3, pp 39-68, (1987).

[BRAI69]: W.S. Brainerd, Tree generating regular systems, *Inf. and Control*, 14 , pp 217-231, (1969).

[CDGV90]: J.L.Coquidé, M.Dauchet, R.Gilleron and S.Vagvölgyi, Tree pushdown automata and Rewrite Systems, submitted paper, *Technical Report I.T.190*, Université Lille, (1990).

[CDT89]: J.L. Coquidé, M.Dauchet and S.Tison, About connections between syntactical and computational complexity, FCT'89, *Lec.Notes.comp.Sci 380*, (1989).

[COGI90]: J.L.Coquidé and R.Gilleron, Proofs and Reachability problems for Rewrite Systems, IMYCS 90, to appear in *Lec.Notes.Comp.Sci*, (1990).

[COMO88]: H. Comon, Unification et désunification: Théorie et Applications, Thèse Université de Grenoble, (1988).

[COMO89]: H. Comon, Inductive Proofs by Specification Transformations, Rewriting Technics and Applications, *Lec. Notes Comp. Sci.*, 355, (1989).

[COMO90]: H. Comon, Equational formulas in order-sorted algebras,in *Proc. ICALP 90*, Springer-Verlag, (1990).

[COUR89]: B. Courcelle,On recognizable sets and tree automata, Resolution of Equations in Algebraic Structures, *Academic Press*, M.Nivat & H. Ait-Kaci edts, (1989).

[DAUC89]: M.Dauchet, Simulation of Turing machines by a left-linear rewrite-rule, Proc 3rd R.T.A, *Lect. Notes Comput. Sci. 355*, (1989).

[DADE89]: M. Dauchet and F. De Comite, A gap between linear and non linear term rewriting systems, *Lec. Notes Comp. Sci.*, 256, (1987).

[DADE89]: M. Dauchet and A. Deruyver, "VALERIANN":Compilation of Ground Term Rewriting Systems and Applications, Proc 3rd R.T.A, *Lect. Notes Comput. Sci. 355*, (1989).

[DATI85]: M. Dauchet and S.Tison, Decidability of Confluence in Ground Term Rewriting Systems, F.C.T 85, *Lec. Notes Comp. Sci.*, 199, (1985).

[DATI90]: M.Dauchet and S. Tison, The theory of Ground Rewrite System is Decidable, in *Proc.5th I.E.E.E symp. on Logic in Computer Science*, (1990).

[DECO84]: F. De Comite, Simulation linéaire des systèmes de réécriture, Thèse Université de Lille I,(1984).

[DEGI89]: A. Deruyver and R. Gilleron, The reachability problem for ground rewrite systems and some extensions, CAAP 89, *Lec. Notes. Comp. Sci*, 351, (1989).

[DEJO90]: N.Dershowitz and J.P. Jouannaud, Rewrite systems, Handbook of Theoretical Computer Science, J.V.Leeuwen editor, North-Holland,.(1990).

[DERS87]: N.Dershowitz, Termination of Rewriting, J. Symbolic Computation 3,(1987).

[DERS89]: N.Dershowitz, Completion and its applications, in Ait-Kaci, M.Nivat eds, *Resolution of Equations in Algebraic Structures*, Vol II: Rewriting Techniques, pp 31-86, (1989).

[DEVI90]: P.Devienne, Weighted graphs, a tool for studying the halting problem and time complexity in logic programming, TCS 75, (1990).

[ENGE75]: J. Engelfriet, Bottom-up and Top-down Tree Transformations, a Comparison, *Math. Systems Theory*, 9, (1975).

[FAGE84]: F.Fages, Le système KB: manuel de référence: présentation et bibliographie, mise en oeuvre, Report R.G.10.84, Greco de Programmation, Bordeaux, (1984).

[FUVA89]: Z. Fülöp and S. Vàgvölgyi, Ground Term Rewriting rules for the Word Problem of Ground Term Equations, submitted paper, (1989).

[FUVA90]: Z. Fülöp and S. Vàgvölgyi, A characterization of irreducible sets modulo left-linear rewriting systems by tree automata, *Fundamenta Informaticae XIII*, (1990).

[GASC85]: J.H. Gallier and K.M. Schimpf, Parsing Tree Languages using Bottom-up Tree Automata with Tree Pushdown Stores, *J. Computer Science*, (1985).

[GEST84]: F. Gecseg and M. Steinby, Tree automata, Akademiai Kiado, (1984).

[GILL90]: R.Gilleron, Reconnaissabilité et fragments décidables en réécriture,Automates à piles d'arbres et calcul de formes normales, Thèse Université de Lille I,(1990).

[GISP68]: S. Ginsburg and E.H. Spanier, Control sets on grammar, *Math. Systems Theory*, 2, (1968).

[GOGU80]:J.A.Goguen, How to prove inductive hypothesis without induction, *Lec. Notes in Comp. Sci.*, 87, (1980).

