Numéro d'ordre : 661

USTL FLANDRES ARTOIS — LIFL U.A 369 du C.N.R.S

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

Thèse

présentée à

L'Université des Sciences et Techniques de Lille Flandres-Artois

pour l'obtention du titre de

Docteur en Informatique

par

Bruno Bogaert

Automates d'arbres avec tests d'égalités

Soutenue le 21 Décembre 1990 devant la commission d'examen :

Irène Guessarian,	**Présidente.**
André Arnold,	
Max Dauchet,	**Rapporteurs.**
Hubert Comon,	
Philippe Devienne,	
Michel Latteux,	
Sophie Tison,	**Examinateurs.**

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 Francis	Systèmes électroniques
M. PARSY Fernand	Mécanique

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

J'exprime toute ma gratitude à Irène Guessarian qui a bien voulu me faire l'honneur de présider le jury de cette thèse.

Je remercie André Arnold qui a accepté la tâche d'être l'un des rapporteurs. Il m'a communiqué des remarques précises et nombreuses qui m'ont aidé dans l'achèvement de ce document.

Au sein du LIFL, Max Dauchet transmet son enthousiasme pour la recherche. Il est aujourd'hui rapporteur de ce travail, j'en suis heureux et l'en remercie vivement.

Mes remerciements vont également à Hubert Comon, qui a accepté de s'intéressser à cette thèse et de participer au jury, à Philippe Devienne, lecteur attentif de ce travail, et à Michel Latteux, dont j'ai pu apprécier les encouragements.

Ce travail doit énormément à Sophie Tison, qui en a suivi la réalisation en prodiguant de nombreux conseils. Ses indications furent indispensables et sa patience remarquable.

Je tiens également à remercier l'ensemble de mes collègues du LIFL, et parmi eux Henri Glanc, qui a assuré la reproduction du mémoire, Anne-Cécile Caron, Jean-Luc Coquidé, Francesco De Comité, et Rémi Gilleron qui partagent avec moi les 25 m^2 du bureau 332.

Introduction

Dans cette étude, nous proposons une extension des reconnaisseurs d'arbres, en incluant la possibilité de tester si deux arbres frères sont égaux. Cette nouvelle définition d'automates permet de manipuler des termes non-linéaires, en imposant des égalités ou des différences parmi les sous-termes. On distingue principalement deux définitions d'automates, ou plus exactement deux stratégies dans l'application des règles de dérivation. Nous nous intéressons aux deux classes d'ensembles d'arbres obtenues et établissons leurs principales propriétés.

N'allons pas plus avant sans présenter d'abord les notions de base (arbres, forêts, reconnaissabilité, morphismes, non-linéarité) puis le cadre dans lequel nous nous situons, les résultats déja connus, les propriétés de décision qui nous intéressent. Ce sera l'occasion de justifier les choix que nous avons faits, la raison pour laquelle les tests interviennent entre arbres frères, et de présenter quelques autres voies possibles.

Un arbre est une structure assez naturelle, d'ailleurs employée très communément dans des domaines divers pour représenter des informations. (arbre généalogique, organisation administrative, énumération de possibilités, calcul à effectuer, ...). C'est une structure néanmoins assez puissante, essentielle dans tous les aspects de l'informatique. Un arbre est donc un objet que l'on peut fabriquer à partir de "matériaux de base" qui sont , d'une part des valeurs -ou des objets qu'il est impossible de décomposer-, et d'autre part des opérateurs -ou encore des objets réclamant plusieurs composantes-. Ces notions de valeur et d' opérateur sont rassemblées en une seule : celle de lettre graduée, que l'on appelle encore lettre avec arité. L'arité indique le nombre de successeurs. Une valeur est donc une lettre d'arité nulle, alors que l'opérateur possède plusieurs successeurs.

L'opération de construction d'un arbre s'appelle concaténation. Elle consiste à prendre deux termes, et à considérer l'un comme successeur de l'autre, à "greffer" un arbre sous un autre. L'arité désigne le nombre d'objets que l'on peut concaténer sous un noeud.

Sont appelés termes clos les arbres dont toutes les branches se terminent par un terme

de arité nulle (on ne peut plus rien leur ajouter). Quand certaines branches sont non achevées, on leur associe un nom de variable. La concaténation revient donc à faire correspondre un terme à une variable.
Un terme est dit linéaire si chaque variable apparaît sur au plus une branche. Concrètement cela signifie que l'on peut concaténer "n'importe quoi, n'importe où" (sous réserve, bien-sûr, de respecter les arités). Si au contraire une même variable figure plusieurs fois, le terme est non-linéaire, la concaténation fera apparaître nécessairement plusieurs sous-arbres identiques.

Le vocabulaire que nous employons est hérité des arbres rencontrés communément : des arbres généalogiques nous avons gardé les termes de père et de fils pour désigner des lettres concaténées; à cause des arbres au sens botanique, nous parlons de feuille pour désigner les lettres de arité 0 et de forêt pour des ensembles d'arbres. C'est donc un vocabulaire hybride (venant à la fois de la botanique et de la généalogie, c'est normal...).

Les arbres que nous avons cités sont finis, mais la définition peut être étendue à des arbres infinis, c'est à dire possédant au moins une branche infinie. Les arbres auxquels nous nous intéressons ici sont tous finis.

D'un point-de-vue plus mathématique, l'ensemble des arbres clos formés sur un alphabet gradué fini (ensemble de lettres graduées) forme une algèbre. C'est une algèbre initiale.
Si une forêt est un ensemble d'arbres, une classe de forêts est un ensemble de forêts qui se distinguent par des propriétés communes.

La classe la plus étudiée est celle des forêts reconnaissables (ou encore rationnelles), nommée REC. Une forêt rationnelle peut être désignée par une expression rationnelle d'arbres. Elle peut aussi être reconnue par un automate d'arbres, c'est à dire que l'on dispose d'une classe simple d'algorithmes permettant de décider si un arbre donné appartient ou non à la forêt. Les automates d'arbres peuvent être ascendants ou descendants, suivant le sens de la reconnaissance (à partir des feuilles ou de la racine).

Comme exemple de forêt reconnaissable, citons notamment les arbres de dérivation d'une grammaire algébrique (de mots, cette fois).
Si l'ensemble des arbres est une algèbre, la classe des forêts reconnaissables est, elle, à rapprocher de celle d' "algèbre sortée". On peut coder le typage d'expressions par un automate ascendant d'arbres.

Nous avons présenté la non-linéarité en disant qu'une même variable figurait plusieurs fois sur les branches d'un terme non clos. On dit encore que la torsion qui est associée est non-injective. Par concaténation avec des termes non-linéaires, on obtient des arbres contenant des sous-arbres égaux.

La non-linéarité est rencontrée fréquemment en programmation logique. Quand intervient la non-linéarité, il y a souvent perte de "bonnes propriétés". Abordée sous l'angle des systèmes de réécriture, l'existence d'un ancêtre commun est polynomiale dans le cas d'un système linéaire gauche, clos à droite, elle est indécidable dans le cas général (Oyamagushi); l'inductive-réductibilité* est polynomiale dans le cas des systèmes de réécritures linéaire à gauche (J.P. Jouannaud, E. Kounalis), mais exponentielle dans le cas général (D. Plaisted, D. Kapur).

On peut aborder la non-linéarité en se donnant des outils de manipulation des égalités de termes, qui constituent un "symptôme" de la non-linéarité. Des outils ont ainsi été présentés par M.J. Mahler, ou par H. Comon, qui étudient l'axiomatisation des algèbres d'arbres et proposent des résultats de décision.

Pour notre part, nous nous sommes dotés d'un outil prenant en compte directement les égalités, en introduisant dans les règles de transition des automates ascendants, les tests entre arbres frères.
Se posent alors deux questions : pourquoi utiliser des automates, et pourquoi faire des tests entre frères ?

Les automates sont des outils assez simples, mais qui ont permis de prouver des résultats assez complexes de façon claire. Ils sont notamment intéressants pour peu que les familles qu'ils engendrent soient closes par opérations booléennes et que l'on dispose d'un algorithme de décision du vide. Nous pensons là, par exemple, à la décision de l'inductive-réductibilité qui peut se ramener à décider si une forêt est vide, après avoir fait quelques opérations booléennes, dont un passage au complémentaire.

Ce qui nous amène à répondre à la deuxième question : pourquoi l'égalité entre frères seulement ? Il s'agissait avant tout de préserver les propriétés que nous venons de citer, la clôture booléenne, la décidabilité du vide. Or nous disposons en cela de plusieurs repères. L'idée d'ajouter des tests d'égalités de sous-termes dans des automates a été introduite par M. Dauchet et J. Mongy en 1981. Les tests portaient alors sur des termes "cousins", chaque règle ne comparant que des cousins "pas trop

---

* *Etant donné un système de réécriture, un terme est inductivement-réductible si toutes ses instances closes sont réductibles*

éloignés", situés à une profondeur bornée en-dessous de la lettre courante. Dans la classe ainsi obtenue, appelée RATEG, la propriété du vide est indécidable. Si chaque règle ne compare que des cousins à profondeur bornée, on en arrive pourtant à lier entre-eux des sous arbres arbitrairement éloignés, par des comparaisons de proche en proche. Ce phénomème de "chevauchement d'égalités" permet, par exemple, de coder le problème de Post (chaque niveau de l'arbre supporte deux séquences de mots, les égalités permettent de vérifier la cohérence entre deux niveaux successifs -les mêmes séquences s'allongent-, en haut de l'arbre l'égalité est testée).

Notons aussi, que ce chevauchement a permis de montrer la puissance des morphismes non-linéaires : l'image par morphismes non-linéaires et opérations booléennes des reconnaissables est l'ensemble des récursivement énumérables, alors qu'en prenant des morphismes linéaires on reste dans les reconnaissables. J.Mongy a ainsi établi qu'il suffit de très peu d'opérations pour engendrer la totalité de la classe des récursivement énumérables. Toute forêt F (r.e.) s'exprime comme $F = \pi(\phi_1(R_1) \cap \phi_2(R_2))$, où ϕ_1 et ϕ_2 sont des morphismes complets. La même technique de propagation d'égalités permet de simuler des dérivations de grammaires d'arbres.

Il convenait donc d'empêcher absolument ce chevauchement. A ces fins, deux voies, au moins, sont possibles :

- On peut restreindre le nombre de tests, tout en comparant des sous-termes cousins. On autorise alors des imbrications d'égalités, mais de façon bornée.
- Une autre voie (c'est celle que nous avons suivie ici) consiste à comparer seulement les fils de la lettre courante. De cette manière chaque arbre n'est comparé qu'à des arbres du même niveau que lui, et le chevauchement non borné est évité.

Notons que W. Thomas suggérait récemment de modifier la nature du test et de regarder si un fils est sous-arbre d'un autre fils. Cela pourrait constituer encore une autre voie.

Dans le premier chapitre, nous donnons la définition formelle des arbres et des automates ascendants d'arbres. Quelques exemples de forêts reconnaissables y sont décrits, ainsi que deux exemples de forêts RATEG : l'un, simple, permet de se familiariser avec le mécanisme de ces automates. Le deuxième exemple est le codage du problème de Post, afin d'illustrer la relation entre recouvrement de conditions et indécidabilité.
Les automates que nous définissons peuvent être présentés sous deux formes : l'une

"sympathique", l'autre plus technique. Dans ce premier chapitre nous donnons des exemples, introduits d'abord sous leur forme "sympathique", et nous définissons ensuite les descriptions, terme plus technique mais qui nous servira tout au long des preuves.
Pour exprimer une condition d'égalité entre éléments, l'idée la plus naturelle est de construire une formule dont le prédicat est celui de l'égalité de termes. Mais cette simplicité d'expression peut être interprétée de différentes manières; si on se donne deux règles :

$\begin{cases} b(q,q) \to q' \text{ quand les deux sous-termes sont égaux.} \\ b(q,q) \to q \end{cases}$

Il peut y avoir ambiguïté sur la deuxième règle : s'applique-t-elle aussi quand les termes #1 et #2 sont égaux, ou est-elle réservée au cas où $\#1 \neq \#2$?
Tranchons, et considérons que la règle 2 peut s'appliquer même en cas d'égalité des sous-arbres; pour imposer une différence il faudra alors l'exprimer explicitement.

Nous proposons alors deux types d'automates :

1) les conditions portent uniquement sur des égalités (les formules ne connaissent pas le prédicat "non"). Nous avons alors affaire à des automates capables d'imposer des égalités, mais pas des différences.

2) les formules peuvent utiliser la négation. Il est alors possible d'imposer une différence entre sous-arbres.

Dans le premier cas, on reconnaîtra par exemple des forêts comme :

- Les arbres binaires bien équilibrés, (avec une seule étiquette sur les noeuds de l'arbre).
- Les arbres tels qu'il existe au moins deux "b" en-dessous desquels les sous-arbres sont égaux.

Par contre il sera impossible de reconnaître :

- Les arbres binaires non équilibrés.
- Les arbres binaires équilibrés, mais dont les noeuds peuvent porter des étiquettes variées.

L'égalité "de forme", n'est obtenue que dans le cas où les étiquettes sont identiques et où une égalité de forme est synonyme d'une égalité de termes.
Dans la reconnaissance d'un arbre, il est toujours possible de ne pas tenir compte d'une égalité, donc aucune différence d'arbres ne peut être imposée.

Il y a, dans ce cas, coïncidence avec les termes engendrés par des algèbres à signature non-linéaire, plus habituellement représentées par des DAG.

Dans le deuxième type d'automates, il est possible d'imposer une différence entre sous-arbres. On pourra, cette fois, reconnaître les arbres binaires déséquilibrés, à une seule étiquette; par contre, bien sûr, toujours pas les arbres équilibrés mais portant des étiquettes diverses.

On sort alors des algèbres à signature non-linéaire.

La suite du chapitre 1 est consacrée à la présentation des descriptions d'égalité. Les descriptions d'égalité constituent un système fini et "normalisé" pour exprimer toutes les conditions d'égalité existant à l'intérieur d'un n-uple d'éléments.
Cette notion "technique" permettra de remplacer les formules telles que celles que nous venons de voir et, en outre, de donner une syntaxe unique pour les deux types d'automates. Par contre, au niveau sémantique, nous introduirons deux comportements (deux façons d'appliquer les règles) pour cette syntaxe unique.

Le chapitre 2 est consacré à l'étude des forêts reconnues par notre première classe d'automate. Nous les définissons comme étant des automates à descriptions d'égalités, munis d'une stratégie à contrôle faible. Le contrôle faible consiste à ne pas imposer la reconnaissance d'une égalité, d'où la coïncidence avec le cas où les conditions sont exprimées par des formules ne connaissant pas la négation.
La classe de forêts obtenue étant ressemblante à REC, mais comportant, en plus, des égalités entre sous-arbres, nous avons utilisé le nom de $REC_{=}$
Cette classe de forêts est stable par les opérations d'union et d'intersection. Elle ne l'est pas par complémentation, l'exemple des arbres équilibrés exposé un peu plus haut, en permet la preuve. Il n'est d'ailleurs pas possible de parler de déterminisme, puisque le type même de stratégie consiste à laisser le choix entre reconnaître une égalité ou ne pas la reconnaître.
Nous montrons que la classe reconnue coïncide avec la clôture de REC par morphisme alphabétique.

Dans la 3ème partie, nous étudions les automates munis d'une stratégie de contrôle

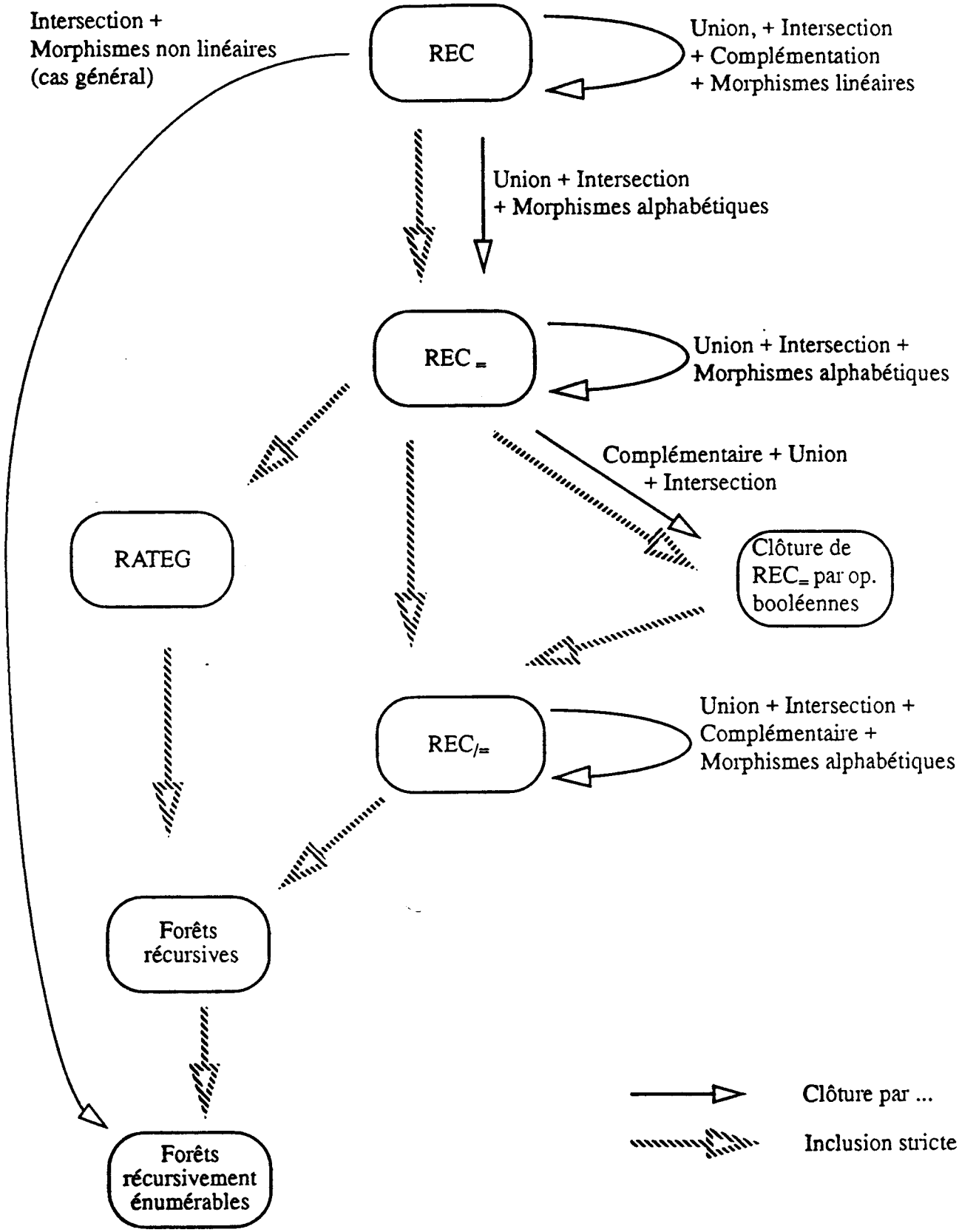
fort : on impose de reconnaître toutes les égalités de sous-arbres. Par défaut, certaines règles ne sont donc accessibles qu'en cas de différence
C'est une stratégie qui permettra de définir des automates réellement déterministes, et donc de prouver que la classe (nommée $REC_{\neq}$) est fermée pour la complémentation. Elle contient strictement la classe $REC_{=}$ et elle est close par union et intersection.
Afin d'établir des résultats de décision, il sera nécessaire de compter le nombre d'arbres atteignant un état. Il ne suffit pas que tous les états en partie gauche d'une règle soient accessibles pour pouvoir décider si il existe un arbre "transitant" par cette règle. Il faut qu'il y en ait suffisamment pour satisfaire éventuellement une condition de différence. Nous donnons donc des outils d'évaluation du nombre d'arbres atteignant chaque état, ce qui nous permet de construire un algorithme de décision du vide.
Ce résultat établi, nous donnons un exemple d'application sur un problème d'inductive-réductibilité.
L'algorithme de décision du vide est de complexité polynomiale pour un automate déterministe. Dans le cas général des automates à tests non déterministes la propriété "du vide" est NP-dure.

Dans le dernier chapitre, nous nous attachons à caractériser les différentes classes de forêts que nous avons évoquées. Nous définissons une relation sur les états de l'automate. Intuitivement, quand deux états sont en relation, c'est que l'un peut être obtenu à la place de l'autre par un arbre qui présente "quelques égalités de plus". Cette relation nous permet d'établir des conditions suffisantes pour les automates à stratégie forte reconnaissant des éléments de REC , de $REC_{=}$ et de la clôture booléenne de $REC_{=}$. Il est ensuite possible de montrer que $REC_{\neq}$ n'est pas la simple clôture booléenne de $REC_{=}$, et qu'il existe encore un fossé entre ces deux classes.

Nous avons montré que les propriétés de décision du vide ou de la finitude sont décidables, il faut aussi s'interroger sur les propriétés d'appartenance à la classe REC ou à la classe $REC_{=}$, (c'est-à-dire l'existence d'un automate classique équivalent, ou celle d'un automate s'exprimant sans différences), qui le sont peut-être aussi. Les caractérisations définies au chapitre 4 peuvent constituer un moyen (entre autres) d'aborder ces problèmes.

Il serait également intéressant d'étudier si la propriété du vide reste décidable dans le cas où l'on s'autorise à comparer des arbres situés plus profondément, mais à une même profondeur (des termes cousins, en quelque sorte). Pour coder le problème de

Post dans les Rateg, on utilise des règles avec des comparaisons entre des arbres à des profondeurs différentes, de même que pour obtenir la classe des récursivement énumérables à partir des reconnaissables, J. Mongy utilise des morphismes dupliquant des variables sur deux niveaux distincts. Ces techniques permettent ainsi d'engendrer des contraintes sur toute la hauteur d'un arbre, quelle qu'elle soit. Si l'on se borne à interdire les comparaisons entre des termes dont les positions seraient à des hauteurs distinctes, peut-être la décision du vide est-elle préservée.



Relations entre quelques classes de forêts

Chapitre 1

Généralités

1. 1 Alphabet gradué, arbres

On appelle **alphabet gradué**, un ensemble de couples (symbole, arité), l'arité étant un entier positif ou nul.

Exemple

$\Sigma = \{a, b, c, +, -, -\}$
$\quad\quad 0\ 0\ 0\ 2\ 2\ 1$

Nous ne nous intéresserons qu'aux alphabets gradués finis.

Σ_i désigne le sous-ensemble de Σ constitué par les termes de arité i.

Les arbres sont formés par concaténation des éléments de Σ, et éventuellement de variables (ou possèdent des branches non terminées, associées à une torsion).

$\mathbf{T}(\Sigma)_n$ désigne l'ensemble des arbres "bien formés" sur l'alphabet Σ et n variables $x_1, \ldots, x_n$. **exemple :** $+(a, -(x_1, +(x_1, -(x_2))))$ est un arbre de $T(\Sigma)_2$

$\mathbf{T}(\Sigma)_0$ désigne donc l'ensemble des arbres sans variable, encore appelés termes clos. La **hauteur** d'un arbre est la longueur de la plus grande branche. La hauteur peut être définie récursivement par :
$h(t) = 0$ si $t \in \Sigma_0$
$h(a(t_1, \ldots, t_n)) = 1 + max\{h(t_i) | i \leq n\}$

1. 2 Automates ascendants d'arbres

Un automate ascendant d'arbres est défini par la donnée d'un quadruplet (Σ, Q, F, R) dans lequel :

Σ est un alphabet gradué fini.

Q est un ensemble fini d'états, pouvant être considérés comme des symboles de arité 1.

$F \subseteq Q$ est un ensemble d'états terminaux.

$R \subseteq \bigcup_i \Sigma_i \times Q^{i+1}$, un ensemble de règles.

Notation : la règle $(a, q_1, \ldots, q_n, q)$ sera presque toujours notée $(a(q_1, \ldots, q_n) \rightarrow q)$

La **partie gauche** d'une règle désigne la lettre et les n états situés à gauche de la flèche. La partie gauche représente les "conditions initiales" et la partie droite le

"résultat" d'application de la règle. Ce vocabulaire sera retenu et adapté ensuite pour les automates à tests.

Transitions

La sémantique des automates ascendants d'arbres est définie par les règles de dérivation :

Soient t et t', deux arbres de $T(\Sigma \cup Q)$,
t se dérive en t', (noté $t \rightarrow t'$) si :
$t = u(a(q_1(t_1), \ldots, q_n(t_n)), \quad t' = u(q(a(t_1, \ldots, t_n)))$ avec
$(a, q_1, \ldots, q_n, q) \in R$

Les arbres **reconnus** par un automate sont les arbres **clos** , tels que l'état atteint à la racine de l'arbre est un état terminal :
t est reconnu par A $\iff \exists q \in F, t \xrightarrow{*} q(t)$,
où "$\xrightarrow{*}$" désigne la clôture réflexive et transitive de "$\rightarrow$"

Une règle peut être apliquée si l'ensemble des états apparaissant en partie gauche apparaissent sous la lettre. On "remonte" ensuite en insérant l'état "d'arrivée" au-dessus de la lettre, d'où l'appellation d'automate ascendant.
Les termes "ascendants" ou "descendants" sont cependant sujets à caution, puisqu'ils font référence à la représentation graphique que l'on prend pour un arbre. Nous représentons toujours un arbre avec la racine en haut et les feuilles en bas (coutume bizarre, il est vrai).
Précisons donc que le terme ascendant est, pour nous, équivalent à "frontier-to-root", et désigne le sens allant des feuilles vers la racine.
Sont définis, parallèlement, des automates descendants (root-to-frontier) d'arbres, où l'état au-dessus de la lettre est l'état de départ, et où l'on insère ensuite n états en-dessous de la lettre où s'effectue la transition.

Classe de forêts reconnues

La classe des forêts reconnues est la même pour les automates ascendants et descendants, cette **classe de forêts reconnues** est notée **REC**.

Les automates descendants n'intervenant pas dans le cadre de ce travail, nous n'y reviendrons pas davantage.

Déterminisme

Un automate ascendant est dit **déterministe** si deux règles distinctes ont des parties gauches distinctes (i.e. pour des conditions initiales fixées, il existe au plus un état d'arrivée). Si cette condition est réalisée, aucun choix n'intervient dans l'application des règles.

Exemples

1-

Le premier exemple représente le typage d'expressions.
Il s'agira de reconnaître toutes les expressions manipulant un type numérique et des valeurs logiques. Les opérateurs et les constantes sont définis par l'alphabet gradué :

$$\Sigma = \{vrai, faux, r_1, \ldots r_n, +, *, =, <, \rceil, \vee, \wedge\}$$
$$\quad 0 \quad 0 \quad 0 \quad 0 \quad 2 \quad 2 \quad 2 \quad 2 \quad 1 \quad 2 \quad 2$$

Les deux types manipulés seront les deux états de l'automate :
$Q = \{Num, Bool\}$

Les règles de l'automate vont nous permettre de préciser les types des valeurs $vrai, faux, (r_i)_{1 \leq i \leq n}$ et les types (paramètre et résultat) des opérateurs.

