Automates d'arbres à jetons

Mathias Samuelides

Université Paris 7 – Denis Diderot
UFR d'informatique

THÈSE

pour l'obtention du titre de
Docteur de l'Université Paris 7
Spécialité Informatique

présentée par
Mathias SAMUELIDES

sur le sujet

Automates d'arbres à jetons

soutenue publiquement le 17 décembre 2007 devant le jury suivant :

Joachim NIEHREN	Rapporteur
Helmut SEIDL	Rapporteur
Olivier CARTON	Examinateur
Stéphane DEMRI	Examinateur
Jean-Eric PIN	Examinateur
Anca MUSCHOLL	Directrice de thèse
Luc SEGOUFIN	Directeur de thèse

Remerciements

Je tiens tout d'abord à remercier très sincèrement Anca Muscholl de m'avoir fait découvrir les machines de Turing et la théorie de la complexité en cours de maîtrise puis d'avoir accepté d'être ma directrice de thèse et de m'avoir ainsi accordé sa confiance tout en me laissant beaucoup d'autonomie. Elle a fixé le cadre de cette thèse et ces orientations générales. Anca m'a également présenté Luc vers la fin de mon DEA et je lui en suis également très reconnaissant.

Luc a su parfaitement co-encadrer mes travaux, il a trouvé les formules exactes pour me relancer lorsque je désespérais de trouver des résultats, ses conseils précieux m'ont énormément appris sur le travail de chercheur. Luc m'a réellement impressionné par sa qualité d'écoute et de compréhension ainsi que par sa sympathie et son humour qui font de lui un être remarquable. Je tiens donc naturellement à lui exprimer ma profonde gratitude et je suis conscient de l'incroyable chance que j'ai eu d'avoir travaillé avec lui.

Je remercie chaleureusement tous les membres du jury :

Merci donc à Helmut Seidl d'avoir accepté de lire et de rapporter cette thèse rédigée en français et à Joachim Niehren pour l'intérêt qu'il a porté à mes travaux malgré l'absence de résultats XML et pour contribuer à prolonger mes recherches en me proposant des thématiques nouvelles et intéressantes.

Merci également à Jean-Eric Pin, à Olivier Carton et à Stéphane Demri de m'avoir fait l'honneur de participer à ce jury.

Je tiens ensuite à exprimer ma reconnaissance à Olivier Serre, mon ange gardien de Chevaleret, pour sa gentillesse, sa bonne humeur, sa bienveillance à mon égard et ses nombreux conseils.

Je remercie Mikolaj, aka the Bo2s of C.S., qui a le bon goût de ne pas aimer les trop longues formules mathématiques et avec qui j'ai eu l'honneur de travailler et

de discuter de musique.

Je remercie Thomas pour avoir relu certains passages de ma thèse et pour m'avoir convaincu de prendre un délicieux double whopper à Budapest.

Sans les résultats de Mikolaj et de Thomas sur les automates cheminants, le chapitre 6 de ce manuscrit n'aurait pas vu le jour.

Je remercie Joost Engelfriet pour l'intérêt et le temps qu'il a porté à mes travaux.

Merci également à Maarten Marx pour son accueil très chaleureux à Amsterdam.

J'ai également une pensée pour Laurent Van Begin et Jean-François Raskin qui m'ont permis de découvrir la recherche et de passer un séjour inoubliable en Belgique avant de commencer cette thèse.

Je remercie aussi Christof Löding : les quelques moments passées à travailler avec lui devant la machine à café de Chevaleret m'ont été très précieux.

Je remercie également Thomas Schwentick avec qui j'ai eu la chance de travailler.

Merci à tous les membres du LIAFA qui permettent à ce laboratoire d'être un lieu de travail particulièrement agréable.

Je remercie en particulier Marie pour sa gentillesse, son excellente humeur et aussi pour avoir relu un chapitre complet (peut-être le seul sans faute de typo !).

Je pense également à toutes les autres personnes avec lesquelles j'ai partagé le bureau 6A09 : Cesara, Hoang Anh, Laura, Selma, Blaise, Hugo (et son ftp), Berke (et sa vitamine C), Philippe et Mathilde.

Je remercie Noëlle pour sa bienveillance et son efficacité. Je n'oublie pas Laifa et Houy qui ont toujours été là pour régler mes difficultés avec le matériel informatique.

Merci aussi à Claire, Florian (sors ton tigrou de ta poche et passe le à la machine le pauvre !) Mohamed Faouzy, Peter, Pierre, Thierry et à tout ceux que j'oublie.

Merci également à toute l'équipe GEMO et à toute l'équipe MOSTRARE.

Merci surtout à ma famille : mon père qui a réussi à s'intéresser, à lire et à comprendre les premiers chapitres de cette thèse, merci aussi à ma mère, ma soeur, mon frère et Nicolas qui ont toujours été là pour me soutenir et sur qui je pourrai toujours compter.

Une pensée pour les habitants de l'ex-bâtiment F de l'ENS Cachan qui m'ont rappelé durant plus de trois ans qu'il y a des problèmes plus difficiles que ceux qui concernent les automates d'arbres.

Enfin je garde le plus beau pour la fin, (ou plutot la plus belle sisi) et je remercie donc Hortense pour m'avoir supporté pendant plus de trois ans et trois mois en espérant que cela continue.

TABLE DES MATIÈRES

CHAPITRE

1

Motivations et état de l'art

1.1 Motivations

Il était une fois, il y a bien longtemps, un petit Poucet perdu dans une forêt, il aimait se promener dans des arbres avec une pile de cailloux qu'il pouvait caler entre deux branches... Les travaux présentés dans cette thèse n'ont pas pour principale motivation l'étude des propriétés du petit Poucet.

1.1.1 XML et les arbres

Les premiers résultats notoires en bases de données datent des années 70 avec les travaux de E.F. Codd. Ses recherches ont abouti à une compréhension poussée des bases de données relationnelles. Les bases de données sont modélisées par des structures relationnelles du premier ordre. Les requêtes sont alors vues comme des applications de l'ensemble des structures relationnelles dans lui même. Si cette approche convient bien à des données très homogènes, la nécessité de manipuler des informations à la structure moins figée a motivé la recherche d'autres formalismes. Avec l'apparition et le développement de XML, la représentation des données sous forme de table laisse la place à une représentation sous forme d'arbres finis ordonnés et étiquetés.

XML (Extensible Markup Language) est une norme permettant de structurer des documents textes sous forme d'arbres au moyen de balises en vue de les stocker, de les transporter et de les échanger [54]. Ce langage est apparu comme un bon compromis entre la richesse de SGML et la simplicité de HTML et s'est actuellement imposé comme un format standard et générique pour les bases de données semi-structurées. SGML (Standard Generalized Markup Language) est un langage de description à balises apparu dans les années 80 dans lequel la structure logique d'un document, qui

```
<personne>
<prenom>Mathias<\prenom>
<date><jour>29<\jour><mois>4<\mois><annee>1980<\annee><\date>
<films>
<dvd><titre>Spartcacus<\titre><annee>1960<\annee><\dvd>
<dvd><titre>STARWARS<\titre><\dvd>
<\films>
<\personne>
```

FIG. 1.1 – Un document XML

est identifiée par des balises insérées dans le document lui-même, est séparée de sa mise en page, qui dépend du support de présentation et qui est définie en dehors du document dans une ou plusieurs feuilles de style [56]. L'objectif d'XML est de définir un langage aussi générique que SGML mais beaucoup plus simple afin de pouvoir servir un plus grand nombre d'utilisateurs et de remplacer le langage HTML pour un grand nombre d'applications. Le langage HTML (Hypertext Markup Language) a été conçu en même temps que http en 1990 par Tim Berners-Lee pour afficher les données d'un document par l'intermédiaire d'un navigateur dont dépend l'affichage [58].

XML est une norme du W3C depuis février 1998. Ce langage informatique libre de droits, et indépendant des plates-formes dont le développement a commencé en 1996 est actuellement largement utilisé par les industries pour la formulation de documents d'échange de données dans des applications de commerce électronique ou de bibliothèques en ligne. XML est un modèle de représentation des données sous forme de textes organisés en arbres au moyen de balises. Contrairement au langage HTML qui définit la signification visuelle de chaque balise et de chaque attribut, le langage XML utilise les balises seulement pour délimiter les éléments de données et laisse l'entière interprétation des données à l'application qui les lit. XML est donc un langage de syntaxe mais pas de sémantique.Pour l'affichage des documents, la norme XML prévoit des feuilles de style qui sont des ensembles de règles spécifiant la réalisation concrète d'un document XML. Cette technique est directement héritée de SGML. XML répond donc à l'attente "write once published anywhere".

Regardons maintenant avec la figure 1.1 à quoi ressemble un document XML.

Un document XML est dans un premier temps une chaîne de caractère ou un mot. Une des conditions de bonne formation des documents XML est le bon parenthésage de ces balises également appelées tags qui permet de voir le document comme un arbre. Un document XML a ainsi une structure d'arbre et le document de l'exemple peut également être représenté par l'arbre de la figure 1.2.

Les nœuds de l'arbre représentant le document XML sont appelés les éléments. Les éléments définissent la structure du document et sont décrits par leur nom leur tag ouvrant $< \cdots >$, leur tag fermant $< \backslash \cdots >$ leur étiquette et leurs attributs. Un attribut est une propriété associée aux éléments. Dans notre exemple, les éléments n'ont pas d'attribut. Nous nous intéressons à la structure des documents XML et non à leur contenu, ainsi nous ne considérons pas les attributs des documents XML.

La lecture du mot de la figure mot correspond à un parcours en profondeur de

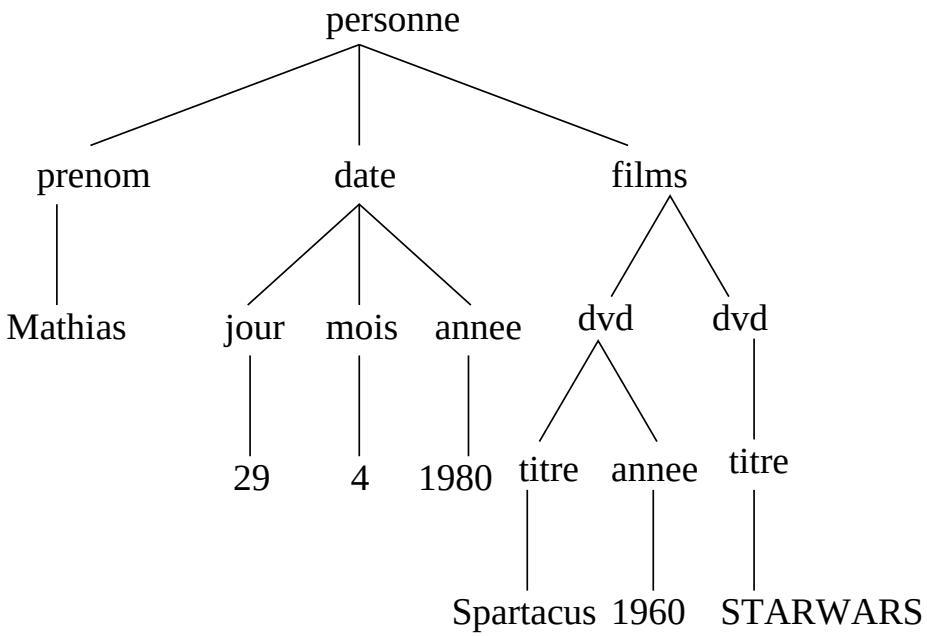

FIG. 1.2 – Arbre représentant le document XML de la figure 1.1

gauche à droite de l'arbre de la figure 1.2 qui représente ce document. Lire un tag ouvrant correspond à visiter l'élément du même nom et lire le tag fermant correspond à remonter au père de l'élément du même nom. La plupart des outils travaillant sur un document XML considèrent la structure d'arbre intrinsèque du document.

Sous cet angle, XML est une syntaxe concrète pour un modèle abstrait d'arbre fini. La spécification du langage XML ne fixe pas l'ensemble des étiquettes autorisées dans un document XML et ne définit pas de sémantique pour les étiquettes. La seule condition que doit impérativement vérifier n'importe quel document XML est d'être bien formé, c'est à dire d'avoir une structure arborescente Un usager peut ainsi définir un type de document XML qui correspond à un langage d'arbres. Les documents XML ont une structure d'arbre fini d'arité non-bornée c'est-à-dire que le nombre de fils d'un nœud n'est pas borné. Les automates d'arbres étudiés dans les prochains chapitres de ce travail sont des automates sur des arbres binaires finis. Il est cependant possible de coder un arbre d'arité non-borné en un arbre binaire et de nombreux problèmes sur les arbres de rang non-bornés se ramènent ainsi à des problèmes analogues sur les arbres binaires. Nous décrirons un tel encodage dans le chapitre 8.

Les automates d'arbres finis, outil essentiel de l'informatique sont des modèles utilisés pour le typage de documents XML. Ils servent également à définir ou à comparer des langages de requêtes pour documents XML.

1.1.2 Les DTD et les automates

Dans un usage particulier, on peut vouloir se limiter à certaines formes de documents. Un usager doit pour cela définir un type XML, c'est-à-dire une description de contraintes sur la structure du document. Par exemple, un ensemble d'étiquettes autorisées et un ensemble de successeurs possibles sont fixés pour un élément dans

```
<!ELEMENT personne (nom, date, films?)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT date (jour, mois, année)>
<!ELEMENT jour (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT année (#PCDATA)>
<!ELEMENT films (dvd*)>
<!ELEMENT dvd (titre, année?)>
<!ELEMENT titre (#PCDATA)>
```

FIG. 1.3 – Une DTD

la structure arborescente. Un type XML donné définit alors le langage des arbres qui satisfont ses contraintes et peut donc être vu comme une grammaire d'arbres ou comme un automate d'arbres. Le typage des documents permet d'améliorer le stockage, de faciliter les requêtes et la navigation dans les données et de protéger les données. Les programmes ont donc recours au typage pour représenter la structure de données et l'échange de données XML est presque toujours dans le contexte d'un type fixé. Deux niveaux de correction peuvent ainsi être distingués pour les documents XML :

- un document est *bien-formé* s'il obéit aux conditions syntaxiques définies par la spécification de XML qui lui permettent d'être interprété comme un arbre.
- un document est *valide* s'il est conforme aux contraintes sur la structure et le contenu spécifiques à une application, c'est-à-dire s'il est respecte un type donné.

Un document valide est toujours bien formé car le type décrit des contraintes sur l'arbre et non sur la représentation textuelle du document XML.

Les DTD (document type definition) constituent le moyen le plus simple et le plus fréquemment utilisé pour valider des échanges de données. Les DTD s'apparentent aux grammaires utilisées en analyse syntaxique. La figure 1.3 présente un exemple de DTD. Une DTD associe à chaque élément une expression régulière sur les types. Les différents indicateurs d'occurrence utilisés dans cette DTD sont le symbole "*" qui signifie 0 ou plusieurs éléments et le symbole " ?" qui signifie 0 ou 1 élément.

Le type "#PCDATA" signifie que l'élément contient des données textuelles. Les grammaires étendues sans contexte sont utilisées pour modéliser les DTD [38].

Le problème de validation peut alors être formalisé de la manière suivante : étant donnés un arbre t et un langage d'arbres régulier défini par un type τ, nous voulons décider si l'arbre t appartient à ce langage. Il s'agit en pratique de vérifier si un document XML est conforme aux contraintes d'un type donné.

Le document de la figure 1.1 est valide relativement au type défini par la DTD de la figure 1.3.

La validité d'un arbre relativement à une DTD peut être testée par les automates d'arbres déterministes top-down. Ces automates sont des machines à états finis avec un contrôle parallèle qui évaluent un arbre de sa racine vers les feuilles. Le modèle séquentiel des automates d'arbres cheminants peut également être utilisé pour le problème de validation. Un automate cheminant est une machine à états finis avec une

unique tête de lecture qui peut se déplacer dans un arbre. Ainsi, il peut effectuer un parcours en profondeur de gauche à droite de l'arbre représentant un document afin de vérifier pour chaque nœud que la suite de ses fils correspond bien aux contraintes imposées par la DTD. Dans le problème de validation en "streaming" d'un document XML, le document XML doit être parcouru en une seule passe (sans revenir en arrière) et avec une mémoire fixée dépendant de la DTD mais pas du document à valider. Un document XML est alors considéré comme un mot d'un langage à balises Le parcours par un automate unidirectionnel d'un document XML considéré comme un mot correspond au parcours en profondeur de l'arbre représentant ce document par un automate cheminant. Il est montré dans [28] que le modèle d'automates sur les mots approprié pour résoudre le problème de validation en streaming d'un document XML est en fait celui des automates à piles avec visibilité (Visibly Pushdown Automata). Dans [33], A Neumann et H Seidl ont étudié les automates de forêts à piles (Pushdown Forest Automata) utilisés dans des algorithmes qui testent la validité d'un document XML pour une DTD ou plus généralement qui évaluent en ligne des requêtes d'arbres régulières.

1.1.3 XSLT, typechecking et transducteurs à jetons

XSLT est un langage permettant de transformer un document XML en un autre document XML ou pour convertir des données XML en document HTML [57]. Neven et Schwentick ont montré dans [34] qu'une transformation XSLT est essentiellement un transducteur cheminant à registres, c'est-à-dire une machine qui parcourt un arbre sur un alphabet infini comme un automate cheminant et qui produit un nouvel arbre en sortie. L'alphabet infini permet de modéliser la présence des attributs dans un document XML. Les transducteurs d'arbres a jetons ont aussi été définis comme un modèle abstrait et général de transformation de langages XML. Un tel transducteur utilise un nombre fixé de jetons pouvant être placés sur les nœuds de l'arbre. Une discipline de pile est imposée aux jetons pour que le typechecking soit décidable : les jetons sont numérotés et doivent être posés dans l'ordre croissant de leur numéro et levés dans l'ordre décroissant de leur numéro. Dans [31], il est montré que les automates d'arbres à jetons ont le pouvoir navigationnel de nombreux langages de requêtes existant pour XML. L'étude des automates cheminants peut aussi permettre de mieux comprendre le pouvoir d'expression de XSLT. Il est affirmé dans [22] que les transducteurs cheminants avec des jetons invisibles peuvent modéliser les mécanismes de récursion de XSLT. Pour cette variante de transducteur cheminant à jetons, un jeton invisible peut être détecté par le transducteur seulement si c'est le dernier jeton dans l'arbre à avoir été posé.

Les automates d'arbres sont également utilisés pour le problème du typechecking qui demande une compréhension des interactions entre la logique et les langages d'arbres. Le problème du typechecking consiste à analyser un programme. Il diffère donc du problème de validation qui concerne l'analyse d'un document. Le problème du typechecking est un problème fondamental pour l'échange de données XML et la conversion des bases de données relationnelles en documents XML. Ce problème est le suivant : étant donnés une transformation de documents XML et un type pour l'arbre de sortie, nous voulons vérifier automatiquement que chaque donnée de

l'entrée a pour image un arbre du type de sortie désiré. Nous considérons donc τ un langage d'arbres régulier donné par un type et T une fonction définie par un programme d'un ensemble d'entrées D vers un ensemble d'arbres et nous voulons décider si pour toute entrée $x \in D$, on a bien $T(x) \in \tau$.

Un typechecker est un module qui analyse simultanément le programme et la sortie XML et qui décide si tous les documents produits par le programme sont valides en retournant oui ou non.

L'étude formelle du problème du typechecking est complexe car les langages de transformation XML et les notions de DTD évoluent constamment.

T Milo, D. Suciu et V. Vianu ont montré que, si on modélise une transformation de documents XML avec un transducteur à k jetons, le problème du typechecking est décidable [31].

1.1.4 Requêtes et XPath

Un des principaux sujets de recherche dans la théorie des bases de données est de caractériser le pouvoir d'expression et la complexité de calcul des langages de requêtes. L'accès aux données d'un document est effectué par l'exécution de requêtes. Ces requêtes sont formalisées par la logique.

L'intérêt des automates d'arbres à jetons s'est récemment accru en raison des liens entre ces automates et les langages de requêtes.

XPath est un langage standard introduit par W3C permettant de sélectionner un ensemble d'éléments dans un document XML et définissant des chemins pour naviguer dans les documents XML [55]. Grâce a son pouvoir d'expression et sa syntaxe compacte, XPath joue un rôle central dans les applications XML. Il est utilisé dans de nombreuses applications. Par exemple, il constitue le noyau du langage de requêtes dans XQuery [59], il permet de sélectionner des nœuds pour les transformations dans XSLT et il est utilisé pour définir des clés dans XML Schema. M. Marx et B. ten Cate ont considéré CoreXPath qui constitue le noyau navigationnel de XPath et l'ont associé avec des expressions régulières de chemins qui correspondent naturellement aux exécutions des automates cheminants. Les expressions XPath les plus simples ressemblent à des chemins de localisation de répertoires. Par exemple, considérons l'expression ci-dessous.

```
/child::personne/child::films
```

Son évaluation retourne l'ensemble des éléments atteint par un chemin partant de la racine imaginaire du document désignée par le premier slash, visitant ensuite un élément "personne" puis un élément "films". Cette requête appliquée à l'exemple de la figure 1.1 sélectionne les éléments dvd correspondant aux films "Starwars" et "Spartacus".