[GREI77]: S. Greibach, Control sets on context-free grammar forms, *J. of Comp. and Syst. Sciences*, 15, (1977).

[GTWW77]: J. Goguen, J.W. Thatcher, E. Wagner and E. Wright, An initial algebra approach to the specification correctness, and implementation of abstract data types, in: *Current Trends in Programming Methodology* Vol. 4 (Prentice-Hall, Englewood Cliffs, NJ), pp 80-149, (1977).

[GUES83]: I. Guessarian, Pushdown Tree Automata, *Math. Systems Theory*, 16, (1983).

[GUKAPU81]: J.V. Guttag, D.Kapur and D.R. Musser, On proving uniform termination and restricted termination of rewriting systems, Technical report n°81 CRD 272, General Electric Company, (1981).

[HUET80]: G. Huet, Confluent Reductions: Abstract properties and applications to Term Rewriting Systems, *J.A.C.M.*, 27, (1980).

[HUHU82]: G.Huet and J.M.Hullot, Proofs by induction in equational theories with constructors, *J. of Comp. and Syst. Sciences*, 25, (1982).

[HULA78]: 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).

[HULE79]: G.Huet and J-J. Levy, Call by need Computation in non-ambiguous linear Term Rewriting Systems, *TR359,* I.N.R.I.A., Le Chesnay, France, (1979).

[HUOP80]: G.Huet and D.C. Oppen, Equations and Rewrite Rules: A survey, in R.V.Book, ed., New York, *Academic Press*, Formal Language Theory: Perspectives and Open Problems, (1980).

[JANT88]: M.Jantzen, Confluent string rewriting, EATCS Monographs on Theoretical Computer Science 14, Springer Verlag, (1988).

[JOKI86]: J.P.Jouannaud and H.Kirchner, Completion of a set of rules modulo a set of equations, *SIAM J. Comput.* 15, (1986).

[JOKO86]:J.P.Jouannaud and E.Kounalis, Automatic proofs by induction in equational theories without constructors, in *Proc. 1st IEEE Symp. on Logic in Computer Science*, (1986).

[JOUA83]:J.P. Jouannaud, Confluent and coherent equational term rewriting systems, application to proofs in abstract data types, *Lec. Notes in Comp. Sci.*, 159, (1983).

[KANA85]: D.Kapur and P.Narendran, An equational approach to theorem proving in first-order predicate calculus, in *Proc. of the 9th International conference on Artificial Intelligence*, pp 1146-1153, (1985).

[KAZH88]: D.Kapur and H.Zhang, An overview of Rewrite Rule Laboratory (RRL), in *Proc of the 8th Conference on Automated deduction*, pp 559-563, (1988).

[KIRC85]: C.Kirchner, Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles, Thèse de Doctorat d'Etat, Nancy, (1985).

[KIRC85]: H.Kirchner, Preuves par complétion dans les variétés d'algèbres, Thèse de Doctorat d'Etat, Nancy, (1985).

[KNBE70]: D.E.Knuth and P.B.Bendix, Simple word problems in universal algebras, *Computational Problems in Abstract Algebra*, pp263-297, (1970).

[KNO90]: D.Kapur, P.Narendran and F.Otto, On ground confluence of term rewriting systems, *Information and Computation* 86, pp 14-31, (1990).

[[KOZE77]: D.Kozen, Complexity of finitely presented algebras, 9th ACM Symposium on theory of computing, Boulder, Colorado, pp 164-177, (1977).

[KRUS60]: J.B.Kruskal, Well-quasi-Ordering, the Tree Theorem and Vazsonyi's conjecture, *Transactions of the American Mathematical Society*, 95, pp 210-225, (1960).

[LABA77]: D.S.Lankford and A.M.Ballantyne, Decision procedures for simple equational theories with permutative axioms: complete sets of permutative reductions, *ATP37*, Dept. of Mathematics and Computer Science, Austin, Texas, (1977).

[LABA77]: D.S.Lankford and A.M.Ballantyne, Decision procedures for simple equational theories with commutative-associative axioms: complete sets of commutative-associative reductions, *ATP39*, Dept. of Mathematics and Computer Science, Austin, Texas, (1977).

[LEBE88]: P.Lebègue, Towards the study of logic programming with weighted graphs, Thèse de l'université de Lille, (1988).

[LEGU80]: B.Leguy, Reductions, transformations et classification des grammaires algébriques d'arbres, Thèse de l'université de Lille, (1980).

[LESC83]:P.Lescanne, Computer experiments with the REVE term rewriting system generator, in *Proc. of the 10th ACM Symp. on Principles of Programming Languages*, pp 99-108, (1983).

[MAHE88]: M.J.Maher, Complete axiomatizations of the algebras of finite, rational and infinite trees, *Proc 3rd IEEE Symp. Logic in Computer Science*, (1988).

[MALC71]: A.I.Mal'cev, Axiomatizable classes of locally free algebras of various types, the *Metamathematics of Algebraic systems*, North-Holland, (1971).