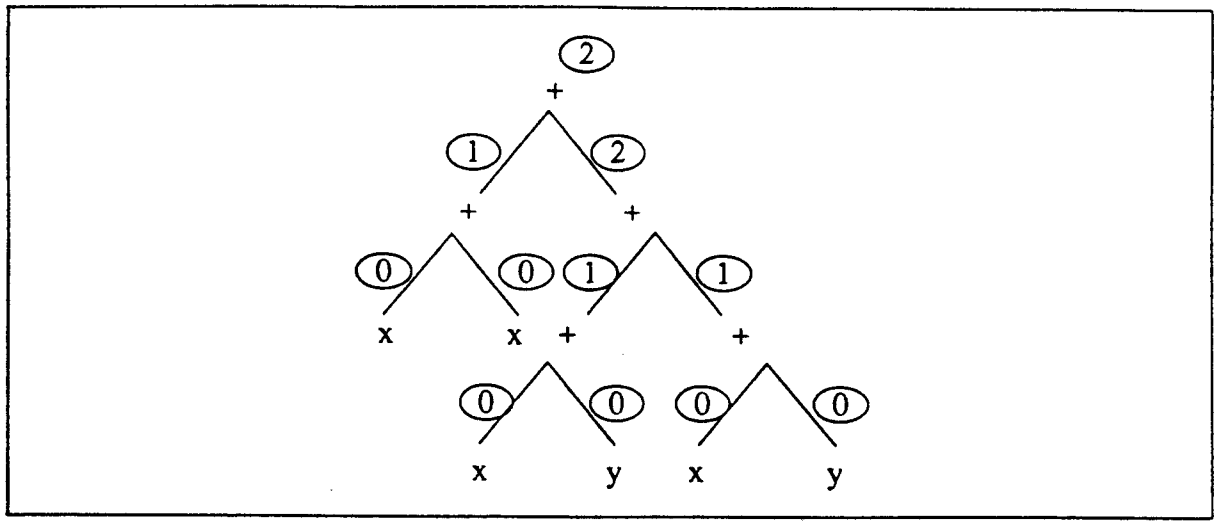
$vrai \rightarrow Bool \quad faux \rightarrow Bool$
$r_i \rightarrow Num$
$+(Num, Num) \rightarrow Num \quad *(Num, Num) \rightarrow Num$
$=(Num, Num) \rightarrow Bool \quad <(Num, Num) \rightarrow Bool$
$\rceil(Bool) \rightarrow Bool$
$\vee(Bool, Bool) \rightarrow Bool \quad \wedge(Bool, Bool) \rightarrow Bool$

Enfin, le choix des états terminaux détermine les expressions reconnues : Si Num et $Bool$ sont terminaux, toutes les expressions respectant le typage seront reconnues, quelque soit le type de leur résultat.

Cet exemple permet d'illustrer la correspondance entre forêts reconnaissables et algèbres sortées, la signature de l'algèbre étant représentée par les états et règles de l'automate.

2-

Voici un deuxième exemple d'automates d'arbres, permettant de calculer le nombre



Nombre de Strahler d'un arbre

Le nombre de Strahler peut être défini par induction de la façon suivante :
Strahler(feuille) = 0
Strahler($b(t_1, t_2)$) = max (Strahler(t_1), Strahler(t_2)), si Strahler(t_1) $\neq$ Strahler(t_2)
1 + Strahler(t_i), sinon.

Ce nombre est, par exemple, celui des registres nécessaires pour calculer une expression arithmétique (nous supposerons que l'addition est la seule opération).

Posons $\Sigma = \{x_1, \ldots, x_n, +\}$
0 0 2

Les états sont les nombres de Strahler inférieurs à une valeur k fixée : $\{0, 1, \ldots, k\}$
Quant aux règles, elles codent le calcul :
$x_i \rightarrow 0$
$+(i,j) \rightarrow max(i,j)$ pour $i \neq j$
$+(i,i) \rightarrow i + 1$ pour i¡k

En prenant $F = [1, k'], k' \leq k$ les arbres reconnus représentent les expressions dont le nombre de Strahler est inférieur ou égal à k'

1.2. 1 Quelques propriétés de la classe REC

Pour toute forêt $F \in REC$, il existe un automate ascendant d'arbres déterministe reconnaissant F.

La propriété "$\mathcal{F}(A) = \emptyset$" est décidable.

La propriété "$\mathcal{F}(A) = \emptyset$" est décidable.

La classe REC est close pour les opérations booléennes, ainsi que par morphisme linéaire d'arbres.

REC est close par morphisme inverse (même non-linéaire).

La clôture de REC par morphisme d'arbres et intersection est l'ensemble de forêts récursivement énumérables.

Toute forêt récursivement énumérable peut être définie par $\pi(\phi_1(R_1) \cap \phi_2(R_2))$, où R_1 et R_2 sont des forêts reconnaissables.

1.2. 2 Définition des morphismes d'arbres

Soient Σ et Σ', deux alphabets gradués. Un **morphisme d'arbres** ϕ de $T(\Sigma)$ dans $T(\Sigma')$ est l'extension, compatible avec la concaténation, d'une application qui, à toute lettre a de Σ, associe :

- $t \in T(\Sigma')_m$
- une fonction $\tau_a : [1, m] \rightarrow [1, n]$ (où n est la arité de a).

La fonction τ représente la torsion appliquée dans l'arbre image aux variables libres de la lettre graduée a.

Exemple :
$\Sigma = \{\bar{a}, a\}$, lettres respectivement de arité 0 et 1.
$\Sigma' = \{\bar{a}, a, b\}$, da arité 0, 1 , 2;

$\phi(\bar{a}) = \bar{a}$
$\phi(a(x)) = b(a(x), a(x))$

L'image par ϕ du "langage" $a^*\bar{a}$ est la forêt de tous les arbres de forme équilibrée, dont chaque branche est un mot de $(ba)^*\bar{a}$.Cet exemple permet de constater que l'image par un morphisme d'une forêt reconnaissable n'est pas nécessairement reconnaissable.

1.2. 3 Morphismes particuliers

Un morphisme d'arbres sera dit **linéaire** si, pour toute lettre a de Σ, τ_a est injective. Donc aucune des variables de a n'apparaît plusieurs fois dans $\phi(a)$.

Un morphisme d'arbres est dit **complet** si, pour toute lettre a de Σ, τ_a est surjective. Toutes les variables de a se retrouvent dans l'image de a.

Un morphisme **strict** est tel que l'image de toute lettre contient au moins une lettre.

Enfin, un morphisme d'arbres est dit **alphabétique** (au sens strict) si toute lettre de Σ possède une image appartenant à Σ'; on dit qu'il est alphabétique au sens large si l'image de a est une lettre ou une variable.

Exemples :

$\phi(a(x)) = a(x) \qquad \phi(b(x_1, x_2)) = a(b(x_1, x_1))$
n'est ni linéaire (x_1 apparaît deux fois à droite), ni complet (x_2 n'apparaît pas), ni alphabétique (l'image de b est un arbre de hauteur 1)

$\phi(a(x)) = b(x, x) \qquad \phi(b(x_1, x_2)) = a(x_1)$
ϕ est alphabétique, mais n'est ni linéaire, ni complet.

$\phi(a(x)) = a(x) \qquad \phi(b(x_1, x_2)) = a(x_2)$
ϕ est alphabétique, linéaire, non complet

$\phi(a(x)) = a(a(x)) \qquad \phi(b(x_1, x_2)) = b(x_2, x_1)$
ϕ est linéaire et complet; il n'est pas alphabétique.

$\phi(a(x) = b(x, x) \qquad \phi(b(x_1, x_2)) = x_2$
ϕ est alphabétique (au sens large : il est effaçant sur a), non-linéaire.

1. 3 Image de *REC* par morphismes non-linéaires

La propriété de non-linéarité pourrait paraître un peu anodine. Il n'en est rien. Les morphismes non-linéaires permettent d'obtenir des forêts très complexes, même appliqués de façon restreinte.

Nous présentons ici un résultat établi par J. Mongy [Mon 81], prouvant qu'il est possible d'obtenir toute forêt récursivement énumérable à partir des reconnaissables, en combinant 3 morphismes et une intersection.
Ce résultat s'énonce précisément par :

Pour toute forêt récursivement énumérable F, il existe R_1 et R_2, deux forêts reconnaissables, ϕ_1 et ϕ_2, deux morphismes complets et injectifs, et π un morphisme alphabétique linéaire non complet, tels que :

$$F = \pi(\phi_1(R_1) \cap \phi_2(R_2))$$

On pourra se reporter à la Thèse de J.Mongy [Mon 81] pour consulter la preuve complète de cette proposition, mais nous donnons ci-dessous une idée de la principale construction, qui éclaire le mécanisme de propagation de contraintes.

L'idée générale est de simuler dans $\phi_1(R_1) \cap \phi_2(R_2)$ les dérivations d'une grammaire d'arbres, et d'obtenir des arbres de la forme : $\#(t, hist)$, où t est un arbre engendré par la grammaire et *hist*, l'historique des dérivations qui ont permis de l'engendrer. Le morphisme π efface ensuite l'historique et la racine, de façon à obtenir dans $\pi(\phi_1(R_1) \cap \phi_2(R_2))$ l'ensemble des arbres engendrés par la grammaire. Sachant que toute forêt récursivement énumérable peut être engendrée par une grammaire d'arbres [ArDa 78], elle pourra donc également être obtenue par cette construction.

Nous décrirons le codage d'un partie du problème : la construction, pour une règle de dérivation *reg* donnée, de $R_1, R_2, \phi_1, \phi_2, \pi$ de manière à obtenir la forêt $\{\#(t', t) \mid t \rightarrow t' \text{ par application de } reg\}$
Posons $reg = (left \rightarrow right)$ où $left$ et $right$ sont des arbres, bien entendu.
Si t donne t', alors il existe u et $(v_i)_{1 \leq i \leq m}$ tels que $t = u(left(v_1, \ldots, v_m))$ et $t' = u(right(v_1, \ldots, v_m))$
Dans la forêt R_1, on code, en quelque sorte, la position dans t et t' du sous-arbre à transformer : c'est à dire l'endroit de u où se fait la greffe de $left$ dans t et de $right$ dans t'. Ce qui revient à exprimer les différentes décompositions de l'arbre, jusqu'à trouver en tête la partie gauche de la règle.

Cette position est indiquée par une suite de choix de sous-arbres à des profondeurs croissantes, par exemple : 2ème branche sous la racine, puis 1ère dans ce sous-arbre, puis 3 ème, ...etc
De façon rigoureuse, maintenant :
Pour chaque lettre de a, on construit les lettres $\tilde{a}^i$, où i est un entier entre 1 et n, qui indique la branche où se fera la dérivation.
Sous la lettre $\tilde{a}^i$, on trouve les branches inchangées représentant les contextes gauches et droits :$g_1 \dots g_{i-1}$ et $d_{i+1}, \dots d_n$; on trouve le sous-arbre à la position i tel qu'il est avant (l) et après (r) application de la règle; une dernière branche, enfin, permet de lier ensemble ces noeuds $\tilde{a}^i_j$ (peigne).
$\tilde{a}^i$ est donc de arité $n+2$ si a est de arité n.
On crée aussi une lettre ρ, dont la arité m est le nombre de variables de $left$ et de $right$.

La forêt reconnaissable R_1 aura donc la forme :
$\dots \tilde{a}^i_j(g_1, \dots, g_{i-1}, l, r, d_{i+1}, \dots d_n, (\tilde{a}^{i'}_{j'}(g'_1 \dots, g'_{i'-1}, l', r', \dots (\dots \rho(v_1, \dots, v_m)))))$

Les arbres $d_k, g_k, g'_k, d'_k, \dots, l, r, l', r', \dots, v_1, \dots, v_m$ sont des arbres quelconques définis sur l'alphabet $\Sigma \cup V$. Il va de soi que nous n'avons ici codé qu'une succession de sous-arbres censée représenter le chemin d'accès $(i, i', \dots)$ au sous-arbre transformé, mais ni la cohérence de ce chemin, ni celle entre l et r n'est assurée. Ce serait évidemment impossible dans une forêt reconnaissable, mais la vérification sera effectuée par application des morphismes et de l'intersection.

Définissons donc le morphisme ϕ_1 qui sera appliqué à R_1. Il laissera identiques toutes les lettres de Σ, et transformera uniquement les lettres $\tilde{a}^i$, par :

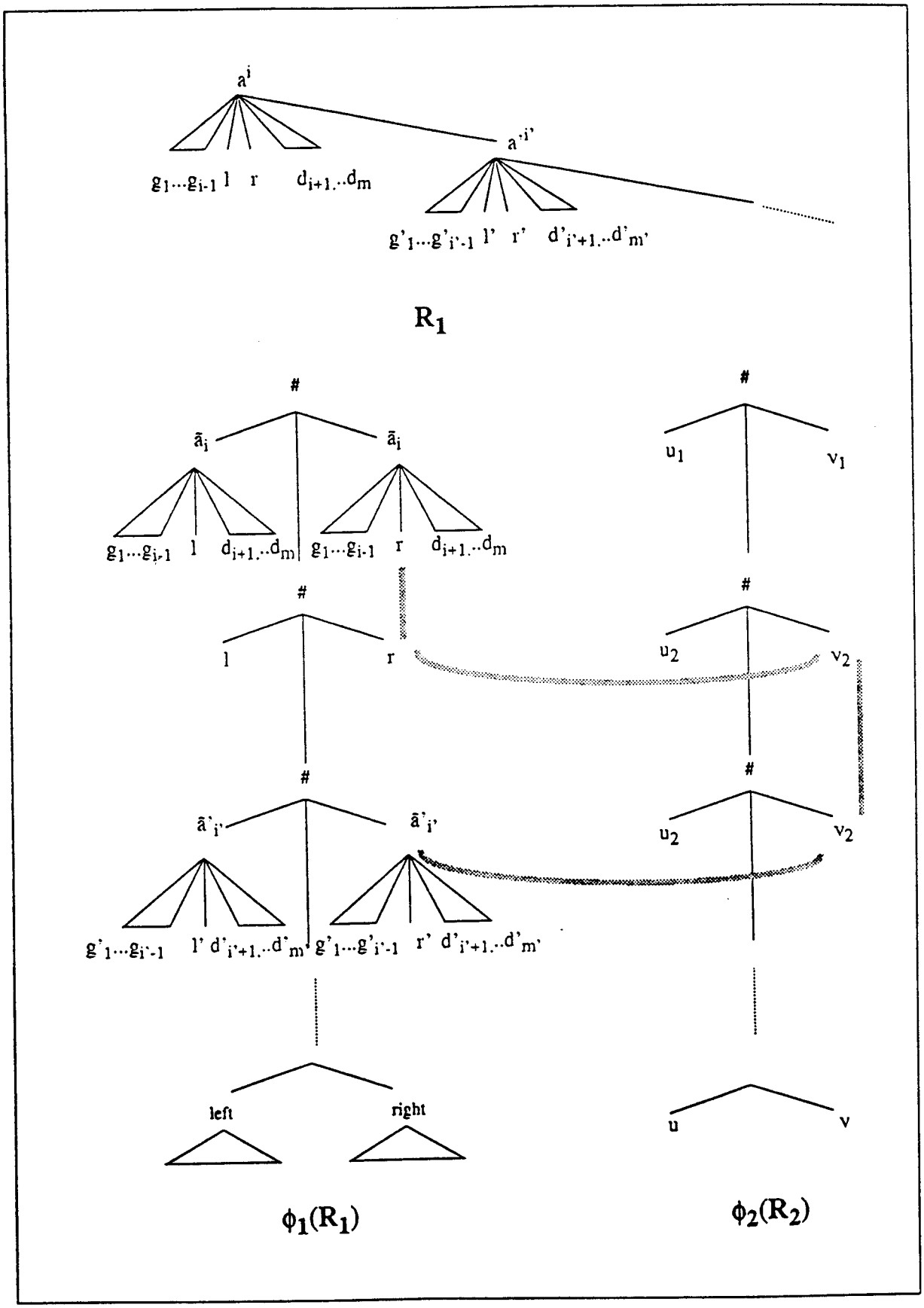
$$\phi_1(a^i(x_1, \dots, x_{i-1}, xl, xr, x_{i+1}, \dots, x_n, y)) =$$

$$\#(a(x_1, \dots, x_{i-1}, xl, x_{i+1}, \dots, x_n), \#(xr, y, xl), a(x_1, \dots, x_{i-1}, xr, x_{i+1}, \dots, x_n))$$

Et la lettre ρ par

$$\phi_1(\rho(x_1, \dots, x_m)) = \#'(right(x_1, \dots, x_m), left(x_1, \dots, x_m))$$

Le morphisme non linéaire crée un arbre portant de part et d'autre d'une branche centrale étiquetée par des #, des sous-arbres "presque identiques", distingués seulement par la branche devant être transformée par la règle ($left \rightarrow right$).
La duplication a permis d'imposer les mêmes contextes avant et après la dérivation.



Simulation d'une règle de grammaire d'arbres

Cependant la cohérence de la dérivation n'est pas encore assurée : chaque "tronçon" de l'arbre (deux # successifs) code une étape de la décomposition et la recomposition correspondante, mais l'enchaînement entre ces étapes n'est pas vérifiée.
Remarquons cependant que le sous-arbre où se déroule la dérivation a été dupliqué : il figure sous deux # consécutifs (le (2p)ième et le (2p+1)ième) : une fois dans son contexte, une fois en dehors, directement sous le #.
C'est ici qu'intervient l'utilisation des chevauchements d'égalités. Si l'on parvient à imposer au sous-arbre sorti du contexte d'être égal au sous-arbre avec contexte qui se trouve immédiatement en-dessous, on vérifie la cohérence entre deux étapes successives, jusqu'à l'extrémité (le #'), où l'on est assuré de trouver une application correcte de la règle de dérivation. ($left(v_1, \ldots, v_m)$ et $right(v_1, \ldots, v_m)$)
On va prendre l'intersection entre la forêt $\phi_1(R_1)$ et une forêt qui impose des égalités "décalées", portant sur le (2p+1)ième # et le (2p+2)ième.

D'où le choix de la forêt R_2 et du morphisme ϕ_2.
R_2 est simplement une forêt de la forme :

$$@(u_1, \#(u_2, \#(u_3, \#(\ldots \#'(u, v)\ldots), v_3)v_2)v_1)$$

à laquelle on applique le morphisme :
$\phi_2(@(x, y, z)) = \#(x, y, z)$
$\phi_2(\#(x, y, z)) = \#(x, \#(x, y, z), z)$
$\phi_2(\#'(x, y)) = \#(x, \#'(x, y), y)$
La lettre de tête assure le décalage, ensuite on trouve des égalités deux par deux.

L'intersection avec $\phi_1(R_1)$ ne conserve que les arbres représentant une application correcte de la règle ($left \rightarrow right$). Deux arbres en vis-à-vis sont les deux situations "avant" et "après" application de la règle. En-dessous, on trouve la totalité de l'"histoire" de la transformation.

Il ne reste plus ensuite qu'à oublier l'histoire (on sait que c'est dangereux), et ne conserver que le résultat. C'est le rôle du morphisme π, qui efface la branche centrale du # : $\pi(\#(x, y, z) = \#(x, z)$

Nous ne donnons que cette version allégée de la construction, non parce que c'est la mode, mais parce qu'elle suffit pour convaincre que l'on peut de la même manière coder l'enchaînement correct de plusieurs règles (la méthode est exactement la même mais à un niveau supérieur) et ensuite effacer l'historique, pour ne conserver que

l'arbre engendré par la grammaire.

1. 4 Automates à tests d'égalité

Dans un reconnaisseur d'arbre classique, seuls interviennent dans les règles les états atteints en dessous (pour les automates ascendants) ou au-dessus (pour les automates descendants) du noeud où s'applique la règle. Dans le cas d'un automate ascendant, par exemple, les "conditions initiales" sont constituées uniquement par la lettre et les états apparaissant en dessous de cette lettre.

Les automates à test sont des automates ascendants dans les règles desquels figurent, en plus, une condition utilisant la comparaison de sous-arbres du noeud considéré. Cette comparaison peut prendre différentes formes, suivant l'endroit où se situent les sous-arbres comparés :

- elle peut ne comparer que les sous-arbres immédiats de la lettre courante : c'est-à-dire comparer des sous-arbres pris à profondeur 0. Ce sera le cas pour les automates que nous étudions dans ce travail.
- la comparaison peut, au contraire, se faire entre des sous-arbres situés plus profondément. On peut encore faire une distinction entre le cas où les arbres comparés se situent à la même distance (profondeur) du noeud considéré et celui où les sous-arbres sont situés n'importe où, éventuellement à des profondeurs différentes.

1.4. 1 Les Rateg : un exemple de comparaisons à profondeur bornée

Les automates (ou classes reconnues, les deux termes sont confondus) RATEG ont été développés par J. Mongy.
Un RATEG est un automate ascendant à tests. Chaque règle est pourvue d'une condition d'application, qui peut être représentée par un arbre fini (et non clos, en général). Ce terme non clos peut être également non linéaire (une même variable peut y figurer plusieurs fois). Cet "arbre-condition" (ce terme n'est pas employé par J.Mongy, mais il représente assez bien la nature de la condition) indique donc la forme du sous-arbre initial apparaissant en-dessous du noeud courant. S'il est non-linéaire, alors certains sous-arbres doivent être égaux pour que la règle puisse s'appliquer.

La classe de forêts RATEG contient (strictement) la classe REC, puisqu'en prenant des arbres-conditions linéaires et réduits à la seule lettre courante, on retrouve un automate ascendant habituel.

Les comparaisons se font à des profondeurs bornées, bien évidemment, puisque les conditions sont des arbres finis. La borne de profondeur est alors donnée, pour l'automate, par la hauteur maximale des conditions.

Voyons un exemple d'automate "RATEG" :

$\bar{a} \rightarrow e$
$a(e) \rightarrow e$
$b(e, e) \rightarrow e'$
$b(e, e') \rightarrow e'$, avec la condition $b(a(x_1), b(x_1, x_2))$
Cette dernière règle s'applique donc en un b en-dessous duquel le sous-arbre initial est sous la forme : $b(a(x_1), b(x_1, x_2))$

Examinons la forêt reconnue : elle contient par exemple
$b(a^p.\bar{a}, a^q.\bar{a})$, $b(a^{p+1}.\bar{a}, b(a^p.\bar{a}, a^q.\bar{a}))$, $b(a^{p+2}.\bar{a}, b(a^p.\bar{a}, a^q.\bar{a})))$, ...
D'une manière générale, on prouve que les arbres sont sous la forme :

$$b(a^{p+i}.\bar{a}, b(a^{p+i-1}.\bar{a}, b(\ldots, b(a^p.\bar{a}, a^q.\bar{a}))))$$

où p, q, i sont des entiers positifs quelconques.
Cet exemple appelle une réflexion : si, au niveau de chaque règle, le test d'égalités ne porte que sur des arbres à profondeur bornée, il est cependant impossible de borner la distance entre des sous-arbres liés par la condition d'égalité. On a ici un exemple de ce genre de situations : le nombre de "a" de la branche linéaire (mot sur $a^*.\bar{a}$) la plus haute dans l'arbre est lié au nombre de "a" de celle qui apparaît le plus bas dans l'arbre, et ce quelle que soit la hauteur de l'arbre. Cette contrainte a pu être propagée sur toute la hauteur par les chevauchements entre "arbres-conditions", d'une application de la règle à une autre. En effet, si l'on opére une unification entre deux conditions successives, on obtient :

$$b(a^2(x_1), b(a(x_1), b(x_1, x_2)))$$

où la variable x_1 apparaît 3 fois, à 3 hauteurs distinctes.

Cette propagation des contraintes d'égalités est, intuitivement, la raison pour laquelle la propriété du vide est indécidable sur les automates RATEG.

Nous donnons, pour bien convaincre de ce résultat, le codage à l'aide des automates RATEG du problème de Post, dont on sait qu'il est indécidable.

Etant donnée une suite finie de mots $(\omega_i)_{1\leq i\leq n}$, on peut construire un automate (de mots) reconnaissant toutes les séquences d'éléments de cette suite. Cet automate comprendra un état initial et final q, tel que $q * \omega_i = q$, pour tout $i \leq n$ (les autres états ont peu d'importance dans la suite, nous supposerons simplement cet automate correctement construit).
Il est "transformé" en automate d'arbres en ajoutant, par exemple, une feuille $\bar{a}$ et une règle $\bar{a}\rightarrow q$.

Pour deux suites finies de mots $(\omega_i)_{1\leq i\leq n}$ et $(\omega'_i)_{1\leq i\leq n}$, ayant même longueur le problème de Post admet une solution si et seulement si il existe deux séquences égales :

$$\omega_{i_1}.\omega_{i_2}.\ldots.\omega_{i_m} = \omega'_{i_1}.\omega'_{i_2}.\ldots.\omega'_{i_m} \quad m \geq 1$$

Nous le codons dans un automate RATEG incluant :
- Deux automates construits suivant la méthode précédente, reconnaissant les séquences de mots des deux suites $(\omega_i)_{1\leq i\leq n}$ et $(\omega'_i)_{1\leq i\leq n}$, et utilisant des ensembles d'états disjoints (on notera q et q' les états atteints par les séquences de mots ω et ω' respectivement).
- Une lettre \$, de arité 3 et les règles :

$$\$(q, base, q)\rightarrow central \quad \bar{a}\rightarrow base$$

$$\$(q, central, q')\rightarrow central \quad cond : \$(\omega_i(x_1)\$(x_1, y, x_2), \omega'_i(x_2))$$

$$\$(q, central, q')\rightarrow final \quad cond : \$(x, y, x)$$

Cet automate reconnaît une forêt d'arbres dont les branches latérales sont des séquences de mots de la suite $(\omega_i)_{1\leq i\leq n}$ à gauche et de $(\omega'_i)_{1\leq i\leq n}$ à droite.
Chaque transition permet de vérifier que les éléments sont pris au même rang i dans chacune des deux suites finies.
Le chevauchement des contraintes d'égalités assure que cette propriété est vérifiée à chaque niveau de l'arbre.
Enfin, une règle permet de tester si les deux branches latérales les plus hautes sont égales.

Il y a donc une solution au problème de Post si et seulement si l'automate reconnaît une forêt non vide. Décider du vide pour un automate RATEG permettrait donc de

décider si le problème de Post admet une solution.

Remarque : Il existe d'autres façons de prouver l'indécidabilité du vide dans les RATEG. J.Mongy se ramène, elle, à son autre résultat prouvant que toute forêt récursivement énumérable s'exprime comme $\pi(\phi_1(R_1) \cap \phi_2(R_2))$ (que nous avons décrit plus haut). Les mécanismes utilisés sont du même ordre. Nous avons utilisé ici le codage direct du problème de Post car il permet de comprendre que c'est le chevauchement des contraintes d'égalités, qui dans un automate, engendre des forêts trop complexes.
Nous nous sommes donc attachés à empêcher cette propagation dans les automates que nous avons étudiés.

1.4. 2 Comparaisons entre fils

C'est le cas que nous développons dans cette thèse. Nous définirons plusieurs classes d'automates, se différenciant entre-elles par la façon dont sont testées les égalités et la façon dont on utilise les tests. Pour illustrer dès à présent cette variété dans les comportements d'automates, nous donnons succintement deux exemples, sans poser les définitions de manière très rigoureuse.