XPath autorise d'autres axes que les arêtes. Les chemins peuvent être définis avec un sous-ensemble des expressions régulières sur les axes de base suivants : "prévious sibling" et "next sibling" pour naviguer entre les nœuds ayant le même père, "child" et "parent" pour naviguer selon les arêtes. Une requête XPath peut également sélectionner un nœud en testant l'existence ou l'absence de chemins. Considérons par exemple l'expression ci-dessous :

`/child::personne/child::films/child:dvd[child::année]`

va sélectionner uniquement les nœuds "dvd" qui ont un élément "année" pour fils. Dans l'exemple, cette requête sélectionne l'élément "dvd" correspondant à "Spartacus".

Core XPath est le corps navigationnel de XPath. Dans [12], B. ten Cate définit Regular XPath comme l'extension de CoreXpath au moyen d'un opérateur de clôture transitive et d'un prédicat pour l'égalité de chemins. Il prouve en utilisant la caractérisation logique des automates à jetons introduite par Engelfriet et Hoogeboom que les automates d'arbres à jetons permettent d'évaluer les expressions de Regular XPath où toutes les occurrences des clôtures transitives sont positives. Un jeton est alors utilisé pour chaque quantificateur et l'automate d'arbres à jetons se déplace le long des chemins définis par une telle expression de Regular XPath sur un document XML.

On voit comment des développements applicatifs récents de thèmes aussi importants que les bases de données ou l'édition de documents XML font appel aux automates d'arbres, notamment aux automates d'arbres cheminants et aux automates d'arbres à jetons. L'objet de cette thèse est de contribuer à l'étude des propriétés de ces derniers objets. La section suivante est consacrée à une brève revue de l'état de l'art qui permet de mieux situer nos résultats. Ceux-ci seront présentés dans le plan de la thèse qui conclut ce chapitre d'introduction.

1.2 Etat de l'art

Le terme de théorie des automates a été introduits par J. von Neumann en 1948. Les machines de Turing définis en 1936 qui sont un modèle d'ordinateur contenaient déja la notion d'automates même s'il s'agit d'un modèle beaucoup plus général. Le premier résultat fondamental de cette théorie est du à Kleene qui montra en 1956 que les langages réguliers sur les mots qui sont définis par les expressions rationnelles sont exactement les langages reconnus par un automate unidirectionnel déterministe sur les mots.

Les mots sont des objets de base très utilisés en informatique. Un automate sur les mots est un modèle pour représenter de façon concise un ensemble potentiellement infini de mots. Un automate fini unidirectionnel sur les mots peut être vu comme une machine de Turing dont le type des transitions autorisées est très restreint : la tête de l'automate ne peut pas écrire sur le ruban, elle part de la gauche du mot et se déplace toujours en avançant d'une position vers la droite. Elle lit ainsi le mot de gauche à droite. Tout comme une machine de Turing, un automate unidirectionnel peut accepter ou rejeter un mot pris en entrée et définit ainsi le langage de l'ensemble des mots acceptés. Comme la tête de lecture d'un automate ne peut pas écrire sur le ruban, une configuration d'un automate sur un mot est donnée par l'état de contrôle et la position du ruban. Un automate déterministe n'a pas de choix à effectuer : étant donnée une configuration, il peut effectuer au plus une transition. Un automate déterministe peut ainsi servir de modèle à un algorithme puisqu'étant donné un mot pris en entrée il existe un unique calcul de l'automate sur ce mot. La notion d'automate non-déterministe a été introduite par J. Myhill en 1957 ainsi que

l'algorithme de déterminisation. Au cours d'un calcul, un automate non-déterministe choisit à chaque étape une transition à effectuer parmi un sous-ensemble de ses transitions qui dépend de la configuration courante. Il est possible de construire à partir d'un automate non-déterministe un automate déterministe qui reconnaît le même langage. Cette construction s'appelle la déterminisation. Les automates unidirectionnels déterministes et non-déterministes ont donc le même pouvoir d'expression mais le modèle non-déterministe est exponentiellement plus concis.

L'intérêt des automates par rapport aux autres méthodes pour décrire des ensembles est qu'ils rendent effectives de nombreuses opérations sur ces ensembles. Par exemple, étant donné deux automates, il est très facile de construire un automate reconnaissant l'union des langages qu'ils définissent et un automate reconnaissant leur intersection. On sait également complémenter un automate donné A c'est-à-dire construire un automate qui reconnaît le complément du langage défini par A.

Les automates bidirectionnels sont une extension des automates unidirectionnels où la tête de lecture peut se déplacer dans les deux directions. Ces automates peuvent être vus comme des machines de Turing qui ne peuvent pas écrire sur leur ruban. Contrairement aux automates unidirectionnels, le mot pris en entrée par un automate bidirectionnel n'est pas consommé lettre par lettre comme des pièces de monnaie dans une machine à café mais l'automate se promène sur ce mot. Le nouveau type de transitions n'augmente pas le pouvoir d'expression bien qu'il offre plus de possibilités pour l'automate.

Les automates à jetons sur les mots sont des automates bidirectionnels qui possèdent un nombre fixé de jetons pouvant être placés sur les lettres du mot. L'automate peut détecter quels sont les jetons posés sur la position de sa tête de lecture. Les jetons servent donc de repères à l'automate (un peu comme les cailloux du petit Poucet).

Le placement des jetons obéit à une hiérarchie de pile : seul le dernier jeton qui a été posé peut être levé par l'automate. Sans cette restriction on obtiendrait un modèle d'automates avec plusieurs têtes de lecture dont le pouvoir d'expression est celui des machines de Turing déterministes utilisant un espace logarithmique par rapport à la longueur du mot pris en entrée et qui reconnaît ainsi des langages non-réguliers [18]. M. Blum and C. Hewitt ont prouvé dans [5] que les langages reconnus par les automates à 1 jeton sur les mots sont réguliers.

N. Globerman et D. Harel ont étudié dans [24] une version des automates à jetons avec discipline de pile sur les mots avec deux restrictions supplémentaires :

– l'automate ne peut pas se déplacer à gauche du dernier jeton qu'il a posé,

– l'automate peut détecter uniquement la présence du dernier jeton qu'il a posé.

Ils ont montré que les automates à jetons ainsi définis reconnaissent seulement des langages réguliers et que les automates à k jetons sont k-exponentiellement plus concis que les automates bidirectionnels.

La logique monadique du second ordre (MSO) est un autre moyen de décrire les langages de mots. La logique du premier ordre quantifie sur les positions d'un mot et la logique du second ordre sur les ensembles de positions. Les formules closes de ces logiques permettent de définir des langages.

Il est montré, par exemple dans [53], que les automates unidirectionnels sur les mots ont le même pouvoir d'expression que la logique monadique au second

ordre. Dès les années 50, les automates ont été utilisés comme outil de décision en logique notamment par Büchi et Rabin. Ils jouent depuis un rôle très important en vérification.

De très nombreuses extensions des automates finis ont été étudiés : les automates alternants qui peuvent effectuer des calculs en parallèle, les automates à piles utilisés en théorie des jeux, les automates sur les mots infinis, les automates à compteurs, les automates à horloges, les automates hybrides, les réseaux de Petri, les automates à registres pour reconnaître des mots sur un alphabet infini...

Des modèles d'automates reconnaissant des ensembles d'arbres ont également été définis. Les arbres sont des objets fondamentaux en informatique. Donald Knuth décrit les arbres comme *the most important non-linear structures that arise in computer algorithms*. Un mot peut être vu comme un arbre particulier dans lequel chaque nœud a au plus un fils. Un tel arbre est dit unaire. Les arbres binaires sont un modèle d'arbre pour lequel chaque nœud a 0 ou 2 fils. Deux types de langages d'arbres sont distingués : les langages d'arbres de rang borné et les langages d'arbre de rang non-borné. Les nœuds des arbres d'un langage d'arbres de rang borné ont un nombre de fils borné par une constante qui dépend du langage alors que dans un langage d'arbres de rang non-borné, chaque nœud peut avoir un nombre arbitraire de fils. L'étude des automates sur les arbres de rang borné se réduit au cas des automates sur les arbres binaires. D'autres modèles d'arbres ont été considérés dans l'étude des automate d'arbres : de nombreux travaux ont été effectués sur les langages d'arbres infinis notamment dans le cadre de la théorie des jeux et de la vérification, sur les langages d'arbres de rang non-borné et sur les langages d'arbres sur un alphabet infini avec les automates cheminants à registres Nous considérons ici les langages d'arbres binaires finis étiquetés par un alphabet fini.

Les automates d'arbres bottom-up ont été introduits par Doner [15] et Thatcher et Wright [50] afin de prouver la décidabilité de la théorie du second ordre avec multiples successeurs. Un automate d'arbres bottom-up a un contrôle parallèle : il évalue un arbre des feuilles à la racine en affectant à chaque nœud un état en fonction de son étiquette et des états affectés à ces fils.

Les automates d'arbres bottom-up constituent le modèle d'automates d'arbres le plus utilisé car ils possèdent de nombreuses propriétés : un automate d'arbre bottom-up peut être déterminisé, complémenté, minimisé et on peut décider si le langage qu'il reconnaît est vide en temps polynomial.

De plus, les automates d'arbres bottom-up reconnaissent les langages réguliers qui sont définis par des formalismes algébriques tels que les grammaires et les morphismes de Σ-algèbres et il est connu depuis les années 60 que la logique MSO définit exactement les langages d'arbres réguliers. Les langages réguliers d'arbres sont les langages d'arbres reconnus par de nombreux autres modèles d'automates d'arbres dont le pouvoir d'expression que les bottom-up : les automates top-down non-déterministes, les automates d'arbres alternants, les automates d'arbres bidirectionnels alternants [23] ... Le modèle des automates d'arbres top-down a été introduits par Rabin. Les automates d'arbres top-down évaluent un arbre de la racine aux feuilles. Ce modèle est moins utilisé que le modèle des automates bottom-up car la variante déterministe des top-down est moins puissante que la variante non-déterministe : il existe des langages d'arbres réguliers qui ne sont pas reconnus par

les automates top-down déterministes. Tous les modèles d'automates qui définissent les langages réguliers ont un comportement parallèle, le contrôle de l'automate est effectué sur plusieurs nœuds de l'automate en même temps.

L'existence d'un modèle d'automates séquentiel reconnaissant les langages d'arbres réguliers est un problème ouvert.

Le modèle d'automate d'arbres séquentiel le plus naturel est celui des automates cheminants (tree-walking automata) introduits par Aho et Ullman en 1971 comme une alternative aux automates d'arbres à branchements [1]. L'étude de ce modèle s'est trouvée récemment relancée avec l'arrivée de XML, et plus particulièrement du langage XPath. Les automates cheminants étendent naturellement les automates bidirectionnels sur les arbres. Un automate cheminant a une unique tête de lecture qui se déplace dans l'arbre d'un nœud à un autre en suivant les arêtes. Il choisit son prochain déplacement en fonction de l'étiquette du nœud courant, de son état courant et du type du nœud courant. Le type d'un nœud est une information finie qui nous indique si le nœud considéré est la racine, un fils droit ou un fils gauche et une feuille ou un nœud interne. Cette information est nécessaire pour pouvoir effectuer un parcours en profondeur [27].

Il est facile de prouver que les langages reconnus par les automates cheminants sont réguliers en utilisant la caractérisation logique des langages. La réciproque est restée pendant longtemps un problème ouvert et est la première des trois questions fondamentales énoncées ci-dessous qu'on peut se poser naturellement sur les automates cheminants.

1. Les automates cheminants reconnaissent-ils tous les langages réguliers ?
2. Peut -on déterminiser n'importe quel automate cheminant ?
3. Peut-on complémenter un automate cheminant ?

Pour la première question, F. Neven et T. Schwentick ont d'abord étudié le cas des automates cheminants qui, au cours d'une exécution, ne retraversent jamais deux fois une arête dans le même sens [34]. Ils ont montré que les automates cheminants avec cette nouvelle restriction ne reconnaissent pas tous les langages réguliers. En 2005, M. Bojańczyk et T. Colcombet ont résolu la question dans le cas général : ils ont montré que la classe des langages reconnus par un automate cheminant est strictement incluse dans la classe des langages réguliers [7]. Ils ont également répondu à la deuxième question en prouvant qu'il existe un langage reconnu par un automate cheminant non-déterministe et qui ne peut être reconnu par aucun automate cheminant déterministe. La troisième question sur la complémentation d'un automate cheminant reste un problème ouvert. Dans [34], F. Neven et T. Schwentick ont établi une caractérisation logique des variantes déterministe et non-déterministe des automates cheminants : ils ont montré que les langages définis par les automates cheminants correspondent aux langages définis par la logique TC(FO) des formes normales de clôtures transitives de formule du premier ordre mais il reste encore à savoir si cette logique sur les arbres est fermée par complément ou non.

Dans [20], J. Engelfriet, H.-J. Hoogeboom et J.-P. Van Best ont travaillé sur des extensions du modèle des automates cheminants : les automates cheminants à pile (pushdown tree-walking automata), les automates cheminants à couleurs (tree walking marble automata), les automates cheminants avec tests MSO et les automates

à jetons. Ils ont introduit les automates d'arbres à jetons afin d'avoir un modèle séquentiel d'automates d'arbres dont le pouvoir d'expression est entre celui des automates cheminants et celui des automates d'arbres bottom-up [18]. Comme dans le cas des automates à jetons sur les mots une discipline de pile est imposée au placement des jetons afin de ne pas dépasser le cadre des langages réguliers (c'est-à-dire pour que les langages définis par les automates d'arbres à jetons soient toujours des langages réguliers)

Dans [19], J. Engelfriet et H.-J. Hoogeboom ont montré le lien très fort entre les automates d'arbres à jetons et la logique du premier ordre augmenté d'un opérateur de clôture transitive sur les arbres. Ils ont introduit un nouveau modèle d'automates à jetons et ont obtenu une caractérisation logique pour les variantes déterministe et non-déterministe de ce modèle. Dans leur nouveau modèle d'automates à jeton appelé le modèle fort, le placement des jetons suit toujours une hiérarchie de pile mais l'automate peut lever le dernier jeton qu'il a posé à distance : un jeton peut être levé au cours d'une exécution à partir de n'importe quel nœud alors que dans le modèle classique des automates à jetons un jeton ne peut être levé que si la tête de l'automate est dessus. Le modèle fort généralise donc le modèle classique que nous appelons le modèle faible. Il est alors naturel de se demander si les automates d'arbres du modèle fort reconnaissent plus de langages que les automates d'arbres du modèle faible. Dans le cas des mots, ces deux modèles d'automates à jetons sont équivalents et reconnaissent exactement les langages réguliers : en effet, tous les langages reconnus par des automates à jetons du modèle fort sont réguliers et les deux modèles d'automates à jetons sur les mots sont des extensions des automates unidirectionnels qui reconnaissent déjà tous les langages réguliers. Dans [18], J. Engelfriet et H.-J. Hoogeboom ont conjecturé que le modèle faible des automates d'arbres à jetons est moins puissant que le modèle des automates d'arbres bottom-up.

Un transducteur cheminant à jetons est une extension des automates à jetons qui produit un arbre en sortie. Dans [31], T. Milo, D. Suciu et V. Vianu ont étudié les transducteurs cheminants à jetons et ont montré que le problème du vide est non-élémentaire pour les automates à jetons lorsque le nombre de jetons fait partie de l'entrée, c'est-à-dire lorsque le nombre de jetons n'est pas fixé pour toutes les instances du problème.

Notons enfin qu'un modèle de transducteur cheminant avec un nombre non-borné de jetons invisibles a été étudié par J. Engelfriet, H.-J. Hoogeboom dans [22] pour modéliser les mécanismes de récursion de XSLT. Les automates correspondant à ces transducteurs reconnaissent tous les langages réguliers.

1.3 Organisation de la thèse et principales contributions

Ce travail porte sur l'étude de deux modèles séquentiels d'automates à jetons sur des arbres binaires finis étiquetés par un alphabet fini.

Le chapitre 2 introduit les langages réguliers et les automates à jetons sur les mots et sur les arbres. La logique MSO sur les mots et sur les arbres est tout d'abord définie dans la section 2.1. La section 2.2 présente ensuite le modèle classique des

automates unidirectionnels sur les mots et celui des automates bidirectionnels sur les mots qui correspondent aux automates à 0 jeton sur les mots. La section 2.3 porte sur les automates d'arbres bottom-up qui reconnaissent les langages d'arbres réguliers et le modèle séquentiel des automates cheminants qui sont en fait les automates à 0 jeton sur les arbres. Nous définissons alors dans la section 2.4 le modèle fort et le modèle faible des automates à jetons sur les arbres et sur les mots. Notons qu'une discipline de pile est imposée au placement des jetons et que, dans le modèle fort, un jeton peut être levé à distance alors que dans le modèle faible un jeton peut être levé uniquement s'il est posé sur le nœud courant. De nombreux exemples illustrent toutes ces définitions. Pour conclure ce chapitre, nous rappelons dans la section 2.5 que les langages réguliers de mots, respectivement d'arbres, sont les langages définis par la logique MSO, puis que tous les modèles d'automates séquentiels sur les mots définis dans ce chapitre ont le même pouvoir d'expression et enfin que les langages d'arbres reconnus par un automate d'arbres à jetons sont réguliers. Nous préciserons ce dernier résultat dans le chapitre 6.

M. Bojańczyk et T. Colcombet ont répondu récemment dans [7] et [6] à deux des trois questions fondamentales qui se sont posées naturellement dès l'introduction des automates cheminants par Aho et Ulman en 1971. La complémentation d'un automate cheminant non-déterministe reste donc le principal problème ouvert sur les automates cheminants. Dans le chapitre 3, nous prouvons que toutes les variantes déterministes des différents modèles d'automates d'arbres séquentiels définis au chapitre précédent sont fermées par complémentation [32]. Rappelons que pour complémenter un automate unidirectionnel déterministe sur les mots il suffit de le compléter puis d'intervertir les états acceptants et les états non-acceptants. Dans le cas des automates cheminants et des automates d'arbres à jetons déterministes nous montrons à l'aide d'un exemple que les exécutions infinies posent problème. Pour résoudre cette difficulté, nous utilisons une idée de Sipser. Nous traitons ainsi dans la section 3.2 le cas des automates d'arbres cheminants déterministes. Nous étendons sans difficulté cette preuve aux automates à jetons du modèle faible déterministe en 3.3.1 : étant donné A un automate à jetons du modèle faible, nous construisons avec une complexité quadratique un automate du modèle faible avec le même nombre de jetons qui reconnaît le complément. Dans le cas du modèle fort, nous adaptons cette construction en 3.3.2 et nous complémentons ainsi un automate à jetons du modèle fort avec une complexité polynomiale en multipliant par 3 le nombre de jetons.

J. Engelfriet et H.J. Hoogeboom ont établi dans [19] une caractérisation logique des variantes déterministe et non-déterministe du modèle fort des automates à jetons. Leur résultat implique que la classe des langages reconnus par un automates à jetons du modèle fort est fermée par complément. Notons que la construction du complément d'un tel automate donnée par cette approche n'est pas efficace en terme de jetons. Le chapitre 4 présente cette caractérisation. La section 4.1 introduit les logiques FO + posTC et FO+DTC qui sont des extensions de la logique du premier ordre au moyen d'un opérateur de clôture transitive. Nous énonçons ensuite dans la section 4.2 la caractérisation logique des automates à jetons du modèle fort de [19] et nous proposons une nouvelle présentation de la construction d'un automate à jetons déterministe reconnaissant le langage défini par une formule FO+DTC dans laquelle nous utilisons les résultats du chapitre 3.

Dans le chapitre 5, nous démontrons que le modèle fort et le modèle faible d'automate d'arbres à jetons ont le même pouvoir d'expression dans le cas déterministe et dans le cas non-déterministe [8]. Nous donnons dans un premier temps les idées principales de la preuve en traitant les cas particuliers des automates non-déterministes à 1 et 2 jetons. Nous introduisons ensuite dans la section 5.2 la notion de comportement d'un automate à jetons du modèle faible sur un arbre que nous utilisons pour le cas général. Etant donné un automate à jetons du modèle fort, nous construisons dans la section 5.3 un automate équivalent avec le même nombre de jetons. La section 5.4 montre comment préserver le déterminisme dans cette construction. Les résultats de ce chapitre impliquent une nouvelle construction du complément d'un automate à jetons du modèle fort dans laquelle on n'augmente pas le nombre de jetons et montrent que la caractérisation logique du modèle fort d'automate à jeton de J. Engelfriet et H.J. Hoogeboom s'applique également au modèle faible.