[MUSS80]: D.Musser, Proving inductive properties of abstract data types, in Proc. 7th ACM Symp. Principles of Programming Languages, (1980).

[NAOT90]: P.Narendran and F.Otto, Some results on equational unification, Lec. Notes in Comp. Sci., 449, pp 276-291, (1990).

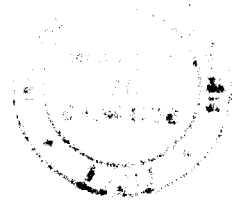
[NEOP80]: G. Nelson and D.C. Oppen, Fast Decision Procedures Based on Congruence Closure, *J.A.C.M.*, 27, (1980).

[NEWM42]: M.H.A.Newman, On theories with a combinatorial definition of equivalence, *Annals of Mathematics*, 43, pp 223-243, (1942).

[OYAM86]: M. Oyamaguchi, The reachability Problem for Quasi-ground Term Rewriting Systems, *Journal of Information Processing*, 9-4, (1986).

[OYAM87]: M.Oyamaguchi, The Church-Rosser property for ground term rewriting systems is decidable, *TCS* 49, pp43-79, (1987).

[OYAM90]: M. Oyamaguchi, The reachability and joinability Problems for right-ground Term Rewriting Systems, to appear in *J. Inf. Process*, (1990).



[OYAM90]: M.Oyamaguchi, On the word problem for right-ground term rewriting systems, *Trans of the IEICE*, E73, pp718-723, (1990).

[PEST81]: G.Peterson and M.Stickel, Complete sets of reductions for some equational theories, *J. Assoc. Comput. Mach.* 28, (1981).

[PLAI85]: D.A. Plaisted, Semantic Confluence Tests and Completion Methods, *Information and Control*, 65, (1985).

[PLOT72]: G.Plotkin, Building in equational theories, *Machine Intelligence*, 7, pp 73-90, (1972).

[RABI69]: M.O.Rabin, Decidability of second order theories and automata on infinite trees, *Amer. Math. Soc.*, (1969).

[RABI77]: M.O.Rabin, Decidable theories, *Handbook of Mathematical Logic*, North Holland eds, pp 595-627, (1977).

[RAOU81]: J-C. Raoult, Finiteness results on rewriting systems, *R.A.I.R.O.* Theoretical Informatics 15, (1981).

[RAVU80]: J-C. Raoult and J. Vuillemin, Operational and Semantic Equivalence between Recursive Programs, *J.A.C.M.* , 27, (1980).

[ROSE73]: B.K. Rosen, Tree-manipulating Systems and Church Rosser Theorems, *J.A.C.M.*, 20, (1973).

[SALO88]: K. Salomaa, Deterministic Tree Pushdown Automata and Monadic Tree Rewriting Systems, *Journal of Comput. and Syst. Sci.*, 37, (1988).

[SCHI82]: K.M. Schimpf, A Parsing Method for Context-free Tree Languages, Ph.D., Univ. of Pennsylvania, (1982).

[STIC86]: M.E.Stickel, The Klaus automated deduction system, in *Proc. of the 8th International Conference on Automated deduction*, pp 703-704, (1986).

[THAT67]: J.W.Thatcher, Characterizing derivation trees of context-free grammars through a generalization of finite automata theory, J. Comput. System Sci., 1, (1967).

[THOM90]: W.Thomas, Automata on infinite objects, to appear in Handbook of Theoretical and Computer Science, (1990).

[TISO89]: S. Tison, The Fair Termination is decidable for Ground Systems, Rewriting Technics and Applications, Chapel Hill, *Lec. Notes Comp. Sci.*, 355, (1989).

[ZHRE85]: H. Zhang and J.L. Remy, Contextual rewriting, *Lec. Notes Comp. Sci.*, 202, (1985).

Titre : Contrôles et preuves dans les systèmes clos.
Automates à piles d'arbres et calcul de formes normales.

Résumé : La réécriture est, à plus d'un titre, un paradigme bien connu de la programmation. Notre but est de contribuer à mieux comprendre les structures algébriques complexes manipulées concernant les arbres et la réécriture dans les arbres.

Nous montrons que les systèmes clos, munis d'un contrôle reconnaissable, ont la même puissance que les systèmes de réécriture généraux. On peut sans difficulté simuler une machine de Turing par un système de réécriture clos à contrôle clos.

Nous prouvons que les ensembles de preuves de dérivation dans les problèmes d'accessibilité pour les systèmes clos peuvent être vus comme des forêts reconnaissables. Nous donnons un algorithme en temps linéaire fournissant un arbre de dérivation pour le problème d'accessibilité.

Nous introduisons une notion qui permet d'éclairer et de généraliser les études antérieures sur les liens entre systèmes de réécriture et automates à piles.

MOTS-CLES : Systèmes clos. Langage reconnaissable. Système contrôlé. Problème d'accessibilité. Automate à piles d'arbres. Système semi-monadique.