1. 5 Premiers exemples

1.5. 1 Avec des formules contenant des égalités

Les conditions sont exprimées au moyen de formules construites à l'aide du prédicat d'égalité, de la conjonction et de la disjonction.
Par exemple :

$a(q_1, \ldots, q_5) \rightarrow q$ quand $\#1 = \#2 \wedge (\#3 = \#4 \vee \#3 = \#5)$

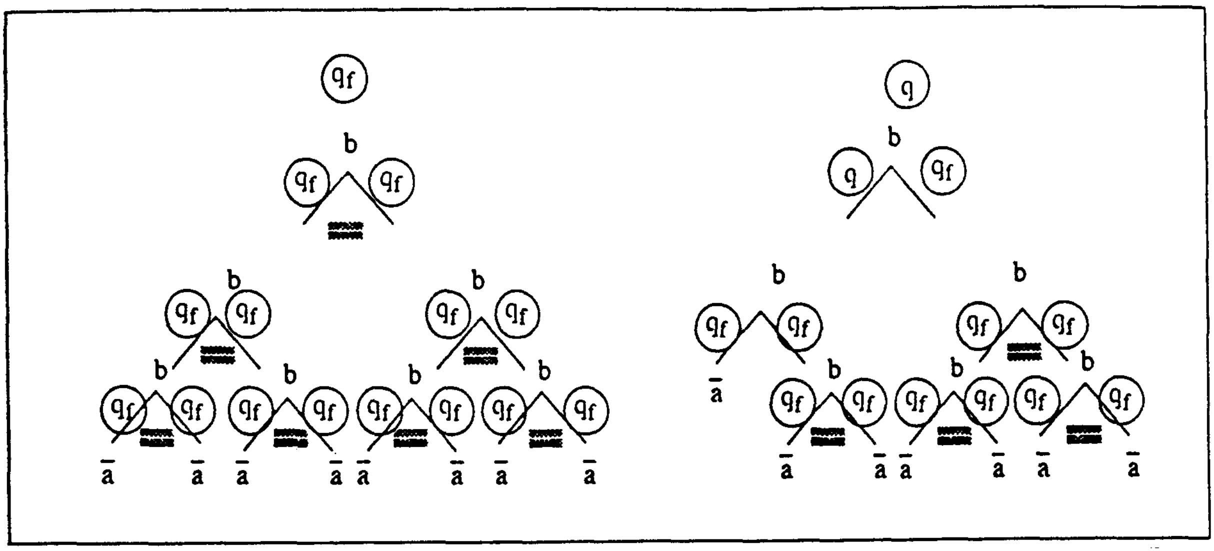
où #i désigne le i-ème fils du noeud étiqueté par a.

Comportement : Une règle peut s'appliquer dès lors que la condition est vérifiée.

La forêt constituée par les arbres équilibrés construits à partir du symbole b de arité 2 et du symbole $\bar{a}$ de arité 0 est reconnue par l'automate suivant :

$\bar{a} \rightarrow q$

$b(q, q) \rightarrow puits$



Seuls les arbres bien équilibrés sont acceptés

$b(q,q) \rightarrow q$ quand $\#1 = \#2$
q est, bien-sûr, état terminal.

La règle en b vérifie simplement qu'il y a égalité entre fils à chaque noeud, sinon la transition n'est pas définie.

Il est naturel de s'intéresser aussi au complémentaire de la forêt que nous venons de définir, et de s'interroger sur la reconnaissance de tous les arbres constitués de b (arité 2) et de a (arité 1) qui présentent, en un point quelconque, un déséquilibre de forme.
Pour construire une règle en b qui laisse passer les arbres en cas de différence mais pas en cas d'égalité, les formules que nous utilisons ici sont insuffisantes : il nous faudrait pouvoir exprimer la différence entre fils. Il s'avère donc impossible de reconnaître les arbres déséquilibrés à l'aide de l'outil exposé ici.
Dans notre deuxième exemple d'automates à tests, nous ajoutons la négation dans les formules .

1.5. 2 avec les négations en plus ...

La formule exprimant la condition est construite à l'aide de l'égalité, la conjonction, la disjonction, et la négation.
On peut donc interdire l'utilisation d'une règle dans le cas d'une égalié entre sous-arbres.
Cette fois il est possible de reconnaître les arbres déséquilibrés. Voici un automate

convenable :

$\bar{a} \rightarrow q$

$b(q,q) \rightarrow q$

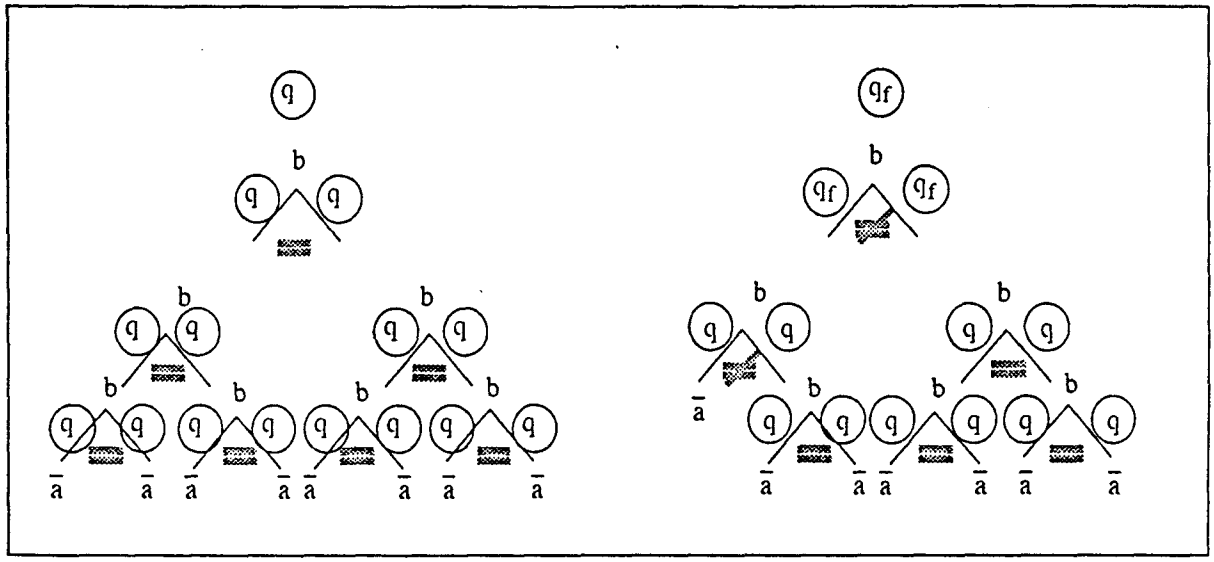
$b(q,q) \rightarrow q_f$ quand $\neg(\#1 = \#2)$

$b(q_f,q) \rightarrow q_f$

$b(q,q_f) \rightarrow q_f$

$b(q_f,q_f) \rightarrow q_f$

Il reconnaît le complémentaire de la forêt citée plus haut.



Reconnaissance des arbres déséquilibrés

1. 6 Les descriptions d'égalités

Dans le cas général d'une lettre de arité n, nous aurons besoin, pour écrire les différentes règles concernant cette lettre, de pouvoir exprimer les cas qui peuvent se présenter, du point de vue des égalités entre sous-arbres. Dans les deux exemples exposés précédemment nous avons utilisé des formules logiques, ce qui est de loin le moyen le plus naturel (?) et le plus habituel pour exprimer ces conditions. Cette représentation possède cependant quelques inconvénients : tout d'abord il n'y a pas unicité de la formule pour exprimer une même condition, ensuite on peut formuler des conditions "incohérentes", c'est-à-dire impossible à réaliser.

Ces deux inconvénients sont peu importants, à priori, pour la définition d'un exemple, on ne construira en général pas de règles avec une condition du style :

"$(\#1 = \#2) \wedge (\#1 \neq \#2)$".
Il est cependant préférable d'éliminer ces cas gênants pour la définition formelle des automates et pour leur manipulation.
Nous nous placerons donc dans un système fini de descripteurs d'égalités cohérent et dans lequel une condition possède une expression unique. Ce qui pourrait revenir à se placer dans le cadre des formes normales conjonctives de formules n'utilisant que l'égalité, et dans lesquelles chaque égalité :"$\#i = \#j$" figurerait au plus une fois, (tout cela modulo la commutativité).
Il est équivalent de considérer, par exemple, l'ensemble de arrangements (avec remise) de n éléments pris parmi n.

Ou encore une classe d'applications de $[1, n]$ dans $\mathbb{N}$ modulo l'équivalence $\equiv$:
$f \equiv g \iff (f(i) = f(j) \iff g(i) = g(j))$

Peu importe, en fait, la représentation personnelle que l'on se fait d'une description cohérente des égalités qui peuvent exister au sein d'un n-uples d'éléments. La définition que nous retiendrons est celle-ci (encore équivalente à celles que nous venons de citer) :
Une description des égalités dans un n-uple d'éléments est une partition de $[1, n]$. Les entiers entre 1 et n désignent une position dans le n-uple. Si deux positions sont dans la même partie, alors les deux éléments correspondant doivent être égaux.
Exemple :
Soient 3 éléments x_1, x_2, x_3. la propriété d'égalité que nous noterons $\{\{1,3\},\{2\}\}$ sera vérifiée quand $x_1 = x_3$

1.6. 1 Définitions

On appelle **description d'égalités** sur n éléments une partition de $[1, n]$

On note $\mathbf{ED_n}$ l'ensemble des descriptions d'égalités sur n éléments, et $\mathbf{ED_{n,m}}$ l'ensemble des descriptions d'égalité de cardinal m, sur n éléments.
Cet ensemble ED_n désigne les descriptions de n-uples où sont représentées m valeurs distinctes.

On pose $ED_0 = \{\perp_0\}$

Nous définissons deux propriétés distinctes d'une description vis-à-vis d'un n-uple

d'éléments :

$$(d \textbf{ est valide pour } (x_i)_{1\leq i\leq n}) \iff (\forall X \in d, \forall i,j \in X, x_i = x_j)$$

$$(d \textbf{ est exacte pour } (x_i)_{1\leq i\leq n})$$

$$\iff$$

$$(d \text{ est valide pour } (x_i)_{1\leq i\leq n} \quad \text{et} \quad \forall X \neq Y \in d, i \in X, j \in Y, x_i \neq x_j)$$

1.6. 2 Ordre sur les descriptions d'égalités

Pour chaque n, on peut définir sur ED_n une relation d'ordre (partiel) $\leq_n$ de la façon suivante :

$$d \leq_n d' \text{ (d est moins précise que d')} \iff (\forall X \in d, \exists Y \in d', X \subseteq Y)$$

ED_n, muni de cette relation est un treillis, et l'on notera $\triangle_n$, l'opération "plus petit majorant". L'ensemble des singletons, le plus petit élément, sera noté $\perp_n$.

1.6. 3 Propriétés:

1) $\perp_n$ est valide pour tout n-uple.
2) Si $(x_i)_{i\leq n}$ vérifient d et $d \geq d'$, alors $(x_i)_{i\leq n}$ vérifient d'.
3) $(x_i)_{i\leq n}$ vérifient d et vérifient d' si et seulement si ils vérifient $d \triangle_n d'$
4) Pour un n-uple donné, la description exacte est la plus précise des descriptions valides.

Le $\triangle_n$ représente donc le "et" sur les conditions, et $\perp_n = \{\{i\} | i \in [1,n]\}$ le "vrai"

Par la suite nous noterons $\leq$ au lieu de $\leq_n$, $\triangle$ au lieu de $\triangle_n$ et $\perp$ au lieu de $\perp_n$, puisqu'il n'y a pas d'ambiguïté possible.

1.6. 4 Exemples

Dans l'ensemble ED_3, considérons la description $d_1 = \{\{1,2\},\{3\}\}$
d_1 est valide et exacte pour a, a, b, elle est valide pour a, a, a
$d_2 = \{\{1\},\{2\},\{3\}\}$ est valide pour a, a, b, et pour a, a, a
$d_2 \leq d_1$, elle est moins précise que d_1
$d_2 = \bot_3$ que l'on note plus simplement $\bot$
$d_3 = \{\{1\},\{2,3\}\}$ et d_2 sont incomparables. $d_3 \bigtriangleup d_2 = \{\{1,2,3\}\}$ est valide pour les 3-uples pour lesquels d_1 est valide et d_2 est valide.
Par exemple $d_1 \bigtriangleup d_2$ est valide pour b, b, b (c'en est même une description exacte).
$d_1 \bigtriangleup d_2 = \top_3$ que l'on note $\top$
$\top$ n'est valide que pour les n-uples dont tous les éléments sont identiques.

Chapitre 2

Stratégie à contrôle faible

2. 1 Définitions et exemples

2.1. 1 Automates à tests d'égalité entre fils

Un automate à tests d'égalités entre fils sera défini par un quadruplet :

Σ , un alphabet gradué

Q, un ensemble fini d'états

F, un ensemble d'états terminaux, sous-ensemble de Q

$\mathcal{R}$, un ensemble fini de règles. $\mathcal{R} \subseteq \bigcup_i \Sigma_i \times ED_i \times Q^{i+1}$

Notation

Les règles seront généralement notées : $(a(q_1, \ldots, q_n)[d] \rightarrow q)$ où d désigne la description d'égalité.

2.1. 2 Transitions : stratégie à contrôle faible

Soit A, un automate, soient $t, t' \in T(\Sigma)_0$. t se dérive en t', (noté $t \rightarrow t'$) si et seulement si :

$\exists a \in \Sigma_n, \quad \exists u \in T(\Sigma)_1, \quad \exists t_1, t_2, ..., t_n \in T(\Sigma_0)$

$\exists (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}$

$t = u(a(q_1(t_1), q_2(t_2), \ldots, q_n(t_n))) \quad t' = u(q(a(t_1, \ldots, t_n)))$

et d est valide pour $(t_i)_{1 \leq i \leq n}$.

Cette définition "sémantique" de l'automate sera qualifiée de **stratégie à contrôle faible**, dans la mesure où le contrôle effectué laisse, en général, une liberté dans le choix de la règle à appliquer : plusieurs descriptions d'égalités peuvent être valides pour un même n-uple, et donc plusieurs règles peuvent être "utilisables".

Notation

$\xrightarrow{*}$ désigne la clôture réflexive et transitive de $\rightarrow$.

2.1. 3 Quelques définitions

Un arbre t de $T(\Sigma_0)$ sera dit **reconnu** par l'automate si $\exists q \in F$, tel que $t \xrightarrow{*} q(t)$.

Nous dirons que deux règles ont **même profil** si elles sont définies sur la même lettre et ont les mêmes états de départ. Elles diffèrent par leur decription d'égalités ou l'état d'arrivée.

On dira qu'elles ont **même partie gauche** quand elles ont le même profil et la même description d'égalité (i.e. elles ne diffèrent que par l'état d'arrivée)

Un automate est dit **complet** si pour toute lettre a, pour tout n-uple $q_1, \ldots, q_n$ (où n est la arité de a), pour toute description $d \in ED_n$, il existe une règle de partie gauche $a(q_1, \ldots, q_n)[d]$.

Propriété.

A tout automate à tests d'égalités, on peut associer un automate complet reconnaissant, par stratégie à contrôle faible, la même forêt.

La complétion se fait de la même manière que pour les automates d'arbres classiques, en ajoutant un état puits, état d'arrivée pour toutes les transitions non définies précédemment. Etant donné le caractère non déterministe de l'automate, la forêt reconnue n'est pas modifiée.

Exemples

$$r_1 = (a(q_1, \ldots, q_n)[\{\{1, \ldots, n\}\}] \rightarrow q)$$
$$r_2 = (a(q_1, \ldots, q_n)[\{\{1\}, \ldots, \{n\}\}] \rightarrow q)$$
$$r_3 = (a(q_1, \ldots, q_n)[\{\{1, \ldots, n\}\}] \rightarrow q')$$
$$r_4 = (a(q'_1, \ldots, q'_n)[\{\{1, \ldots, n\}\}] \rightarrow q)$$

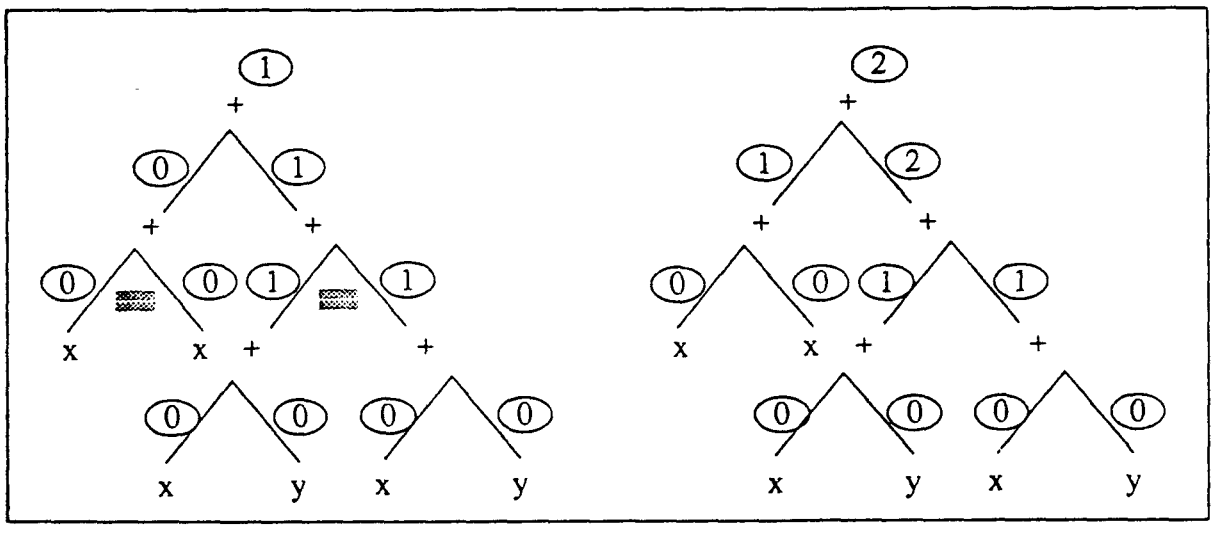
r_1 et r_2 ont même profil, mais pas même partie gauche.
r_1 et r_3 ont même partie gauche, donc même profil.
r_4 n'a pas le même profil que r_1 , ni r_2 ni r_3

2.1. 4 Classe de forêts reconnues : $REC_=$

On notera $\mathbf{REC_=}$, la famille des forêts reconnues par des automates à tests, par stratégie à contrôle faible.

Exemple
Nous allons compléter l'automate cité dans le chapitre de généralités, qui calculait le nombre de Strahler d'un arbre.

Si l'on se réfère à l'interprétation en termes de registres de calculs, on peut affiner l'évaluation en remarquant que $x + x$ ne nécessite pas de registre supplémentaire, puisque l'opération peut se faire par décalage du registre contenant x.



Nombres de Strahler réduits : exemples de parcours possibles

Ce qui nous amène donc à ajouter une règle dans l'automate :

$+(i,i)[\{\{1,2\}\}]\rightarrow i$
Que l'on pourra exprimer également (voir section suivante) par
$+(i,i)[\#1 = \#2]\rightarrow i$

ce qui signifie simplement que les deux branches doivent être égales.

Remarquons bien que l'automate est devenu non-déterministe.
L'arbre $+(x_1,x_1)$ peut atteindre aussi bien 1 que 0

Cependant la forêt reconnue en prenant $Fin = [1,k']$ comprendra tous les arbres pouvant utilser moins de k' registres, en utilisant le décalage.

On peut aussi se poser la question de l'utilité de faire figurer la règle
$+(i,j)[\#1 = \#2] \rightarrow max(i,j)$ quand $i \neq j$
La forêt reconnue serait exactement la même.
Nous verrons un peu plus loin qu'il est toujours possible de se priver de telles règles (les automates seront alors dits normalisés).
Cependant, dans le cas général, on ne peut pas faire de suppression pure et simple de ces règles et, contrairement à ce qui se passe ici, il faut souvent créer de nouvelles règles "normales" avant la suppression des autres.

2. 2 Expression des conditions à l'aide de formules

Dans l'introduction, nous donnions succintement une définition d'automate à tests, dans lequel les conditions seraient exprimées par des formules construites à l'aide de l'égalité, de la conjonction et de la disjonction.
Cette formulation est équivalente à la définition d'automates donnée en section 2.1, où les règles utilisent des descriptions d'égalités et non des formules.
Nous avons expliqué au chapitre 1 la raison de l'utilisation des descriptions, raison essentiellement technique, ainsi que la nécessité de disposer d'un système fini de conditions. Afin de pouvoir utiliser indifféremment l'une ou l'autre des formulations, nous donnons une définition rigoureuse d'automates utilisant des formules, et nous en prouvons l'équivalence avec la précédente.

Définition
Tout automate à contrôle faible peut être exprimé par un quadruplet :

Σ, alphabet gradué.
Q, ensemble fini d'états
F, ensemble d'états terminaux
$\mathcal{R}$ ensemble de règles, inclus dans $\bigcup_i \Sigma_i \times F_{=,i} \times Q^{i+1}$

Où $F_{=,i}$ désigne l'ensemble de formules sur i variables, utilisant égalité, conjonction et disjonction.

Exemple
$\#1 = \#2 \wedge \#2 = \#4$ indique que les éléments de numéro 1, 2 et 4 doivent être égaux.

Transitions

Les transitions se définissent comme suit :

$t \rightarrow t'$ si et seulement si :

$\exists a \in \Sigma_n, \quad \exists u\ T(\Sigma)_n, \quad \exists t_1, t_2, ...t_n \in T(\Sigma)_0$

$\exists (a, f, q_1, \ldots, q_n, q) \in \mathcal{R}$

$t = u(a(q_1(t_1), q_2(t_2), \ldots, q_n(t_n))) \quad t' = u(q(a(t_1, \ldots, t_n)))$

et $f(t_1, \ldots, t_n)$ est vraie.

Equivalence des deux définitions

Montrons que les deux définitions sont équivalentes.

Pour toute formule f de $F_{=,i}$, il existe un sous-ensemble D_f dans ED_i qui vérifie

$$f(x_1, \ldots, x_n) \text{ est vraie } \iff (\exists d \in D_f),\ \text{d est valide pour } x_1, \ldots, x_n$$

Considérons maintenant un automate $A_{formules}$ dont les conditions sont exprimées par des formules. On construit un automate équivalent $A_{descriptions}$ avec les mêmes ensembles d'états. Ses règles seront définies en associant à toute règle r de $A_{formules}$ de la forme :

$$r = a(q_1, \ldots, q_n) \rightarrow q \text{ [quand } f(t_1, \ldots, t_n) \text{ est vraie]}$$

l'ensemble des règles :

$$\{(a(q_1, \ldots, q_n)[d] \rightarrow q) | d \in D_f\}$$

De cette manière, la règle r peut s'appliquer en un noeud si et seulement si l'une des règles ainsi construites peut s'appliquer.

Inversement, à toute description d, on peut associer la formule conjonctive:

$$f = \bigwedge_{\{i,j | \exists p \in d, i \in p, j \in p\}} \#i = \#j$$

cette formule est vraie pour $(t_i)_{1 \leq i \leq n}$ si et seulement si d est valide pour $(t_i)_{1 \leq i \leq n}$

Donc, pour un automate $A_{descriptions}$ on peut construire un automate $A_{formules}$ en remplaçant chaque description par la formule conjonctive équivalente.

Exemples

A la description $\{\{1,2\},\{3,4,5\}\}$, on fait correspondre la formule

$\#1 = \#2 \wedge \#3 = \#4 \wedge \#4 = \#5$, qui représente la même condition.

A la formule $\#1 = \#2 \vee \#2 = \#3 \vee \#1 = \#3$, on associe plusieurs descriptions (on considère qu'il n'y a que 3 variables)
$\{\{1,2\},\{3\}\}$, $\{\{1,3\},\{2\}\}$, $\{\{1\},\{2,3\}\}$, $\{\{1,2,3\}\}$
Remarque : la dernière description est inutile pour ce type d'automate : si les 3 variables sont égales, alors l'une quelconque des 3 autres decriptions est valide.

Voici l'automate à descriptions qui reconnaît la forêt constituée par les abres équilibrés, sur l'alphabet $\Sigma = \{\bar{a}, a, b\}$
0 1 2
Il s'agit d'une traduction de l'automate donné en exemple au chapitre 1.
$Q = \{q, q_f\}$ $Fin = \{q_f\}$
L'ensemble de règles :

$\bar{a}[\bot] \rightarrow q_f$
$a(q_f)[\bot] \rightarrow q_f$
$b(q,q)[\{\{1,2\}\}] \rightarrow q_f$

Remarque : On omettra parfois le $[\bot]$ pour écrire plus simplement $profil \rightarrow$ *état d'arrivée*

2. 3 Contrôle faible et déterminisme

Du fait de la stratégie à contrôle faible, le non-déterminisme peut prendre deux formes.
Considérons un arbre $a(q_1(t_1), \ldots, q_n(t_n))$.
Pour choisir la règle à appliquer il faut :

- Choisir une partie gauche $q_1, \ldots, q_n, d$, où d est une description valide pour $t_1, \ldots, t_n$. Un choix est donc possible entre plusieurs descriptions.
- Choisir une règle possédant cette description. Plusieurs peuvent exister, comme dans le cas des automates classiques.

Eliminer cette deuxième cause de non déterminisme, n'est donc pas éliminer le choix entre plusieurs règles.
Dans le cas des automates de mots, ou des automates d'arbres classiques, il y a coïncidence entre le sens habituel du mot déterminisme et "pour une partie gauche donnée, il existe au plus une règle".

Ce n'est pas le cas pour la stratégie à contrôle faible des automates à tests d'égalités. Aussi, ne parlerons-nous pas de déterminisme pour désigner cette propriété, mais plutôt de pseudo-déterminisme.

Exemple
Si un automate comprend les règles :
$b(q,q)[\#1 = \#2] \rightarrow q$
$b(q,q)[\bot] \rightarrow q'$

et si l'on considère $b(t,t)$ avec $t \xrightarrow{*} q$, on a le choix entre utiliser la règle étiquetée par $[\#1 = \#2]$, ou celle étiquetée par $[\bot]$.

2.3. 1 Pseudo-déterminisme

Définition.

Un automate est dit **pseudo-déterministe** *si pour toute partie gauche p, il existe au plus une règle.*

Exemple
Un automate comprenant les règles :

$c(q,q,q)[\{\{1,2\},\{3\}\}] \rightarrow q$
$c(q,q,q)[\{\{1,2\},\{3\}\}] \rightarrow q'$

n'est pas pseudo-déterministe, on peut obtenir deux états distincts avec des règles ayant même partie gauche.

Proposition.

Pour tout automate à contrôle faible, il existe un automate à contrôle faible pseudo-déterministe reconnaissant la même forêt.