Dans le chapitre 6, nous étendons aux automates d'arbres à jetons les résultats de [6] et [7] sur les automates d'arbres cheminants [8]. M. Bojańczyk et T. Colcombet ont prouvé que les automates d'arbres cheminants non-déterministes reconnaissent plus de langages que les automates d'arbres cheminants déterministes. Ils ont également montré que les automates cheminants ne reconnaissent pas tous les langages réguliers. Nous montrons en 6.2.2 que le pouvoir d'expression des automates d'arbres à jetons augmente strictement avec le nombre de jetons et en 6.2.3 que, pour tout entier k, il existe un langage reconnu par un automate cheminant non-déterministe qui n'est reconnu par aucun automate déterministe à k jetons. Pour cela, nous utilisons deux familles de langages définies de manière inductive dans la section 6.1 à partir des langages de séparation définis dans [7] et [6]. Nous démontrons dans la section 6.3 que les automates d'arbres à jetons ne reconnaissent pas tous les langages réguliers. Ce résultat était conjecturé par J. Engelfriet and H.J. Hoogeboom dans [18]

Le chapitre 7 porte sur la complexité des problèmes de décision du vide et de l'inclusion pour les variantes déterministe et non-déterministe du modèle fort et du modèle faible d'automates d'arbres à k jetons où le nombre de jetons k est fixé. Dans [31], T. Milo, D. Suciu and V. Vianu montrent que ce problème est non-élémentaire si l'entier k fait partie de l'entrée. La section 7.1 introduit les notions et les théorèmes classiques de complexités en espace et en temps utilisés dans la suite du chapitre. Nous rappelons ensuite la complexité du problème du vide pour les automates d'arbres bottom-up et la transformation d'un automate cheminant en un automate bottom-up équivalent qui est présentée dans [14] et qui implique que le problème du vide est ExpTime pour les automates cheminants. La section 7.3 étend cette transformation aux automates à jetons du modèle fort et établit ainsi la borne supérieure des problèmes du vide et de l'inclusion pour les automates à k jetons du modèle fort non-déterministes. Dans la section 7.4, pour tout entier k strictement positif, nous simulons une machine de Turing alternante à espace $(k-1)$-exponentiel avec un automate à k jetons déterministe du modèle faible. Nous prouvons ainsi que les problèmes du vide et de l'inclusion sont des problèmes k-ExpTime complets pour tous les modèles d'automates d'arbres à k jetons considérés [42]. Nous montrons aussi que ces deux problèmes sont $(k-1)$-ExpSpace complets pour les automates à k jetons sur les mots.

Pour conclure, le chapitre 8 rappelle les résultats de ce travail et en dégage quelques perspectives. Nous exposons d'abord quelques problèmes ouverts posés dans le cadre abordé . Nous proposons ensuite des extensions des automates d'arbres à jetons qu'il nous semble important d'étudier maintenant au regard des motivations exposées au début de ce premier chapitre.

CHAPITRE

2

Des automates sur les mots aux automates d'arbres à jetons

Sommaire

Ce chapitre d'état de l'art est consacré aux automates et à la logique sur les arbres et les mots. Nous introduisons dans un premier temps les mots et les arbres : nous définissons les structures relationnelles qui leur sont associées et la logique monadique du second ordre sur ces différents modèles de structures de données. Nous présentons différents modèles pour les arbres dans la section 2.1 : les arbres binaires, les arbres de rang borné et les arbres de rang arbitraire. Notons que tous les arbres que nous considérons ensuite sont des arbres binaires finis. Nous introduisons ensuite deux modèles d'automates sur les mots dans la section 2.2 et deux modèles d'automates sur les arbres dans la section 2.3 en utilisant de nombreux exemples. Les automates unidirectionnels et bidirectionnels sont des automates séquentiels sur les mots. Les automates d'arbres que nous appelons les automates d'arbres bottom-up et que nous notons BU effectuent des calculs en parallèle partant des feuilles et remontant à la racine. La classe des automates d'arbres cheminants qui sont appelés tree-walking automata en anglais et que nous notons TWA est un modèle d'automates séquentiels qui étend sur les arbres le modèle des automates bidirectionnels sur les mots. Pour tous les modèles d'automates que nous définissons, nous distinguons les variantes déterministes où les transitions sont des fonctions et les variantes non-déterministes. Nous présentons enfin dans la section 2.4 les automates d'arbres à jetons qui sont appelés pebble tree automata en anglais et que nous notons PTA. Les automates d'arbres à jetons font l'objet de ce travail et sont définis de manière similaire sur les mots. Nous considérons deux modèles d'automates d'arbres à jetons : le modèle fort pour lequel le jeton est considéré comme un pointeur et le modèle faible où le jeton est vu comme un objet physique. La dernière section établit des rapports entre les langages définis par les différents modèles d'automates que nous avons présentés et la logique monadique du second ordre. Nous montrons ainsi que les langages d'arbres reconnus par un automate d'arbres à jetons sont réguliers et que tous les automates séquentiels sur les mots que nous avons présentés ont le même pouvoir d'expression. Nous verrons dans le chapitre 6 que les différents modèles d'automates d'arbres (BU, TWA et PTA) que nous définissons n'ont pas tous le même pouvoir d'expression.

2.1 Les arbres, les mots et la logique MSO

2.1.1 Les mots et les arbres : des structures de données

Un *alphabet* est un ensemble fini de symboles. Les éléments d'un alphabet sont des *lettres*.

Un *mot* fini sur un alphabet Σ est une suite finie de lettres de Σ. Un mot est représenté par la juxtaposition des lettres qui le composent. Par exemple, *logique* représente le mot a sept lettres associé à la suite l, o, g, i, q, u, e. Le mot vide, c'est-à-dire le mot associé à la suite vide, est noté ϵ. Par la suite tous les mots que nous considérons seront des mots finis.

Etant donnés deux mots u et v, la concaténation de u et de v sera notée $u \cdot v$ ou tout simplement uv.

Un mot u est un *préfixe* d'un mot v s'il existe un mot w tel que $v = uw$. Un

mot u est un *suffixe* d'un mot v s'il existe un mot w tel que $v = wu$. Un préfixe *propre* de u (respectivement un suffixe *propre* de u) est un préfixe (respectivement un suffixe de u) distinct de u. Un ensemble L de mots est fermé par préfixe si, pour tout mot u dans L, tout préfixe de u est dans L.

Soit Σ un alphabet, on note Σ^* l'ensemble des mots sur Σ.

Notation 2.1 *On note $\mathbb{N}_{>0}$ l'ensemble des entiers strictement positifs et on désigne par ${\mathbb{N}_{>0}}^*$ l'ensemble des suites finies d'entiers strictement positifs. Soient i et j deux entiers, l'ensemble des entiers compris entre i et j est noté $[i, j]$ et si n est un entier $[n]$ désigne l'ensemble $[1, n]$ des entiers strictement positifs et inférieurs ou égaux à n.*

Nous avons défini un mot comme une suite finie de lettres. Un mot de *longueur* n sur un alphabet Σ peut ainsi être vu comme une application de $[n]$ dans Σ associant à tout i, $1 \leq i \leq n$, la i-ème lettre du mot.

Un *domaine d'arbre* $\mathcal{N}$ sur $\mathbb{N}_{>0}$ est un sous-ensemble de ${\mathbb{N}_{>0}}^*$ tel que si $vi \in \mathcal{N}$ avec $v \in {\mathbb{N}_{>0}}^*$ et $i \geq 1$ alors $v \in \mathcal{N}$ et si de plus $i > 1$ alors $v(i-1) \in \mathcal{N}$. Le mot vide notée ϵ représente la racine. On appelle les éléments de $\mathcal{N}$ des *nœuds*. La profondeur d'un nœud est égale à sa longueur. La profondeur de la racine d'un arbre est donc nulle. Un nœud w est un *fils* d'un nœud v et v est le *père* de w s'il existe $i > 0$ tel que $w = vi$. Des nœuds v et w sont des *frères* s'ils ont le même père. Un nœud w est un *descendant* d'un nœud v et v est un *ancêtre* de w s'il existe $u \in {\mathbb{N}_{>0}}^*$ tel que $w = vu$.

Un *arbre* fini sur un alphabet Σ est un couple $t = (\mathcal{N}, \lambda)$ où $\mathcal{N}$ est un domaine d'arbre fini sur $\mathbb{N}_{>0}$ et λ est une application de $\mathcal{N}$ vers Σ. Tous les arbres que nous considérons seront finis. L'*arité* ou le *degré* d'un nœud est le nombre de fils de ce nœud. Un *nœud interne* est un nœud d'arité strictement positive. Une *feuille* est un nœud d'arité nulle. L'arité ou le degré d'un arbre est le nombre maximum de fils de ses nœuds. Un arbre *unaire* est un arbre dont tous les nœuds internes sont d'arité 1 et un tel arbre correspond à un mot. La *taille* d'un arbre est le nombre de ses nœuds et la *hauteur* d'un arbre est la profondeur maximale de ses nœuds. Par la suite, nous travaillons sur des arbres de degré 2. Cette restriction est importante car les arbres représentant des documents XML n'ont pas un degré fixé. Nous reviendrons sur cette restriction en 8.2 On peut définir de différentes manières les arbres d'arité bornée. Pour $n > 0$ un *arbre de degré* n est un arbre dont le domaine est un sous-ensemble de $\{1, \cdots, n\}^*$. Dans de nombreux ouvrages, on utilise la notion d'alphabet gradué pour introduire les arbres d'arité bornée. Un *alphabet gradué* Σ_d est un couple (Σ, d) où Σ est un alphabet fini et d est une fonction de Σ vers $\mathbb{N}$. Un arbre sur l'alphabet gradué Σ est un arbre sur l'alphabet Σ tel que chaque nœud v a exactement $d(v)$ fils. Nous pouvons remarquer qu'un arbre sur l'alphabet gradué (Σ, d) est un arbre de degré d_m où d_m est la valeur maximale de la fonction d sur Σ.

Exemple 2.1 Nous représentons dans la figure 2.1 un arbre de degré 2 sur l'alphabet $\{a, b, c\}$. Un nœud interne peut avoir un ou deux fils.

Pour simplifier encore d'avantage les difficultés techniques des preuves de nos résultats nous nous restreignons aux arbres binaires pour lesquels chaque nœud a soit zéro successeur soit deux successeurs qui sont son fils gauche et son fils droit

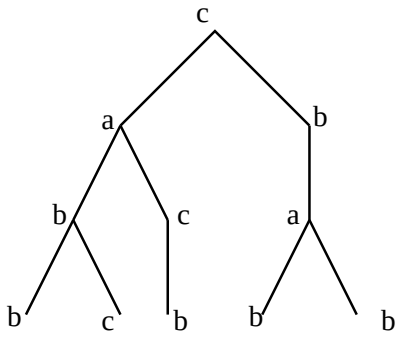

FIG. 2.1 – Représentation d'un arbre de degré 2

mais tous les résultats que nous prouverons dans le cadre des arbres binaires restent en fait vrais dans le cadre des arbres de degré fixé. Les nœuds d'un arbre vont être représentés par des mots sur l'alphabet $\{1, 2\}$ où 1 signifie branche gauche et 2 signifie branche droite. Un domaine d'arbre binaire fini est un ensemble fini $\mathcal{N}$ de $\{1, 2\}^*$ fermé par préfixe tels que pour chaque $w \in \mathcal{N}$, $w1 \in \mathcal{N}$ si et seulement si $w2 \in \mathcal{N}$. Un nœud interne a donc toujours deux fils. La *racine* d'un arbre, notée ϵ est le nœud qui correspond au mot vide sur l'alphabet $\{1, 2\}$. Tous les arbres que nous considérons dans les chapitres 3 à 7 sont des arbres binaires.

Exemple 2.2 Nous représentons dans la figure 2.2 un arbre binaire sur l'alphabet $\{a, b, c\}$.

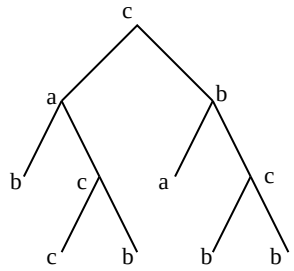

FIG. 2.2 – Représentation d'un arbre binaire

On définit un *graphe* orienté en indexant les *sommets*, qui sont aux graphes ce que les nœuds sont aux arbres, par les éléments d'un ensemble muni d'une relation binaire. Un graphe orienté est donc un couple $G = (V, E)$ où V est un ensemble de sommets et $E \subseteq V \times V$ est une relation binaire sur V appelée ensemble des *arcs*. Par la suite, tous les graphes que nous verrons sont orientés.

La définition précédente ne décrit que la structure et pas un étiquetage éventuel des sommets ou des arcs. Un graphe dont les sommets sont étiquetés par un alphabet Σ est alors un triplet (V, λ, E) tel que (V, E) est un graphe et λ est une application de V dans Σ. Un graphe qui possède un sommet lié à tous les autres sommets par un unique chemin est un *arbre non-ordonné* dont le sommet lié à tous les autres est la racine. Un arbre $(\mathcal{N}, \lambda)$ peut aussi être vu comme le graphe $(\mathcal{N}, \lambda, E)$ tel que les successeurs d'un sommet sont ordonnés et tel que l'ensemble des arcs E correspond à la relation "fils" dans l'arbre : $E = \{(v, vi) \mid i > 0 \text{ et } vi \in \mathcal{N}\}$. Pour représenter

un graphe, on représente un sommet u par un cercle dans lequel on écrit u et un arc (v, w) par une flèche partant du sommet v et allant au sommet w. La *taille* d'un graphe est la somme du nombre de ses nœuds et du nombre de ses arêtes. Attention, la taille d'un arbre n'est donc pas égale à la taille du graphe qui lui correspond.

Exemple 2.3 Nous représentons dans la figure 2.3 le graphe (V, E) avec $V = \{1, 2, 3, 4\}$ et $E = \{(1, 2), (2, 1), (2, 4)(3, 2), (4, 1)\})$.

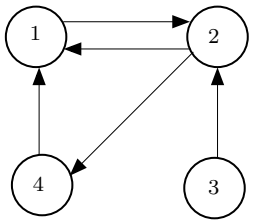

FIG. 2.3 – Représentation du graphe (V, E)

2.1.2 Des modèles pour les mots et les arbres

Une *signature* $\mathcal{S}$ est un ensemble fini de symboles relationnels, chaque symbole $R \in \mathcal{S}$ possédant une arité entière notée $|R|$. Une *structure relationnelle* $\mathbb{S}$ sur une signature $\mathcal{S}$ est une paire $(\mathcal{U}, \mathcal{I})$ où $\mathcal{U}$ est un ensemble appelé *univers* et $\mathcal{I}$ est une application associant a chaque symbole R de la signature son *interprétation*, c'est à dire une partie de $\mathcal{U}^{|R|}$.

Définition 2.1 *Soit Σ un alphabet et soit $w = w_1 \cdots w_n$ un mot sur Σ. Le mot w est représenté par la structure relationnelle $([n], \mathcal{I})$ sur la signature $\{S, <, (P_a)_{a\in\Sigma}\}$ où*

- *$\mathcal{I}(S)$ est la relation successeur sur $[n]$ avec $(i, i+1) \in \mathcal{I}(S)$ pour $1 \leq i < n$,*
- *$\mathcal{I}(<)$ est l'ordre naturel sur $[n]$ et*
- *pour chaque $a \in \Sigma$, $\mathcal{I}(P_a)$ est une relation unaire contenant les positions des lettres a dans w.*

Par la suite on utilisera la même notation pour un mot w et sa structure relationnelle.

Définition 2.2 *Un arbre binaire $t = (\mathcal{N}, \lambda)$ sur l'alphabet fini Σ est représenté par la structure relationnelle $(\mathcal{N}, \mathcal{I})$ sur la signature $(S_1, S_2, <, (P_a)_{a\in\Sigma})$ où*

- *$\mathcal{I}(S_1)$ est la relation "fils gauche" sur $\mathcal{N}$ avec $(u, u1) \in \mathcal{I}(S)$ pour $u1 \in \mathcal{N}$,*
- *$\mathcal{I}(S_2)$ est la relation "fils droit" sur $\mathcal{N}(w)$ avec $(u, u2) \in \mathcal{I}(S_2)$ pour $u2 \in \mathcal{N}$,*
- *$\mathcal{I}(<)$ est la relation "ancêtre" sur les nœuds avec $(u, v) \in \mathcal{I}(<)$ pour $v \in \mathcal{N}$ et u préfixe de v et*
- *pour chaque $a \in \Sigma$, $\mathcal{I}(P_a)$ est une relation unaire contenant les nœuds étiquetés par a dans t.*

Par la suite on utilisera la même notation pour un arbre t et sa structure relationnelle.

Remarque 2.1 On peut étendre ce modèle sans difficulté pour les arbres de degré borné. Le modèle pour les arbres d'arité k est défini en remplaçant les prédicats binaires S_1 et S_2 par k prédicats $S_1, \cdots, S_k$ tels que pour chaque $i \in [k]$ l'interprétation de S_i est la relation "i-ème fils" qui contient tous les couples (u, ui) tels que ui est un nœud de l'arbre.

Remarque 2.2 Afin de définir un modèle pour les arbres d'arité non-bornée, on ne peut rajouter un nombre infini de prédicats puisqu'une signature est un ensemble fini de symboles relationnels. On introduit alors un nouveau prédicat qui sera interprété par la relation "frère suivant". Ainsi, un arbre d'arité non-borné $t = (\mathcal{N}, \lambda)$ sur l'alphabet fini Σ est représenté par la structure relationnelle $(\mathcal{N}, \mathcal{I})$ sur la signature $(S_\downarrow, S_\rightarrow, <, (P_a)_{a\in\Sigma})$ où

- $\mathcal{I}(S_\downarrow)$ est la relation "premier fils" sur $\mathcal{N}$ avec $(u, u1) \in \mathcal{I}(S)$ pour $u1 \in \mathcal{N}$,
- $\mathcal{I}(S_\rightarrow)$ est la relation "frère suivant "sur $\mathcal{N}$ avec $(ui, u(i+1)) \in \mathcal{I}(S_\rightarrow)$ pour $ui, u(i+1) \in \mathcal{N}$,
- $\mathcal{I}(<)$ est la relation ancêtre sur les nœuds avec $(u, v) \in \mathcal{I}(w)$ pour $v \in \mathcal{N}$ et u préfixe de v et
- pour chaque $a \in \Sigma$, $\mathcal{I}(P_a)$ est une relation unaire contenant les nœuds étiqueté par a dans t.

Tous les modèles ci-dessus pour les arbres peuvent être redéfinis sans le prédicat $<$ interprété par la relation ancêtre mais les modèles que nous considérons par la suite sont ceux avec le prédicat $<$.

2.1.3 Logique du premier ordre

Les propriétés des mots, des arbres et des graphes peuvent être formalisées avec des formules logiques. Nous commençons par la logique du premier ordre également appelée logique FO qui est à la base de toutes les autres logiques considérées ici.

Définition 2.3 *On se fixe un ensemble dénombrable de variables du premier ordre notées $x, y, \cdots$ Etant donnée une signature $\mathcal{S}$, les formules du premier ordre sur $\mathcal{S}$ sont définies inductivement au moyen des constructions ci-dessous :*

- *la constante* vrai,
- *la conjonction : $\phi \wedge \psi$ où ϕ et ψ sont des formules du premier ordre,*
- *la négation : $\neg\phi$ où ϕ est une formule du premier ordre,*
- *la quantification existentielle du premier ordre : $\exists x.\phi$ où x est une variable du premier ordre et ϕ une formule du premier ordre,*
- *l'égalité de variables du premier ordre : $x = y$ avec x et y variables du premier ordre,*
- *la relation entre variables du premier ordre : $R(x_1, \cdots, x_{|R|})$ où R est un symbole relationnel de $\mathcal{S}$ d'arité $|R|$ et x_1,$\cdots$,$x_{|R|}$ sont des variables du premier ordre.*

Les autres connecteurs classiques ne sont que des combinaisons des précédents :

- la constante faux équivaut à $\neg$vrai
- la disjonction : $\phi \vee \psi$ équivaut à $\neg((\neg\phi) \wedge (\neg\psi))$,
- l'implication : $\phi \Rightarrow \psi$ équivaut à $(\neg\phi) \vee \psi$

– l'équivalence : $\phi \Leftrightarrow \psi$ équivaut à $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$
– la quantification universelle du premier ordre : $\forall x\ \phi$ équivaut à $\neg(\exists x \neg\phi)$.

On omet le plus souvent de préciser la signature $\mathcal{S}$ quand il n'y a pas d'ambiguïté. Dans une formule logique, une variable qui n'est pas liée à un quantificateur existentiel ou universel est dite *libre*. De manière plus formelle, étant donnée une formule ϕ, on définit l'ensemble $\mathcal{VL}(\phi)$ des variables libres par induction sur la structure de ϕ comme suit :

– $\mathcal{VL}(\text{vrai}) = \emptyset$
– $\mathcal{VL}(\phi \wedge \psi) = \mathcal{VL}(\phi) \cup \mathcal{VL}(\psi)$
– $\mathcal{VL}(\neg\phi) = \mathcal{VL}(\phi)$
– $\mathcal{VL}(\exists x\phi) = \mathcal{VL}(\phi) - \{x\}$
– $\mathcal{VL}(x = y) = \{x, y\}$
– $\mathcal{VL}(R(x_1, \cdots, x_{|R|}) = \{x_1, \cdots, x_{|R|}\}$

Une formule est close si elle n'a aucune variable libre. Par abus de langage et pour un ensemble V de noms de variables, on dit qu'une formule ϕ a pour variables libres V si V contient $\mathcal{VL}(\phi)$.

Etant données une structure relationnelle $\mathbb{S} = (\mathcal{U}, \mathcal{I})$, une formule du premier ordre ϕ de variables libres V et une application γ de V dans $\mathcal{U}$ appelé valuation, on définit par récurence sur ϕ la relation de satisfaction de ϕ par $\mathbb{S}$ dans le contexte γ, notée $\gamma, \mathbb{S} \models \phi$ de la manière suivante :

– $\gamma, \mathbb{S} \models \text{vrai}$
– $\gamma, \mathbb{S} \models \phi \wedge \psi$ si $\gamma, \mathbb{S} \models \phi$ et $\gamma, \mathbb{S} \models \psi$
– $\gamma, \mathbb{S} \models \neg\phi$ si $\gamma, \mathbb{S} \not\models \phi$ (c'est-à-dire si l'on n'a pas $\gamma, \mathbb{S} \models \phi$)
– $\gamma, \mathbb{S} \models R(x_1, \cdots, x_{|R|})$ si $(\gamma(x_1), \cdots, \gamma(x_{|R|})) \in \mathcal{I}(R)$
– $\gamma, \mathbb{S} \models x = y$ si $\gamma(x) = \gamma(y)$
– $\gamma, \mathbb{S} \models \exists x\phi$ s'il existe $u \in \mathcal{U}$ tel que $\gamma_{x \to u}, \mathbb{S} \models \phi$ où $\gamma_{x \to u}$ représente la fonction partout égale a γ sauf pour x où elle vaut u.

Pour une formule close ϕ, si $f_\emptyset, \mathbb{S} \models \phi$ où $f_\emptyset$ est l'application de domaine vide, on dit que $\mathbb{S}$ satisfait ϕ ou que $\mathbb{S}$ est un modèle de ϕ ce que l'on note simplement $\mathbb{S} \models \phi$.

Il est parfois utile de rendre explicite les variables libres d'une formule. Ainsi, on notera $\phi(x_1, \cdots, x_n)$ une formule logique de variables libres $x_1, \cdots, x_n$. Soient $e_1, \cdots, e_n$ des éléments de $\mathcal{U}$ et γ la valuation qui fait correspondre à chaque variable x_i l'élément e_i.

$\mathbb{S}, e_1, \cdots, e_n \models \phi(x_1, \cdots, x_n)$ signifie alors que $\gamma, \mathbb{S} \models \phi$. S'il n'y a pas d'ambiguité sur $\mathbb{S}$, on dit que ϕ est satisfaite par $e_1, \cdots, e_n$.

Afin d'illustrer les notions définies ci-dessus nous donnons maintenant quelques exemples de formule du premier ordre sur les mots et sur les arbres.

Exemple 2.4 Considérons les formules FO pre, der et mil à une variable libre sur les mots étiquetés par un alphabet Σ définies respectivement par :

– $\text{pre}(x) = \neg(\exists y\ S(y, x))$
– $\text{der}(x) = \neg(\exists y\ S(x, y))$
– $\text{mil}(x) = \neg(\text{pre}(x) \vee \text{der}(x))$

La formule pre est satisfaite par la première position d'un mot, la formule der par la dernière position et la formule mil par toutes les autres positions du mot.

Exemple 2.5 Soient $\mathrm{ra}, \mathrm{in}, \mathrm{fe}, \mathrm{fg}$ et fd les cinq formules suivantes à une variable libre sur les arbres binaires étiquetés par un alphabet Σ :

- $\mathrm{ra}(x) = \neg\exists y\ (S_1(y,x) \vee S_2(y,x))$
- $\mathrm{in}(x) = \exists y\ S_1(x,y)$ et $\mathrm{fe}(x) = \neg\mathrm{in}(x)$
- $\mathrm{fg}(x) = \exists y\ S_1(y,x)$ et $\mathrm{fd}(x) = \exists y\ S_2(y,x)$

La formule ra est satisfaite par la racine de l'arbre, la formule in par les nœuds internes, la formule fe par les feuilles, la formule fg par les nœuds qui sont des fils gauches et la formule fd par ceux qui sont des fils droits.

Exemple 2.6 Voici enfin une formule du premier ordre sans variables libres satisfaites par les arbres dont toutes les feuilles sont étiquetées par la lettre a de Σ.

$$\forall x\ ((\neg(\exists y\ x < y)) \Rightarrow P_a(x))$$

Remarque 2.3 Avec la logique FO sur les mots, on peut exprimer la relation successeur à l'aide de la relation ancêtre. En effet la formule $S(x,y)$ est équivalente à la formule $(x < y) \wedge \neg(\exists z\ (x < z) \wedge (z < y))$.

2.1.4 Logique MSO

La *logique au second ordre monadique* étend la logique du premier ordre par des variables interprétées comme des parties de l'univers. Il est possible de quantifier existentiellement et universellement sur ces ensembles. Cela correspond à une augmentation importante du pouvoir d'expressivité par rapport à la logique du premier ordre.

Définition 2.4 *On se fixe un ensemble dénombrable V_m de variables du second ordre monadique. Elles sont notées en lettres majuscules : $X, Y, \ldots$. Les formules monadiques du second ordre ou les formules* MSO *sont les formules du premier ordre étendues par la quantification existentielle du second ordre monadique $\exists X.\phi$ et la proposition atomique d'appartenance $x \in X$.*

Comme précédemment on s'autorise à utiliser la quantification universelle du second ordre monadique $\forall X\ \phi$ comme un raccourci équivalent à $\neg(\exists X \neg\phi)$. Les définitions données pour la logique du premier ordre sont réutilisables sans grandes modifications. Ainsi les variables libres peuvent être monadiques tout comme du premier ordre et une valuation γ, en plus d'associer à chaque variable libre du premier ordre un élément de l'univers associe à chaque variable monadique un sous-ensemble de l'univers.

La définition de la satisfaction d'une formule est alors complétée par les règles suivantes :

- $\gamma, \mathbb{S} \models \exists X\phi$ s'il existe $U_X \subseteq U$ tel que $\gamma, X \to U_X, \mathbb{S} \models \phi$
- $\gamma, \mathbb{S} \models x \in X$ si $\gamma(x) \in \gamma(X)$.

Une formule monadique du second ordre $\phi(X_1, \cdots, X_n)$ ayant $X_1, \cdots, X_n$ comme variables libres est interprétée dans une nouvelle structure relationnelle obtenue en ajoutant n prédicats unaires $P_1, \cdots, P_n$.

Remarque 2.4 Avec la logique MSO sur les mots, on peut exprimer la relation ancêtre à l'aide de la relation successeur. En effet la formule $x < y$ est équivalente à la formule $\forall X(\forall x_1, x_2(S(x_1, x_2) \wedge x_1 \in X) \Rightarrow x_2 \in X \wedge x \in X) \Rightarrow y \in X$.

Remarque 2.5 La logique MSO permet d'exprimer l'inclusion de deux ensembles. Si X et Y sont des variables du second ordre $X \subseteq Y$ désigne la formule $\forall x(x \in X \Rightarrow x \in Y)$.

Remarque 2.6 Soit $\phi(X_1, \cdots, X_n)$ une formule MSO sur les arbres étiquetés par un alphabet A ayant $X_1, \cdots, X_n$ comme variables libres. On peut interpréter cette formule du second ordre comme une formule du premier ordre dans la structure relationnelle des arbres étiquetés par l'alphabet $\Sigma \times \{0,1\}^n$ en codant l'appartenance d'un nœud à un ensemble correspondant à une variable X_i par la i-ème composante binaire de l'étiquette d'un nœud. De la même façon, une formule MSO avec n variables libres sur les mots étiquetés par un alphabet Σ peut être interprétée sur les mots étiquetés par l'alphabet $\Sigma \times \{0,1\}^n$.

On utilisera en 2.5.1 la logique MSO_{ens} sur les mots et les arbres qui a le même pouvoir d'expression que la logique monadique du second ordre et une syntaxe plus simple. L'idée est de simuler les quantifications sur des éléments par des quantifications sur des ensembles. Les formules atomiques de la logique MSO_{ens} sur les mots sont $\text{Sing}(X)$, $X \subseteq Y$, $\text{Suc}(X, Y)$, $X \subseteq P_a$ (pour $a \in A$) qui signifient respectivement que X est un singleton, que X est un sous-ensemble de Y, que X et Y sont des singletons $\{x\}$ et $\{y\}$ avec $S(x, y)$ et que X est un sous-ensemble de P_a. Pour définir la logique MSO_{ens} sur les arbres binaires on utilisera les formules atomiques $\text{Suc}_0(X, Y)$ et $\text{Suc}_1(X, Y)$à la place de $\text{Suc}(X, Y)$. La transformation d'une formule MSO dont les variables libres sont toutes du second ordre en une formule MSO_{ens} et la transformation inverse se font facilement par induction sur la formule.

Exemple 2.7 La formule MSO_{ens} $\text{Sing}(X)$ correspond à la formule MSO suivante :

$$\forall x, y \ (x \in X \wedge y \in X) \Rightarrow (x = y)$$

La formule MSO $\exists z(z \in X)$ correspond à la formule MSO_{ens} suivante :

$$\exists Z \ \text{Sing}(Z) \wedge Z \subseteq X$$

Donnons maintenant un exemple d'une formule MSO sur les arbres.

Exemple 2.8 On rappelle que pre et der sont les formules du premier ordre à une variable libre sur les mots définies dans l'exemple 2.4 et satisfaites respectivement par la première et la dernière position. Considérons maintenant la formule MSO sans variable libre définie ci-dessous sur les mots étiquetés par un alphabet Σ :

$$\exists X, Y \{\neg(\exists z \ z \in X \wedge z \in Y) \wedge \exists x(\text{pre}(x) \wedge x \in X) \wedge \exists y(\text{der}(y) \wedge y \in Y)$$
$$\wedge \forall z_1 z_2 S(z_1, z_2) \Rightarrow [(z_1 \in X \wedge z_2 \in Y) \vee (z_1 \in Y \wedge z_2 \in X)]\}$$

Cette formule est satisfaite par tous les mots de longueur paire. En effet, dans cette formule, les variables X et Y correspondent respectivement à l'ensemble des positions impaires et à l'ensemble des positions paires du mot.

2.2 Automates sur les mots

Dans cette section, nous définissons les langages rationnels, les automates unidirectionnels et les automates bidirectionnels sur les mots.

2.2.1 Langages rationnels de mots

Soit Σ un alphabet. Un *langage* désigne un ensemble de mots et sera en général noté par une lettre majuscule romaine, par exemple L. Tous les langages de mots que nous considérons sont des langages de mots finis. Un langage sur Σ est donc un sous-ensemble de Σ^*.

Plusieurs opérations peuvent être effectuées sur les langages. Les langages étant des ensembles, nous rappelons ci-dessous les opérations booléennes.

Etant donnés deux langages L_1 et L_2,

- le langage $L_1 \cup L_2 = \{u \in \Sigma^* | u \in L_1 \text{ ou } u \in L_2\}$ est l'*union* de L_1 et L_2,
- le langage $L_1 \cap L_2 = \{u \in \Sigma^* | u \in L_1 \text{ et } u \in L_2\}$ est l'*intersection* de L_1 et L_2,
- le langage ${L_1}^c = \{u \in \Sigma^* | u \notin L_1\}$ est le *complémentaire* de L_1.

Outre les opérations booléennes, il existe d'autres opérations naturelles sur les langages fondées sur la concaténation pour les mots.

Etant donnés deux langages L_1 et L_2, le *produit* de L_1 et de L_2 est le langage $\{u_1u_2 \in \Sigma^* | u_1 \in L_1 \text{ et } u_2 \in L_2\}$ que l'on note L_1L_2.

Etant donné L un langage, on définit par récurrence la suite des langages $(L^i)_{i\in\mathbb{N}}$ par $L^0 = \{\epsilon\}$ et $L^{i+1} = L^iL$. Ces langages sont appelés les *puissances* de L et l'union des puissances de L est appelée l'*étoile* de L et est notée L^*.

Remarque 2.7 La notation Σ^* définie dans la première partie prend alors tout son sens en considérant Σ comme l'ensemble des mots sur Σ réduits à une lettre.

Les langages étant des ensembles potentiellement infinis et arbitrairement complexes, ils ne peuvent pas tous être décrits par un modèle raisonnablement simple. Nous construisons maintenant des langages et leur représentation inductivement à partir de langages de base et d'opérations que l'on appellera opérations rationnelles. Nous obtenons ainsi une classe intéressante de langages appelés *langages rationnels*.

Définition 2.5 *Étant donné un alphabet Σ, la classe des langages rationnels sur Σ notée Rat Σ^* est la plus petite classe de langages satisfaisant les propriétés suivantes :*

- *$\emptyset \in$ Rat Σ^**
- *pour toute lettre $a \in \Sigma$, $\{a\} \in$ Rat Σ^**
- *Rat Σ^* est fermée pour l'union, le produit et l'étoile.*

Nous avons ainsi défini de manière récurente la classe des langages rationnels. Pour représenter de tels langages, on utilise les *expressions rationnelles*.

Définition 2.6 *Les expressions rationnelles et leurs langages associés sont définis par induction :*

- *$\emptyset$ est une expression rationnelle associée au langage $\emptyset$.*

- *pour toute lettre $a \in \Sigma$, a est une expression rationnelle associée au langage $\{a\}$*
- *si E_1 et E_2 sont des expressions rationnelles respectivement associées aux langages L_1 et L_2, alors $(E_1 \cup E_2)$, $(E_1.E_2)$ et E_1^* sont des expressions rationnelles respectivement associées aux langages $L_1 \cup L_2$, $L_1.L_2$ et L_1^*.*

Lorsqu'il n'y a pas d'ambiguïté, on supprime les parenthèses inutiles.

Dans la suite, nous confondrons une expression rationnelle avec le langage qu'elle représente.

Exemple 2.9 Pour tout alphabet Σ, l'ensemble des mots réduits à une lettre sur cet alphabet est un langage rationnel que l'on note également Σ. En effet, chacun des mots de ce langage fini est un langage rationnel et la classe des langages rationnels est fermée par union finie.

Exemple 2.10 Soit Σ un alphabet et a une lettre de Σ. Le langage $\Sigma^* a$ est l'ensemble des mots qui se terminent par la lettre a et le langage $(\Sigma\Sigma)^*$ est l'ensemble des mots de longueur paire.

2.2.2 Automates unidirectionnels

La notion d'expression rationnelle permet de représenter de façon finie un langage rationnel de mots. Une autre approche pour représenter un langage rationnel est l'utilisation des automates finis.

Les automates finis permettent de définir les *langages réguliers*. De manière informelle, un *automate fini unidirectionnel* est une machine à états finis qui lit un mot donné en entrée lettre par lettre de gauche à droite et qui décide, après avoir lu la dernière lettre, si ce mot appartient au langage représenté par l'automate. Un tel automate correspond à un graphe orienté dont les arêtes sont étiquetées par les lettres d'un alphabet et dans lequel deux sous-ensembles de sommets ont été distingués.

Définition 2.7 *Un automate fini unidirectionnel sur les mots, ou un* 1NFA*, est un quintuplet $(Q, \Sigma, I, F, \delta)$ où Q est un ensemble fini d'états, Σ est un alphabet, $I \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états initiaux, $F \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états finaux et $\delta \subseteq Q \times \Sigma \times Q$ est un sous-ensemble de $Q \times \Sigma \times Q$ appelé ensemble des transitions.*

L'ensemble des transitions δ peut aussi être défini comme une application de $Q \times \Sigma$ dans l'ensemble des parties de Q noté 2^Q . La représentation graphique d'un automate unidirectionnel $A = (Q, \Sigma, I, F, \delta)$ est celle du graphe dont les arcs sont étiquetés par Σ ayant Q pour ensemble de sommets et δ pour ensemble d'arcs. On ajoute une flèche entrante à chaque sommet associé à un état initial et une flèche sortante à chaque sommet associé à un état final. La *taille* d'un automate est la taille du graphe correspondant c'est à dire la somme du nombre d'états et du nombre de transitions.

Etant donné un 1NFA $A = (Q, \Sigma, I, F, \delta)$, on dit qu'il y a une transition d'un état p à un état q d'étiquette a ce que l'on note $p \xrightarrow{a} q$ si $(p, a, q) \in \delta$. On dit aussi que p est l'origine et q l'extrémité de la transition (p, a, q).

Un *calcul* c de A est une suite de transitions $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{n-1} \xrightarrow{a_n} q_n$ telles que l'origine de chacune coïncide avec l'extrémité de la précédente. Le mot $a_1 a_2 \cdots a_n$ est appelé l'étiquette du calcul c. On dit également que c est une *exécution* de q_0 à q_n d'étiquette $a_1 \cdots a_n$. Le calcul c est *acceptant* si q_0 est un état initial et q_n un état final.

Un mot fini u est *accepté* par un 1NFA A s'il est l'étiquette d'au moins un calcul acceptant de A. On note $L(A)$ l'ensemble des mots acceptés par l'automate A et $L(A)$ est appelé *langage reconnu* par A. Deux automates qui reconnaissent le même langages sont *équivalents*.

Exemple 2.11 On considère encore l'alphabet $\Sigma = \{a, b\}$.

Les 1NFA $A_1 = (\{q_1, q_2\}, \Sigma, \{q_1\}, \{q_2\}, \delta_1)$ et $A_2 = (\{q_1, q_2\}, \Sigma, \{q_1\}, \{q_1\}, \delta_2)$ sont représentés dans la figure 2.4 avec

- $\delta_1 = \{(q_1, a, q_1), (q_1, b, q_1), (q_1, a, q_2)\}$,
- $\delta_2 = \{(q_1, a, q_2), (q_2, a, q_1), (q_1, b, q_2), (q_2, b, q_1)\}$.

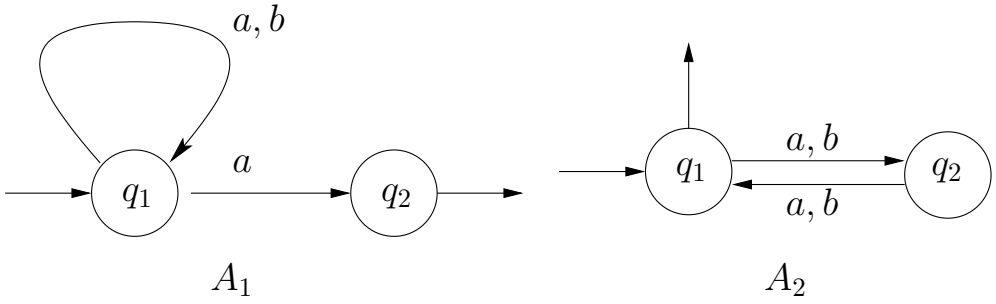

FIG. 2.4 – Représentations graphiques des 1NFA A_1 et A_2

Les automates A_1 et A_2 reconnaissent respectivement les langages rationnels $\Sigma^* a$ et $(\Sigma\Sigma)^*$.

Les automates apportent donc un moyen de représenter de façon finie des langages de mots.

Définition 2.8 *Soit L un langage de mots. On dit que L est un langage reconnaissable s'il existe un* 1NFA *qui le reconnaît. Étant donné un alphabet Σ, on note $Rec\ \Sigma$ l'ensemble des langages reconnaissables sur l'alphabet Σ.*

Le lien entre les langages rationnels et les langages reconnaissables nous est donné par le théorème de Kleene.

Théorème 2.1 (Kleene) *Pour tout alphabet fini Σ, les langages rationnels de Σ et les langages reconnaissables de Σ coïncident :*

$$Rat\ \Sigma^* = Rec\ \Sigma^*$$

Considérons l'automate A_1 défini dans l'exemple 2.11. Le mot ba est accepté puisque le calcul $q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_2$ est un calcul acceptant d'étiquette ba . Cependant le calcul $q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_1$ est un calcul d'étiquette ba qui commence dans un état initial mais qui n'est pas acceptant.

Etant donné un 1NFA, il peut y avoir plusieurs calculs de même étiquette dont certains sont acceptants et d'autres non. Cela provient du fait que l'ensemble des états initiaux n'est pas un singleton et que l'ensemble des transtions ne correspond pas à une fonction de $Q \times \Sigma$ à valeurs dans Q. D'un point de vue algorithmique, il serait plus simple de ne pas avoir cette ambiguïté que l'on appelle non-déterminisme. Pour cela, on définit les *automates déterministes*.

Définition 2.9 *Soit $A = (Q, \Sigma, I, F, \delta)$ un* 1NFA. *L'automate A est un automate déterministe, ou un* 1DFA, *s'il vérifie les deux conditions suivantes :*

1. *l'ensemble des états initiaux I est un singleton.*
2. *pour tout état q et pour toute lettre a, s'il existe des états r et s tels que $(q, a, r) \in \delta$ et $(q, a, s) \in \delta$, alors $r = s$.*

Si q_0 est l'état initial, on pourra noter $A = (Q, \Sigma, q_0, F, \delta)$ à la place de $A = (Q, \Sigma, \{q_0\}, F, \delta)$.

Ainsi, dans un automate déterministe, deux transitions ayant la même origine et la même étiquette ont la même extrémité. L'ensemble des transitions s'identifie alors à une fonction partielle de $Q \times \Sigma$ à valeurs dans Q. Comme le montre le théorème ci-dessous, les automates déterministes reconnaissent les mêmes langages que les automates non-déterministes. Cependant le prix à payer est qu'ils sont moins concis.