2.3. 2 Algorithme de pseudo-déterminisation

Soit A un automate à contrôle faible. L'automate $\bar{A}$, pseudo-déterministe, sera défini sur le même alphabet gradué. Les ensembles d'états et les règles en sont définis de la manière suivante :

$Q_0 \leftarrow \{X | X = \{q | \exists \bar{a} \in \Sigma_0, (\bar{a} \rightarrow q) \in \mathcal{R}\}\}$
$\mathcal{R}_0 \leftarrow \{(\bar{a}[T] \rightarrow X) | X = \{q | (a[\bot] \rightarrow q) \in \mathcal{R}\}\}$
$i \leftarrow 0$

répéter :
  $i \leftarrow i+1$
  $\bar{Q}_i \leftarrow \bar{Q}_{i-1} \cup$
    $\{X |\ \exists a \in \Sigma_n, \exists (X_i)_{1 \leq i \leq n}$ dans $Q_{i-1}, \exists d \in ED_n,$
      $X = \{q | \exists (q_i)_{1 \leq i \leq n}, q_k \in X_k, \quad (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}\}$
    $\}$

  $\bar{\mathcal{R}}_i \leftarrow \bar{\mathcal{R}}_{i-1} \cup$
    $\{(a(X_1, \ldots, X_n)[d] \rightarrow X) |$
    $a \in \Sigma_n, \quad (X_i)_{1 \leq i \leq n}$ dans $\bar{Q}_{i-1}, \quad d \in ED_n,$
    $X = \{q | \exists (q_i)_{1 \leq i \leq n}$ dans $Q, q_k \in X_k,$
      $(a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}$
    $\}$
  $\}$
jusqu'à ce que $\bar{Q}_i = \bar{Q}_{i-1}$ et $\bar{\mathcal{R}}_i = \bar{\mathcal{R}}_{i-1}$
$\bar{Q} \leftarrow \bar{Q}_i$
$\bar{\mathcal{R}} \leftarrow \bar{\mathcal{R}}_i$
$\bar{F} \leftarrow \{X | X \cap F \neq \emptyset\}$

Preuve de l'équivalence des deux automates

1- Nous prouvons par récurrence sur la hauteur de l'arbre que

$$\forall X \in 2^Q, \quad (t \xrightarrow[\bar{A}]{*} X) \Rightarrow (\forall q \in X, t \xrightarrow[A]{*} q)$$

- Si t est de hauteur 0, t est une lettre de Σ_0.
  $t \xrightarrow[\bar{A}]{*} \{q | (t \rightarrow q) \in \mathcal{R}\}$
  (Construction de $\bar{\mathcal{R}}_0$)

- Supposons la propriété vraie pour un arbre de hauteur n, et considérons t de hauteur n+1
  Posons $t = a(t_1, \ldots, t_n)$ où n est la arité de la racine de t.

$t \xrightarrow[\bar{A}]{*} X$

$\Rightarrow$

$\exists (X_i)_{1 \leq i \leq n}, \exists d \in ED_n$ *valide pour* $t_1, \ldots, t_n, \exists (a(X_1, \ldots, X_n)[d] \rightarrow X) \in \bar{\mathcal{R}}$

$t_i \xrightarrow[\bar{A}]{*} X_i$

$\Rightarrow$

$\exists (X_i)_{1 \leq i \leq n}, \exists d \in ED_n$ *valide pour* $t_1, \ldots, t_n$,

- $\forall q \in X, \exists (q_i)_{1 \leq i \leq n}, q_i \in X_i, (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}$ [construction]
- $\forall q_i \in X_i, t_i \xrightarrow[A]{*} q_i$ [hyp. récurrence]

$\Rightarrow$

$\forall q \in X, t \xrightarrow[A]{*} q(t)$

2-

$$\forall t \in T(\Sigma)_0,\ \forall q \in Q,\ (t \xrightarrow[A]{*} q \Rightarrow \exists X,\ t \xrightarrow[\bar{A}]{*} X,\ q \in X)$$

C'est évidemment vrai pour un arbre de hauteur nulle.

Si $t = a(t_1, \ldots, t_n) \xrightarrow[A]{*} q$, alors, il existe $q_1, \ldots, q_n$ et une description d valide pour $t_1, \ldots, t_n$ tels que $t_i \xrightarrow[A]{*} q_i$ et $(a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}$

Si la propriété est vérifiée pour chaque sous-arbre t_i, $(t_i \xrightarrow[\bar{A}]{*} X_i, q_i \in X_i)$, la méthode de construction nous assure de l'existence d'une règle $(a(X_1, \ldots, X_n)[d] \rightarrow X)$, où $q \in X$ si $q_i \in X_i$

2.3. 3 Egalités d'arbres , égalités d'états

Quand on construit une automate à tests, il est assez rare que l'on construise des règles imposant une égalité entre des branches atteignant des états différents, du type $b(q_1, q_2)[quand\ \#1 = \#2] \rightarrow q)$

C'est pourtant tout à fait possible au regard de la définition que nous avons donnée en début de chapitre.

Il se trouve qu'il est intéressant (et simplificateur) dans certaines preuves de pouvoir supposer que les égalités ne sont imposées que sur des sous-arbres pouvant atteindre un état identique.

Définition.

Un automate A d'ensemble de règles $\mathcal{R}$ est dit **normalisé** *si*

$$(a(q_1,\ldots,q_n)[d]\rightarrow q) \in \mathcal{R} \Rightarrow d \text{ est valide pour } (q_i)_{1\leq i\leq n}$$

En général, un automate ne peut pas être complet et normalisé (sauf si toutes les lettres sont de arité 1, ou s'il n'y a qu'un seul état). Nous introduisons une nouvelle définition, celle d'automates "les plus complets possibles", tout en étant normalisés.

Définition.

Un automate est dit **normalisé-complet** *si pour toute lettre a, tout n-uple $q_1,\ldots,q_n$, toute description $d \in ED_n$* **valide pour $\mathbf{q_1},\ldots,\mathbf{q_n}$**, *il existe une règle de partie gauche $a(q_1,\ldots,q_n)[d]$.*

Proposition.

Toute forêt de $REC_=$, peut être reconnue par un automate normalisé.

Preuve

Pour prouver la proposition, nous améliorons l'algorithme de pseudo- déterminisation. Cette fois-ci, un état rassemblera l'ensemble des choix possibles dans l'automate de départ, même avec des descriptions distinctes.
Cette même construction nous permettra dans le chapitre suivant de montrer qu'il est possible de se ramener à un automate totalement déterministe (à stratégie forte).

$Q_0 \leftarrow \{X | \exists \bar{a} \in \Sigma_0,\ X = \{q \mid (\bar{a}[\bot] \rightarrow q) \in \mathcal{R}\}\}$
$\mathcal{R}_0 \leftarrow \{(\bar{a}|[\bot] \rightarrow X) | X = \{q | (a \rightarrow q) \in \mathcal{R}\}\}$
$i \leftarrow 0$

répéter :
  $i \leftarrow i + 1$
  $\bar{Q}_i \leftarrow \bar{Q}_{i-1} \cup$
    $\{X |\ \exists a \in \Sigma_n, \exists (X_i)_{1 \leq i \leq n}$ dans $Q_{i-1}, \exists d \in ED_n,$
      $X = \{q | \exists (q_i)_{1 \leq i \leq n},\ q_k \in X_k,\ \exists d' \leq d,\ (a(q_1, \ldots, q_n)[d'] \rightarrow q) \in \mathcal{R}\}$
    $\}$

  $\bar{\mathcal{R}}_i \leftarrow \bar{\mathcal{R}}_{i-1} \cup$
    $\{(a(X_1, \ldots, X_n)[d] \rightarrow X) |$
      $a \in \Sigma_n,\quad (X_i)_{1 \leq i \leq n}$ dans $\bar{Q}_{i-1},\quad d \in ED_n,$
      $X = \{q | \exists (q_i)_{1 \leq i \leq n},\ q_k \in X_k,\ \exists d' \leq d,\ (a(q_1, \ldots, q_n)[d'] \rightarrow q) \in \mathcal{R}\}$
    $\}$
jusqu'à ce que $\bar{Q}_i = \bar{Q}_{i-1}$ et $\bar{\mathcal{R}}_i = \bar{\mathcal{R}}_{i-1}$
$\bar{Q} \leftarrow \bar{Q}_i$
$\tilde{\mathcal{R}} \leftarrow \mathcal{R}_i$
$\bar{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \setminus \{(a(X_1, \ldots, X_n)[d] \rightarrow X) |\ d$ *non valide pour* $X_1, \ldots, X_n\}$
$\bar{F} \leftarrow \{X | X \cap F \neq \emptyset\}$

Equivalence des automates

Nous montrons que l'automate $\tilde{A} = (\Sigma, Q, F, \mathcal{R})$ reconnaît la même forêt que A

1-

$$\forall X \in 2^Q,\quad (t \xrightarrow[\tilde{A}]{*} X) \Rightarrow (\forall q \in X, t \xrightarrow[A]{*} q)$$

La preuve se fait exactement de la même manière que pour la pseudo-déterminisation.

2-

$$\forall t \in T(\Sigma)_0, t \xrightarrow[\tilde{A}]{*} \{q | t \xrightarrow[A]{*} q\}$$

La preuve se fait encore par récurrence. La propriété est évidente pour les lettres de arité 0. Si on la suppose vraie jusqu'à la hauteur n, et si l'on considère un arbre $a(t_1, \ldots, t_n)$ de hauteur n+1, on montre que la propriété est encore vraie, en considérant la description exacte des t_i. Sur cette description exacte des sous-arbres est définie une règle vers un état rassemblant toutes les possibilités de l'automate de départ.

3- Il est alors possible d'éliminer les règles imposant une égalité sur des branches portant des états différents, car si un arbre t peut atteindre deux états q et q', alors il peut atteindre dans le nouvel automate un état qui contient q et q'.

2.3. 4 Automates à tests pour *REC*

Proposition.

REC$_=$ contient REC

Preuve : soit une forêt reconnaissable F, reconnue par un automate $A = (\Sigma, Q, F, \mathcal{R})$. La forêt reconnue par A est également reconnue par :
$A_= = (\Sigma, Q, F, R_=)$ où $R_= = \{(a(q_1, \ldots, q_n)[\bot] \rightarrow q) | (a(q_1, \ldots, q_n) \rightarrow q) \in R\}$.
$A_=$ est muni de la stratégie de transitions à contrôle faible. L'automate avec tests $A_=$ ne contient que des règles utilisant la description $\bot$, il n'y a donc aucune différence sémantique entre cet automate et l'automate de départ.

2. 4 Propriétés booléennes de $REC_=$

2.4. 1 Clôture par union

Proposition.

La famille $REC_=$ est **close par union.**

Preuve

Soient $F \in REC_=$ et $F' \in REC_=$ reconnues respectivement par les automates $A = (\Sigma, Q, Fin, \mathcal{R})$ et $A' = (\Sigma', Q', Fin', \mathcal{R}')$. L'union de ces deux forêts sera reconnue par l'automate union de A et A', c'est à dire que si, bien-sûr, Q et Q' sont disjoints, $F \cup F'$ sera reconnue par $(\Sigma, Q \cup Q', Fin \cup Fin', \mathcal{R} \cup \mathcal{R}')$.

2.4. 2 Clôture par intersection

Proposition.

La famille $REC_=$ est **close par intersection.**

Preuve

Nous construisons l'automate produit.

Soit F reconnue par $A = (\Sigma, Q, , Fin, \mathcal{R})$ et F' par $A' = (\Sigma, Q, Fin, \mathcal{R}))$ (si les alphabets sont différents, on peut se limiter à l'intersection des deux).

L'automate $A \times A' = (\Sigma, Q \times Q', Fin \times Fin', \mathcal{R}_\times)$ reconnaît $A \times A'$.

Pour construire l'ensemble de règles $\mathcal{R}_\times$ on fait deux à deux le produit des règles des automates A et A'. La règle "produit" portera la description "et" des descriptions des deux règles composantes. Ce qui nous donne la définition :

$\mathcal{R}_\times = \{(a((q_1, q'_1), \ldots (q_n, q'_n))|[d \triangle d'] \rightarrow (q, q')) \mid (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R},$
$(a(q'_1, \ldots, q'_n)[d'] \rightarrow q)' \in \mathcal{R}'$

$\}$

Montrons que :

$(t \xrightarrow[A]{*} q(t)$ et $t \xrightarrow[A']{*} q'(t)) \iff (t \xrightarrow[A_\times]{*} (q, q')(t))$

- la propriété est trivialement vraie quand t est une lettre de arité 0.
- Supposons-la vraie pour un arbre de hauteur n et considérons $t = a(t_1, \ldots, t_n)$ de hauteur n+1. t se dérive en q(t) (dans A) et en q'(t) (dans A') si et seulement si

  $\exists q_1, \ldots, q_n$ dans Q et $q_1, \ldots, q'_n$ dans Q' tels que

  . pour tout i, $t_i \xrightarrow[A]{*} q_i(t_i), t_i \xrightarrow[A']{*} q'_i(t_i)$

  . $\exists r = (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}, \exists r' = (a(q'_1, \ldots, q'_n)[d'] \rightarrow q)' \in \mathcal{R}'$ avec d et d', deux descriptions valides pour $(t_i)_{1 \leq i \leq n}$.

  La propriété est supposée vraie pour les t_i, donc $t \xrightarrow[A]{*} q(t)$ et $t \xrightarrow[A']{*} q'(t)$ si et seulement si

  $t_i \xrightarrow[A_\times]{*} (q_i, q'_i)(t)$ et $\exists (a((q_1, q'_1), \ldots, (q_n, q'_n))|[d \triangle d'] \rightarrow (q, q'))$ dans $\mathcal{R}_\times$

  Et

  (d et d' sont valides pour $(t_i)_{1 \leq i \leq n}$) $\iff$ ($d \triangle d'$ est valide pour $(t_i)_{1 \leq i \leq n}$)

Exemple

$\Sigma = \{\bar{a}, a, \alpha, c\}$
$\quad\quad 2\ 1\ 1\ 3$

Un premier automate reconnaît la forêt dont les arbres possèdent un a au dessus d'un c dont les deux premières branches sont égales.
Le deuxième est similaire, mais reconnaît les arbres ayant un α au-dessus d'un c dont les deux dernières branches sont égales.

$\mathcal{R}_1$:
$\bar{a} \rightarrow q$
$c(q,q,q)[\perp] \rightarrow q$
$c(q,q,q)[\{\{1,2\},\{3\}\}] \rightarrow q_=$
$a(q_=) \rightarrow q_f$
$c(e_1,e_2,e_3)[\perp] \rightarrow q_f$ si q_f apparaît dans e_1, e_2, e_3

$\mathcal{R}_2$:
$\bar{a} \rightarrow q$
$c(q,q,q)[\perp] \rightarrow q$
$c(q,q,q)[\{\{1\},\{2,3\}] \rightarrow q_=$
$\alpha(q_=) \rightarrow q_f$
$c(e_1,e_2,e_3)[\perp] \rightarrow q_f$ si q_f apparaît dans e_1, e_2, e_3

Dans l'automate produit on trouvera notamment les règles $\bar{a} \rightarrow (q,q)$
$c((q,q),(q,q),(q,q))[\perp] \rightarrow (q,q)$
$c((q,q),(q,q),(q,q))[\{\{1,2\},\{3\}\}] \rightarrow (q_=,q)$
$c((q,q),(q,q),(q,q))[\{\{1\},\{2,3\}\}] \rightarrow (q,q_=)$
$c((q,q),(q,q),(q,q))[\{1,2,3\}] \rightarrow (q_=,q_=)$
$a((q_=,q)) \rightarrow (q_f,q)$
$\alpha((q,q_=)) \rightarrow (q,q_f)$
Ainsi que toutes les règles comprenant en partie gauche les états
$(q_=,q),(q,q_=),(q_=,q_=)\ldots$

2.4. 3 Complémentation

Proposition.

$REC_=$ n'est **pas fermée par complémentation.**

Preuve

Rappelons (exemple cité en introduction) que $REC_=$ contient la forêt B_{equil} définie par tous les arbres bien équilibrés construits sur une lettre de arité 0 ($\tilde{a}$) et une lettre de arité 2 (b). Nous montrons que le complémentaire de cette forêt n'appartient pas à $REC_=$

Supposons que l'automate $A = (\Sigma, Q, Fin, \mathcal{R})$ reconnaît $B_{desequil} = T(\Sigma)_0$ - B_{equil}. Pour tout arbre t, il existe q tel que $t \xrightarrow{*} q$, puisque qu'un arbre équilibré peut toujours être greffé sur un autre pour former un arbre déséquilibré, donc reconnu.

On appelle t_i, l'arbre équilibré de profondeur i, et $Q(t_i)$, l'ensemble des états atteints par t_i.
Pour chaque i, et chaque j avec $i \neq j$, $b(t_i, t_j)$ est reconnu, $b(t_i, t_i)$ ne l'est pas, donc $\exists q \in Q(t_i) \setminus Q(t_j), \exists q' \in Q(t_j) \setminus Q(t_i),\ b(q,q') \rightarrow q_f,\ q_f \in Fin$.

Les états étant en nombre fini, mais non les arbres équilibrés, il existe un état q_0 atteint par une infinité d'arbres équilibrés. Soit $N_0 = \{i | t_i \xrightarrow{*} q_0\}$.

Nous allons montrer que l'on peut construire une suite infinie d'états distincts $(q_k)_{k \geq 0}$ et d'ensembles infinis $N_k = \{i | \forall k' \leq k, t_i \xrightarrow{*} q_{k'}\}$.

Supposons la suite définie jusqu'au rang k fixé. Soit i_k tel que $i_k \in N_k$, et $X_k = Q(t_{i_k})$, les états atteints par t_{i_k}. Remarquons que pour tout $k' \leq k$, $q_{k'} \in X_k$.
On pose $N'_k = N_k \setminus \{i_k\}$
N_k est, par hypothèse de récurrence, infini; N'_k l'est également. Pour chaque $i \in N'_k$, $b(t_{i_k}, t_i)$ est reconnu.
On pose $Y_k = \{q' |\ \exists i \in N'_k,\ \exists q \in X_k,\ \exists q_f \in Fin,\ t_i \xrightarrow{*} q',\ b(q,q') \rightarrow q_f\}$
Y_k ne pouvant être infini, il comprend au moins un état q_{k+1} atteint par une infinité d'arbres t_i, $i \in N'_k$. On pose $N_{k+1} = \{i | i \in N'_k, t_i \xrightarrow{*} q_{k+1}\}$, ensemble infini.
D'autre part $Y_k \cap X_k = \emptyset$, sinon il y aurait des arbres équilibrés reconnus.
Comme $\{q_{k'} | k' \leq k\} \subseteq X_k$, $q_{k+1} \neq q'_k,\ k' \leq k$.

On pourrait ainsi construire une suite infinie d'états distincts.

2. 5 Stabilité par morphisme alphabétique

La définition des morphismes d'arbres a été donnée dans le premier chapitre. Nous considérerons maintenant des morphismes alphabétiques (l'image d'une lettre est une lettre) non nécessairement linéaires ni complets. Ces morphismes font apparaître ou disparaître la non linéarité immédiatement en-dessous d'une lettre. Nous montrons qu'il y a coïncidence entre les forêts reconnues par nos automates et les images par morphismes alphabétiques des forêts reconnaissables.

2.5. 1 Morphismes stricts

Proposition.

La famille $REC_=$ est **close par morphisme alphabétique strict.**

Preuve

Soit F, une forêt reconnue par l'automate $A = (\Sigma, Q, Fin, \mathcal{R})$ et $\phi : T(\Sigma) \to T(\Sigma')$, un morphisme alphabétique d'arbres.

Nous supposerons l'automate A complètement accessible. (pour tout état, il existe un arbre l'atteignant)

$\phi(F)$ sera reconnue par l'automate A_ϕ défini sur Σ' par :

$Q_\phi = Q$

$Fin_\phi = Fin$

$\mathcal{R}_\phi = \{(\phi(a)(q_{\tau_a(1)}, \ldots, q_{\tau_a(m)})[\tau.d] \to q) | (a(q_1, \ldots, q_n)[d] \to q) \in \mathcal{R}\}$

La forêt reconnue par l'automate A_ϕ sera notée F'

1) $(t \xrightarrow[A]{*} q) \Rightarrow (\phi(t) \xrightarrow[A\phi]{*} q)$

- si t est de hauteur nulle, alors $t = \bar{a} \in \Sigma_0$; il existe $(\bar{a}, [\bot] \to q) \in \mathcal{R}$ et $(\phi(\bar{a}))[\bot] \to q) \in \mathcal{R}_\phi$
- supposons la propriété vraie pour une hauteur n. Soit $t = a(t_1, \ldots, t_n)$ de hauteur n+1.

  $t \xrightarrow[A]{*} q(t) \Rightarrow$

  $\exists (a(q_1, \ldots, q_n)[d] \to q) \in \subset \mathcal{R}$,

  d est valide pour $(t_i)_{1 \leq i \leq n}$, et $\forall i \leq n, t_i \xrightarrow[A]{*} q_i$

  $\Rightarrow$

  $\exists (\phi(a)(t_{\tau(1)}, \ldots t_{\tau(n)})[\tau.d] \to q) \in R_\phi$, $\tau.d$ est valide pour $(t_{\tau(i)})_{1 \leq i \leq m}$

avec, d'après l'hypothèse de récurrence : $\phi(t_i) \xrightarrow[A_\phi]{*} q_i$

$\Rightarrow$

$\phi(t) \xrightarrow[A_\phi]{*} q$

Donc si t est reconnu par A, $\phi(t)$ est reconnu par A_ϕ, $\phi(F) \in F'$

2) $(t' \xrightarrow[A_\phi]{*} q) \Rightarrow (\exists t \in T(\sigma), \quad \phi(t) = t', \quad t \xrightarrow[A]{*} q)$

- t'=$\bar{a}'$ est une lettre de arité 0.
  $(\bar{a}'[\perp] \rightarrow q) \in \mathcal{R}_\phi \Rightarrow \exists (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}, \phi(a) = \bar{a}'$
  A est supposé complètement accessible, et donc $\exists t = a((t_i)_{1 \leq i \leq n})$ se dérivant en q. $\phi(t) = \bar{a}'$
- $t' = a'(t'_1, \ldots t'_m)$ de hauteur n+1, la propriété étant supposée vraie jusqu'à la hauteur n.
  $\exists (a'(q'_1, \ldots, q'_m)[d'] \rightarrow q) \in \mathcal{R}_\phi$
  d' est valide pour $t'_1, \ldots, t'_n, \quad t'_i \xrightarrow[A_\phi]{*} q'_i$
  $\Rightarrow$
  $\exists (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}, \quad \phi(a) = a', \quad \tau.d = d', \quad \forall i \leq m, q_{\tau(i)} = q'_i$
  $t'_i \xrightarrow[A_\phi]{*} q_{\tau(i)} \Rightarrow$
  pour chaque i entre 1 et n, l'automate étant complètement accessible, $\exists t_i, t_i \xrightarrow[A]{*} q_i(t_i)$.
  Si, de plus, $i \in \tau([1, m])$, alors, d'après l'hypothèse de récurrence, $\exists t_i, t_i \xrightarrow[A]{*} q_i, \phi(t_i) \xrightarrow[A_\phi]{*} q_i$
  $\Rightarrow$
  $\phi(a((t_i)_{1 \leq i \leq n})) = a'(t_{\tau(1)}, \ldots t_{\tau(m)})$ et $a((t_i)_{1 \leq i \leq n}) \xrightarrow[A]{*} q(t)$
  $F' \in \phi(F)$

2.5. 2 Image de *REC*

Proposition.

REC$_=$ *est l'***image** *de REC par morphisme alphabétique.*

Corollaire.

REC$_=$ *est la clôture de* **REC** *par union, intersection et morphisme alphabétique strict.*

Preuve

Une forêt F de la famille $REC_=$, reconnue par $A = (\Sigma, Q, Fin, \mathcal{R})$. Soit l'automate ascendant classique A', défini de la façon suivante :

- $\Sigma' = \bigcup_{n \geq 0} \Sigma'_n$, $\quad \Sigma'_n = \{(a,d) | a \in \Sigma_m, d \in ED_{m,n}\}$
- $Q' = Q \quad Fin' = Fin$
- $\mathcal{R}' = \{((a,c)(q'_1, \ldots, q'_n) \rightarrow q) | \exists (a(q_1, \ldots, q_m)[d] \rightarrow q) \in \mathcal{R}, q_{d(i)} = q'_i\}$

et le morphisme ϕ, qui à (a,d), de arité n, associe a, de arité m, avec la surjection d de $[1,m]$ dans $[1,n]$: à tout élément de $ED_{n,m}$ on associe une application de $[1,n]$ dans $[1,m]$).

Pour prouver que $F = \phi(F')$, on se ramène à la preuve précédente en remarquant que l'automate A' peut être assimilé à un automate à tests dont toutes les descriptions d'égalité seraient $\perp$. En lui appliquant alors la construction décrite à la section précedente, on obtient l'automate A.

2.5. 3 Morphismes effaçants

Définition.

Nous appellerons morphisme gommant, un morphisme d'arbres tel que l'image de toute lettre est soit une variable, soit la lettre elle-même, avec la torsion identité.

La définition est légèrement différente de celle de morphisme "effaçant", puisque l'on impose à un morphisme gommant d'être égal à l'identité sur les lettres non effacées.

Lemme.

Tout morphisme alphabétique se décompose en un morphisme alphabétique strict et un morphisme gommant.

Pour établir la clôture par morphisme alphabétique, il nous a paru plus clair d'utiliser la décomposition d'un morphisme alphabétique quelconque en un morphisme gommant et un morphisme alphabétique strict éventuellement non linéaire.

La décomposition d'un morphisme quelconque ϕ peut se faire de la façon suivante :

- si $\phi(a(x_1, \ldots, x_n)) = u(x_{i_1}, \ldots, x_{i_m})$, on choisit la décomposition :
  $\phi_1(a(x_1, \ldots, x_n)) = a(x_1, \ldots, x_n)$ et $\phi_2(a(x_1, \ldots, x_n)) = u(x_{i_1}, \ldots, x_{i_m})$

- si $\phi(a(x_1,\ldots,x_n)) = x_i$, alors $\phi_1(a(x_1,\ldots,x_n)) = x_i$

Proposition.

$REC_=$ est stable par morphisme gommant

Corollaire.

$REC_=$ est la clôture de REC par morphisme alphabétique (au sens large)

Preuve de la proposition

$A = (\Sigma, Q, F, \mathcal{R})$ est un automate reconnaissant une forêt $\mathcal{F}$

Pour chaque état $q \in Q$, on construit l'ensemble des états q' tels que soit $q' = q$, soit il existe un arbre u, avec $u(q_1,\ldots,q_n) \xrightarrow{*} q$, $\phi(u(x_1,\ldots,x_n)) = x_i$,et $q' = q_i$

Nous notons $Rempl(q)$ cet ensemble.

L'automate A' reconnaissant l'image de F que nous construisons possède les mêmes états que A.

Pour l'ensemble des états terminaux, on prend :

$F' = \bigcup_{q \in F} Rempl(q)$

et pour les règles de transition :

$\{(a(q'_1,\ldots,q'_n)[d] \to q) \mid (a(q_1,\ldots,q_n)[d] \to q) \in \mathcal{R},$
$q'_i \in Rempl(q_i), d \text{ valide pour } q'_1,\ldots,q'_n\}$

La dernière condition assure que, quand q est "impliqué" dans une égalité, on a choisi le même remplaçant pour chaque occurrence.

2.5. 4 Morphismes inverses

Proposition.

La classe $REC_=$ n'est pas stable par morphisme alphabétique inverse

Preuve

Considérons $\Sigma\{b, a, \alpha, \bar{a}\}$ et $\Sigma' = \{b, a, \bar{a}\}$
$\quad 2\ 1\ 1\ 0 \qquad\qquad 2\ 1\ 0$

et un morphisme ϕ, identité sur b et $\bar{a}$, et tel que l'image de a et de α est a.

L'image inverse par ϕ de $\{b(a^*.\bar{a}, a^*.\bar{a})|$ les deux branches sont égales$\}$ est l'ensemble des arbres dont les branches, constituées de a et α, sont nécessairement de longueurs égales. Cet ensemble ne peut clairement pas être reconnu par les automates que nous avons définis.

Chapitre 3

Stratégie à contrôle fort

3. 1 Définitions

Les automates de la classe $REC_=$ présentent un non-déterminisme en raison de la possiblité de choisir entre une règle "forte", c'est à dire reconnaissant une relation d'égalité forte entre les sous-arbres, et des règles plus faibles (de descriptions moins précises) mais également valides.

D'autre part, la classe des forêts reconnues n'est pas close par complémentaire. D'une façon informelle, c'est également dû à la possibilité d'utiliser une règle moins forte, quand elle existe.

Dans ce chapitre nous choisissons une autre stratégie dans l'application des règles de l'automate, la finalité étant de pouvoir doter les automates d'un comportement déterministe, et d'obtenir une classe de forêts close par complémentaire. Nous obtiendrons des automates capables d'imposer des différences entre sous-arbres.

3.1. 1 Stratégie à contrôle fort

Un arbre t est reconnu par **stratégie à contrôle fort**, par un automate à tests d'égalités si et seulement si il existe un état final q tel que $t \xrightarrow{*} q(t)$, où $\xrightarrow{*}$ désigne la clôture réflexive et transitive de $\rightarrow$:

$(t \rightarrow t') \iff$

$\exists u \in T(\Sigma)_1, \quad \exists a \in \Sigma_n, \quad \exists (t_i)_{1 \leq i \leq n} \in T(\sigma)_0,$

$\exists q$ et $(q_i)_{1 \leq i \leq n} \in Q$

$\exists (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}$ tels que

$t = u(a(q_1(t_1), \ldots, q_n(t_n))), \quad t' = u(q(a(t_1, \ldots, t_n)))$

d est valide pour $(t_i)_{1 \leq i \leq n}$

$\forall (a(q_1, \ldots, q_n)[d'] \rightarrow q') \in \mathcal{R}, \; d' > d,$ alors d' non valide pour $t_1, \ldots, t_n$

C'est à dire que pour un profil donné, on ne peut appliquer qu'une des règles, dont la description d'égalités est de précision maximale.

3.1. 2 Complétion

La définition d'un automate complet est indépendante de la stratégie choisie. Par contre la méthode de complétion est différente. On ne peut ajouter simplement une règle menant à un état puits, puisque cette règle est susceptible d'être appliquée en remplacement d'une autre, de description moins précise.

Il faut, ici, traiter les règles non de façon isolée, mais en considérant globalement

toutes les règles ayant même profil.

Notons $\mathcal{R}_{a,q_1,\ldots,q_n} = \{(a(q_1,\ldots,q_n)[d]\rightarrow q) | d \in ED_n, q \in Q\}$
et $ED_{a,q_1,\ldots,q_n} = \{d | d \text{ figure dans } \mathcal{R}_{a,q_1,\ldots,q_n}\}$.
Les états qu'aurait pu atteindre un arbre $a(t_1,\ldots,t_n)$ tel que d est la description exacte de $t_1,\ldots,t_n$, sont tous les états figurant en partie droite de règles des descriptions d', où d' est un minorant maximal de d.

L'ensemble $\mathcal{R}_{a,q_1,\ldots,q_n}$ va alors être complété par :
$\{(a(q_1,\ldots,q_n)[d]\rightarrow q) \mid \exists (a(q_1,\ldots,q_n)[d']\rightarrow q) \in \mathcal{R}_{a,q_1,\ldots,q_n}$, d' est un minorant maximal de d dans l'ensemble $ED_{a,q_1,\ldots,q_n}\}$

Un état puits est ajouté pour les descriptions minimales :
$\{(a(q_1,\ldots,q_n)[d]\rightarrow puits) \mid \nexists d' \in ED_{a,q_1,\ldots,q_n}, d' \leq d\}$

Propriété.

Pour un automate complet, de stratégie à contrôle fort, les règles pouvant s'appliquer à un arbre $a(t_1,\ldots,t_n)$ sont les règles portant la description exacte de $t_1,\ldots,t_n$

3.1. 3 Expression des conditions à l'aide de formules

Comme dans les cas des automates à stratégie faible, on peut exprimer les conditions à l'aide de formules dont les variables désignent les fils directs de la lettre. Ces expressions peuvent utiliser la conjonction, la disjonction et la négation, donc peuvent exprimer la différence entre sous-arbres.
L'équivalence entre les deux formulations se montre à partir d'une construction tout à fait analogue à celle décrite dans le chapitre précédent à propos des formules ne contenant pas de négation : Chaque formule f peut être exprimée "en extension", c'est à dire qu'on lui associe un ensemble fini de descriptions correspondant à toutes les combinaisons d'arbres validant la formule :

$$E_f = \{d | (\text{d est valide pour } (x_i)_{1\leq i\leq n} \qquad f(t_1,\ldots,t_n) \text{ est vraie}\}$$

Pour toute règle r de formule f, d'un automate $A_{formules}$, on construit les règles à

descriptions

$$\mathcal{R}_r = \{(a(q_1, \ldots, q_n)[d] \rightarrow q) \mid (a((q_i)_{1 \leq i \leq n})[d] \rightarrow q,\ d \in E_f\}$$

Mais il faut, cette fois, compléter $\mathcal{R}_r$ par :

$$\{(a(q_1, \ldots, q_n)[d] \rightarrow puits) | d \notin E_f\}$$

pour tenir compte de ce que les formules contiennent des négations. Si un n-uple d'arbres ne respecte pas ces "négations d'égalité", il faut lui fournir une règle adéquate, menant à un état puits.
De cette manière la stratégie à contrôle fort qui impose de tenir compte de toutes les égalités, force à utiliser cette règle.

L'opération inverse, associant un automate à formules à un automate à descriptions est plus délicate que dans le cas de la stratégie faible. Nous supposerons que l'automate $A_{descriptions}$ est complet.
Cette supposition est indispensable pour que la construction ci-dessous soit valable.
On peut alors transformer chaque règle à description en une règle à formule. La formule associée à une description d sera la formule conjonctive :

$$f_d = \Big(\bigwedge_{\{i,j \mid \exists p \in d, i \in p, j \in p\}} \#i = \#j\Big) \wedge \Big(\bigwedge_{\{i,j \mid \nexists p \in d, i \in p, j \in d\}} \neg(\#i = \#j)\Big)$$

Exemples

Nous donnons un exemple de transformation d'un automate dont les conditions sont exprimées par des descriptions en son équivalent utilisant des formules.
Nous écrirons les règles concernant une lettre c de arité 3 et un seul état q en partie gauche.
Si les règles sont les suivantes :
(1) $c(q,q,q)[\{1,2,3\}] \rightarrow q_f$
(2) $c(q,q,q)[\{1,2\},\{3\}] \rightarrow q$
(3) $c(q,q,q)[\{1\},\{2\},\{3\}] \rightarrow q_f$
Il faut ajouter, pour que l'automate soit complet :
(4) $c(q,q,q)[\{1\},\{2,3\}] \rightarrow q_f$
(5) $c(q,q,q)[\{1,3\},\{2\}] \rightarrow q_f$
(6) $c(q,q,q)[\{1\},\{2,3\}] \rightarrow q_f$

La condition de la règle 1 est transformée en $\#1 = \#2 \wedge \#1 = \#3 \wedge \#2 = \#3$

Celle de la règle 2 en
$\#1 = \#2 \wedge \#2 \neq \#3 \wedge \#1 \neq \#3$
Pour la règle 3
$\#1 \neq \#2 \wedge \#1 \neq \#3 \wedge \#2 \neq \#3$
On voit l'importance de considérer un automate complet. Si on se limite aux traductions des règles 1, 2 et 3, aucune transition n'est prévue dans le cas où $\#2 = \#3$, tous deux différents de $\#3$, alors que dans l'automate à descriptions on peut utiliser la règle 3 pour un tel cas de figure.
Pour l'automate à formules, il faut donc compléter par les traductions des règles 4, 5 et 6.

Nous donnons l'ecriture de l'automate avec descriptions qui reconnaît les forêts d'arbres déséquilibrés sur les trois lettres graduées $\{\bar{a}, a, b\}$
0 1 2

$Q = \{q, q_f\}$ $Fin = \{q_f\}$
Les règles de l'automate :
$\bar{a}[\perp] \rightarrow q$
$a(q)[\perp] \rightarrow q$
$a(q_f)[\perp] \rightarrow q_f$
$b(q,q)[\perp] \rightarrow q_f$ en raison de l'existence de la règle suivante, celle-ci n'est applicable que si les sous-arbres sont distincts
$b(q,q)[\{\{1,2\}\}] \rightarrow q$
$b(q,q_f)[\perp] \rightarrow q_f$
$b(q_f,q)[\perp] \rightarrow q_f$
$b(q_f,q_f)[\perp] \rightarrow q_f$

3.1. 4 $REC_{\neq}$: classe reconnue

Définition.

On notera $\mathbf{REC}_{\neq}$ *la famille des forêts reconnues par stratégie forte, par un automate à tests.*

3. 2 Déterminisme

Définition.

Un automate à test est dit **déterministe pour la stratégie forte** *si et seulement si pour toute partie gauche* $a, q_1, \ldots, q_n, [d]$

- Il existe au plus une règle dans $\mathcal{R}$

- Pour toute $d' \in ED_n$, *si il existe une règle de partie gauche* $a, q_1, \ldots, q_n, [d']$, *alors il existe une règle de partie gauche* $a, q_1, \ldots, q_n, [d \Delta d']$

Cette définition assure un vrai déterminisme dans la reconnaissance d'un arbre : pour chaque n-uple d'arbres $(t_i)_{1 \leq i \leq n}$ et chaque n-uple d'états $(q_i)_{1 \leq i \leq n}$, on peut trouver une règle unique de description maximale (au sens de la précision).

Proposition.

Pour toute forêt F reconnue par stratégie forte, il existe un automate déterministe A tel que F soit reconnue par A muni de la stratégie forte.

3.2. 1 Algorithme de déterminisation

On suppose que A est un **automate complet** reconnaissant F. L'algorithme de déterminisation est identique à celui de pseudo-déterminisation développé au chapitre précédent.

$Q_0 \leftarrow \{X | \exists \bar{a} \in \Sigma_0,\ X = \{q \mid (\bar{a}[\bot] \rightarrow q) \in \mathcal{R}\}\}$
$\mathcal{R}_0 \leftarrow \{(\bar{a}[\bot] \rightarrow X) | X = \{q | (a \rightarrow q) \in \mathcal{R}\}\}$
$i \leftarrow 0$

répéter :
  $i \leftarrow i+1$
  $\bar{Q}_i \leftarrow \bar{Q}_{i-1} \cup$
    $\{X |\ \exists a \in \Sigma_n, \exists (X_i)_{1 \leq i \leq n}$ dans $Q_{i-1}, \exists d \in ED_n,$
      $X = \{q | \exists (q_i)_{1 \leq i \leq n},\ q_k \in X_k, (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}\}$
    $\}$

  $\bar{\mathcal{R}}_i \leftarrow \bar{\mathcal{R}}_{i-1} \cup$
    $\{(a(X_1, \ldots, X_n)[d] \rightarrow X) |$
      $a \in \Sigma_n,\quad (X_i)_{1 \leq i \leq n}$ dans $\bar{Q}_{i-1},\quad d \in ED_n,$
      $X = \{q | \exists (q_i)_{1 \leq i \leq n},\ q_k \in X_k,\ (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}\}$
    $\}$
jusqu'à ce que $\bar{Q}_i = \bar{Q}_{i-1}$ et $\bar{\mathcal{R}}_i = \bar{\mathcal{R}}_{i-1}$
$\bar{Q} \leftarrow \bar{Q}_i$
$\tilde{\mathcal{R}} \leftarrow \mathcal{R}_i$
$\bar{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \setminus \{(a(X_1, \ldots, X_n)[d] \rightarrow X) |\ d$ *non valide pour* $X_1, \ldots, X_n\}$
$\bar{F} \leftarrow \{X | X \cap F \neq \emptyset\}$

Preuve

Soit t, un arbre défini sur $T(\Sigma)$, et X, un état de $\bar{Q}$, nous allons montrer que :
$(X = \{q | t \xrightarrow[A]{*} q(t)\}) \iff (t \xrightarrow[\bar{A}]{*} X(t))$
- si $t = \bar{a}$ est de hauteur nulle, $\bar{\mathcal{R}}_0$ contient une seule règle $(\bar{a}[\bot] \rightarrow X)$ où X est l'ensemble des états pouvant être atteints par $\bar{a}$ dans A.
- On suppose la propriété vraie pour une hauteur n et on pose $t = a(t_1, \ldots, t_n)$, supposé de hauteur n+1.
d désignera la description exacte de $t_1, \ldots, t_n$
On remarquera que l'automate $\bar{A}$ est également un automate complet.
$t \xrightarrow[\bar{A}]{*} X(t)$
$\iff$
$\exists (X_i)_{1 \leq i \leq n} \in \bar{Q},$
$(a(X_1, \ldots, X_n)[d] \rightarrow X) \in \bar{\mathcal{R}}, t_i \xrightarrow[\bar{A}]{*} X_i$
$\iff$ ***(appliquant l'hypothèse de récurrence)***
$t_i \xrightarrow[\bar{A}]{*} X_i = \{q_i | t_i \xrightarrow[A]{*} q_i\},$
$(a(X_1, \ldots, X_n)[d] \rightarrow X) \in \bar{\mathcal{R}},$

$\Longleftrightarrow$
$t_i \xrightarrow[\bar{A}]{*} X_i = \{q_i | t_i \xrightarrow[A]{*} q_i\}$,
$(q \in X \iff \exists q_1, \ldots, q_n, q_i \in X_i, (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R})$
$\Longleftrightarrow$
$X = \{q | t \xrightarrow[A]{*} q\}$

L'ensemble des états terminaux est constitué par les ensembles contenant un état terminal de l'automate de départ.
La forêt reconnue est donc la même.

3.2. 2 Algorithme de complétion-déterminisation

Il est possible de réaliser en une seule opération la déterminisation et la complétion d'un automate. L'algorithme est alors le suivant

$\bar{Q}_0 \leftarrow \{X | \exists \bar{a} \in \Sigma_0, X = \{q |\ (\bar{a}[\perp] \rightarrow q\} \in \mathcal{R}\}$
$\bar{\mathcal{R}}_0 \leftarrow \{(\bar{a}[\perp] \rightarrow X) | \bar{a} \in \Sigma_0, X = \{q |\ q \in X, (\bar{a}[\perp] \rightarrow q) \in \mathcal{R}\}$
$i \leftarrow 0$
Répéter :
$i \leftarrow i + 1$
$\bar{Q}_i \leftarrow \bar{Q}_{i-1} \qquad \mathcal{R}_i \leftarrow \mathcal{R}_{i-1}$
Pour tout a dans Σ, tout $(X_i)_{1 \leq i \leq n}$ dans $\bar{Q}_{i-1}$, tout d dans ED_n
faire
$X \leftarrow \{q |$
$\exists (q_i)_{1 \leq i \leq n}, \qquad \exists d' \leq d, q_i \in X_i,$
$(a(q_1, \ldots, q_n)[d'] \rightarrow q) \in \mathcal{R},$
$\nexists d'', d' < d'' \leq d, (a(q_1, \ldots, q_n)[d''] \rightarrow q'') \in \mathcal{R}$
$\}$
$\bar{Q}_i \leftarrow \bar{Q}_i \cup \{X\}$
$\bar{\mathcal{R}}_i \leftarrow \mathcal{R}_i \cup \{(a(X_1, \ldots, X_n)[d] \rightarrow X)\}$
fait
Jusqu'à ce que $\bar{Q}_i = \bar{Q}_{i-1}$ et $\bar{\mathcal{R}}_i = \bar{\mathcal{R}}_{i-1}$
$\bar{Q} \leftarrow \bar{Q}_i \quad \bar{\mathcal{R}} \leftarrow \bar{\mathcal{R}}_i$
$\bar{Fin} \leftarrow \{X | X \cap Fin \neq \emptyset\}$

Nous ne développons pas la preuve qui est assez ressemblante à celle que nous venons

de voir.

3.2. 3 Automates normalisés

La définition des automates normalisés et des normalisés-complets a été donnée au chapitre précédent.

Proposition.

Toute forêt F, reconnue par stratégie forte, pourra être reconnue par un automate déterministe et normalisé-complet, c'est à dire :
$\forall a \in \Sigma_n, \quad \forall (q_i)_{1 \leq i \leq n} \in Q, \quad \forall d \in ED_n,$
il existe une règle (unique) $(a(q_1, \ldots, q_n)[d] \rightarrow q)$ *si et seulement si d est valide pour* $(q_i)_{1 \leq i \leq n}$

Preuve

Soit A, un automate déterministe reconnaissant une forêt F. La normalisation de l'automate déterministe est simple : elle revient à supprimer toutes les règles $(a(q_1, \ldots, q_n)[d] \rightarrow q)$ telles que d n'est pas valide pour $q_1, \ldots, q_n$. Ces règles imposent, pour s'appliquer, l'existence d'un arbre t atteignant deux états distincts ce qui, dans un automate déterministe, est impossible.

On remarquera que dans le cas de la classe $REC_=$ la normalisation demandait une construction plus complexe, puisque nous n'avions pas d'automates déterministes. L'algorithme de normalisation que nous avions donné consistait, sans le dire, à construire un automate qu'on aurait pu aisément rendre déterministe, si on l'avait de surcroît muni de la stratégie forte. Il donnait la construction de base permettant de prouver que $REC_=$ est inclus dans $REC_{\neq}$ et que tous ses éléments peuvent être reconnus par un automate déterministe normalisé.

3.2. 4 Inclusion de $REC_=$

Proposition.

$REC_{\neq}$ *contient* $REC_=$.

Preuve

Tout d'abord, il est bien évident qu'un automate donné reconnaît, dans le cas général, des forêts différentes suivant qu'on le dote d'un comportement à stratégie forte ou faible.

Considérant une forêt F de $REC_{=}$, reconnue par un automate A (stratégie faible), nous pouvons construire un automate $\bar{A}$ non-déterministe, à stratégie forte, reconnaissant F.

On complète l'ensemble de règles $\mathcal{R}$ en permettant, pour chaque description, d'atteindre tous les états qui pouvaient l'être dans A en utilisant une règle de même profil, mais de description moins précise.

$$\bar{\mathcal{R}} = \{(a(q_1,\ldots,q_n)[d]\rightarrow q) \mid \exists d' \leq d,\ (a(q_1,\ldots,q_n)[d']\rightarrow q) \in \mathcal{R}\}$$

Cet automate est complet.

On montre alors que $(t \xrightarrow[A]{*} q \iff t \xrightarrow[\bar{A}]{*} q)$

Pour les lettres de arité 0, la stratégie ne joue aucun rôle.

Si l'on suppose la proprété vraie pour une hauteur n, examinons les états atteints par $a(t_1,\ldots,t_n)$ (d désigne la description exacte des t_i) : $t \xrightarrow[A]{*} q$

$\iff$

$\exists d' \in ED_n$ valide pour $t_1,\ldots,t_n$, $\exists (q_i)_{1\leq i\leq n}$,

$(a(q_1,\ldots,q_n)[d']\rightarrow q) \in \mathcal{R}$, $t_i \xrightarrow[A]{*} q_i$

$\iff$

$(a(q_1,\ldots,q_n)[d]\rightarrow q) \in \bar{\mathcal{R}}$

$t_i \xrightarrow[\bar{A}]{*} q_i$

$\iff$

$t \xrightarrow[\bar{A}]{*} q$

3. 3 Propriétés de clôture booléenne

3.3. 1 Par union et intersection

Proposition.

La famille $\mathbf{REC}_{\neq}$ *est* **close par union et par intersection.**

Preuve

La preuve de la stabilité par union et intersection se fait par un produit d'automates. Soient F et F', deux forêts de $\mathbf{REC}_{\neq}$, reconnues respectivement par les automates $A = (\Sigma, Q, Fin, \mathcal{R})$, et $A' = (\Sigma, Q', Fin', \mathcal{R}')$
A et A' seront supposés être déterministes et normalisés-complets.

Les automates "produit" $A_\cup$ et $A_\cap$ sont définis par :

$$Q_\cup = Q_\cap = Q \times Q'$$
$$\mathcal{R}_\cup = \mathcal{R}_\cap = \{(a(q_1, q'_1), \ldots, (q_n, q'_n))[d] \rightarrow (q, q'))| \\ (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}, (a(q'_1, \ldots, q'_n)[d] \rightarrow q') \in \mathcal{R}'\}\}$$
$$Fin_\cup = Q \times Fin' \cup Fin \times Q'$$
$$Fin_\cap = Fin \times Fin'$$

Les automates étant supposés normalisés-complets, on est assuré de l'existence d'une règle pour chaque n-uple d'états et chaque description (si la description est valide pour le n-uple d'états). Le déterminisme nous permet de ne faire le produit que de règles portant la même description, le produit d'automates est donc plus simple que dans le cas des automates non déterministes, ou à stratégie faible.
Pour tout arbre t, $t \rightarrow (q, q')$ dans l'automate produit si et seulement si $t \rightarrow q$ dans A, et $t \rightarrow q'$ dans A'. A fortiori, par clôture transitive et reflexive,
$t \xrightarrow[A_\cup]{*} (q, q') \iff t \xrightarrow[A_\cap]{*} (q, q') \iff (t \xrightarrow[A]{*} q$ et $t \xrightarrow[A']{*} q')$

3.3. 2 Clôture par complémentation

Proposition.

La famille $\mathbf{REC}_{\neq}$ *est close par complémentaire.*

Nous avons montré qu'il était toujours possible de prendre des automates déterministes et normalisés, donc complets. Pour tout arbre t de Σ_0, il existe donc un unique état q tel que $t \xrightarrow[A]{*} q(t)$. Comme dans tous les cas de reconnaisseurs déterministes et complets, en définissant $Fin' = Q - Fin$, on obtient un automate reconnaissant le complémentaire.

3. 4 Compter les arbres

Dans les algorithmes suivants, nous aurons besoin de compter les arbres susceptibles d'atteindre un état donné.
Nous avons des règles qui imposent des différences. Or, pour satisfaire ces différences, il faut disposer de suffisamment de "candidats". On peut très bien construire la règle $\bar{a} \rightarrow q$ comme seule règle dont q est partie droite, et ensuite imposer par une règle $(b(q,q)[\#1 \neq \#2] \rightarrow \ldots)$ une différence entre des arbres atteignant q, mais cette condition ne sera jamais satisfaite, faute de l'existence de suffisamment d'arbres atteignant q.
Pour pouvoir décider du vide, on ne peut se contenter de savoir si un état est accessible, il est nécessaire d'avoir une indication sur le nombre d'arbres qui l'atteignent. Ce sera l'objet de la section suivante, mais précisons que ce comptage est également indispensable pour construire un automate reconnaissant l'image par morphisme d'un élément de $REC_{\neq}$
Nous présentons donc un certain nombre de définitions et de propriétés dont l'objet est de simplifier l'écriture des comptages effectués par la suite.

3.4. 1 Description réduite

Définition.

Soit d, une description appartenant à ED_n, et Pos, un ensemble d'indices de $[1,n]$. On appellera **description réduite** *aux positions de Pos l'ensemble $d/Pos = \{p \cap Pos | p \in d, p \cap Pos \neq \emptyset\}$,*
donc la "projection" de d sur un certain nombre de positions.

Nous nous servirons des descriptions réduites pour séparer, dans une règle, les différentes parties d'une description, suivant l'état concerné.

Définition.

Soit $(q_i)_{1 \leq i \leq n}$, un n-uple d'états, d, une description appartenant à ED_n, et q, un état apparaissant dans le n-uple. Nous appellerons description **réduite à q relativement au n-uple $q_1, \ldots, q_n$**, *la description d réduite à l'ensemble des positions de q à l'intérieur du n-uple.*

Notation :

La description d réduite à q, relativement au n-uple $q_1, \ldots, q_n$ sera généralement

notée
$d/q = d/Pos(q)$, où $Pos(q) = \{i | q_i = q\}$
Dans cette notation simplifiée le n-uple n'apparaît pas. Il n'y aura, en principe, pas d'ambiguïté sur le n-uple que l'on considère.

3.4. 2 Cas des automates normalisés

Nous considèrerons toujours que nous avons affaire à des automates déterministes normalisés. Dans ce cadre :

- la description d est toujours valide pour le n-uple d'états
- la description exacte des sous-arbres $t_1, \ldots, t_n$ est inférieure à celle de $q_1, \ldots, q_n$

Chaque partie de la description désigne différentes positions d'un même état. Une description réduite est alors un sous-ensemble de la description globale.

On a, de plus,
$d = \bigcup_{\text{q, état du n-uple}} d/q$

Il en ressort la propriété suivante :

Propriété.

Soit d, une description, et $(q_i)_{1 \leq i \leq n}$, un n-uple d'états. Soient n éléments $x_1, \ldots, x_n$, dont la description exacte est inférieure à celle des q_i ($q_i \neq q_j \Rightarrow x_i \neq x_j$)

d est une description valide (resp. la plus précise parmi les éléments de ED_n) pour $(x_i)_{1 \leq i \leq n}$ si et seulement si pour chaque état q du n-uple, d/q est une description valide (resp. la plus précise parmi les réductions des éléments de ED_n) pour les éléments situés aux positions $Pos(q)$.

Ces propriétés justifient que nous nous intéressions séparément aux arbres atteignant des états différents.
Nous supposerons donc maintenant que les descriptions sont liées à des règles "homogènes à gauche", c'est à dire dont tous les états en partie gauche sont identiques. (si tel n'était pas le cas, il faudrait considérer toutes les descriptions réduites). Chacun des points développés ci-dessous peut être transposé pour des descriptions réduites.

Etant donnés une description d, le cardinal de d, noté $|d|$, vérifie notamment :

Propriété.

Si d est la description la plus précise de $x_1, \ldots, x_n$, alors figurent parmi les x_i au moins $|d|$ valeurs distinctes.

De cette propriété, en somme triviale, retenons trois conséquences.
Si $r = (a(q, q, \ldots, q)[d] \rightarrow q')$, alors
- Il faudra que , au minimum, $|d|$ sous-arbres distincts atteignent q pour que d puisse être la description la plus précise et que la règle r puisse s'appliquer.

- Si $|d|$ sous-arbres distincts $t_1, \ldots, t_{|d|}$ atteignent q, alors la règle r concerne $|d|!$ arbres, obtenus en permutant les valeurs $t_1, \ldots, t_{|d|}$

(ces deux propriétés servent pour décider si un état est accessible).

- Si un arbre t possède $|d|$ antécédents distincts pour un morphisme ϕ, alors on peut construire un n-uple d'antécédents de t, tel que d est la description la plus précise pour $t_1, \ldots t_n$.

Inversement, on peut définir l'ensemble des descriptions susceptibles d'être les plus précises pour $x_1, \ldots, x_n$ quand k valeurs distinctes figurent parmi $x_1, \ldots, x_n$. Nous noterons cet ensemble :
$Desc(k) = \{d | d \in ED_n, \quad |d| = k\}$

Ainsi, si $\top$ est la description la plus précise de $x_1, \ldots, x_n$, (i.e. ils sont tous égaux), et si x est l'image de k valeurs distinctes (pour nous il s'agira d'image par morphisme), alors pour toute description d supérieure (ou égale) à un élément de $Desc(k)$, on peut construire un n-uple d'antécédents de x, dont d est la description la plus précise.