Théorème 2.2 *Soit L un langage reconnu par un* 1NFA *dont l'ensemble d'états est Q. Il existe un* 1DFA *qui reconnaît L dont le nombre d'états est $2^{|Q|}$. De plus, pour tout entier $n > 1$, il existe un langage L reconnu par un* 1NFA *à n états tel que tout automate déterministe qui reconnaît L possède au moins 2^n états.*

Complémenter un automate A donné signifie construire un automate A^c qui reconnaît le complément du langage reconnu par A. Afin de complémenter un 1DFA on le transforme en un 1DFA équivalent tel que chaque mot soit l'étiquette d'un calcul.

Définition 2.10 *Soit A un* 1NFA. *A est complet si pour tout état q et pour toute lettre a il existe un état q' tel que (q, a, q') est une transition.*

Compléter un 1NFA *signifie construire un* 1NFA *complet équivalent.*

En ajoutant un nouvel état rejetant, il est facile de compléter un 1NFA donné.

Les 1DFA complets sont simples à complémenter. En effet, si on intervertit l'ensemble des états finaux et l'ensemble des états non-finaux d'un tel automate, les calculs qui étaient acceptants dans A deviennent rejetants dans A^c et réciproquement. On peut alors complémenter un 1NFA après l'avoir déterminisé et complété.

Théorème 2.3 *Soit L un langage reconnu par un* 1NFA *A dont l'ensemble d'états est Q. Il existe un automate déterministe A^c dont l'ensemble d'états est 2^Q qui reconnaît le complément de L .*

Exemple 2.12 Le 1NFA $A_2 = (\{q_1, q_2\}, q_1, \{q_1\}, \delta_2)$ de l'exemple 2.11 est déterministe et reconnaît l'ensemble des mots de longueur paire. L'ensemble des mots de longueur impaire est le langage $\Sigma(\Sigma\Sigma)^*$ reconnu par le 1DFA $A_3 = (\{q_1, q_2\}, q_1, \{q_2\}, \delta_2)$ représenté figure 2.5.

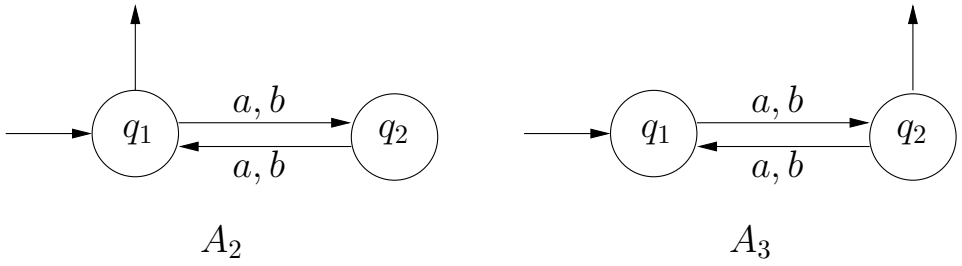

FIG. 2.5 – Représentation graphique du 1DFA A_2 et de son complément A_3

2.2.3 Automates bidirectionnels

Un 1NFA est un automate avec une tête de lecture qui parcourt le mot de gauche à droite. En dotant les automates unidirectionnels de la faculté de déplacer leur tête dans les deux directions, on obtient un modèle de calcul bidirectionnel qui est toujours confiné aux langages réguliers mais dont la concision augmente exponentiellement. Les automates bidirectionnels ont été introduits par M.O. Rabin et D. Scott [41]. Avant de définir formellement les automates bidirectionnels, nous définissons le *type* d'une position d'un mot.

Définition 2.11 *Le type d'une position d'un mot est une information finie qui indique si la position donnée est la première du mot, la dernière du mot ou bien une autre position. Plus formellement, soit w un mot de longueur n, pour toute position $i \in [n]$, le type de i noté $\theta_w(i)$, ou $\theta(i)$ s'il n y a pas d'ambiguïté sur le mot w, est un élément de l'ensemble* $\{\text{pre}, \text{mil}, \text{der}\}$ *tel que* $\theta(1) = \text{pre}$, $\theta(n) = \text{der}$ *et* $\theta(i) = \text{mil}$ *sinon. On note* TYPE_M *l'ensemble des types* $\{\text{pre}, \text{mil}, \text{der}\}$.

Remarque 2.8 Si w est un mot réduit à une lettre, la seule lettre du mot w est à la fois la première et la dernière lettre. Dans ce cas particulier, la position correspondante à cette lettre a deux types à la fois. Il faudrait donc ajouter un nouvel élément à TYPE_M uniquement pour ce cas particulier si on veut que la notation $type(v)$ désigne toujours un unique élément.

Remarque 2.9 Le type d'une position peut être défini en utilisant les formules du premier ordre pre et der définies dans l'exemple 2.4. Par exemple, une position d'un mot w est de type mil si et seulement si elle satisfait la formule à une variable libre que l'on note $\text{mil}(x)$ et qui est définie par $\neg\text{der}(x) \wedge \neg\text{pre}(x)$.

Définition 2.12 *Un automate bidirectionnel sur les mots, ou un* 2NFA *est un quintuplet $(Q, \Sigma, I, F, \delta)$ où Q est un ensemble fini d'états, Σ est un alphabet, $I \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états initiaux, $F \subseteq Q$ est un*

sous-ensemble de Q appelé ensemble des états finaux et δ est un sous-ensemble de $(Q \times \Sigma \times \mathrm{TYPE}_M) \times (Q \times \{-1, 0, +1\})$ appelé ensemble des transitions.

Un automate bidirectionnel A choisit un nouvel état et un déplacement de sa tête de lecture, -1 pour aller à la position précédente (à gauche), 0 pour rester sur place et $+1$ pour aller à la position suivante (à droite) en fonction de l'état courant ainsi que du type et de l'étiquette de la position de la tête de lecture. La position de la tête de lecture d'un automate séquentiel sur les mots est appelée la *position courante*.

Soit $w \in \Sigma^*$ un mot de longueur $n > 0$, une configuration de A sur w est un couple (q, i) où $q \in Q$ correspond à l'état de controle et $i \in [n]$ donne la position de la tête de lecture. Une configuration initiale de A sur w est une configuration de $I \times \{1\}$, c'est-à-dire une configuration telle que la tête de lecture de l'automate est sur la première lettre et l'état de contrôle est un état initial. Une configuration finale de A sur w est une configuration de $F \times \{n\}$, c'est-à-dire une configuration telle que la tête de lecture est sur la dernière lettre de w et l'état de contrôle est un état final. On dit qu'une configuration (r, j) est un successeur d'une configuration (q, i) ce que l'on note $(q, i) \xrightarrow{A} (r, j)$ s'il existe une transition $((q, a, \theta), (r, k))$ telle que $k = j - i$, a est l'étiquette de la lettre à la position i et telle que $\theta = \mathrm{pre}$ si $i = 1$, $\theta = \mathrm{der}$ si $i = n$ et $type = \mathrm{mil}$ sinon. Une exécution de A sur w est une suite de configurations successives $c_1 \to c_2 \cdots \to c_m$. S'il existe une exécution partant de la configuration c_1 et terminant dans la configuration c_m, on dit que c_m est accessible à partir de c_1 et initial et on note cela $c_1 \xrightarrow{A,*} c_m$

Une exécution est acceptante si elle part d'une configuration initiale et termine dans une configuration finale. Un mot non-vide est accepté par l'automate s'il existe une exécution acceptante de l'automate sur ce mot. Le mot vide est accepté par l'automate s'il existe un état qui est à la fois un état initial et un état final.

Comme pour les automates unidirectionnels, nous définissons la variante déterministe du modèle des automates bidirectionnels.

Définition 2.13 *Un automate bidirectionnel déterministe sur les mots, ou un* 2DFA*, est un automate bidirectionnel $(Q, \Sigma, I, F, \delta)$ tels que I est un singleton et δ est une fonction de $Q \times \Sigma \times \mathrm{TYPE}_M$ à valeurs dans $Q \times \{-1, 0, +1\}$*

Remarque 2.10 Un 2NFA ne peut pas se déplacer à gauche de la première lettre ou à droite de la dernière lettre. Si Q est l'ensemble des états, les transitions de $(Q \times \Sigma \times \{\mathrm{pre}\}) \times (Q \times \{-1\})$ et de $(Q \times \Sigma \times \{\mathrm{der}\}) \times (Q \times \{+1\})$ ne peuvent donc pas être effectuées.

Remarque 2.11 Un automate unidirectionnel peut être considéré comme un automate bidirectionnel qui se déplace vers la droite jusqu'à la dernière lettre. En effet, étant donné A un 1NFA, il est très facile de construire un 2NFA A' qui reconnaît le même langage : A' se comporte comme A jusqu'à la dernière position à partir de laquelle il ne peut appliquer que les transitions de A qui lui permettent d'atteindre un état final. Nous verrons avec le théorème 2.9 de la section suivante que les 2NFA et les 1NFA reconnaissent en fait les mêmes langages.

Exemple 2.13 Soit Σ un alphabet, nous définissons un 2DFA A_4 qui reconnaît le langage $(\Sigma^6)^*$ des mots dont la longueur est divisible par 6. Cet automate commence

par lire le mot de gauche à droite et vérifie ainsi qu'il est de longueur paire puis il parcourt le mot dans l'autre sens et vérifie que sa longueur est divisible par 3. Enfin, si la longueur du mot est bien divisible par 2 et par 3, l'automate retourne à la fin du mot et accepte.

On pose ainsi $A_4 = (\{q_0, q_1, q'_0, q'_1, q'_2, q_f\}, \Sigma, q_0, \{q_f\}, \delta)$ où δ est l'ensemble des transitions de la forme :

- $((q_0, \sigma, \theta), (q_1, +1))$ et $((q_1, \sigma, \text{mil}), (q_0, +1))$ avec $\sigma \in \Sigma$ et $\theta \in \{\text{pre}, \text{mil}\}$,
- $((q_1, \sigma, \text{der}), (q'_1, -1))$ et $((q'_1, \sigma, \text{mil}), (q'_2, -1))$ avec $\sigma \in \Sigma$,
- $((q'_2, \sigma, \text{mil}), (q'_0, -1))$ et $((q'_0, \sigma, \text{mil}), (q'_1, -1))$ avec $\sigma \in \Sigma$,
- $((q'_2, \sigma, \text{pre}), (q_f, +1))$ et $((q_f, \sigma, \text{mil}), (q_f, +1))$ avec $\sigma \in \Sigma$.

Nous représentons ci-dessous une exécution acceptante de A_4 sur un mot de six lettres. Elle commence au début du mot dans l'état q_0 et se termine à la fin du mot dans l'état q_f

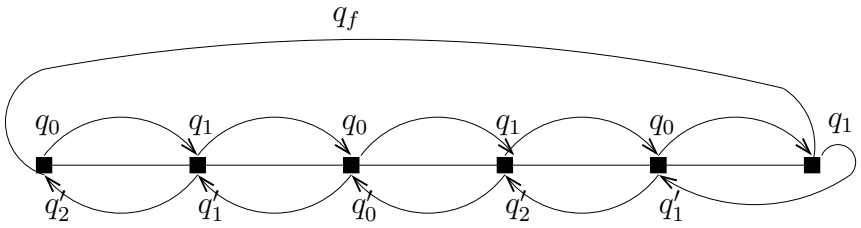

FIG. 2.6 – Représentation de l'exécution du 2DFA A_4 sur un mot à 6 lettres

Nous verrons avec le théorème 2.9 que les automates bidirectionnels ont le même pouvoir d'expression que les automates unidirectionnels. Etant donné un 2DFA, il est possible de construire un 1NFA équivalent [41, 51]. Dans la partie 7.2.1, nous décrivons la construction de [14] qui permet d'obtenir à partir d'un automate cheminant un automate d'arbres bottom-up équivalent. Cette construction s'adapte facilement au cas des mots. On peut ainsi transformer un automate bidirectionnel A_1 en un automate unidirectionnel A_2 équivalent. L'idée de cette transformation est la suivante. Pour chaque position i d'un mot $w_1 \cdots w_n$, A_2 calcule pas à pas dans son état d'une part l'ensemble des couples d'états (q, q') correspondant aux exécutions de A_1 démarrant à la position i dans l'état q et terminant à la position i dans l'état q' telles que A_1 ne quitte pas le préfixe $w_1 \cdots w_i$ et d'autre part l'ensemble des états dans lesquels A_1 peut atteindre la position i à partir d'une configuration initiale en restant toujours dans le préfixe $w_1 \cdots w_i$.

2.3 Automates sur les arbres

Nous définissons dans un premier temps les contextes et les sous-arbres, puis nous présentons le modèle classique d'automates d'arbres que nous appelons automates d'arbres bottom-up et enfin le modèle séquentiel des automates cheminants.

2.3.1 Arbres finis de rang borné

Dans le chapitre précédent, nous avons défini trois classes d'arbres finis : les arbres binaires, les arbres de rang borné et les arbres de rang non-borné étiquetés. Pour chacune de ces classes, un langage d'arbres est un ensemble d'arbres. Comme pour les langages de mots, les opérateurs booléens union, intersection et complément peuvent être définis sur les langages d'arbres. Par contre, le produit de deux langages d'arbres et l'étoile d'un langage d'arbres ne peuvent pas être définis comme pour les langages de mots car la concaténation de deux arbres n'est pas définie. On peut alors se demander comment composer des arbres. Nous introduisons donc les notions de sous-arbre, de contexte et de substitution.

Définition 2.14 *Soient $t = (\mathcal{N}, \lambda)$ un arbre binaire étiqueté par l'alphabet Σ et v un nœud de t. Le sous-arbre de t enraciné en v est l'arbre $(\mathcal{N}', \lambda')$ noté $t_{|v}$ tel que*
- *$\mathcal{N}' = \{u \in \{1,2\}^* | \ vu \in \mathcal{N}\}$ et*
- *λ' est la fonction définie sur $\mathcal{N}'$ telle que $\lambda'(u) = \lambda(vu)$.*

Si v est un nœud interne le sous-arbre gauche de v est le sous-arbre de t enraciné en $v1$, le fils gauche de v, et le sous-arbre droit de v est le sous-arbre enraciné en $v2$, le fils droit de v.

A partir de maintenant, on considère un alphabet Σ et un symbole $\circledcirc$ qui n'appartient pas à Σ.

Définition 2.15 *Un contexte sur l'alphabet Σ est un arbre sur $\Sigma \cup (\Sigma \times \{\circledcirc\})$ tel que le symbole "$\circledcirc$" apparaît seulement sur une seule feuille ; tous les autres nœuds sont étiquetés par des lettres de Σ. La feuille dont l'étiquette contient le symbole spécial "$\circledcirc$" est appelée le trou du contexte. Soient $t = (\mathcal{N}, \lambda)$ un arbre et v un nœud. Le contexte de t pointé en v noté $C_{t,v}$, ou C_v s'il n'y a pas d'ambiguïté pour le choix de t, est le contexte obtenu en remplaçant dans t le sous-arbre enraciné en v par une feuille étiquetée par $(\lambda(v), \circledcirc)$. Plus formellement, C_v est l'arbre $(\mathcal{N}', \lambda')$ sur l'alphabet $\Sigma \cup \{\circledcirc\}$ où*
- *$\mathcal{N}' = \{u \in \mathcal{N} | \ v$ n'est pas un préfixe propre de $u\}$ et*
- *λ' a la même valeur que λ pour tout nœud de $\mathcal{N}'$ distinct de v et vaut $(\lambda(v), \circledcirc)$ pour v.*

Ainsi le contexte $C_{t,v}$ est obtenu en supprimant tous les descendants propres de v dans t et en remplaçant l'étiquette de v par $(\lambda(v), \circledcirc)$.

Exemple 2.14 La figure 2.7 représente un arbre t sur l'alphabet $\{a, b, c\}$, le sous-arbre $t_{|v}$ et le contexte C_v où v est un nœud de t que l'on a marqué par un carré.

Si l'on compose un contexte C et un arbre t en remplaçant le trou de C par t, on obtient un arbre comme dans la figure 2.8

Définition 2.16 *Soient $C = (\mathcal{N}_C, \lambda_C)$ un contexte, v le trou de C et $t = (\mathcal{N}_t, \lambda_t)$ un arbre sur l'alphabet Σ. On dit que C et t sont compatibles si $\lambda_C(v) = \lambda_t(\epsilon)$. On appelle alors composition de C et t, que l'on note $C[t]$, l'arbre $(\mathcal{N}, \lambda)$ tel que*
- *$\mathcal{N} = dom_C \cup \{vu | u \in \mathcal{N}_t\}$*
- *$\forall u \in \mathcal{N}_C \setminus \{v\}$, $\lambda(u) = \lambda_C(u)$ et $\forall u \in \mathcal{N}_t$, $\lambda(vu) = \lambda_t(u)$.*

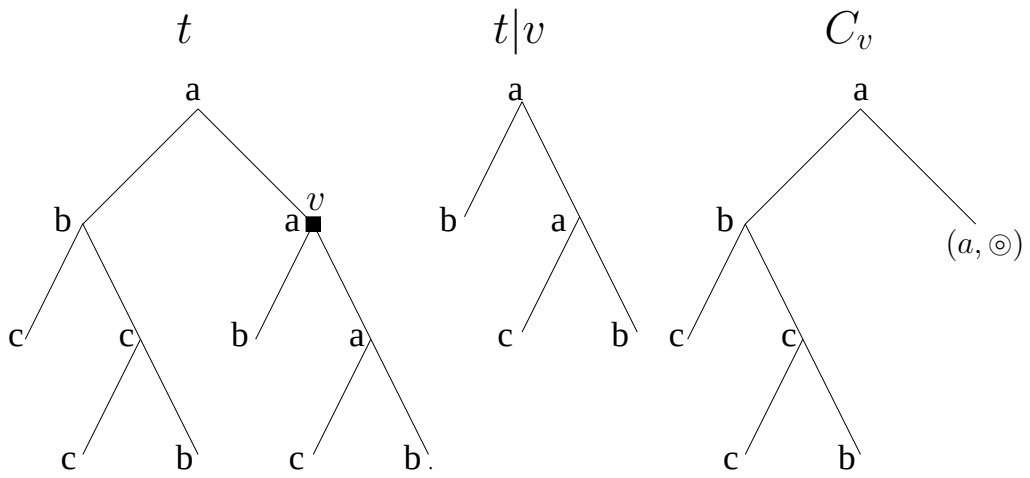

FIG. 2.7 – Représentation d'un arbre, d'un des ses sous-arbres et d'un de ses contextes.

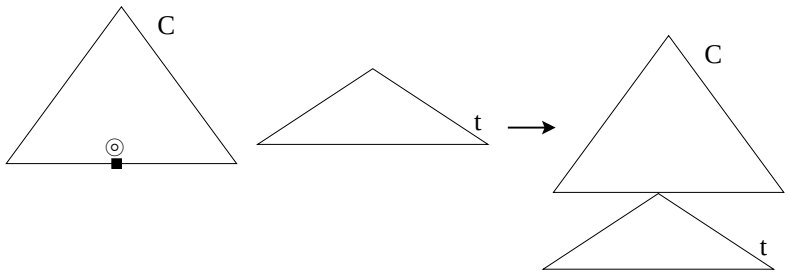

FIG. 2.8 – Composition d'un contexte et d'un arbre

2.3.2 Langages d'arbres réguliers et automates d'arbres bottom-up

Les automates d'arbres bottom-up ont été introduits par J. Doner [15] et par J.W. Thatcher et J.B. Wright [49].

Définition 2.17 *Un automate d'arbres bottom-up, ou un* BU, *est un quintuplet* $(Q, \Sigma, q_0, F, \delta)$ *où* Q *est un ensemble fini d'états,* Σ *est un alphabet,* $q_0 \in Q$ *est un état de* Q *appelé état initial,* $F \subseteq Q$ *est un sous-ensemble de* Q *appelé ensemble des états acceptants et* δ *est un sous-ensemble de* $(\Sigma \times Q \times Q) \times Q$ *appelé ensemble des transitions.*

Soient t un arbre binaire étiqueté par Σ et $B = (Q, \Sigma, q_0, F, \delta)$ un BU. Un calcul de B sur t est une fonction ρ de l'ensemble des nœuds de t à valeurs dans Q telle que, pour chaque nœud x d'étiquette σ, on a

- $((\sigma, q_0, q_0), \rho(x)) \in \delta$ si x est une feuille et
- $((\sigma, \rho(x_1), \rho(x_2)), \rho(x)) \in \delta$ si x a pour fils droit x_1 et pour fils gauche x_2,.

Un calcul de B sur t est acceptant si l'état affecté à la racine est acceptant, on dit alors que l'arbre t est accepté par B. Le langage reconnu par B est l'ensemble des arbres pour lesquels il existe un calcul acceptant de B. La famille des langages d'arbres reconnus par des automates d'arbres bottom-up est appelée la classe des langages réguliers d'arbres.

Il existe d'autres modèles d'automates d'arbres qui permettent de définir les langages d'arbres réguliers : par exemple les automates qu'on appelle "automates top-down" en anglais parce qu'ils évaluent un arbre de la racine aux feuilles, les automates d'arbres alternants, les automates qui peuvent effectuer des tests MSO...