Il faut généraliser cette propriété : si d est la description la plus précise pour $x_1, \ldots, x_n$, et si chaque x_i est l'image de k valeurs distinctes, alors pour toute description d' comprise (au sens large) entre d et un élément de $Desc_d(n)$, il existe un n-uple d'antécédents de $x_1, \ldots, x_n$ dont d' est la description la plus précise.
Avec $Desc_d(k) = \{d' |\ d' \leq d,\ \forall p \in d, p$ *contient au plus k parties de* $d'\}$
Remarque : $Desc(k) = Desc_\top(k)$

3. 5 Problèmes de décision

Nous présentons les solutions pour deux problèmes de décision : les problèmes du vide et de la finitude d'une forêt reconnue par l'un de nos automates.

3.5. 1 La forêt reconnue est-elle vide ?

Proposition.

Pour tout automate à tests A, la propriété "$\mathcal{F}(A) = \emptyset$" est décidable.

$\mathcal{F}(A)$ désigne la forêt reconnue par stratégie à contrôle fort.

Nous supposerons l'automate A **déterministe**.
Pour les autres cas (forêt reconnue par stratégie faible ou automate non-déterministe) nous avons donné les constructions permettant de se ramener à un automate déterministe muni de la stratégie à contrôle fort.
Nous développerons donc un algorithme pour les automates déterministes, et nous étudierons ensuite la complexité dans le cas général.

Dans le cas d'automates ascendants classiques, une règle peut être satisfaite si et seulement si tous ses états en partie gauche sont accessibles. L'algorithme de décision du vide, ou encore de décision d'accessibilité d'un état, consiste donc à construire les ensembles d'états accessibles, jusqu'à stabilisation.
Dans l'algorithme que nous construisons ici, il faudra faire intervenir le nombre d'arbres pouvant accéder à q. En effet, une règle ne peut pas être satisfaite si le nombre d'arbres atteignant q est insuffisant. C'est le cas quand cette règle "impose" une différence entre certains fils. Nous retrouvons ici une notion que nous avons déjà introduite dans la section précédente, à savoir le nombre minimum d'arbres devant atteindre q, en dessous duquel une description donnée ne peut être "la plus précise".

Nous supposerons que l'automate est déterministe et normalisé, puisqu'il existe une méthode de déterminisation et de complétion. Cette supposition n'est, bien sûr, pas sans conséquence pour la complexité de l'algorithme. Celui qui est développé ici est linéaire par rapport au nombre d'états, ce qui ne saurait être le cas sur des automates non-déterministes. Notons que l'algorithme de déterminisation est, comme dans le cas des automates classiques, exponentiel.

Rappelons que, relativement à un n-uple d'états $(q_i)_{1 \leq i \leq n}$, on peut définir la description d réduite à q, comme l'ensemble des parties dans lesquelles une position de q est présente. Pour une règle r de description d, la quantité $|d/q|$ représentera le nombre d'états devant atteindre q, pour pouvoir construire un n-uple d'arbres dont d/q serait la description exacte. On peut donc énoncer :

3.5. 2 Quelques lemmes techniques

Lemme.

Une règle r, de description d est satisfaite si et seulement si chaque état apparaissant en partie gauche est atteint par au moins $|d/q|$ arbres distincts.

Nous compterons donc les arbres, mais heureusement de façon bornée ...

Lemme.

Soit amax, l'arité maximale de l'alphabet Σ, si un état est accessible par au moins amax arbres distincts, alors pour toute description, on peut construire un k-uple de ces arbres dont d/q est la description exacte.

En effet d est de cardinal maximal $amax$, ainsi donc que toutes ses réductions.
Pour pouvoir compter les arbres, il manque encore un outil : si une règle est accessible, combien d'arbres "produit-elle" ?
Là encore, étudions les différentes descriptions réduites :
- Si un état est atteint par moins de $|d/q|$ arbres, il n'existe aucun k-uple de description exacte d/q.
- S'il est atteint par plus de $|d/q|$ arbres (soit n), on peut construire $A_n^{|d/q|}$, le nombre d'arrangements de $|d/q|$ éléments pris parmi n.

Notation

Pour toute règle r, de description d, et dont la partie gauche concerne l états distincts, $q_1, \ldots, q_l$. Nous noterons :
$Prod_r(n_1, \ldots, n_l) = \prod A_{n_l}^{|d/q_l|}$
le nombre de productions de r quand q_l est atteint par n_l arbres.

Lemme.

Si $Prod_r(n_1, \ldots, n_l) \neq 0$, et si $\exists j, n_j > amax$ alors $Prod_r(n_1, \ldots, n_l) \geq amax$.

Il suffit de remarquer que $|d/q| \leq amax$ et donc $A_{amax}^{|d/q|} \geq amax$

Lemme.

Si $Prod_r(n_1, \ldots, n_l) \neq 0$, *alors* $Prod_r(n'_1, \ldots n'_l)0$, *où* $n'i = min(n_i, amax)$

Ce lemme découle du précedent. Si l'un des n_j atteint ou dépasse $amax$, alors $Prod_r$ est soit nulle, soit $\geq amax$

Pour déterminer si une règle produit au moins un arbre, il faut donc établir le comptage jusqu'à $amax$.

3.5. 3 Algorithme de décision

Soit $A = (\Sigma, Q, Fin, \mathcal{R})$, un automate déterministe. L'algorithme qui suit permet de déterminer si A reconnaît par stratégie à contrôle fort une forêt vide.

$nombre_q^i$ désignera le nombre d'arbres de hauteur au plus (i-1), pouvant atteindre l'état q

Pour tout état q, $nombre_q^0 \leftarrow 0$
$i \leftarrow 0$

Répéter
  $i \leftarrow i + 1$
  Pour tout état q
    $nombre_q^i \leftarrow 0$
  Pour toute règle r, d'état d'arrivée q :
    $nombre_q^i \leftarrow min(\ amax,\ nombre_q^i + Prod_r(nombre_{q_1}^{i-1}, \ldots, nombre_{q_l}^{i-1})\ )$
Jusqu'à ce que, pour tout q, $nombre_q^i = nombre_q^{i-1}$
  ou $\exists q \in Fin, nombre_q^i \neq 0$
si $\forall q \in Fin,\ nombre_q = 0$, alors $\mathcal{F}(A) = \emptyset$
  sinon $\mathcal{F}(A) \neq \emptyset$
fsi

Preuve de l'algorithme

1) *$nombre_q^i$ est le minimum de amax et du nombre d'arbres de hauteur au plus (i-1) atteignant q.*

Nous considérons une lettre de arité nulle comme ayant une hauteur nulle. La propriété est donc vraie pour i=1, à la première exécution, ne peuvent avoir une production, que les règles n'ayant pas d'états en partie gauche.

On suppose cette propriété vraie au rang n. Le nombre d'arbres de hauteur n atteignant q est égal au nombre de productions des règles dont il est partie droite, à partir des arbres de hauteur n-1. L'algorithme énumère toutes les règles possibles et calcule leur production, $Prod_r$.
Cette production est ajoutée à $nombre_q^n$, à concurrence de $amax$.

2) L'algorithme s'arrête : le nombre d'exécution de la boucle externe est borné grossièrement (cf. plus loin) par $amax \times |Q|$, celui de la boucle interne par le nombre de règles. Il est donc linéaire par rapport au nombre d'états × nombre de règles

3) Nous avons montré que s'il existe un l-uple $n_1, \ldots n_l$ tel que $Prod_q(n_1, \ldots, n_l) = amax$, alors il existe un l-uple $n'_1, \ldots n'_l$ tel que $n'_i \leq amax$, et assurant également une production de $amax$ arbres.

Corollaire.

Pour un automate déterministe, la propriété "$\mathcal{F}(A) = \emptyset$" est polynomiale.

On peut affiner la limite ($amax \times |Q|$) que nous indiquions pour le nombre d'exécution de la boucle externe. L'algorithme s'arrête après au plus $1 + amax \times (|Q| - 1)$ exécutions.
En effet, à chaque exécution de la boucle (sauf éventuellemnt pour la dernière), l'un des $nombre_q^i$ est incrémenté. S'il est fait $1 + amax \times (|Q| - 1)$ exécutions, alors soit $nombre_q^i \neq 0$ pour tout q, soit aucun nombre n'a été incrémenté lors de la dernière exécution. Dans tous les cas, il y a arrêt.

La complexité maximale de l'algorithme est donc donnée par
$|\mathcal{R}| \times (1 + amax \times (|Q| - 1))$

Corollaire.

Si $\mathcal{F}(A) \neq \emptyset$, alors il existe $t \in F$, tel que $hauteur(t) \leq amax \times (|Q| - 1)$

Nous avons montré que $nombre_q^i$ représentait le nombre d'arbres de hauteur $i - 1$, atteignant q, puis que i pouvait être limité à $1 + amax \times (|Q| - 1)$

Il s'agit de la plus petite borne que nous puissions fournir. Il existe des automates, tels que le plus petit arbre reconnu est de cette hauteur.

Exemple

$\Sigma = \{\bar{t}, t, o, m, @\}$
0 1 1 1 3
$Q = \{q_t, q_o, q_m, final\}$

$amax \times (|Q| - 1) = 9$

Les règles de l'automate :
$\bar{t} \rightarrow q_t$
$t(q_m) \rightarrow q_t$
$o(q_t) \rightarrow q_o$
$m(q_o) \rightarrow q_m$
$@(q_m, q_m, q_m)[\#1 \neq \#2 \wedge \#2 \neq \#3 \wedge \#1 \neq \#3] \rightarrow final$

les arbres atteignant q_m sont les éléments de $(mot)^* mo\bar{t}$
Pour satisfaire la condition de la dernière règle, il faut 3 éléments distincts atteignant q_m, donc l'un aura nécessairement la hauteur 8 ($motmotmo\bar{t}$).
L'un des plus petits arbres reconnu sera :
$@(motmotmo\bar{t}, motmo\bar{t}, mo\bar{t})$, de hauteur 9.

Il y a là une différence avec les automates classiques, où le plus petit arbre reconnu est au plus de hauteur $|Q| - 1$
Remarquons enfin que la forêt de notre exemple reconnaît 6 arbres (3!) de hauteur 9, obtenus par permutation des sous-arbres.

3.5. 4 Exemple d'application à l'inductive réductibilité

L'exemple que nous citons est emprunté à H. Comon [Com 90]. Il s'agit de considérer le problème de l'inductive réductibilité pour le système de réécriture suivant :

$$
\begin{aligned}
R = \{ \quad & p(s(x)) \rightarrow x \qquad s(p(x)) \rightarrow x \qquad s(s(0)) \rightarrow 0 \\
& x + x \rightarrow 0 \qquad 0 + x \rightarrow x \qquad x + 0 \rightarrow x \\
& p(x) + y \rightarrow p(x + y) \qquad x + p(y) \rightarrow p(x + y) \\
\} &
\end{aligned}
$$

Rappelons qu'un terme t est inductivement réductible pour un système R si et seulement si chacune de ses instances closes est réductible par R.

Termes clos réductibles

Il nous est possible de construire un automate reconnaissant l'ensemble des termes clos réductibles.

q_r est l'état atteint par tout arbre réductible. Il sera le seul état terminal.
Au sommet des arbres non-réductibles, on trouvera les états :

q_0 atteint immédiatement au dessus des feuilles
q'_0 atteint par s(0)
q_1 atteint au-dessus d'un p
q_2 au-dessus d'un s
q_3 au-dessus d'un +

On obtient l'automate

$0 \rightarrow q_0$
$s(q_0) \rightarrow q'_0 \quad s(q'_0) \rightarrow q_r$
$s(q_1) \rightarrow q_r \quad s(q_2) \rightarrow q_2 \quad s(q_3) \rightarrow q_2$
$s(q_r) \rightarrow q_r$
$p(q_0) \rightarrow q_1 \quad p(q'_0) \rightarrow q_r$
$p(q_1) \rightarrow q_1 \quad p(q_2) \rightarrow q_r \quad p(q_3) \rightarrow q_1$
$p(q_r) \rightarrow q_r$
$+(q_r, q) \rightarrow q_r \quad +(q, q_r) \rightarrow q_r$
$+(q_1, q) \rightarrow q_r \quad +(q, q_1) \rightarrow q_r$
$+(q_0, q) \rightarrow q_r \quad +(q, q_0) \rightarrow q_r$
} pour tout q
$+(q_r, q_r)[\#1 \neq \#2] \rightarrow q_r$
$+(q, q)[\#1 \neq \#2] \rightarrow q_3 \qquad q \neq q_r, q \neq q_r$
$+(q, q)[\#1 = \#2] \rightarrow q_r \qquad$ pour tout q
$+(q, q') \rightarrow q_3 \qquad q \neq q'$, q et q' différents de q_0, q_1, et q_r

Inductive-réductibilité

Pour décider si un terme u est inductivement réductible, on modifie l'automate, de manière à reconnaître l'ensemble des arbres réductibles de la forme $u(t_1, \ldots, t_n)$, que nous appellerons Red_u
u est inductivement réductible si
$Red_u = T_u = \{u(t_1, \ldots, t_n) | t_i \in T(\Sigma)_0\}$
donc si $T_u \cap (T(\Sigma) - Red_u)$ est vide.

Reprenons l'exemple fourni par H. Comon : soit à étudier l'inductive réductibilité de $u = x + y$

Un automate reconnaissant Red_u est obtenu en :
- remplaçant chaque q_r apparaissant au-dessus d'un + par q_{r+}
- dupliquant toutes les règles où q_{r+} apparaît en partie gauche, et substituant q_{r+} à q_r (à gauche)

q_{r+} devient seul état final.

L'automate auquel nous avons affaire est déterministe : une seule des règles comporte un test : nous avons explicitement interdit d'atteindre l'état q_3 en cas d'égalité de sous-arbres. Il est, de plus, complet.
En prenant $\{q_0, q_0', q_1, q_2, q_3, q_r\}$ comme ensemble d'états terminaux, on définit un automate reconnaissant le complémentaire.
Il reste encore à prendre l'intersection avec T_u, c'est à dire à n'accepter que les arbres commençant par un +.
Ce peut être fait, en remarquant que $(t \xrightarrow{*} q_f|q_3) \iff t = +(x, y)$
En prenant q_3 comme seul état terminal, on reconnaît donc la forêt $T_u \cap (T(\Sigma) - Red_u)$

On obtient l'automate

$0 \to q_0$
$s(q_0) \to q_0' \quad s(q_0') \to q_r$
$s(q_1) \to q_r \quad s(q_2) \to q_2 \quad s(q_3) \to q_2$
$s(q_r) \to q_r$
$s(q_{r+}) \to q_r$
$p(q_0) \to q_1 \quad p(q_0') \to q_r$
$p(q_1) \to q_1 \quad p(q_2) \to q_r \quad p(q_3) \to q_1$
$p(q_r) \to q_r$
$p(q_{r+}) \to q_r$

$$\left.\begin{array}{l} +(q_r, q) \to q_{r+} \quad +(q, q_r) \to q_{r+} \\ +(q_{r+}, q) \to q_{r+} \quad +(q, q_{r+}) \to q_{r+} \\ +(q_1, q) \to q_{r+} \quad +(q, q_1) \to q_{r+} \\ +(q_0, q) \to q_{r+} \quad +(q, q_0) \to q_{r+} \end{array}\right\} \text{ pour tout } q$$

$+(q_r, q_r)[\#1 \neq \#2] \to q_{r+}$
$+(q_{r+}, q_{r+})[\#1 \neq \#2] \to q_{r+}$
$+(q, q)[\#1 \neq \#2] \to q_3 \qquad q \neq q_r, q \neq q_{r+}$
$+(q, q)[\#1 = \#2] \to q_{r+} \qquad$ pour tout q
$+(q, q') \to q_3 \qquad q \neq q'$, q et q' différents de q_0, q_1, q_r et q_{r+}

Avec $Fin = \{q_2, q_3\}$

Nous pouvons maintenant lui appliquer l'algorithme de décison du vide

Nous noterons n_i au lieu de $nombre_q^i$ qui figure dans l'algorithme. Chaque étape de l'algorithme nécessite de consulter les 68 règles que comprend notre automate. (NB : l'algorithme de décision du vide que nous avons présenté peut être rendu plus efficace dans certains cas, mais sans changer la borne maximale).
Nous obtenons le résultat ci-dessous :

Etape 1 :
$n_0 = 1$
Etape 2 :
$n_0 = 1 \quad n'_0 = 1 \quad n_1 = 1 \quad n_{r+} = 1$
Etape 3 :
$n_0 = 1 \quad n'_0 = 1 \quad n_1 = 1 \quad n_{r+} = 2 \quad n_r = 2$
Etape 4 :
$n_0 = 1 \quad n'_0 = 1 \quad n_1 = 1 \quad n_{r+} = 2 \quad n_r = 2$

L'algorithme s'arrête donc là. Ni q_3, ni q_2 ne sont atteints. La forêt reconnue est vide, et u= x+y est inductivement réductible.

3.5. 5 Complexité dans le cas général

Proposition.

Dans le cas général d'un automate non-déterministe, la propriété "$\mathcal{F}(A) = \emptyset$" est NP-dure.

Nous pouvons y réduire le problème de la satisfiabilité des expressions booléennes. Soit une expression booléenne $Ex(x_1, \ldots, x_n)$. Nous lui associons un automate sur l'alphabet :

$$\begin{array}{lccccccccc} \Sigma = \{ & Vrai, & Faux, & x_1, \ldots, & x_n, & \vee, & \wedge, &], & @ & \} \\ & 0 & 0 & 1 & 1 & 2 & 2 & 1 & n+2 & \end{array}$$

L'idée est d'associer à chaque expression booléenne Ex un automate qui reconnaît les arbres $@(t_{Ex}, \ldots, t_{Ex})$ (n+2 fois t_{Ex}), où t_{Ex} est la représentation arborescente

usuelle de l'expression Ex, munie d'une valuation correcte (chaque extrémité de branche est constituée d'une variable x_i munie de sa valuation, une valeur (Vrai, Faux) qui est la feuille).
Si t_{Ex} figure n+2 fois en dessous de @, c'est parce que chaque exemplaire subit une partie du contrôle : pour l'un, on vérifie que l'arbre représente bien l'expression booléenne Ex; pour le deuxième, on vérifie que la valuation donne bien pour résultat "Vrai"; les n derniers exemplaires servent à contrôler la cohérence de la valuation, c'est à dire à vérifier que tous les exemplaires d'un x_i donné se sont bien vu affecter la même valeur.
Chacun de ces contrôles, pris individuellement, peut être fait par un automate ascendant classique, dont la complexité est polynomiale par rapport à l'expression donnée.

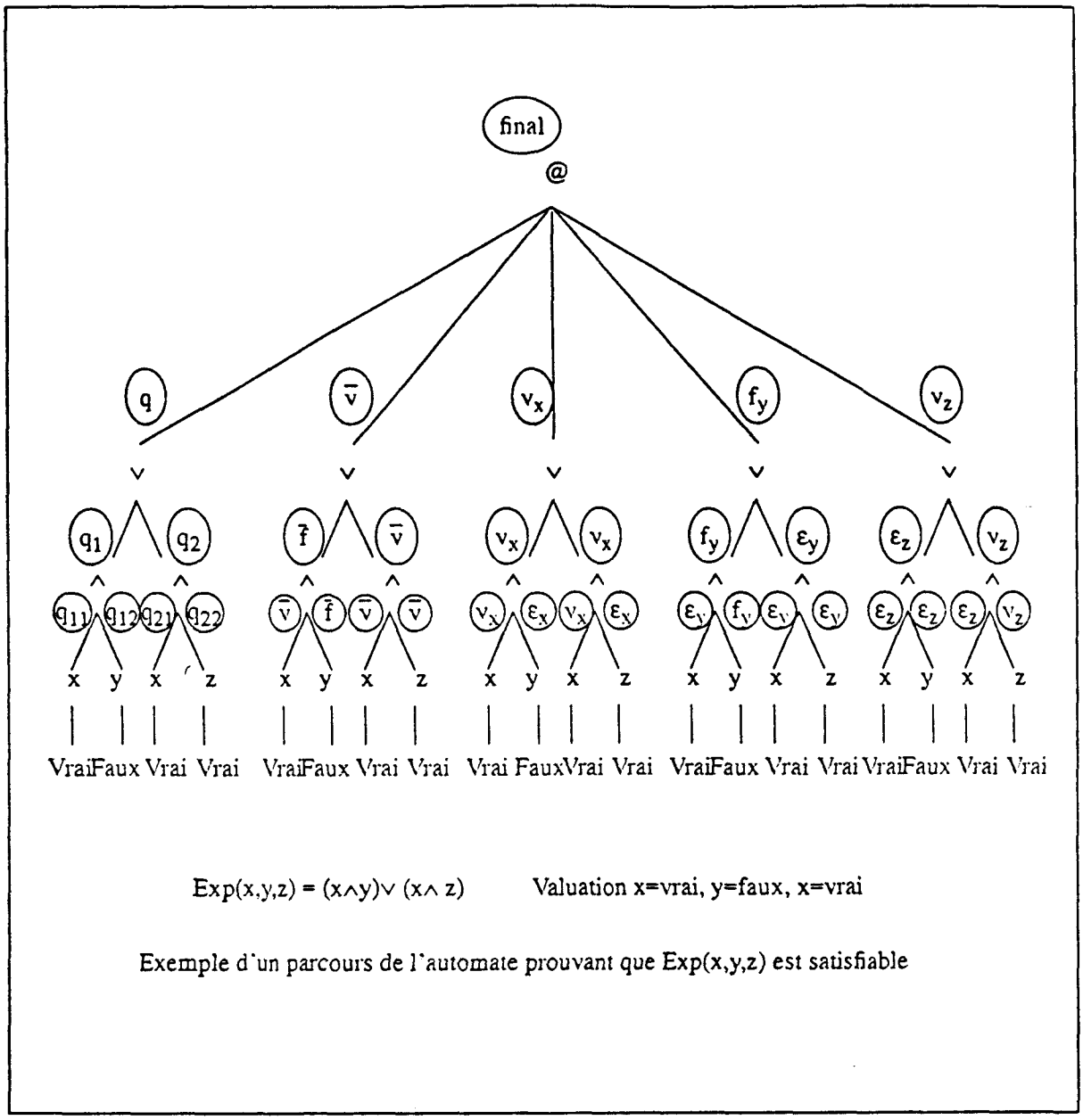
A la racine, nous ajoutons la lettre @, munie d'une règle qui vérifie qu'il y a égalité entre les sous-arbres, et que c'est bien le même arbre qui a répondu à chacune des vérifications.
De cette manière, si il existe un arbre t_{Ex} :
- qui représente l'expresion Ex
- dont la valuation est cohérente
- dont l'évaluation vaut Vrai

alors l'expression booléenne Ex est satisfiable.

Voici de façon plus précise quelles seront les règles de l'automate associé à Ex:

$Vrai \rightarrow v \qquad Faux \rightarrow f$

$x_i(v) \rightarrow v_i$	$x_i(f) \rightarrow f_i$	Règles vérifiant la cohérence de la valuation :
$x_i(v) \rightarrow \epsilon_j$	$x_i(f) \rightarrow \epsilon_j$	L'état v_i (resp f_i) est atteint par les arbres dans lesquels tous les x_i sont associés à la valeur "vrai" (resp. "Faux")
$\rceil(v_i) \rightarrow v_i$	$\rceil(f_i) \rightarrow f_i$	
$\vee(v_i, v_i) \rightarrow v_i$	$\vee(f_i, f_i) \rightarrow f_i$	
$\wedge(v_i, v_i) \rightarrow v_i$	$\wedge(f_i, f_i) \rightarrow f_i$	
$\vee(v_i, \epsilon_i) \rightarrow v_i$	$\vee(\epsilon_i, v_i) \rightarrow v_i$	
$\vee(f_i, \epsilon_i) \rightarrow f_i$	$\vee(\epsilon_i, f_i) \rightarrow f_i$	
$\vee(v_i, v_j) \rightarrow puits$	$\vee(f_i, f_j) \rightarrow puits$	ϵ_i est atteint par les arbres ne comportant pas de valuation pour la variable x_i
$\vee(f_i, v_j) \rightarrow puits$	$\vee(v_i, f_j) \rightarrow puits$	
$\vee(v_i, f_i) \rightarrow puits$	$\vee(f_i, v_i) \rightarrow puits$	
$\wedge(v_i, f_i) \rightarrow puits$	$\wedge(f_i, v_i) \rightarrow puits$	
$(i \neq j)$		



Satisfiabilité des expressions booléennes

$x_i(v) \rightarrow \bar{v} \qquad x_i(f) \rightarrow \bar{f}$
$\rceil(\bar{v}) \rightarrow \bar{f} \qquad \rceil(\bar{f}) \rightarrow \bar{v}$
$\vee(\bar{f},\bar{f}) \rightarrow \bar{f} \qquad \vee(\bar{f},\bar{v}) \rightarrow \bar{v}$
$\vee(\bar{v},\bar{f}) \rightarrow \bar{v} \qquad \vee(\bar{v},\bar{v}) \rightarrow \bar{v}$
$\wedge(\bar{f},\bar{f}) \rightarrow \bar{f} \qquad \wedge(\bar{f},\bar{v}) \rightarrow \bar{f}$
$\wedge(\bar{v},\bar{f}) \rightarrow \bar{f} \qquad \wedge(\bar{v},\bar{v}) \rightarrow \bar{v}$

Calcul de la valeur de l'expression

Il ne manque plus que les règles permettant de vérifier qu'un arbre représente l'expression booléenne Ex. La construction d'un automate reconnaissant une expression

booléenne est très classique. Nous donnons ci-dessous un algorithme qui construit un ensemble d'états et un ensemble de règles, qui viendront s'ajouter aux états et règles déjà cités.

L'algorithme de construction de l'automate peut être donné par :

$Q \leftarrow \{q_{Ex}\}$
Compléter (Ex, q_{Ex})

où compléter s'écrit de la façon suivante :
Compléter $(Expression, q_\omega)$

Si $Expression = x_k$ alors
$\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{x_k(v) \rightarrow q_\omega, x_k(f) \rightarrow q_\omega\}$

sinon_si $Expression = \rceil(Expression')$ alors
$\quad Q \leftarrow Q \cup \{q_{\omega.1}\}$
$\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{\rceil(q_{\omega.1}) \rightarrow q_\omega\}$
$\quad$ compléter $(Expression', q_{\omega.1})$

sinon_si $Expression = \vee(Expression1, Expression2)$ alors
$\quad Q \leftarrow Q \cup \{q_{\omega.1}, q_{\omega.2}\}$
$\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{\vee(q_{\omega.1}, q_{\omega.2}) \rightarrow q_\omega\}$
$\quad$ compléter $(Expression1, q_{\omega.1})$ compléter $(Expression2, q_{\omega.2})$

sinon - - $Expression = \wedge(Expression1, Expression2)$
$\quad Q \leftarrow Q \cup \{q_{\omega.1}, q_{\omega.2}\}$
$\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{\wedge(q_{\omega.1}, q_{\omega.2}) \rightarrow q_\omega\}$
$\quad$ compléter $(Expression1, q_{\omega.1})$ compléter $(Expression2, q_{\omega.2})$

fsi

Et, enfin, les règles de tête permettant de vérifier que le même arbre a atteint :

- l'état q_{Ex} sur la première branche, il représente l'expression Ex,
- l'état $\bar{v}$ sur la deuxième, l'évaluation donne "vrai",
- soit v_i, soit f_i sur la (i+1)ième branche pour la cohérence.