Remarque 2.12 Pour les arbres de rang borné, il est facile d'étendre la définition d'automates BU :

Définition 2.18 *Un automate d'arbres de rang borné bottom-up, B est un quintuplet $(Q, \Sigma, q_0, F, \delta)$ où Q est un ensemble fini d'états, $\Sigma = (A, d)$ est un alphabet gradué de rang r, $q_0 \in Q$ est un état de Q appelé état initial, $F \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états acceptants et δ est un sous-ensemble de $\bigcup_{i\in\{0,\cdots,m\}}(Q^i \times A) \times Q$ appelé ensemble des transitions tel que si $((q_1, \cdots, q_j), a, q) \in \delta$ avec $0 \leq j \leq r$ alors j est le degré de la lettre a.*

Soit t un arbre sur l'alphabet gradué (A, d). Un calcul de B sur t est une fonction ρ de l'ensemble des nœuds de t à valeurs dans Q tellle que, si x est un nœud d'étiquette σ dont les fils de gauche à droite sont $x_1, \cdots, x_d$, on a $((\rho(x_1), \cdots, \rho(x_d)), \sigma, \rho(x)) \in \delta$. On définit alors comme pour les BU les calculs acceptants et les langages réguliers d'arbres de rang borné.

Pour les arbres de rang non-borné, on peut encore définir des automates d'arbres qui effectuent des calculs en parallèle en procédant des feuilles à la racine mais la définition de la fonction de transition doit être adaptée pour que cette fonction soit représentable de manière finie. Nous introduirons les automates d'arbres de rang non borné dans le dernier chapitre.

Nous définissons maintenant la variante déterministe des automates d'arbres binaires.

Définition 2.19 *Un* BU $B = (Q, \Sigma, q_0, F, \delta)$ *est déterministe si δ est une fonction partiellement définie de $(\Sigma \times Q \times Q)$ à valeurs dans Q. L'automate déterministe B est alors complet si δ est partout définie.*

Les BU déterministes reconnaissent tous les langages réguliers comme le montre le théorème ci-dessous.

Théorème 2.4 *Soit L un langage reconnu par un* BU B *dont l'ensemble d'états est Q. Il existe un automate déterministe qui reconnaît L dont le nombre d'états est $2^{|Q|}$.*

Les BU peuvent donc être déterminisés contrairement aux automates d'arbres qui procèdent de la racine aux feuilles. Cette propriété très importante nous a conduits à choisir d'introduire les langages d'arbres réguliers avec les automates BU. Tout comme les 1DFA les BU déterministes sont faciles à complémenter, il suffit de les compléter et d'intervertir l'ensemble des états finaux et l'ensemble des états non-finaux. On en déduit le théorème suivant :

Théorème 2.5 *Soit L un langage reconnu par un* BU B *dont l'ensemble d'états est Q. Il existe un automate déterministe qui reconnaît le complément de L dont le nombre d'états est $2^{|Q|}$.*

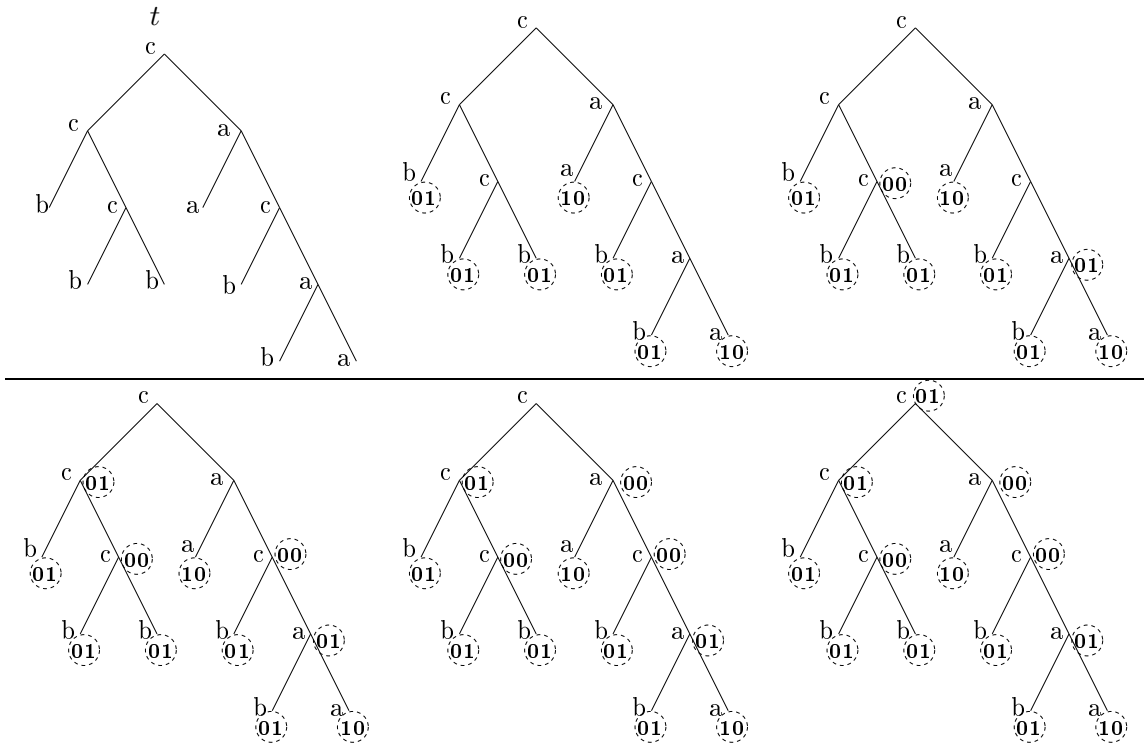

FIG. 2.9 – Les différentes étapes d'une exécution de B_1 sur t

Exemple 2.15 On note tout d'abord $\bigoplus$ l'addition sur l'ensemble des booléens $\{0,1\}$ tel que $0 \bigoplus 0 = 1 \bigoplus 1 = 0$ et $1 \bigoplus 0 = 0 \bigoplus 1 = 1$ On considère ensuite L_1 le langage des arbres binaires sur l'alphabet $\Sigma = \{a, b, c\}$ qui vérifient la propriété suivante : un nœud est étiqueté par la lettre c si et seulement si son sous-arbre droit contient un nombre pair de nœuds étiquetés par la lettre a et son sous-arbre gauche contient un nombre impair de nœuds étiquetés par la lettre b. On peut vérifier que l'arbre t de la figure 2.9 appartient à L_1. Le langage L_1 est reconnu par le BU déterministe $B_1 = (Q, \Sigma, 00, Q, \delta)$ avec $Q = \{00, 01, 10, 11\}$ et δ définie par

- $\forall x_a, x_b, y_a, y_b \in \{0,1\}$, $\delta(a, x_a x_b, y_a y_b) = (x_a \bigoplus y_a \bigoplus 1)(x_b \bigoplus y_b)$
- $\forall x_a, x_b, y_a, y_b \in \{0,1\}$, $\delta(b, x_a x_b, y_a y_b) = (x_a \bigoplus y_a)(x_b \bigoplus y_b \bigoplus 1)$
- $\forall x_a, y_b \in \{0,1\}$,$\delta(c, x_a 1, 0 y_b) = x_a(y_b \bigoplus 1)$

L'état affecté à un nœud par l'automate est un nombre de deux chiffres binaires, le premier code la parité du nombre de nœuds étiquetés par a dans le sous-arbre droit, le second code la parité du nombre de nœuds étiquetés par b dans le sous-arbre gauche.

On représente figure 2.9 les différentes étapes de l'exécution acceptante de B_1 sur l'arbre t.

2.3.3 Automates d'arbres cheminants

Nous présentons maintenant un modèle d'automates d'arbres séquentiels introduit par Aho et Ullman qui étend le modèle des automates bidirectionnels aux arbres. Il s'agit des *automates cheminants* appelés tree-walking automata en anglais. Un automate cheminant est donc un automate avec une tête de lecture qui se déplace dans un arbre d'un nœud à un autre en suivant les arêtes. Nous avons vu que les transitions d'un automate bidirectionnel sur les mots prennent en compte le type de la position de la tête de lecture afin de détecter le début et la fin du mot parcouru. Un automate cheminant a également accès à une information finie concernant la position dans l'arbre du nœud sur lequel est sa tête de lecture. Ce nœud est appelé *nœud courant* et cette information est encore appelée le *type* de ce nœud.

Définition 2.20 *Soit t un arbre de rang n et de domaine et u un nœud de t. Le type de u est l'élément noté $\theta(u)$ de l'ensemble* $\mathrm{TYPE} = \{in, fe\} \times (\{ra\} \cup [n])$ *tel que*

- *si t est réduit au nœud u, ce nœud est à la fois la racine et une feuille et $\theta(u) = (fe, ra)$,*
- *si u est à la fois un nœud interne et la racine, $\theta(u) = (in, ra)$,*
- *si u est une feuille et $u = xi$ avec $1 \leq i \leq n$, alors $\theta(u) = (fe, i)$ et*
- *si u est un nœud interne et $u = xi$ avec $1 \leq i \leq n$, alors $\theta(u) = (in, i)$.*

Un automate cheminant sur les arbres binaires sait donc à chaque étape d'un calcul si le nœud sur lequel est sa tête de lecture est un fils gauche, un fils droit ou la racine et si ce nœud est une feuille de l'arbre. Un automate cheminant choisit ainsi son nouvel état en fonction de l'état courant ainsi que du type et de l'étiquette du nœud courant. Les déplacements possibles de la tête de lecture sont les suivants : $\uparrow$ pour remonter au père, $\circlearrowleft$ pour rester sur place, $\swarrow$ et $\searrow$ pour descendre respectivement au fils gauche et au fils droit. Nous donnons maintenant une définition formelle des automates cheminants.

Définition 2.21 *Un automate cheminant sur les arbres binaires, ou un* TWA *(pour Tree Walking Automaton), est un quintuplet $(Q, \Sigma, I, F, \delta)$ où Q est un ensemble fini d'états, Σ est un alphabet, $I \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états initiaux, $F \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états finaux ou ensemble des états acceptants et δ est un sous-ensembe de*

$(Q \times \Sigma \times \mathrm{TYPE}) \times (Q \times \{\uparrow, \swarrow, \searrow, \circlearrowleft\})$ *appelé ensemble des transitions.*

Soient t un arbre et $A = (Q, \Sigma, I, F, \delta)$ un TWA. Une *configuration* de A sur t est un couple (q, u) où $q \in Q$ correspond à l'état de contrôle et est appelé *état courant* et u est le *nœud courant* qui donne la position de la tête de lecture. Une configuration initiale de A sur t est une configuration de $I \times \{\epsilon\}$, c'est-à-dire une configuration telle que la tête de lecture de l'automate est sur la racine et telle que l'état de contrôle est un état initial de l'automate. Une configuration acceptante de A sur t est une configuration de $F \times \{\epsilon\}$. Une configuration (r, v) est un successeur d'une configuration (q, u) ce que l'on note $(q, u) \xrightarrow{A} (r, v)$ s'il existe une transition $((q, a, \theta), (\mu, r))$ telle que θ est le type de u, a est l'étiquette du nœud u et

- $v = u$ si $\mu = \circlearrowleft$,
- v est le père de u si $\mu = \uparrow$,
- v est le fils gauche de u si $\mu = \swarrow$ et
- v est le fils droit de u si $\mu = \searrow$.

Une exécution de A sur t est une suite de configurations successives $c_1 \xrightarrow{A} c_2 \xrightarrow{A} \cdots c_m$. S'il existe une exécution partant de la configuration c_1 et terminant dans la configuration c_m, on dit que c_m est accessible à partir de c_1 et on note cela $c_1 \xrightarrow{A,*} c_m$. Une exécution est acceptante si elle part d'une configuration initiale et termine dans une configuration acceptante. On dit alors que A accepte t.

Définition 2.22 *Un automate cheminant déterministe sur les arbres, ou un* DTWA*, A est un automate cheminant $(Q, \Sigma, I, F, \delta)$ tel que I est un singleton et δ est une fonction de $Q \times \Sigma \times$ TYPE à valeurs dans $Q \times \{\uparrow, \swarrow, \searrow, \circlearrowleft\}$.*

Remarque 2.13 Un TWA accepte à la racine d'après la définition d'une configuration finale. Cette convention ne diminue pas le pouvoir d'expression des TWA. En effet, si on suppose qu'un TWA accepte dès qu'il atteint un état final quel que soit le nœud courant, alors, pour tout TWA A, on peut construire un TWA qui simule A jusqu'à ce qu'un état final soit atteint et qui, à partir de cette configuration remonte toujours à la racine.

Remarque 2.14 Comme pour les BU on peut facilement adapter la définition des TWA pour les arbres de rang borné. Un automate cheminant sur des arbres unaires correspond alors à un automate bidirectionnel qui accepte au début du mot. Afin d'étendre le modèle des automates cheminants aux arbres de rang non-borné, nous devrons introduire pour les arbres de rang non-borné une nouvelle définition du type d'un nœud et un nouvel ensemble de déplacements. En effet, la fonction de transition doit toujours être décrite de manière finie. Nous verrons comment on peut définir les TWA pour les arbres de rang non-borné dans le chapitre 8.

Exemple 2.16 Soient Σ un alphabet, a une lettre de Σ et L_5 le langage des arbres dont toutes les feuilles sont étiquetés par a. Considérons A_5 l'automate cheminant déterministe suivant : $A_5 = (\{q_\swarrow, q_\uparrow, q_\searrow\}, \Sigma, q_\swarrow, \{q_\uparrow\}, \delta)$ où δ est définie par :

- $\forall \sigma \in \Sigma$, $\delta(q_\swarrow, \sigma, (in, \begin{smallmatrix} ra \\ 1 \\ 2 \end{smallmatrix})) = (q_\swarrow, \swarrow)$ et $\delta(q_\swarrow, a, (fe, \begin{smallmatrix} 1 \\ 2 \end{smallmatrix})) = (q_\uparrow, \circlearrowleft)$
- $\forall \sigma \in \Sigma$, $\delta(q_\uparrow, \sigma, (\begin{smallmatrix} in \\ \text{fe} \end{smallmatrix}, 1)) = (q_\searrow, \uparrow)$ et $\delta(q_\uparrow, \begin{smallmatrix} \sigma \\ a \end{smallmatrix}, (\begin{smallmatrix} in \\ fe \end{smallmatrix}, 2)) = (q_\uparrow, \uparrow)$
- $\forall \sigma \in \Sigma$, $\delta(q_\searrow, \sigma, (in, \begin{smallmatrix} 1 \\ 2 \end{smallmatrix})) = (q_\swarrow, \searrow)$

L'automate défini ci-dessus effectue un parcours en profondeur d'un arbre donné et visite ainsi toutes ses feuilles pour vérifier qu'elles sont étiquetées par la lettre a. Dans l'état initial $q_\swarrow$, il descend le plus à gauche possible sans changer d'état jusqu'à ce qu'il atteigne une feuille. Si cette feuille est étiquetée par a, l'automate passe alors dans l'état $q_\uparrow$. Dans l'état $q_\uparrow$, A_5 remonte dans l'arbre en restant dans cet état tant que le nœud courant est un fils droit. Si A_5 remonte ainsi jusqu'au fils gauche d'un nœud v, il a alors visité tout le sous-arbre gauche de v, il remonte alors en v et passe dans l'état $q_\searrow$ pour vister le sous-arbre droit de v. Si, à partir de l'état $q_\uparrow$, l'automate remonte jusqu'à la racine, il accepte. Dans l'état $q_\searrow$, à partir

d'un nœud v, l'automate descend au nœud u le fils droit de v et passe dans l'état $q_{\swarrow}$ pour visiter le sous-arbre enraciné en u. La figure 2.10 représente une exécution de A_5 sur un arbre de L_5. Nous avons représenté les sous-exécutions où l'automate est dans l'état $q_{\swarrow}$ par des flèches bleues, celles où l'automate est dans l'état $q_{\uparrow}$ par des flèches rouges et celles où l'automate est dans l'état $q_{\searrow}$ par des flèches vertes.

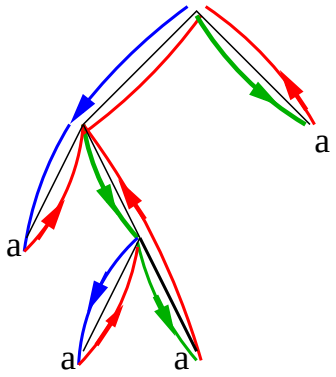

FIG. 2.10 – Représentation de l'exécution du DTWA A_5 sur un arbre de L_5

Kamimura et Slutzky ont montré dans [27] qu'un automate cheminant qui n'a pas l'information fils gauche ou fils droit sur le nœud ne peut pas reconnaître le langage L_5. En effet, un automate cheminant tel que les transitions effectuées à partir d'un nœud autre que la racine ne dépendent pas de la deuxième composante du type du nœud courant qui vaut 1 pour fils gauche et 2 pour fils droit ne peut pas vérifier que toutes les feuilles d'un arbre en entrée sont étiquetées par la lettre a.

Exemple 2.17 On considère maintenant le langage L_6 des arbres tels qu'il existe un chemin de la racine à une feuille dans lequel tous les nœuds sont étiquetés par la lettre a. Nous définissons A_6 un automate non-déterministe qui devine un chemin de la racine à une feuille pour reconnaître ce langage. A_6 a deux états : son état initial q_0 et son état final q_1 et la relation de transition δ. Dans l'état q_0, si l'automate est sur un nœud interne étiqueté par la lettre a, il reste dans cet état et descend au fils gauche ou au fils droit. Si l'automate atteint une feuille étiquetée par a dans l'état q_0, il passe dans l'état q_1. Dans l'état final, l'automate remonte jusqu'à la racine et reste dans cet état pour accepter l'arbre.

Nous avons représenté dans la figure 2.11 la sous-exécution où l'automate est dans l'état q_0 par une flèches bleue et celles où l'automate est dans l'état q_1 par une flèche rouge.

Il est facile de construire un automate cheminant non-déterministe qui reconnaît l'union et l'intersection de deux langages reconnus par des automates cheminants non-déterministes.

Proposition 2.1 *La classe des langages reconnus par un automate cheminant est fermée par union et intersection.*

Preuve.

Etant donné A_1 et A_2 deux automates cheminants, l'automate qu'on construit pour l'union choisit de manière non-déterministe un des deux automates A_i avec $i \in$

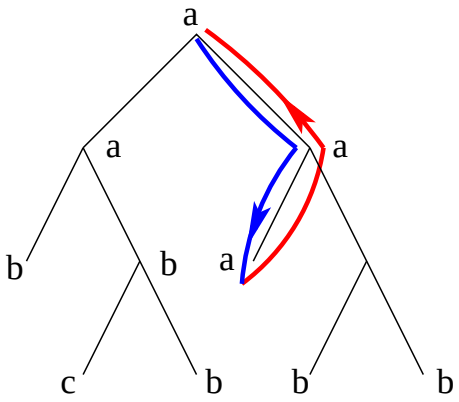

FIG. 2.11 – Représentation d'une exécution acceptante de A_6 sur un arbre de L_6

$\{1, 2\}$, le simule et accepte si l'automate A_i accepte. L'automate pour l'intersection simule d'abord l'automate A_1, si A_1 accepte il simule alors l'automate A_2 et accepte si A_2 accepte. □

Il est également simple de construire un automate cheminant déterministe qui reconnaît l'intersection de deux langages reconnus par des automates cheminants déterministes.

Proposition 2.2 *La classe des langages reconnus par un automate cheminant déterministe est fermé par intersection.*

Preuve. Etant donné A_1 et A_2 deux automates cheminants, l'automate pour l'intersection simule d'abord l'automate A_1, si A_1 accepte il simule alors l'automate A_2 et accepte si A_2 accepte. □

La preuve pour reconnaître l'union de deux langages reconnus par des automates cheminants non-déterministes ne fonctionnent plus dans le cas déterministe. En effet, l'automate ne peut plus deviner lequel des deux automates il doit simuler et il ne peut pas simuler les deux automates l'un après l'autre à cause des exécutions infinies. Nous verrons comment résoudre ce problème dans le chapitre 3.

2.4 Automates à jetons

2.4.1 Automates d'arbres à jetons

Nous présentons maintenant un modèle d'automates d'arbres séquentiels introduits par J. Engelfriet et H.J. Hoogeboom dans [18] : les automates d'arbres à jetons. Une variante de ce modèle d'automates sur les mots avait été étudiée par N. Globerman and D. Harel [24]. Les définitions des automates d'arbres à jetons qui suivent s'adaptent sans difficulté au cas des mots.

Informellement, un automate d'arbres à jetons est un automate séquentiel avec une tête de lecture qui se déplace dans l'arbre comme celle d'un automate cheminant et avec un nombre fixé de jetons qui peuvent être placés sur les nœuds de l'arbre. Nous imposons ci-dessous une restriction importante sur le placement des jetons.

Les jetons sont numérotés de 1 à k. A chaque étape d'une exécution, les jetons qui ont été placés par l'automate dans l'arbre sont les jetons dont les numéros sont

supérieurs ou egaux à i pour un certain entier i. Notons que $i = 1$ si tous les jetons sont placés dans l'arbre et $i = (k + 1)$ si aucun jeton n'est posé dans l'arbre. Si les jetons posés dans l'arbre sont les jetons numérotés de i à k, le seul jeton que l'automate peut lever est le jeton i (si $i \leq k$) et le seul jeton que l'automate peut poser est le jeton $(i - 1)$ (si $i > 1$). On dit que le placement des jetons suit une discipline de pile car, à chaque étape d'une exécution, le jeton qui peut être levé par l'automate est le dernier jeton qui a été posé.

Les transitions que l'automate peut effectuer à partir d'une configuration dépendent de l'état courant, du type et de l'étiquette du nœud courant ainsi que de l'ensemble des jetons posés sur ce nœud. Nous distinguons deux modèles d'automates à jetons. Dans le premier modèle, que nous appelons le modèle faible, un jeton peut être levé par l'automate seulement si sa tête de lecture est sur la position où ce jeton est posé. Le jeton est alors vu comme un objet physique. Dans le second modèle que nous appelons le modèle fort, l'automate peut lever le dernier jeton qu'il a posé à partir de n'importe quel nœud. Le jeton peut ainsi être appelé à distance et il est donc vu comme un pointeur.

Remarque 2.15 Sans la restriction sur le placement des jetons, il est facile de simuler une machine à deux compteurs avec un automate à deux jetons sur les mots. Ainsi, si l'on supposait que les jetons peuvent être posés et levés dans n'importe quel ordre, les automates à jetons reconnaîtraient des langages non-réguliers car ils caractériseraient les langages des classes NLogSpace pour les automates non-déterministes et DLogSpace pour les automates déterministes qui désignent respectivement la classe des langages reconnus par une machine de Turing à espace logarithmique et celle des langages reconnus par une machine de Turing déterministe à espace logarithmique. Ces notions seront définies formellement dans le chapitre 7.

Remarque 2.16 Nous avons choisi que les jetons posés dans l'arbre au cours d'une exécution soient numérotés de k à i. Cette convention ne semble pas très intuitive : on peut en effet se demander pourquoi nous avons numéroté les jetons de telle sorte que l'automate pose en premier le jeton k, puis le jeton $(k - 1)$ et ainsi de suite au lieu de poser les jetons dans l'ordre croissant de leur numéro. Il se trouve que nos preuves par induction vont être simplifiées par cette convention avec laquelle un automate à k jetons se comporte comme un automate à $(i - 1)$ jetons à partir du moment où il a posé le jeton i jusqu'à ce qu'il lève ce jeton.

On rappelle que, étant donnés des entiers n et m, $[n]$ et $[n, m]$ désignent respectivement l'ensemble $\{1, \cdots, n\}$ des n premiers entiers strictement positifs et l'ensemble $\{n, \cdots, m\}$ des entiers compris entre n et m, $2^{[n]}$ désigne ainsi l'ensemble des parties de $\{1, \cdots, n\}$. Nous définissons maintenant formellement les automates à jetons sur les arbres binaires. Cette définition s'adapte sans difficulté pour les arbres de rang borné et donc pour les mots.

Définition 2.23 *Un automate à k jetons sur les arbres binaires, ou un* PTA_k*, est un quintuplet $(Q, \Sigma, I, F, \delta)$ où Q est un ensemble fini d'états, Σ est un alphabet, $I \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états initiaux, $F \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états finaux et δ est un sous-ensemble de*

$$(Q \times \Sigma \times \mathrm{TYPE} \times [0, k] \times 2^{[k]}) \times (Q \times \mathrm{ACTION})$$

appelé ensemble des transitions où ACTION *est l'ensemble fini* $\{\uparrow, \circlearrowleft, \swarrow, \searrow, \text{pose}, \text{leve}\}$.

Un automate d'arbres à jetons choisit donc un nouvel état et une action en fonction de l'état courant, de l'étiquette et du type du neud courant, du nombre de jetons qu'il peut encore poser dans l'arbre ainsi que des jetons qui sont placés sur le nœud courant. Les différentes actions possibles d'un automate d'arbres à jetons sont $\uparrow$, $\circlearrowleft$, $\swarrow$ et $\searrow$ pour se déplacer comme dans le cas des automates cheminants, "pose" pour poser un jeton sur le nœud courant et "leve" pour lever de l'arbre le dernier jeton qui a été posé.

Soit t un arbre. Etant donné un sous-ensemble des jetons $J \in 2^{[k]}$, une *affectation, des jetons* de J ou une J-affectation est une fonction qui attribue à chaque élément de J un nœud de t. Pour $0 \leq i \leq k$, une *i-configuration* est un triplet (q, u, f) où u est le nœud courant qui donne la position de la tête de lecture, $q \in Q$ est l'état courant et f est une affectation des jetons de $\{(i+1), \cdots, k\}$ appelée le *placement courant* des jetons. Notons que pour $k = i$, nous considérons que $\{k+1, \cdots, k\}$ désigne l'ensemble vide. Une *configuration* est une i-configuration pour un i donné.

Notation 2.2 *Pour désigner la configuration (q, u, f) on pourra utiliser la notation $(q, u, u_k, \cdots, u_{i+1})$ où $u_j = f(j)$ pour $i+1 \leq j \leq k$ et, lorsque k est fixé, pour désigner une suite de positions $u_k, \cdots, u_i$, on utilisera la notation $\vec{u}_i$. Une k-configuration $(q, u, \emptyset)$ sera noté (q, u) Ainsi la configuration (q, u, f) telle que $u_j = f(j)$ pour $i+1 \leq j \leq k$ se note également $(q, u, \vec{u}_{i+1})$.*

Par défaut, nous considérons qu'un automate à jetons appartient au modèle fort.

Soit $A = (Q, \Sigma, I, F, \delta)$ un PTA$_k$. Une i-configuration (q, u, f) a pour successeur une configuration (q', u', f'), ce que l'on note $(q, u, f) \xrightarrow{A} (q', u', f')$ dans les 3 cas suivants :

1. les affectations de jetons f et f' sont identiques et il existe une transition $((q, a, \theta, i, f^{-1}(u)), (q', D))$ où $D \in \{\uparrow, \circlearrowleft, \swarrow, \searrow\}$ tel que a est l'étiquette du nœud u, θ est le type de u et $k = \circlearrowleft$ si $u = u'$, $k = \uparrow$ si u' est le père de u, $k = \swarrow$ si u' est le fils gauche de u et $k = \searrow$ si u' est le fils droit de u.
2. les positions u et u' sont identiques, les affectations de jetons sont telles que $f' = f \cup \{(i, u)\}$ et il existe une transition $((q, a, \theta, i, f^{-1}(u), (q', \text{pose}))$ où a et θ sont respectivement l'étiquette et le type du nœud u.
3. les positions u et u' sont identiques, l'entier i est strictement inférieur à k, f' est une affectation des jetons de $\{i+2, \cdots, k\}$ telle que pour tout $i+2 \leq j \leq k$, $f(j) = f'(j)$ et il existe une transition $((q, a, \theta, i, f^{-1}(u)), (q', \text{leve}))$ telle que a et θ sont respectivement l'étiquette et le type de la position u.

Le premier cas décrit un déplacement de la tête de lecture comme dans le cas des automates cheminants et les deux autres cas correspondent respectivement au placement et à la levée d'un jeton dans l'arbre. On remarque que pour $0 \leq i \leq k$, l'automate A ne peut effectuer une transition de la forme $(Q \times \Sigma \times \text{TYPE} \times \{i\} \times 2^{[k]}) \times (Q \times \text{ACTION})$ qu'à partir d'une i-configuration.

Dans *le modèle faible* d'automates à jetons, on impose une restriction supplémentaire sur la levée d'un jeton. Une transition qui lève le dernier jeton qui a été posé, c'est-à-dire une transition de $(Q \times \Sigma \times \text{TYPE} \times \{i\} \times 2^{[k]}) \times (Q \times \{\text{leve}\})$ avec

$0 \leq i \leq k$ ne peut être appliquée à une configuration (q, u, f) que si $f(i+1) = u$. Les autres transitions sont appliquées comme dans le cas du modèle fort. Ainsi les transitions qui lèvent un jeton sont de la forme $(q, a, \theta, i, J), (q', \text{leve})$ avec $(i+1) \in J$. On en déduit que si A est un automate à k jetons du modèle faible, pour $0 \leq i \leq k$, une i-configuration (q, u, f) a pour successeur une configuration (u', q', f') ce que nous notons $(u, q, f) \xrightarrow{A,t} (q', u', f')$ dans les 3 cas suivants :

1. les affectations de jetons f et f' sont identiques et il existe une transition $((q, a, \theta, i, f^{-1}(u)), (q', D))$ où $D \in \{\uparrow, \circlearrowleft, \swarrow, \searrow\}$ tel que a est l'étiquette du nœud u, θ est le type de u et $k = \circlearrowleft$ si $u = u'$, $k = \uparrow$ si u' est le père de u, $k = \swarrow$ si u' est le fils gauche de u et $k = \searrow$ si u' est le fils droit de u.
2. les positions u et u' sont identiques, les affectations de jetons sont telles que $f' = f \cup \{(i, u)\}$ et il existe une transition $((q, a, \theta, i, f^{-1}(u)), (q', \text{pose}))$ où a et θ sont respectivement l'étiquette et le type du nœud u.
3. les positions u et u' sont identiques, l'entier i est strictement inférieur à k, f' est une affectation des jetons de $\{i+2, \cdots, n\}$ telle que $f(j) = f'(j)$ pour tout $i+2 \leq j \leq n$, $f(i+1) = u$, $u' = u$ et il existe une transition $((q, a, \theta, i, f^{-1}(u)), (q', \text{leve}))$ telle que a et θ sont respectivement l'étiquette et le type de la position u.

Les deux premiers cas sont les mêmes que pour le modèle faible. Le modèle fort et le modèle faible d'automates ne diffèrent que pour la levée d'un jeton.

Notation 2.3 *Etant donnés un arbre t et un automate à jetons A, la notation $(u, q, f) \xrightarrow{A,*} (u', q', f')$ signifie qu'il existe une exécution de A dans t de la configuration (u, q, f) à la configuration (u', q', f').*

Une exécution d'un automate d'arbres à k jetons A sur un arbre t est acceptante si elle part d'une k-configuration de la forme (q_i, ϵ) où q_i est un état initial et si elle termine dans une k-configuration (q_f, ϵ) où q_f est un état final. L'arbre t est alors accepté par A.

Pour chacun des deux modèles d'automates à jetons, on peut définir une variante déterministe.

Définition 2.24 *Un automate à k jetons déterministe sur les arbres ou un* DPTA_k *A est un automate à k jetons $(Q, \Sigma, I, F, \delta)$ tel que I est un singleton et δ est une fonction partiellement définie de $Q \times \Sigma \times \text{TYPE} \times \{0, \cdots, k\} \times 2^{[k]}$ à valeurs dans $Q \times \text{ACTION}$.*

Remarque 2.17 Nous avons choisi par convention qu'un automate accepte à la racine avec tous les jetons levés. Cette convention n'a pas d'influence sur le pouvoir d'expression des automates à jetons. En effet, quand un automate à jetons du modèle faible ou du modèle fort choisit d'accepter un arbre il peut facilement lever tous les jetons qui sont posés dans l'arbre et remonter à la racine dans un état final.

Comme nous l'avons signalé dans la remarque 2.16, un automate à k jetons avec $k > 0$ se comporte comme un automate à $(k-1)$ jetons à partir du moment où il pose le jeton k jusqu'à ce qu'il le lève. Dans certaines preuves, on décomposera

une exécution d'un automate à k jetons en sous-suites de k-configuration et en sous-exécutions d'automates à $(k-1)$ jetons que nous appelons $(k-1)$-exécutions.

Définition 2.25 *Soit A un automate à k jetons et $0 \leq i \leq k$. Une i-exécution de A sur un mot est une exécution partant d'une i-configuration (x, q, f) et terminant dans une i-configuration (x', q', f) durant laquelle l'automate ne lève pas le jeton $i+1$.*

Nous décrivons maintenant à titre d'exemple le comportement d'un automate à 1 jeton du modèle faible sur les arbres binaires et celui d'un automate à 2 jetons du modèle fort sur les mots.

Exemple 2.18 Etant donné un arbre binaire t, on appelle *chemin le plus à droite* de t la suite de nœuds qui va de la racine à la feuille la plus à droite et qui est l'ensemble de tous les nœuds du langage 2^* ; de même *le chemin le plus à gauche* de t est l'ensemble de tous les nœuds du langage 1^* qui forme le chemin de la racine à la feuille la plus à gauche. On considère maintenant L_7 le langage des arbres sur l'alphabet $\{a, b\}$ qui vérifient la propriété suivante : chaque nœud du chemin le plus à droite étiqueté par b a un sous-arbre gauche qui contient au moins une feuille étiquetée par a. Ainsi l'arbre de la figure 2.12 appartient à ce langage car les sous-arbres enracinés aux nœuds 1, 21 et 2221 contiennent tous une feuille étiquetée par a.

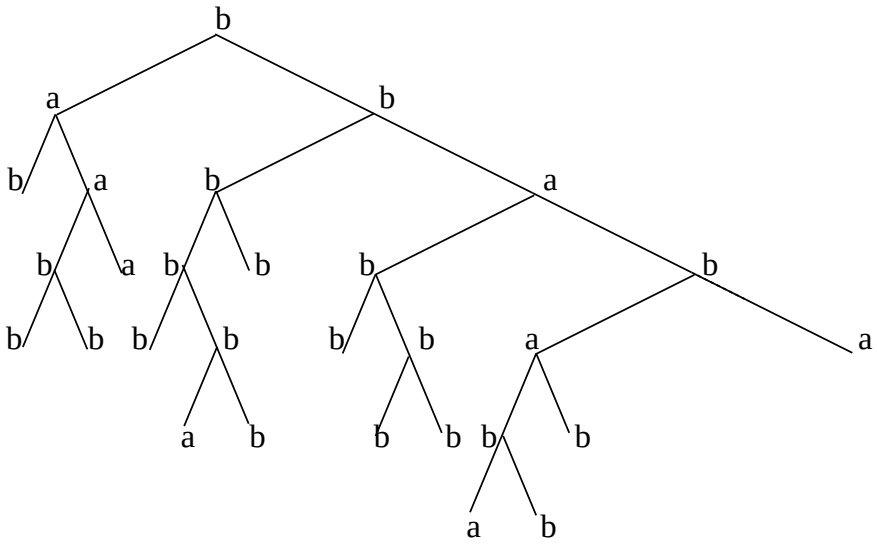

FIG. 2.12 – Un arbre de L_7

Nous définissons maintenant $A_7 = (Q, \Sigma, I, F, \delta)$ un automate déterministe à 1 jeton du modèle faible qui reconnaît le langage L_7. On pose $Q = \{q_I, q_{d1}, q_{d2}, q_m, q_1, q_F\}$, q_I est l'état initial et q_F l'état final. L'automate dans l'état q_I est toujours sur le chemin le plus à droite et si le nœud courant u est un nœud interne étiqueté par la lettre b l'automate pose son jeton, descend au fils gauche et passe dans l'état q_{d1} pour vérifier que le sous-arbre gauche de u contient une feuille étiquetée par a. Si A est dans l'état initial q_I sur un nœud interne étiqueté par a, A ne change pas d'état et descend au fils droit pour continuer de visiter le chemin le plus à droite de

l'arbre. Si l'automate atteint une feuille étiquetée par a dans l'état initial, il a bien vérifié que chaque nœud du chemin le plus à droite étiqueté par b a un sous-arbre gauche qui contient une feuille étiquetée par la lettre a et l'automate passe alors dans l'état final q_F à partir duquel il remonte jusqu'à la racine pour accepter. Les états q_{d1},q_{d2} et q_m permettent à A, une fois qu'il a posé un jeton sur un nœud u du chemin le plus à droite, d'effectuer un parcours en profondeur du sous-arbre gauche de u comme dans l'exemple 2.16. Si au cours de ce parcours, une feuille étiquetée par a est visitée, l'automate passe dans l'état q_1, remonte jusqu'au nœud où le jeton est placé, lève ce jeton puis descend au fils droit et repasse dans l'état initial pour visiter les autres nœuds du chemin le plus à droite.

2.4.2 Automates à jetons sur les mots

Les automates à jetons sont définis de manière similaire sur les mots.

Définition 2.26 *Un automate à k jetons sur les mots , ou un* PA_k*, est un quintuplet* $(Q, \Sigma, I, F, \delta)$ *où Q est un ensemble fini d'états, Σ est un alphabet, $I \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états initiaux, $F \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états finaux et δ est un sous-ensemble de*

$$(Q \times \Sigma \times \mathrm{TYPE}_M \times [0, \cdots, k] \times 2^{[k]}) \times (Q \times \mathrm{ACTION})$$

appelé ensemble des transitions où ACTION *est l'ensemble fini* $\{-1, 0, +1, \mathrm{pose}, \mathrm{leve}\}$.

Les définitions de configurations et de calculs d'un automate d'arbres à jetons s'adaptent sans difficultés sur les mots. Nous distinguons toujours le modèle fort et le modèle faible d'automates à jetons. Nous considérons qu'un automate à jetons accepte à la fin du mot avec tous ses jetons levés mais cette convention n'influence pas le pouvoir d'expression des automates à jetons.

Exemple 2.19 Soit Σ un alphabet, on considère l'ensemble des mots w pour lesquels il existe un mot v de trois lettres et des mots u_1, u_2, u_3 et u_4 tels que $w = u_1 v u_2 v u_3 v u_4$. Ce langage est régulier, il correspond à l'expression rationnelle $\bigcup_{v \in \Sigma^3} \Sigma^* v \Sigma^* v \Sigma^* v \Sigma^*$. Un automate non déterministe peut reconnaître ce langage en devinant le mot v puis en parcourant w afin de chercher trois occurrences distinctes de v. Nous décrivons dans cet exemple le comportement d'un automate à deux jetons du modèle fort A_8 qui n'a pas besoin de retenir dans son état un mot de trois lettres. Avec le non-déterminisme, l'automate commence par choisir une position x_2 du mot pour poser le jeton 2 et se déplace d'au moins deux positions vers la droite pour poser le jeton 1 sur une lettre x'_2. A partir de là, A_8 vérifie que les lettres qui étiquettent x_2 et x'_2 sont identiques et effectue la même vérification pour x_1 et x'_1 les positions qui sont respectivement les prédécesseurs de x_2 et de x'_2 et enfin pour x_3 et x'_3 les successeurs respectifs de u et de u'. Une fois que toutes ses vérifications sont faites, si les facteurs $x_1x_2x_3$ et $x'_1x'_2x'_3$ sont étiquetés par le même mot, l'automate retourne au jeton 2 lève les deux jetons du mot, le jeton 2 est alors levé de manière forte à partir de x'_2, puis A_8 repose le jeton 2 en $x'2$ et se déplace d'au moins deux positions vers la droite pour poser le jeton 1 sur une nouvelle position x''_2. L'automate va finalement verifier si les facteurs $x''_1x''_2x''_3$ et $x'_1x'_2x'_3$ sont étiquetés par le même mot

où x_1'' et x_3'' sont respectivement le prédécesseur et le successeur de x_2''. On représente dans la figure ci-dessous les différentes étapes du placement des deux jetons dans une exécution acceptante de A_8 sur un mot de $\Sigma^* abc \Sigma^* abc \Sigma^* abc \Sigma^*$. Pour chaque étape représentée, la flèche verticale représente la tête de lecture de l'automate.

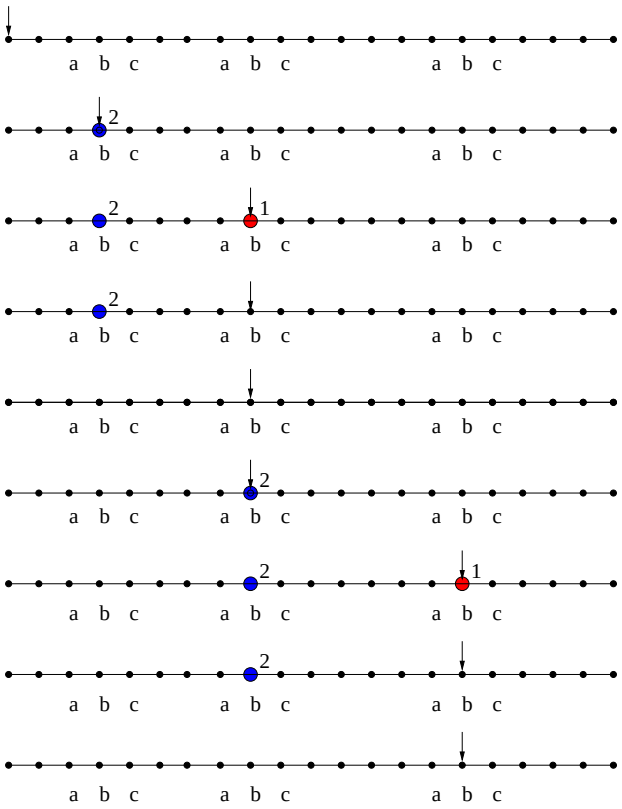

FIG. 2.13 – Différentes étapes du placement des jetons dans une exécution de A_8.

Cette figure montre comment le modèle fort permet de "construire pas à pas" un chemin avec deux jetons. Les deux jetons sont représentés par des petits disques numérotés.

Les deux propositions ci-dessous se prouvent de la même manière que les propositions 2.1 et 2.2.