$@(q_{Exp}, \bar{v}, \alpha_1, \ldots, \alpha_n)[\#1 = \#2 = \cdots = \#(n+2)] \rightarrow final$
où α_i peut être soit v_i, soit f_i

Toutes les règles non écrites donnent l'état puits.

Il existe donc une valuation correcte de Ex si et seulement si il existe un arbre reconnu par notre automate.

3.5. 6 Finitude d'une forêt reconnue

Proposition.

Pour un automate à tests A, la propriété "$\mathcal{F}(A)$ est finie" est décidable.

Nous montrerons la propriété dans le cas où l'automate A est déterministe, et muni de la stratégie forte. Les autres cas peuvent s'y ramener (de manière non polynomiale...) par les algorithmes de déterminisation.
Nous supposerons donc toujours que A est déterministe.

Comme dans le cas d'un automate d'arbres classiques, on réduit le problème à la recherche d'arbres tels qu'un même état est atteint deux fois sur une branche.

Proposition.

$\mathcal{F}_q(A)$ *est infini si et seulement si*
$\exists u \in T(\Sigma)_0,\ \exists v \in T(\Sigma)_1,\ h(v) \neq 0,\ \exists w \in T(\Sigma)_1,\ \exists q' \in Q$
$u \xrightarrow[A]{*} q',\ v.u \xrightarrow[A]{*} q',\ w.v.u \xrightarrow[A]{*} q$

Proposition (équivalente).

$\mathcal{F}_q(A)$ *est infini si et seulement si*
$\exists u \in T(\Sigma)_0,$
$\exists \tilde{v} \in T(\Sigma)_n,\ h(\tilde{v}) \neq 0,\ u$ *non sous-arbre de* $\tilde{v}$
$\exists \tilde{w} \in T(\Sigma)_{n'},\ \tilde{v}(u,\ldots,u)$ *non sous-arbre de* $\tilde{w}$
$\exists q' \in Q$
$u \xrightarrow[A]{*} q',\ \tilde{v}(u,\ldots,u) \xrightarrow[A]{*} q',\ \tilde{w}(\tilde{v}(u,\ldots,u),\ldots,\tilde{v}(u,\ldots,u)) \xrightarrow[A]{*} q$

L'équivalence des deux propositions est sans difficulté : $\tilde{v}$ est obtenu à partir de v en remplaçant chaque occurrence de u par une variable. De même pour obtenir $\tilde{w}$ à partir de w, on enlève les occurrences de $\tilde{v}(u,\ldots,u)$

Preuve

• Dans le sens "$\mathcal{F}_q(A)$ infini $\Rightarrow$ il existe un état atteint deux fois sur la même branche", la preuve est triviale, la hauteur des arbres ne pouvant être bornée.

• Posons $u_0 = u$, $u_{i+1} = \tilde{v}(u_i, \ldots, u_i)$ pour $i \geq 0$.
Nous allons montrer que pour tout i, $u_i \xrightarrow[A]{*} q'$.

Remarquons tout d'abord que u n'étant pas sous-arbre de $\tilde{v}$, mais étant sous-arbre de u_i pour tout i, aucun des u_i n'est sous-arbre de $\tilde{v}$.

La propriété "$u_i \xrightarrow[A]{*} q'$" est vraie pour u_0 et u_1. Supposons la vraie jusqu'à un k fixé, $k \geq 1$.
Soient $\tilde{t}_1$ et $\tilde{t}_2$, deux sous-arbres de $\tilde{v}$, contenant éventuellement des variables.
$\tilde{t}_1(u_k, \ldots, u_k) = \tilde{t}_2(u_k, \ldots, u_k) \iff \tilde{t}_1(u_{k-1}, \ldots, u_{k-1}) = \tilde{t}_2(u_{k-1}, \ldots, u_{k-1})$
est évident, u_k n'apparait pas dans $\tilde{v}$, donc il n'apparait ni dans $\tilde{t}_1$, ni dans $\tilde{t}_2$. Il en va de même pour u_{k-1}
Comme $u_k = \tilde{v}(u_{k-1}, \ldots, u_{k-1})$ et $u_{k+1} = \tilde{v}(u_k, \ldots, u_k)$, les mêmes situations d'égalités ou de différences de sous-arbres existent dans u_k et dans u_{k+1}.
D'autre part, d'après l'hypothèse de récurrence,
$u_{k-1} \xrightarrow[A]{*} q'$ et $u_k = \tilde{v}(u_{k-1}, \ldots, u_{k-1}) \xrightarrow[A]{*} q'$, donc $u_{k+1} = \tilde{v}(u_k, \ldots, u_k) \xrightarrow[A]{*} q'$
On a construit une suite infinie d'arbres distincts ($h(\tilde{v}) \neq 0$), atteignant q'.

L'argument de preuve utilisé dans la récurrence peut également s'appliquer à $\tilde{w}(u_1, \ldots, u_1)$ et $\tilde{w}(u_i, \ldots u_i)$.
On retrouve les mêmes situations d'égalités et de différences, pour les sous-arbres dont la racine se trouve dans $\tilde{w}$.
u_1 et u_i atteignant le même état q', $\tilde{w}(u_i, \ldots, u_i) \xrightarrow[A]{*} q$.

Algorithme de décision

Par rapport à l'algorithme de décision du vide, on ajoute pour chaque état un ensemble de taille maximale $amax$, contenant les "successeurs" de chaque état.
$Successeurs(q) = \{q' | \exists v, u \; u \xrightarrow{*} q, \; v.u \xrightarrow{*} q'\}$
Les états seront associés à $nombre_q^i \in [0, amax]$. Dès lors qu'un état est atteint deux fois sur une même branche, la forêt qui l'atteint est infinie.

On marque donc les états par une valeur logique : $infini(q)$

Pour tout état q, $nombre_q^0 \leftarrow 0 \quad Successeurs(q) \leftarrow \emptyset$
$i \leftarrow 1$

Répéter
| Pour tout état q
| $nombre_q^i \leftarrow 0$
| Pour toute règle r, d'état d'arrivée q
| $x \leftarrow Prod_r(nombre_{q_1}^{i-1}, \ldots, nombre_{q_l}^{i-1})$
| $nombre_q^i \leftarrow min(\ amax,\ x\ )$
| si $x \neq 0$ alors
| Pour q', en partie gauche,
| si $infini(q')$ ou $q' = q$ alors $marquer\_infini(q)$;
| sinon_si $q' \in Successeurs(q)$ alors $marquer\_infini(q')$;
| sinon $Successeurs(q') \leftarrow Successeurs(q') \cup Successeurs(q) \cup \{q\}$;
| fsi
| fsi
| $i \leftarrow i + 1$
Jusqu'à ce que, pour tout q, $nombre_q^i = nombre_q^{i-1}$
ou $\exists q \in Fin,\ infini(q)$
$\mathcal{F}(A)$ est infinie si et seulement si $\exists q \in Fin,\ infini(q)$

avec
$marquer\_infini(q)$
| Si $\neg infini(q)$ alors
| $infini(q) \leftarrow vrai$
| $nombre_q^i \leftarrow amax$
| Pour $q' \in Successeurs(q)$: $marquer\_infini(q')$
| fsi

Preuve

Si $\neg infini(q)$, $nombre_q^i$ *représente le nombre d'arbres de hauteur plus petite que* i, *atteignant l'état* q.
Si $infini(q)$, *il existe* u, v, w, *il existe un état* q' *tel que* u *atteint* q', $v.u$ *atteint* q', $w.v.u$ *atteint* q

Cette propriété est invariante dans l'algorithme. La première partie de l'assertion est identique à celle développée pour la preuve de l'algorithme de décision du vide. On remarque aisément que si $nombre_q^i$ est un nombre entre 0 et $amax$, le calcul est identique à celui fait dans l'algorithme de décision du vide.

$Successeurs(q)$ est l'ensemble des états atteints par les arbres de hauteur au plus i,

tel que l'un des sous-arbres atteint q. Le marquage $nombre_q^i \leftarrow infini$ intervient dès lors qu'un arbre de hauteur i atteint q, et que l'un de ses sous-arbres atteint q, ou un état q' avec $infini(q')$ vrai.

L'algorithme s'arrête, on peut, très grossièrement, borner le nombre d'exécutions par $3 \times |Q|$. Remarquons qu'il accélère la décision du vide, en permettant de tenir compte de états atteints deux fois sur une même branche (au détriment de la complexité spatiale).

Reprenons l'exemple

$\bar{t} \rightarrow q_t$
$t(q_m) \rightarrow q_t$
$o(q_t) \rightarrow q_o$
$m(q_o) \rightarrow q_m$
$@(q_m, q_m, q_m)[\#1 \neq \#2 \wedge \#2 \neq \#3 \wedge \#1 \neq \#3] \rightarrow final$

Il suffit alors de savoir que $\bar{t} \xrightarrow{*} q_t$ et $tmo\bar{t} \xrightarrow{*} q_t$ pour conclure que la forêt reconnue est non vide, et même infinie.

3. 6 Morphismes alphabétiques d'arbres

La notion de morphismes d'arbres a été définie au chapitre précédent. Ici aussi, nous nous intéresserons aux morphismes alphabétiques, non nécessairement linéaires ni complets, qui conserveront la famille $REC_{\neq}$.

3.6. 1 Stabilité de $REC_{\neq}$

la famille $\mathbf{REC}_{\neq}$ est **stable par morphisme alphabétique strict**

Pour une forêt F appartenant à $REC_{\neq}$ et un morphisme alphabétique ϕ, l'automate reconnaissant $\phi(F)$ sera construit à partir d'un automate déterministe normalisé reconnaissant F.

Construisons tout d'abord l'ensemble de règles suivant :

$$\mathcal{R}_\phi^0 = \{(\phi(a)(q_{\tau_a(1)}, \ldots, q_{\tau_a(m)})[\tau.d] \rightarrow q) | (a(q_1, \ldots, q_n)[d] \rightarrow q) \in \mathcal{R}$$

Les règles définies de cette manière sont insuffisantes pour permettre de reconnaître $\phi(F)$. Le problème provient de la non-injectivité du morphisme : telles quelles, ces

règles imposent des différences là où le morphisme peut les avoir effacées. Voici un exemple :
la forêt F est l'ensemble des arbres $b((a+\alpha)^*\bar{a},(a+\alpha)^*\bar{a})$ dont les deux branches sont différentes.
$F \in REC_{\neq}$, elle est reconnue par l'automate :

$\bar{a}\rightarrow q \quad a(q)\rightarrow q \quad \alpha(q)\rightarrow q$

$b(q,q)[\#1 \neq \#2]\rightarrow q_f \quad b(q,q)[\#1 = \#2]\rightarrow puits$

le morphisme ϕ conservera toutes les lettres, sauf α qui sera transformé en a.
La méthode de construction de $\mathcal{R}^0_\phi$ donne exactement les mêmes règles que dans l'automate de départ, sauf la règle $\alpha(q)\rightarrow q$ qui n'existe plus.
L'automate ainsi construit rejette donc, par exemple $b(a\bar{a},a\bar{a})$ qui, pourtant, est l'image par ϕ de $b(\alpha\bar{a},a\bar{a})$ et devrait donc être reconnu.
Il est possible de distinguer le cas des arbres qui peuvent être l'image par le morphisme de plusieurs éléments distincts. Un marquage des états est possible quand la règle utilisée est issue de plusieurs règles de l'automate de départ :
dans notre exemple, la règle
$(a(q)\rightarrow q)$
est issue de $(a(q)\rightarrow q)$ et de $(\alpha(q)\rightarrow q)$.
On peut donc assurer un marquage de q, après utilisation de cette règle.
Cette information n'est cependant pas encore suffisante. Soit la forêt
$c((\alpha+a)^*\bar{a},(\alpha+a)^*\bar{a},(\alpha+a)^*\bar{a})$,
où les trois branches sont différentes deux à deux.
Il faut distinguer le cas de $b(a\bar{a},a\bar{a},a\bar{a})$ qui ne peut être l'image d'un arbre de F, puisque seulement $a\bar{a}$ et $\alpha\bar{a}$ ont pour image $a\bar{a}$. Le marquage simple est donc insuffisant, il faudra, pour chaque état, compter le nombre d'images réciproques pour les arbres qui l'atteignent. Le comptage peut s'arrêter à l'arité maximale de Σ.

Exemple :
$a(q^1)\rightarrow q^2 \qquad a(q^2)\rightarrow q^m$
$c(q^1,q^1,q^1)[\{1\},\{2\},\{3\}]\rightarrow q_f \qquad c(q^1,q^1,q^1)$[autres descriptions]$\rightarrow puits$
$c(q^2,q^2,q^2)[\{1\},\{2\},\{3\}]\rightarrow q_f \qquad c(q^2,q^2,q^2)$[autres descriptions]$\rightarrow puits$
$c(q^m,q^m,q^m)$[toutes descriptions]$\rightarrow q_f$

3.6. 2 Construction d'un automate reconnaissant $\phi(F)$

Avant de décrire l'algorithme de construction, nous précisons la méthode de comptage présentée informellement.

Comptage

Pour cette construction, chaque état de l'automate de départ sera associé à un facteur, qui représentera le nombre d'antécédents par ϕ de chaque arbre atteignant cet état. On se limitera au facteur amax, arité maximale de l'alphabet Σ.

Rappelons et étendons les définitions et propriétés établis en 3.4. Nous avions notamment défini

$$Desc_d(k) = \{d' | d' \leq d, \forall p \in d, p \textit{ contient au plus } k \textit{ parties de } d'\}$$

Si d est supposée être la description d'une règle homogène (un seul état en partie gauche) d'un automate normalisé, alors on a la propriété :
Si d est la description la plus précise d'un n-uple x_i, et si chaque x_i est l'image de k valeurs distinctes, alors il existe pour toute description $d' \in Desc_d(k)$ un n-uple d'antécédents de $x_1, \ldots, x_n$, dont d' est la description exacte.

Soit d, une partition, d' un élément de $Desc_d(k)$, et $x_1, \ldots, x_n$ un n-uple dont d est la description exacte. Déterminons le nombre de n-uples $x'_1, \ldots, x'_n$ dont d' est la description exacte et tels l'image de $x'_1, \ldots, x'_n$ est $x_1, \ldots, x_n$

Chaque élément p de d correspond à une valeur du n-uple. p contient i_p parties de d', avec $i_p \leq k$. Il faut donc trouver i_p valeurs distinctes dont les images sont identiques. Il y a donc $k!/i_p!$ façons de choisir ces i_p valeurs (de façon ordonnée).
Ce choix doit se faire pour chaque élément de la partition d. Le nombre de n-uples $x'_1, \ldots, x'_n$ convenables est donc

$$\prod_{p \in d} k!/i_p!$$

Nous noterons $Mult_{d,d'}(k)$ cette quantité.

Il reste encore à généraliser ces notions pour les règles non homogènes à gauche. Il faut alors faire intervenir toutes les descriptions réduites, pour chaque état q présent dans le n-uple. Nous noterons

$$Desc_d(k_{q_1}, \ldots, kq_m) = \bigcup Desc_d(k_q)$$

$$Mult_{d,d'}(k_{q_1}, \ldots, k_{q_m}) = \prod Mult_{d,d'}(k_q)$$

D'autre part nous supposons disposer de $nombre_q$ pour chaque état q de l'automate de départ. Ce calcul peut avoir été effectué par un algorithme tel que celui de décision, que nous avons décrit.

Algorithme

Soit un automate $A = (\Sigma, Q, Fin, \mathcal{R})$, reconnaissant une forêt F, Il sera supposé normalisé et déterministe. Nous construisons un automate $\bar{A} = (\Sigma', \bar{Q}, \bar{Fin}, \bar{\mathcal{R}})$ reconnaissant $\phi(F)$.

Ses états seront des éléments de $Q \times [1, amax]$, et seront notés q^x

(Voir figure)

Preuve

1) La propriété

$$t \xrightarrow[A_i]{*} q^x \Rightarrow \exists u_1, \ldots, u_x,\ \phi(u_k) = t,\ t_k \xrightarrow[A]{*} q$$

est invariante dans l'algorithme.

i désigne le nombre d'exécutions de la boucle intermédiaire (la plus petite boucle construisant une règle). A_i est l'automate tel qu'il est construit à la i^{eme} exécution de la boucle.

Chaque arbre atteignant q^x est l'image par le morphisme ϕ de x arbres (exactement) qui atteignent q.

La preuve de l'invariance de la propriété découle des propositions et lemmes techniques établis dans les sections précedentes. La boucle interne construit une règle $(a(q_1, \ldots, q_n)[d'] \rightarrow q^x)$.

Soit $t = a(t_1, \ldots, t_n)$, un arbre atteignant q^x par la dernière règle construite.

Chaque t_k est l'image de i_k arbres atteignant q_k.

On note $e_1^{y_1}, \ldots, e_p^{y_p}$ les différentes valeurs des états $q_1^{i_1}, \ldots, q_n^{i_n}$.

d est une description valide de $t_1, \ldots, t_n$. D'après la propriété énoncée en fin de section 3.4.2, $t_1, \ldots, t_n$ sont les images de $Mult_{d.\tau.\tilde{d}}(y_1, \ldots, y_p)$ n-uples $\tilde{t}_1, \ldots, \tilde{t}_n$, dont $\tau.\tilde{d}$ est une description exacte.

Enfin chaque n-uple $\tilde{t}_1, \ldots, \tilde{t}_n$ correspond à plusieurs m-uples, si l'on tient compte des branches effacées (il faut alors tenir compte du produit des nombres d'arbres sur ces branches effacées).

Le nombre d'arbres "convenables" dont t est l'image est donc calculée dans x. t est l'image de x arbres atteignant q dans l'automate de départ.

2)

$$t \xrightarrow[A]{*} q \Rightarrow \exists x, \phi(t) \xrightarrow[\bar{A}]{*} q^x$$

$\bar{Q}_0 \leftarrow \emptyset$

$\bar{\mathcal{R}}_0 \leftarrow \emptyset$

$k \leftarrow 0$

Répéter

$k \leftarrow k+1$

Pour toute partie gauche $a', q_1^{x_1}, \ldots, q_n^{x_n}, [d]$ telle que

$a' \in \Sigma'_n$, $d \in ED_n$, $q_1^{x_1}, \ldots, q_n^{x_n} \in Q_{k-1}$

d est valide pour $q_1^{x_1}, \ldots, q_n^{x_n}$

On note $e_1^{y_1}, \ldots, e_p^{y_p}$ les valeurs des états présents en partie gauche. faire

$x \leftarrow 0$

Pour toute règle $r = (a(\tilde{q}_1, \ldots, \tilde{q}_m)[\tilde{d}] \rightarrow q)$ telle que

$\phi(a) = a'$

$q_i = \tilde{q}_{r(i)}$

$d \in Desc_{r,\tilde{d}}(y_1, \ldots, y_p)$

faire

$$x \leftarrow x + Mult_{d,r,\tilde{d}}(y_1, \ldots, y_p) \times \prod_{\tilde{q} \text{ sur branche effacée}} nombre_{\tilde{q}}$$

fait

$x \leftarrow min(x, amax)$

Si $x \neq 0$ alors

$\bar{Q}_k \leftarrow \bar{Q}_k \cup \{q^x\}$

$\bar{\mathcal{R}}_k \leftarrow \bar{\mathcal{R}}_k \cup \{a(q_1^{x_1}, \ldots, q_n^{x_n})[d] \rightarrow q^x\}$

fsi

fait

Jusqu'à ce que $\bar{Q}_k = \bar{Q}_{k-1}$ et $\bar{\mathcal{R}}_k = \bar{\mathcal{R}}_{k-1}$

$\bar{Q} \leftarrow \bar{Q}_k$

$\bar{\mathcal{R}} \leftarrow \bar{\mathcal{R}}_k$

$\bar{F} \leftarrow \{q^x | q \in F\}$

Algorithme de construction

La propriété se montre par une récurrence sur la hauteur de l'arbre.

Elle est évidemment vraie pour les feuilles, la première exécution de la boucle principale construisant toutes les règles sur des lettres de arité nulle, par simple

renommage de la lettre.

Si la propriété est vraie pour des sous-arbres d'une hauteur h fixée, et si $t = a(t_1, \ldots, t_m)$ est un arbre de hauteur $h+1$, chaque sous-arbre t_k atteint un état q_k, et $\phi(t_k)$ atteint un état $q_k^{i_k}$. Dans l'algorithme, il est alors construit une règle :

$$a(q_{\tau(1)}^{i_{\tau(1)}}, \ldots q_{\tau(n)}^{i_{\tau(n)}})[d'] \rightarrow q^x$$

avec $d' \leq \tau.d$.

telle que d' est valide pour les images $\phi(t_{\tau(1)}), \ldots, \phi(t_{\tau(n)})$

donc l'arbre $\phi(t)$ atteint l'état q^x

3.6. 3 Morphismes effaçants

Proposition.

$REC_{\neq}$ est stable par morphismes gommants.

Corollaire.

$REC_{\neq}$ est stable par morphismes alphabétiques (au sens large)

Comme dans le chapitre précédent, nous décomposons un morphisme alphabétique quelconque en un morphisme alphabétique strict et un morphisme gommant.

Preuve de la proposition

Nous considèrerons une forêt reconnue par un automate $A = (\Sigma, Q, Fin, \mathcal{R})$ déterministe, normalisé-complet.

Soit a, une lettre telle que $\phi(a(x_1, \ldots, x_n)) = x_i$, et r, une règle $(a(q_1, \ldots, q_n)[d] \rightarrow e)$. Soient t et $t' = \phi(t)$, où t est un arbre atteignant e, tel que que la règle dernière règle utilisée est r

t' est l'image par ϕ de $Prod_r(nombre_{\tilde{q}_1}, \ldots, nombre_{\tilde{q}_m})/nombre_{q_i}$ arbres distincts utilisant la règle r à la racine.

($\tilde{q}_1, \ldots, \tilde{q}_m$ sont les valeurs distinctes des états $q_1, \ldots, q_n$)

Pour deux états q et e, on définit l'ensemble

$\mathcal{R}(q, e) = \{r = (a(q_1, \ldots, q_n)[d] \rightarrow e) \in \mathcal{R},\ q = q_i,\ \phi(a(x_1, \ldots, x_n)) = x_i\}$

Puis le quantités :

$Subst^0(q, e) = 0$ si $q \neq e$, 1 sinon.

$Subst^1(q, e) =$

$min(amax, Subst^0(q,e) + \sum_{r \in \mathcal{R}(q,e)}(nombre_{\tilde{q}_1}, \ldots, nombre_{\tilde{q}_m})/nombre_q)$

Et pour $i > 1$:

$Subst^i(q,e) = min(amax, Subst^{i-1}(q,e) + \sum_{q'} Subst^{i-1}(q,q') \times Subst^{i-1}(q',e))$

$Subst^i(q,e)$ est une suite finie, puisque croissante et bornée par $amax$. Notons $Subst^*(q,e)$ sa limite.

$Subst^*(q,e)$ représente pour chaque arbre u atteignant q, le nombre d'arbres $v.u$, tels que $v.u \xrightarrow[A]{*} e, \phi(v.u) = u$ (jusqu'à amax).

Les états de l'automate A' qui reconnaît $\phi(F)$ seront

$Q' = Q \times [0, amax]$

$Fin' = Fin \times [1, amax]$

et les règles

$\mathcal{R}'\{(a(q_1^{x_1}, \ldots, q_n^{x_n})[d] \rightarrow e^x) \mid (a(q_1, \ldots, q_n)[d'] \rightarrow q) \in \mathcal{R},$

$x = Mult_{d,d'}(y_1, \ldots, y_m) \times Subst^*(q,e)\}$ $y_1, \ldots, y_m$ sont les valeurs des x_i pour les valeurs distinctes $\tilde{q}_1, \ldots, \tilde{q}_m$ parmi $q_1, \ldots, q_n$

Prouvons que :

(t est l'image par ϕ de x arbres atteignant q) $\iff$ $t' \xrightarrow[A']{*} q^x$

• Si $h(t) = 0$, $t = \bar{a} \in \Sigma_0$.

$\bar{a}$ ne peut être l'image que d'arbres de la forme $u.\bar{a}$

Pour chaque état e, on connaît par $Subst^*(q,e)$ le nombre d'arbres $v.\bar{a}$ tels que $v.\bar{a}$ atteint e, et $\phi(v.\bar{a}) = \bar{a}$

$(\bar{a}[\perp] \rightarrow q) \in \mathcal{R} \iff (\bar{a}[\perp] \rightarrow e^x) \in \mathcal{R}', x = Subst^*(q,e)$

• Si la propriété est vraie pour une hauteur fixée, considérons $t = a(t_1, \ldots, t_n)$.

t ne peut être l'image que d'arbres de la forme $v.a.(u_1, \ldots, u_n)$, avec $\phi(u_i) = t_i$

$t \xrightarrow[A']{*} e$

$\iff$

$(t_i \xrightarrow[A']{*} q_i$ *et* $(a(q_1^{x_1}, \ldots, q_n^{x_n})[d] \rightarrow e) \in \mathcal{R}'$

d est la description exacte de $t_1, \ldots, t_n$

$\iff$

$\exists q \in Q,$

$\exists (a(q_1, \ldots, q_n)[d'] \rightarrow q) \in \mathcal{R}$

$t_i \xrightarrow[A']{*} \rightarrow q_i$

$x = Mult_{q,q'}(x_1, \ldots, x_m) \times Subst^*(q,e)$

$\iff$

$\exists q$

$\exists Subst^*(q,e)$ arbres $v.t,\ \phi(v.t) = t,\ v.t \xrightarrow[A]{*} q$

$\exists x_i$ arbres $u_i,\ \phi(u_i) = t_i,\ u_i \xrightarrow[A]{*} q_i$

$\exists d', (t_1, \ldots, t_n)$ est l'image de $Mult_{d,d'}(x_1, \ldots, x_m)$ $(t_1, \ldots, t'_n)$ dont d est la description exacte.

$\Longleftrightarrow$

$\exists x$ arbres $v.a(u_1, \ldots, u_n), v.a(u_1, \ldots, u_n) \xrightarrow[A]{*} e,\ \phi(v.a(u_1, \ldots, u_n)) = t$

Ce qui démontre la propriété au rang supérieur.

En prenant $Fin' = Fin \times [1, amax]$, on reconnaît donc l'image par ϕ de la forêt reconnue par A.

3.6. 4 Morphismes inverses

Proposition.

$REC_{\neq}$ n'est pas stable par morphisme alphabétique inverse.

Preuve

Elle est identique à celle développée dans le chapitre 2 à propos de la classe $REC_{=}$. On considère le même contre-exemple permettant d'engendrer par morphisme inverse une forêts d'arbres dont les branches sont de même longueur, mais dont les étiquettes sont différentes.

Chapitre 4

Relations entre les classes

4. 1 Relation induite par les règles sur les états

4.1. 1 Compatibilité avec les règles : définition

Soit un automate à test A. Une relation *Rel*, définie sur l'ensemble de ses états sera dite **compatible avec** $\mathcal{R}$ si:
Si $\forall(a(q_1,\ldots,q_n)[d]\rightarrow q) \in \mathcal{R} \qquad \forall(a(q'_1,\ldots,q'_n)[d']\rightarrow q') \in \mathcal{R}$
telles que $d \leq d'$ et $q_i R q'_i$,
on a qRq'.

4.1. 2 Relation induite

On appelle relation induite par $\mathcal{R}$ sur Q, la plus petite relation réflexive et compatible avec $\mathcal{R}$. Cette relation sera notée ∇_A

Construction de la relation induite
Il convient de justifier l'existence d'une plus petite relation compatible. On construit la clôture compatible comme on construit la clôture transitive.

$Rel \leftarrow Id$
Répéter
$Rel' \leftarrow \{(q,q')|$
$\exists(a(q_1,\ldots,q_n)[d]\rightarrow q) \in \mathcal{R},\ \exists(a(q'_1,\ldots,q'_n)[d']\rightarrow q') \in \mathcal{R}$
avec $d \leq d'$ et $q_i\ Rel\ q'_i$
$\}$
$Rel \leftarrow Rel'$
jusqu'à stabilisation.
$\nabla_A \leftarrow Rel$

"toute relation réflexive et compatible avec $\mathcal{R}$ contient *Rel*" est invariant dans la boucle de l'algorithme.
D'autre part, si l'on suppose l'existence de deux relation minimales, l'intersection des deux est encore une relation compatible. Il y a donc unicité de la relation minimale.

4.1. 3 Une propriété de ∇_A

$$\left.\begin{array}{l} \exists u \in T(\sigma)_n,\ \exists (q_i)_{1\leq i\leq n},\ \exists (q'_i)_{1\leq i\leq n} \in Q, \\ q_i\ \nabla_A\ q'_i,\ u(q_1,\ldots,q_n) \xrightarrow{*} q,\ u(q'_1,\ldots,q'_n) \xrightarrow{*} q' \end{array}\right\} \Rightarrow \quad q\ \nabla_A\ q'$$

Cette propriété se montre par récurrence sur la hauteur de l'arbre u :
- si u est une lettre, c'est la définition d'une relation compatible avec $\mathcal{R}$ qui nous donne le résultat.
- si la propriété est vraie au rang n, et si $u = (a(u_1,\ldots,u_m))$, on applique l'hypothèse à chaque état $\bar{q}_i$ et $\bar{q'}_i$ atteint par u_i.
$q\ \nabla_A\ q'$ à cause des deux règles qui sont définies pour a : $(a(\bar{q}_1,\ldots,\bar{q}_m)[d]\rightarrow q)$ et $(a(\bar{q'}_1,\ldots,\bar{q'}_m)[d]\rightarrow q')$

4. 2 Une caractérisation de *REC* ...

Proposition.

Une forêt F appartient à REC si et seulement si il existe un automate à tests complet A tel que :

A reconnaît F par stratégie forte,

∇_A est la relation identité.

Preuve

Si $F \in REC$, notons A_{rec} l'automate qui la reconnaît. On construit aisément un automate à tests en prenant chaque règle de A_{des}, et en la "décorant" par toutes les descriptions possibles.

$\mathcal{R}_{tests} = \{(a(q_1,\ldots,q_n)[d]\rightarrow q) | (a(q_1,\ldots,q_n)\rightarrow q) \in \mathcal{R}, d \in ED_n\}$

L'automate A_{tests} aura les mêmes états que A_{rec}

La relation identité, plus petite relation réflexive, est ici compatible avec les règles de l'automate : deux règles ayant même profil ont même état d'arrivée.

Réciproquement, si A est un automate vérifiant $\nabla_A = Id$, alors toutes les règles ayant même profil ont aussi même partie droite, de manière évidente. Et si toutes les règles de même profil ont même état d'arrivée, l'automate est équivalent à un automate classique, reconnaissant une forêt rationnelle.

4. 3 ...et une caractérisation de $REC_=$

Proposition.

Un forêt F appartient à $REC_=$ si et seulement si il existe un automate à tests déterministe et complet A tel que :

A reconnaît F par stratégie forte

∇_A est une relation sans circuit.

$q \in Fin$ et $q \nabla_A q'$ $\Rightarrow$ $q' \in Fin$

On rappelle qu'une relation sans circuit est telle que sa clôture transitive est anti-symétrique.

La proposition sera montrée par les deux propositions suivantes.

4.3. 1 Automates "déterminisés"

Si A est un automate obtenu par déterminisation d'un automate à stratégie faible, alors ∇_A est une relation sans circuit.

Le reconnaisseur par stratégie forte d'une forêt de $F_=$ peut être construit par déterminisation. Nous allons montrer que cet automate est bien celui qui vérifie les propriétés énoncées dans la proposition : La relation ∇_A étant plus petite que la relation d'inclusion entre les ensembles d'états.

Rappelons l'algorithme de normalisation développé au chapitre 2.

$Q_0 \leftarrow \{X | \exists \bar{a} \in \Sigma_0,\ X = \{q \mid (\bar{a}[\bot] \rightarrow q) \in \mathcal{R}\}\}$
$\mathcal{R}_0 \leftarrow \{(\bar{a}|[\bot] \rightarrow X) | X = \{q | (a \rightarrow q) \in \mathcal{R}\}\}$

répéter :
$\bar{Q}_i \leftarrow \bar{Q}_{i-1} \cup$
$\{X |\ \exists a \in \Sigma_n, \exists (X_i)_{1 \leq i \leq n}$ dans $Q_{i-1}, \exists d \in ED_n,$
$X = \{q | \exists (q_i)_{1 \leq i \leq n},\ q_k \in X_k,\ \exists d' \leq d,\ (a(q_1, \ldots, q_n)[d'] \rightarrow q) \in \mathcal{R}\}$
$\}$

$\bar{\mathcal{R}}_i \leftarrow \bar{\mathcal{R}}_{i-1} \cup$
$\{(a(X_1, \ldots, X_n)[d] \rightarrow X) |$
$a \in \Sigma_n,\quad (X_i)_{1 \leq i \leq n}$ dans $\bar{Q}_{i-1},\quad d \in ED_n,$
$X = \{q | \exists (q_i)_{1 \leq i \leq n},\ q_k \in X_k,\ \exists d' \leq d,\ (a(q_1, \ldots, q_n)[d'] \rightarrow q) \in \mathcal{R}\}$
$\}$
jusqu'à ce que $\bar{Q}_i = \bar{Q}_{i-1}$ et $\bar{\mathcal{R}}_i = \bar{\mathcal{R}}_{i-1}$
$\bar{Q} \leftarrow \bar{Q}_i$
$\tilde{\mathcal{R}} \leftarrow \mathcal{R}_i$
$\bar{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \setminus \{(a(X_1, \ldots, X_n)[d] \rightarrow X) |\ d$ *non valide pour* $X_1, \ldots, X_n\}$
$\bar{F} \leftarrow \{X | X \cap F \neq \emptyset\}$

Cet algorithme représente, de fait, une déterminisation de l'automate. Si nous munissons le résultat de la stratégie à contrôle fort, la forêt reconnue est encore la même.

Considérons la relation d'inclusion entre les états de l'automate. elle est, bien-sûr, réflexive, transitive, antisymétrique. Toute relation plus petite sera sans circuit. Vérifions qu'elle est compatible avec les règles de l'automate :
Soit deux règles : $(a(X_1, \ldots, X_n)[d] \rightarrow X)$ et $(a(X'_1, \ldots, X'_n)[d'] \rightarrow X')$ telles que $X_i \subset X'_i$
Les ensembles X et X' ont été construits de la façon suivante :
$X \leftarrow \{q | \exists (q_i)_{1 \leq i \leq n},\quad \exists \delta \leq d, q_i \in X_i, (a(q_1, \ldots, q_n)[\delta] \rightarrow q) \in \mathcal{R}\quad \}$

$X' \leftarrow \{q | \exists (q_i)_{1 \leq i \leq n},\quad \exists \delta \leq d', q_i \in X'_i, (a(q_1, \ldots, q_n)[\delta] \rightarrow q) \in \mathcal{R}\quad \}$

Pour tout état q de X , $\exists (a(q_1, \ldots, q_n)[\delta] \rightarrow q)$ avec
$q_i \in X_i \subset X'_i$ et $\delta \leq d \leq d'$
Donc $q \in X'$
L'inclusion est donc compatible avec les règles de l'automate.
Fin est constitué par les ensembles qui contiennent un état terminal de l'automate

de départ. La propriété $X \in Fin$ est donc conservée par l'inclusion.
La relation ∇_A, plus petite que la relation d'inclusion, est donc sans circuit et elle conserve l'appartenance d'un état à l'ensemble Fin.

4.3. 2 Proposition

Dans un automate complet, si ∇_A est une relation sans circuit, et si ($q \in Fin, q \nabla_A q' \Rightarrow q' \in Fin$) alors A reconnaît la même forêt que s'il était doté de la stratégie à contrôle faible

Tout d'abord la forêt reconnue par stratégie forte est incluse dans celle reconnue par le même automate muni de l'autre stratégie, puisque dans l'un on oblige à utiliser la description la plus précise, alors que dans l'autre il suffit d'une description valide. Ce résultat est d'ailleurs vrai pour tout automate, indépendamment de la relation ∇_A.

D'autre part, si un arbre atteint par stratégie faible des états qu'il n'atteint pas par stratégie forte, c'est en utilisant des règles dont les descriptions sont moins précises. Or si l'on utilise une règle dont la description est moins précise, on atteint une état lié par la relation ∇_A.
Si l'état atteint par stratégie forte n'est pas terminal, les autres ne le seront donc pas non plus.

Nous montrons cette propriété par récurrence sur la hauteur de l'arbre.
Remarquons d'abord que si l'automate est déterministe pour la stratégie forte, il est pseudo-déterministe pour la stratégie faible. Le choix entre plusieurs règles n'existe donc que pour des règles de descriptions différentes.

1) pour un arbre de hauteur nulle, un seul état est donc atteint par stratégie faible (pseudo-déterminisme), le même que pour la stratégie forte.

2) Si la propriété est vraie pour un arbre de hauteur n.
   Soit $t = a(t_1, \ldots, t_n)$, Posons q_i l'état atteint par chaque t_i, pour la stratégie forte.
   Pour tout n-uple $(q'_i)_{1 \leq i \leq n}$ d'états atteints par $(t_i)_{1 \leq i \leq n}$ pour l'autre stratégie, on a donc $q'_i \nabla_A q_i$.
   Soit q, état atteint par t (str. forte) et q' atteint par t par str. faible. Alors il existe deux règles :
   $(a(q_1, \ldots, q_n)[d] \rightarrow q)$ et $(a(q'_1, \ldots, q'_n)[d] \rightarrow q)'$,

telles que $d' \leq d$, puisque d est la description le plus précise des t_i, et $q'_i \nabla_A q_i$.
Donc $q' \nabla_A q$

Tout état atteint par stratégie faible est donc plus "faible" (au sens de ∇^*_A) que celui atteint par stratégie forte !...
Comme $q' \in Fin$ et $q' \leq q \Rightarrow q \in Fin$, tout arbre reconnu par stratégie faible est aussi reconnu par stratégie forte.

La famille $REC_=$ peut donc être définie comme l'ensemble des forêts reconnues selon la stratégie forte, par un automate déterministe, dont ∇_A est une relation sans circuit qui conserve l'appartenance à Fin

4. 4 La clôture booléenne de $REC_=$

Proposition.

Toute forêt appartient à la clôture booléenne de $REC_=$ si et seulement si il existe un automate à tests déterministe complet A, qui reconnaît F selon la stratégie forte, et tel que ∇_A est une relation sans circuit

Preuve

A) Sens direct : Nous montrons que la propriété "∇_A est une relation sans circuit" est conservée par produit d'automates et par passage au complémentaire.
C'est trivial pour le passage au complémentaire, puisque cette opération ne concerne que la définition de Fin. (Il s'agit d'automates déterministes)
Concernant le produit d'automates, on peut se reporter au chapitre 3, pour trouver la construction.
Sur $Q \times Q'$, on définit la relation d'ordre :
$(q_1, q'_1) \sqsubseteq (q_2, q'_2)$ si et seulement si $q_1 \nabla_A q_2$ et $q'_1 \nabla_{A'} q'_2$
Cette relation est compatible avec les règles :
Pour tout couple de règles de l'automate produit :
$a((q_1, q'_1), \ldots, (q_n, q'_n))[d] \rightarrow (q, q')$,
$a((\bar{q}_1, \bar{q}'_1), \ldots, (\bar{q}_n, \bar{q}'_n))[\bar{d}] \rightarrow (\bar{q}, \bar{q}')$

Il existe respectivement dans A et A' les deux couples de règles :
$(a(q_1, \ldots, q_n)[d] \rightarrow q) \quad (a(\bar{q}_1, \ldots, \bar{q}_n)[\bar{d}] \rightarrow \bar{q})$
$(a(q'_1, \ldots, q'_n)[d] \rightarrow q') \quad (a(\bar{q}'_1, \ldots, \bar{q}'_n)[\bar{d}'] \rightarrow \bar{q}')$
Si $(q_i, q'_i) \sqsubseteq (\bar{q}_i, \bar{q}'_i)$ pour tout i et $d \leq d'$, alors

$q_i \nabla_A \bar{q}_i$ et $q'_i \nabla_{A'} \bar{q}'_i$ donc $q \nabla_A \bar{q}$ et $q' \nabla_A \bar{q}'$
puis $(q, q') \sqsubseteq (\bar{q}, \bar{q}')$

La propriété "∇ est une relation sans circuit" est conservée par produit et complémentaire. Donc toute forêt appartenant à la clôture boolénne de $REC_=$ admet un reconnaisseur par stratégie forte, dont la relation ∇ est une relation sans circuit.

B) Inversement, tout automate A dont la relation ∇_A est antisymétrique reconnaît un élément de $REC_=$:
Pour tout état final q, A_q désignera un automate ayant les mêmes états et mêmes règles que A, mais avec q comme seul état final.
Comme $\mathcal{F}(A) = \bigcup_{q \in Fin} \mathcal{F}(A_q)$, et que chaque automate A_q possède évidemment la même relation ∇_A (puisque seuls les états terminaux diffèrent), on peut réduire le problème et supposer que A possède un seul état final q_f.

A_{sup} désignera l'automate obtenu en prenant comme terminaux tous les états supérieurs à q_f : $Fin_{sup} = \{q | q_f \nabla_A^* q\}$
A_{sup} reconnaît un élément de $REC_=$, d'après la proposition 4.3
On construit également $Fin_{\bar{inf}} = Q \setminus \{q | q \nabla_A^* q_f\}$
$q \in Fin_{\bar{inf}}$ et $q \leq q' \Rightarrow q' \in Fin_{\bar{inf}}$ L'automate correspondant reconnaît donc aussi une forêt de $REC_=$
et l'automate A_{inf} reconnaît la forêt complémentaire.
Il nous reste à remarquer que :
$\mathcal{R}(A) = \mathcal{R}(A_{sup}) \cap \mathcal{R}(A_{inf})$
Ces deux automates ayant mêmes règles et mêmes états, l'intersection est faite en prenant simplement l'intersection des ensembles d'états terminaux, c'est à dire ici le singleton $\{q_f\}$
Tout automate dont la relation ∇ est sans circuit reconnaît donc un élément de la clôture booléenne de $REC_=$

4. 5 Entre la clôture booléenne de $REC_=$ et $REC_{\neq}$

Proposition.

$REC_{\neq}$ n'est pas la clôture booléenne de la famille $REC_=$

La preuve est faite en exhibant une forêt de $REC_{\neq}$ qui ne peut être reconnue par un automate dont la relation ∇ est antisymétrique.

Considérons les arbres suivants : les extrémités de branches sont des mots de $a^*\bar{a}$ et à "la base" on trouve des c, lettre de arité 3.
Parmi ces arbres, on sélectionnera ceux qui possèdent au moins un c dont les deux premiers fils sont identiques, mais différents du troisième. Peu importe la place où se trouve ce c.
Cette forêt appartient à $REC_{\neq}$, elle est reconnue, par exemple, par l'automate $\bar{a} \rightarrow q_0$
$a(q_0) \rightarrow q_0$

$c(q_1,q_2,q_3)[d_{inf}] \rightarrow q_{nf}$	
$c(q_1,q_2,q_3)[d_{correcte}] \rightarrow q_f$	si q_f n'apparaît pas parmi q_1,q_2,q_3
$c(q_1,q_2,q_3)[d_{sup}] \rightarrow q_{nf}$	
$c(q_1,q_2,q_3)[d] \rightarrow q_f$	ssi q_f apparaît parmi q_1,q_2,q_3; d quelconque

La relation ∇ définie sur cet automate n'est évidemment pas antisymétrique, puisque $q_{nf} \nabla q_f \nabla q_{nf}$
Montrons qu'on ne peut supprimer l'antisymétrie.
Soit A, un automate déterministe quelconque reconnaissant F, nous supposerons que la relation ∇ définie sur ses états est antisymétrique.
Appelons q_0 l'état atteint par les mots $a^*\bar{a}$. Nous pouvons supposer cet état unique, dans la mesure où il n'intervient que dans les extrémités des branches d'arbres et qu'il ne peut se retrouver après avoir rencontré un c, et cela quel que soit l'automate A choisi.
Il doit nécéssairement exister trois règles :
$c(q_0,q_0,q_0)[d_{inf}] \rightarrow q_1$
$c(q_0,q_0,q_0)[d_{correcte}] \rightarrow q_2$
$c(q_0,q_0,q_0)[d_{sup}] \rightarrow q_3$
Avec $q_1 \nabla q_2 \nabla q_3$, états qui doivent, pour respecter l'antisymétrie, être distincts deux à deux (ils ne peuvent être confondus car q_2 est terminal, et non les deux autres).
Ecrivons maintenant l'ensemble des règles faisant intervenir ces trois états:
$c(q_1,q_1,q_1)[d_{inf}] \rightarrow q_{11}$
$c(q_1,q_1,q_1)[d_{correcte}] \rightarrow q_{12}$
$c(q_1,q_1,q_1)[d_{sup}] \rightarrow q_{13}$

et

$c(q_2, q_2, q_2)[d_{inf}] \rightarrow q_{22}$
$c(q_2, q_2, q_2)[d_{correcte}] \rightarrow q_{23}$
$c(q_2, q_2, q_2)[d_{sup}] \rightarrow q_{23}$

enfin,
$c(q_3, q_3, q_3)[d_{inf}] \rightarrow q_{31}$
$c(q_3, q_3, q_3)[d_{correcte}] \rightarrow q_{32}$
$c(q_3, q_3, q_3)[d_{sup}] \rightarrow q_{33}$

Nous en déduisons les relations suivantes :
$q11 \nabla q12 \nabla q13 \nabla q23 \nabla q33$
avec $q11 \neq q12 \neq q13 \neq q23 \neq q33$
(ils sont, en alternance, terminaux et non-terminaux)

Pour respecter l'antisymétrie, ils doivent donc être tous distincts. En considérant les règles faisant intervenir ces 5 nouveaux états, on montrerait alors de la même façon l'existence de 7, puis 9 ,etc,...états distincts.

D'une manière générale, on montre donc que l'on peut construire un ensemble d'états
$E_j = \{q_\omega | \omega = 1^k.\alpha.3^{i-k-1}, \alpha = 0, 1, 2\ k \leq i\}$
tous distincts, tels que
$q_{1^k.1.3^{i-k-1}} \nabla q_{1^k.2.3^{i-k-1}} \nabla q_{1^k.3.3^{i-k-1}} \nabla q_{1^{k-1}.1.3^{i-k}}$

On a donc vérifié cette propriété pour i=1 (et 2). Si elle est vraie pour un rang n fixé, on considère les règles définies sur les états de E_n :
$c(q_\omega, q_\omega, q_\omega)[d_{inf}] \rightarrow q_{\omega.1}$
$c(q_\omega, q_\omega, q_\omega)[d_{correcte}] \rightarrow q_{\omega.2}$
$c(q_\omega, q_\omega, q_\omega)[d_{sup}] \rightarrow q_{\omega.3}$

Ce qui nous donne, pour l'état q_{1^n} les relations : $q_{1^n.1} \nabla q_{1^n.2} \nabla q_{1^n.3}$ tous distincts puisque $q_{1^n.2}$ est terminal et non les autres. Remarquons d'ailleurs que les seuls états terminaux sont ceux donc l'indice contient un "2", qui représente l'utilisation de la règle de $d_{correcte}$

Les autres relations viennent en considérant que si $q_\omega \nabla q_{\omega'}$ et $q_\omega \neq q_{\omega'}$, c'est encore

vrai pour $q_{\omega.3}$ et $q_{\omega'.3}$.

La propriété est montrée, et il faudrait donc une infinité d'états pour construire un automate reconnaissant F et dont la relation ∇ soit antisymétrique.

Bibliographie

[ArDa 76] A. Arnold & M. Dauchet : " Bitransductions de forêts."
Third International Colloquium on Automata, Languages and Programming Edinburg University Press (1976)

[ArDa 78] A. Arnold & M. Dauchet : " Forêts algébriques et homomorphismes inverses."
Inform. and Control, 37, pp 182-196 (1978)

[BoDaWa 88] F. Bossut, M. Dauchet, B. Warin : " Rationality and recognizability on planar acyclic graphs."
MFCS'88, Lecture Notes in Computer Sc. 324 (1988)

[Bra 68] W.S. Brainerd : " The minimalization of tree automata."
Inf. and Control, 13, p 484 (1968)

[CoDaTi 89] J.L. Coquidé, M. Dauchet, S. Tison : " About connections between syntactical and computational theory."
FCT 89, Lecture Notes in Computer Science, 380 (1989)

[Com 88] H. Comon : " Unification et Disunification. Théorie et applications."
Thèse de l'INPG (Grenoble) (1988)

[Com 89] H. Comon : " Inductive proofs by specification transformations."
Proc. Rewriting technics and applications, Lecture Notes in Computer Sc. 375 (1989)

[Com 90] H. Comon : " Equational formulas on order-sorted algebras."
Proc. ICALP 90, Springer-Verlag, pp 674-688 (1990)

[Coq 90] J.L. Coquidé : " Controles et preuves dans les systèmes clos. Automates à piles d'arbres et calculs de formes normales."
Thèse de l'Université de Lille (1990)

[Cou 89] B. Courcelle : " On recognizable sets and tree automata."
Resolutions of equations in algebraic structures. Academic Press, M.Nivat & Ait-Kaci ed. (1989)

[DaHeLeTi 87] M. Dauchet, P. Heuillard, P. Lescanne, S. Tison : " Decidability of the confluence of ground term rewriting systems."
2nd Symp. on Logic in Computer Sc., IEEE Comp. Soc. Press (1987)

[DaTi 90] M. Dauchet & S. Tison : " Réduction de la non-linéarité des morphismes d'arbres."
Rapport IT 196, LIFL, Université de Lille (1990)

[Dau 89] M. Dauchet : " Simulations of Türing machines by a left-linear rewrite rule."
Proc 3rd RTA, Lect Notes in Computer Science, 355 (1989)

[DeGi 89] A. Deruyver & R. Gilleron : " The reachability problem for ground rewrite systems and some extension."
CAAP 89, Lect. Notes in Computer Sc. (1989)

[Dev 90] P. Devienne : " Weighted graphs, a tool for studying the halting problem and time complexity in logic programming."
TCS 75 (1990)

[EiWr 67] S. Eilenberg & J. Wreight : " Automata in general algebra."
Inf. and Control, 11, pp 52-70 (1967)

[JoKo 86] J.P. Jouannaud & E. Kounalis : " Automatic proofs by induction in equationnal theory without constructors."
Proc. 1st IEEE symp. Logic in Computer Science, Cambridge, Mass. (1986)

[GéSt 84] F. Gécseg & M. Steinby : " Tree automata."
Akadémiai Kiadó, Budapest (1984)

[Gil 90] R. Gilleron : " Reconnaissabilité et fragments décidables en réécriture. Automates à piles et calculs de formes normales."
Thèse de l'Université de Lille (1990)

[Gue 83] I. Guessarian : " Pushdown tree automata."
Math. systems theory, 16 (1983)

[HoUl 79] J.E. Hopcroft, J.D. Ullman : " Introduction to automata theory, languages and computation."
Addison-Wesley Publ. Comp. (1979)

[KaNaZh 85] D. Kapur, P. Narendran, H. Zhang : " On sufficient completeness and related properties of term rewriting systems."
Research Report TR 87-26, Computer science department, State University of New-York at Albany (1985)

[Kou 90] E. Kounalis : " Testing for inductive (co-)reductibility."
Proc CAAP 90 (1990)

[Mah 88] M.J. Maher : " Complete axiomatizations of the algebras of finite, rationnal and infinite trees."
Proc 3rd IEEE Symp. Logic in Computer Sc. (1988)

[Mon 81] J. Mongy : " Transformations de noyaux reconnaissables d'arbres. Forêts RATEG."
Thèse de l'Université de Lille (1981)

[MuSaSc 88] D.E. Muller, A. Saoudi, P.E. Schupp : " Weak alterning automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time."
Proc. 3rd IEEE Symp. Logic in Computer Sc., p 422, (1988)

[MuSc 87] D.E. Muller & P.E. Schupp : " Alterning automata on infinite trees."
Theoretical Computer Sc., 54, p 287 (1987)

[Oya 87] M. Oyamagushi : " The Church-Roser property for ground term rewriting systems is decidable."
TCS 49, pp 43-79 (1987)

[Oya 90] M. Oyamagushi : " The reachability and joinability problems for right-ground term rewriting systems."
J. Inf. Process (to appear)

[Pla 85] D.A. Plaisted : " Semantic confluence tests and completion method."
Inf. and control, 65 (1985)

[Rab 69] M.O. Rabin : " Decidability of second order theories and automata on infinite trees."
American Math. Soc. (1969)

[Rab 77] M.O. Rabin : " Decidable theories."
Handbook of Mathematical Logic North-Holland ed. pp 595-627 (1977)

[Rém 82] J.L. Rémy : " Etudes des systèmes de réécriture conditionnels et applications aux types abstraits algébriques."
Thèse de l'INPL (Nancy) (1982)

[Tha 73] J.W. Thatcher : " Tree automata : an informal survey."
Currents in the theory of computing. A.V. Aho ed. Prentice Hall (1973)

[Tho 90] W. Thomas : " Automata on infinite objects."
Handbook of Theoretical and Computer Sc. (to appear)

[Tis 89] S. Tison : " The fair termination is decidable for ground systems."
Rewriting Technics and Applications, Lect. Notes in Computer Sc., 355 (1989)

[Tis 90] S. Tison : " Automates comme outils de décision dans les arbres."
Mémoire d'Habilitation, Université de Lille (1990)

Table des Matières

Dans la classe des ensembles rationnels d'arbres, les automates permettent de décider de l'appartenance d'un terme. Ils sont assimilables à une signature dans les algèbres sortées initiales.

Nous proposons une extension des automates d'arbres, en ajoutant à chaque règle une condition portant sur les égalités entre termes frères.

Nous définissons deux classes d'automates. Dans la première, les règles ne peuvent imposer que des égalités, pas de différences. La classe des forêts reconnues coïncide alors avec celle des algèbres à signature non-linéaire.

Une deuxième classe, strictement plus large, est obtenue en permettant d'imposer des différences. Il s'agit alors d'une algèbre de Boole, close par morphismes alphabétiques d'arbres. Nous prouvons que chaque forêt peut être reconnue par un automate à tests déterministe, et que l'on peut décider si la forêt reconnue par un automate à tests est vide. La propriété du vide est NP-dure dans le cas général. Nous donnons un algorithme de complexité polynomiale dans le cas où les automates sont déterministes.

Laboratoire d'Informatique Fondamentale de Lille
U.A. 369 du CNRS
Université des Sciences et Techniques de Lille Flandres-Artois
BP 36 - 59655 Villeneuve d'Ascq cedex
Téléphone : 20 43 44 92 Télécopie : 20 43 65 66