Proposition 2.3 *Soit $k \in \mathbb{N}$, la classe des langages reconnus par un automate à k jetons du modèle fort (respectivement du modèle faible) est fermée par union et intersection.*

Proposition 2.4 *Soit $k \in \mathbb{N}$, la classe des langages reconnus par un automate déterministe à k jetons du modèle fort (respectivement du modèle faible) est fermée par intersection.*

2.5 Automates et logique MSO

Nous avons choisi de présenter les automates et la logique MSO dans le même chapitre et nous établissons maintenant des liens entre la logique MSO sur les mots,

respectivement sur les arbres et les automates que nous avons définis sur les mots, respectivement sur les arbres. Nous démontrons d'abord que les automates unidirectionnels correspondent à la logique MSO sur les mots et les automates d'arbres bottom-up à la logique MSO sur les arbres. Nous prouvons ensuite que tous les langages de mots reconnus par des automates à jetons sur les mots sont réguliers. Cette démonstration s'étend sans difficulté au cas des arbres. Dans le cas des mots où les automates unidirectionnels sont des cas particuliers d'automates à jetons, nous pouvons déduire que la classe des langages reconnus par des automates à jetons est l'ensemble des langages réguliers. Dans le cas des arbres, nous verrons dans le chapitre que les automates d'arbres à jetons ne reconnaissent pas tous les langages d'arbres réguliers.

2.5.1 Langages réguliers

Les formules de logique sur les mots sans variables libres permettent de définir des langages. On peut donc parler de pouvoir d'expression d'une logique tout comme on parle de pouvoir d'expression pour une classe d'automates. Par exemple, la formule de l'exemple 2.8 définit le langage reconnu par l'automate A_2 de l'exemple 2.11.

Définition 2.27 *Soit ϕ une formule logique close sur les arbres (respectivement sur les mots). Le langage défini par ϕ est l'ensemble des arbres (respectivement des mots) qui correspondent à des modèles satisfaisant ϕ.*

Par la suite, lorsque nous considérons une formule logique qui définit un langage, cette formule est nécessairement close.

Les langages réguliers correspondent à la logique MSO sur les mots et sur les arbres. Nous prouvons ce résultat d'abord sur les mots et ensuite sur les arbres binaires. La preuve se généralise sans difficulté au cas des arbres de rang borné.

Théorème 2.6 *Un langage de mots est reconnu par un* 1NFA *si et seulement s'il est défini par une formule* MSO *et les transformations d'un* 1NFA *en une formule* MSO *et d'une formule* MSO *en un* 1NFA *sont effectives*

Preuve. Considérons d'abord $A = (Q, \Sigma, I, F, \delta)$ un 1NFA. Nous pouvons supposer que $Q = [n]$. Nous devons trouver une formule monadique du second ordre qui est vérifiée par un modèle représentant un mot si et seulement si A accepte ce mot. Etant donné un mot $w = w_1 \cdots w_m$, la formule va donc vérifier l'existence d'un calcul acceptant de A sur w. Un tel calcul est de la forme $q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \cdots q_{m-1} \xrightarrow{w_m} q_m$ avec $q_0 \in I$, $q_n \in F$ et $(q_i, w_i, q_{i+1}) \in \delta$ pour $i < m$. Nous codons une telle suite d'états par un n-uplet $(X_1, \cdots, X_n)$ de sous-ensembles de $\{1, \cdots, m\}$ deux à deux disjoints tel que X_i est l'ensemble des positions x de w pour lesquelles l'automate est dans l'état i juste avant d'avoir lu la lettre à la position x. On considère $\mathrm{pre}(x)$ et $\mathrm{der}(x)$ les formules du premier ordre définies dans l'exemple 2.4 qui vérifient respectivement si la position correspondant à la variable x est la première et la dernière position du mot. Il est alors facile de prouver qu'un mot non-vide w est accepté par A si et seulement s'il satisfait la formule suivante :

$$\exists X_0 \cdots \exists X_n \ (\bigwedge_{i \neq j} \forall x \neg(x \in X_i \wedge x \in X_j) \wedge \forall x(\mathrm{pre}(x) \Rightarrow \bigvee_{i \in I} x \in X_i) \qquad (a)$$

$$\wedge \forall x \forall y(S(x,y) \Rightarrow \bigvee_{(i,a,j) \in \delta} (x \in X_i \wedge P_a(x) \wedge y \in X_j)) \qquad (b)$$

$$\wedge \forall x(\mathrm{der}(x) \Rightarrow \bigvee_{\exists j \in F | (i,a,j) \in \delta} (x \in X_i \wedge P_a(x)))) \qquad (c)$$

En effet, la partie (a) de la formule ci-dessus signifie que chaque position appartient à au plus un des k ensembles qui correspondent aux états et que la première position du mot appartient à un ensemble associé à un état initial. La partie (b) de la formule vérifie ensuite que deux positions successives sont dans des ensembles associés à des états qui correspondent à des configurations successives. Enfin, la partie (c) indique que la dernière position appartient à un ensemble correspondant à un état permettant d'atteindre un état final. La conjonction des parties (a), (b) et (c) traduit donc bien l'existence d'un calcul acceptant de l'automate.

Le mot vide satisfait la formule ci-dessus quel que soit l'automate A. Aussi, si A n'accepte pas le mot vide on rajoute la clause $\exists x \ (x = x)$.

Pour montrer l'implication inverse du théorème, nous utilisons la logique MSO_{ens} qui est définie en 2.1.4 et qui est équivalente à la logique MSO. Nous voulons donc construire par induction, pour chaque formule ϕ de MSO_{ens} à k variables libres, un automate qui accepte les mots $w \in (A \times \{0,1\}^k)^*$ satisfaisant la formule ϕ. Rappelons qu'une formule de MSO_{ens} est une formule du second ordre et sans quantificateur du premier ordre avec les prédicats atomiques $Sing(X_i)$, $Suc(X_i, X_j)$, $X_i \subseteq X_j$ et $X_i \subseteq P_a$. Il est facile de définir un automate pour reconnaître les langages définis par les formules de bases : $\mathrm{Sing}(X_i)$, $\mathrm{Suc}(X_i, X_j)$, $X_i \subseteq X_j$ et $X_i \subseteq P_a$. Par exemple, l'automate correspondant à la formule $X_i \subseteq X_j$ vérifie simplement que lorsque la i-ème composante binaire d'une position a pour valeur 1, la j-ème composante binaire de cette position a la même valeur. Pour l'induction il suffit de considérer les deux opérateurs $\neg$ et $\vee$ et la quantification existentielle sur des ensembles qui permettent d'exprimer les autres opérateurs et la quantification universelle sur des ensembles. Comme la classe des langages réguliers est fermée par complément et par union il ne nous reste plus qu'à considérer la projection pour la quantification existentielle. Supposons donc que le langage sur l'alphabet $\Sigma \times \{0,1\}^k$ défini par la formule $\phi(X_1, \cdots, X_k)$ soit reconnu par l'automate A. L'automate sur l'alphabet $\Sigma \times \{0,1\}^{k-1}$ qui correspond à la formule $\exists X_i \ \phi(X_1, \cdots, X_k)$ devine avec le non-déterminisme pour chaque position dans le mot un bit qui correspondrait à la i-ème composante binaire d'un mot sur $\Sigma \times \{0,1\}^n$ et simule A sur ce mot. □

Nous montrons maintenant que les langages réguliers d'arbres correspondent également à la logique MSO. Prouvons d'abord que la classe des langages d'arbres réguliers est fermée pour les opérateurs booléens union, intersection et complément.

Théorème 2.7 *Soient L_1 et L_2 deux langages réguliers sur Σ. Les langage $L_1 \cup L_2$, $L_1 \cap L_2$ et ${L_1}^c$ sont réguliers.*

Preuve. Soit $B_1 = (Q_1, \Sigma, q_1, F_1, \delta_1)$ et $B_2 = (Q_2, \Sigma, q_2, F_2, \delta_2)$ des automates d'arbres qui reconnaissent respectivement L_1 et L_2. On peut supposer que Q_1 et Q_2

sont disjoints.

Pour reconnaître le complément de L_1, d'après le théorème 2.4, on peut déterminiser B_1. Soit donc B_1' un automate déterministe qui reconnaît L_1. Il suffit maintenant d'intervertir l'ensemble des états finaux et des états non finaux de B_1' et l'automate ainsi construit accepte le complément de L_1.

Pour reconnaître l'union de L_1 et L_2, nous construisons un nouvel automate dont l'ensemble d'états sera la réunion des ensembles d'états des deux automates. Pour reconnaître l'intersection, nous construisons l'automate qu'on appelle produit de B_1 et B_2.

Posons $Q_\cup = Q_1 \cup Q_2 \cup \{q\}$ où q est un nouvel état qui n'appartient ni à Q_1 ni à Q_2 et $Q_\cap = Q_1 \times Q_2$. On note ensuite

- $\delta_\cup = \delta_1 \cup \delta_2 \cup \{((q,q,\sigma),q_3) | ((q_1,q_1,\sigma),q_3) \in \delta_1 \text{ ou } ((q_2,q_2,\sigma),q_3) \in \delta_2\}$
- $\delta_\cap = \{(((p_1,p_2),(p_1',p_2'),\sigma),(p_3,p_3')) | (p_1,p_1',\sigma),p_3) \in \delta_1 \text{ et } (p_2,p_2',\sigma),p_3') \in \delta_2\}$

Nous pouvons alors vérifier facilement que les automate $(Q_\cup, \Sigma, q, F_1 \cup F_2, \delta_\cup)$ et $(Q_\cap, \Sigma, (q_1,q_2), F_1 \times F_2, \delta_\cap)$ reconnaissent respectivement $L_1 \cup L_2$ et $L_1 \cap L_2$. □

Remarque 2.18 Pour montrer que la classe des langages réguliers est fermée par intersection, on pouvait aussi remarquer que $L_1 \cap L_2 = ({L_1}^c \cup L_2^c)^c$.

On peut maintenant montrer que les automates d'arbres ont le même pouvoir d'expression que la logique MSO de la même manière que nous avons montré que cette logique correspondait aux 1NFA sur les mots.

Théorème 2.8 *Un langage d'arbres est reconnu par un* BU *si et seulement s'il est défini par une formule* MSO *et les transformations d'un* BU *en une formule* MSO *et d'une formule* MSO *en un* BU *sont effectives*

Preuve. La preuve ci-dessous pour les langages réguliers d'arbres binaires s'adapte sans difficulté pour les langages réguliers d'arbres de rang borné. Nous considérons donc $B = (Q, \Sigma, q_0, F, \delta)$ un BU avec $Q = [n]$ et $q_0 = 1$. Nous allons construire une formule vérifiant l'existence d'un calcul acceptant de B sur un arbre t donné. Nous codons un calcul de B sur t par un k-uplet $(X_1, \cdots, X_k)$ de sous-ensembles des nœuds de t deux à deux disjoints tel que X_i est l'ensemble des nœuds de t auxquels l'automate attribue l'état i dans le calcul. Il est facile de prouver qu'un arbre non-vide t est accepté par B si et seulement s'il satisfait la formule suivante dans laquelle $fe(x)$ et $ra(x)$ sont les formules du premier ordre définies dans l'exemple 2.5 qui vérifient respectivement si le nœud correspondant à la variable x est une feuille et la racine de l'arbre :

$$\exists X_0 \cdots X_n \; (\bigwedge_{i \neq j} \forall x \neg (x \in X_i \wedge x \in X_j) \wedge \forall x (\mathrm{fe}(x) \Rightarrow \bigvee_{\substack{\sigma \in \Sigma \\ i \in Q \,|\, ((1,1,\sigma),i) \in \delta}} P_\sigma(x) \wedge x \in X_i) \qquad (a)$$

$$\wedge \forall xyz (S_1(x,y) \wedge S_2(x,z) \Rightarrow \bigvee_{((i,j,a),k) \in \delta} y \in X_i \wedge P_a(x) \wedge z \in X_j \wedge x \in X_k) \qquad (b)$$

$$\wedge (\mathrm{ra}(x) \Rightarrow \bigvee_{\exists j \in F} (x \in X_i))) \qquad (c)$$

La formule ci-dessous a en effet été construite comme la formule de la preuve du théorème 2.8.

Le mot vide satisfait la formule ci-dessus quel que soit l'automate A. Aussi, si A n'accepte pas le mot vide on rajoute la clause $\exists x\ (x = x)$

Pour montrer l'implication inverse du théorème, nous utilisons le théorème 2.7 et nous procédons exactement comme dans le cas des mots. □

2.5.2 Automates à jetons et logique MSO

Pour terminer ce chapitre, nous montrons maintenant que les langages reconnus par un automate à jetons du modèle fort sont des langages réguliers. Attention, dans le cas des arbres la réciproque est fausse : nous montrerons en effet dans le chapitre 6 qu'il existe un langage d'arbres régulier qui n'est pas reconnu par un automate d'arbres à jetons. Nous nous intéressons d'abord au cas des automates sur les mots avec 0 jeton : les 2NFA.

Théorème 2.9 *Un langage de mots est reconnu par un* 2NFA *si et seulement s'il est défini par une formule* MSO. *Les transformations de l'automate en formule et de la formule en automate sont effectives.*

Preuve. D'après le théorème 2.8, pour chaque formule MSO, on peut construire un automate 1NFA qui reconnaît le langage défini par cette formule. De plus, d'après la remarque 2.11, un 1NFA est aussi un 2NFA. Il ne reste donc plus qu'à construire pour tout 2NFA une formule MSO qui définit le langage reconnu par l'automate. Considérons donc $A = (Q, \Sigma, I, F, \delta)$ un 2NFA. Nous pouvons encore supposer que $Q = [n]$ avec $n > 0$.

Soit un mot $w = w_1 \cdots w_m$. La formule que nous construisons doit vérifier l'existence d'un calcul acceptant de A. Nous codons l'ensemble des calculs de A sur w par le n-uplet $(X_1, \cdots, X_n)$ de sous-ensembles de $\{1, \cdots, m\}$ tels que pour chaque état j, X_j est l'ensemble des positions x du mot w pour lesquelles il existe une exécution d'une configuration initiale à la configuration (j, x).

Nous considérons maintenant $\phi_I(X_1, \cdots, X_n)$ la formule suivante signifiant que les ensembles associés à un état initial contiennent la première position du mot : $\forall x\ \mathrm{pre}(x) \Rightarrow \bigwedge_{i \in I}\ X_i(x)$

Nous notons ensuite $\phi_\delta(X_1, \cdots X_n)$ la formule ci dessous :

$$\bigwedge_{(i,a,j,\theta,0)\in\delta} \forall x(\theta(x) \wedge P_a(x) \wedge x \in X_i) \Rightarrow x \in X_j$$

$$\wedge \bigwedge_{(i,a,j,\theta,+1)\in\delta} \forall x \forall y(\theta(x) \wedge P_a(x) \wedge x \in X_i \wedge S(x,y)) \Rightarrow y \in X_j$$

$$\wedge \bigwedge_{(i,a,j,\theta,-1)\in\delta} \forall x \forall y(\theta(x) \wedge P_a(x) \wedge x \in X_i \wedge S(y,x)) \Rightarrow y \in X_j$$

La formule $\phi_\delta(X_1, \cdots, X_n)$ signifie que deux positions successives x et y appartiennent à des ensembles qui correspondent à des configurations successives. Les

trois cas correspondent à la position de x par rapport à y : on peut avoir $x = y$, $x = y - 1$ ou bien $x = y + 1$.

On pose maintenant $\phi_{I,\delta}(X_1, \cdots, X_n)$ l'intersection des formules ϕ_δ et ϕ_I.

$$\phi_{I,\delta}(X_1, \cdots, X_n) = \phi_\delta(X_1, \cdots, X_n) \wedge \phi_I(X_1 \cdots, X_n)$$

Cette formule est satisfaite si et seulement si pour chaque état j, l'ensemble associé à la variable X_j contient au moins toutes les positions x pour lesquelles la configuration (j, x) est accessible à partir d'une configuration initiale. Aussi, le plus petit n-uplet de sous-ensembles satisfaisant la formule $\phi_{I,\delta}$ code l'ensemble des exécutions de A.

Nous désignons enfin par $\phi_F(X_1, \cdots, X_n)$ la formule suivante :

$$\bigvee_{f \in F} \exists x \ \mathrm{der}(x) \wedge x \in X_f$$

La formule $\phi_F(X_1, \cdots, X_n)$ signifie que la dernière position du mot appartient à un ensemble correspondant à un état final. Un mot non-vide est accepté si le n-uplet qui code les exécutions de A satisfait la formule ϕ_F. On peut alors vérifier qu'un mot non-vide w est accepté par A si et seulement s'il vérifie la formule suivante :

$$\forall X_1 \cdots X_n \ (\phi_{I,\delta}(X_1, \cdots, X_n) \Rightarrow \phi_F(X_1, \cdots, X_n))$$

□

En adaptant au cas des arbres la construction d'une formule MSO à partir d'un automate bidirectionnel dans la preuve ci-dessus, on prouve sans difficulté que le langage reconnu par un automate cheminant est régulier : il suffit d'utiliser les formules du premier ordre qui permettent de définir le type d'un nœud et de tenir compte de tous les mouvements possibles d'un TWA. Nous prouvons maintenant que les automates à jetons sur les mots ont le même pouvoir d'expression que les automates unidirectionnels à l'aide de la proposition ci-dessous.

Proposition 2.5 *Pour tout $k \in \mathbb{N}$, un langage de mots est reconnu par un automate à k jetons du modèle fort si et seulement s'il est défini par une formule* MSO. *Les transformations de l'automate en formule et de la formule en automate sont effectives.*

Notation 2.4 *Dans la preuve ci-dessous, nous définirons des formules avec des variables libres $x_k, \cdots, x_{i+1}$ correspondant à la position des $(k - i)$ premiers jetons posés. Afin d'améliorer la lisibilité des formules nous étendons la notation vectorielle utilisée pour des suites de positions à des suites de variables libres. Ainsi nous utilisons la notion $\vec{x}_i$ pour désigner la suite des $(k-i+1)$ variables libres $x_k, \cdots, x_i$.*

Notation 2.5 *Pour chaque transition η d'un automate, on note* $\mathrm{ent}(\eta)$ *l'état à partir duquel η peut être appliquée et* $\mathrm{sor}(\eta)$ *l'état atteint lorsque l'automate effectue η.*

Notation 2.6 *Soient $A = (Q, \Sigma, I, F, \delta)$ un* PA_k, *$i \in [k]$ et $\alpha \in \{+0, +1, -1, \mathrm{pose}, \mathrm{leve}\}$, on note $\delta_{\alpha,i}$ l'ensemble des transitions de $(Q \times \Sigma \times \mathrm{TYPE}_M \times \{i\} \times 2^{[k]}) \times (Q \times \{\alpha\})$ et on note δ_i les transitions de $\delta_{+0,i} \cup \delta_{+1,i} \cup \delta_{-1,i} \cup \delta_{\mathrm{pose},i} \cup \delta_{\mathrm{leve},i}$.*

Notation 2.7 *Considérons $0 \leq i \leq k$ et η une transition de δ_i . Il existe une formule du premier ordre ϕ_η à $(k-i+1)$ variables libres $(x, \vec{x}_{i+1})$ qui est satisfaite par les $(k-i+1)$-uplets $(u, \vec{u}_{i+1})$ tels que A peut effectuer η à partir d'une configuration de $(q_1, u, \vec{u}_{i+1})$. En effet, il est facile de vérifier avec une formule du premier ordre le type et l'étiquette de la position u correspondant à la variable x ainsi que l'égalité entre la position u et les positions de $\vec{u}_{i+1}$ qui correspondent aux variables $\vec{x}_{i+1}$. Ainsi pour une transition η de la forme $(Q \times a \times \{\theta\} \times \{i\} \times \{J\}) \times (Q \times \text{ACTION})$ avec $a \in \Sigma$, $\theta \in \text{TYPE}_M$, $0 \leq i \leq k$ et $J \in 2^{[k]}$, la formule $\phi_\eta(x, \vec{x}_{i+1})$ est la formule $\bigwedge_{j \in J}(x = x_j) \wedge \theta(x) \wedge P_a(x)$.*

Preuve. Soit k un entier positif. D'après le théorème 2.8, il suffit de construire pour tout automate à k jetons sur les mots une formule MSO satisfaite par les mots reconnus par cet automate. Considérons donc $A = (Q, \Sigma, I, F, \delta)$ un PTA_k sur les mots. Nous montrons d'abord par induction sur i que pour tout entier $0 \leq i \leq k$ et pour tout couple d'états $(p, q) \in Q \times Q$, il existe une formule MSO à $(k-i+2)$ variables libres $\phi^i_{p,q}$ satisfaite par les $(k-i+2)$-uplets $(u, v, \vec{u}_{i+1})$ pour lesquels il existe une i-exécution de A de la configuration $(p, u, x_n, \vec{u}_{i+1})$ à la configuration $(q, v, \vec{u}_{i+1})$

Montrons d'abord le cas $i = 0$. Un automate à jetons se comporte comme un automate bidirectionnel au cours d'une 0-exécution.

Nous reprenons et adaptons donc la preuve du théorème 2.9. Nous posons ainsi $n = |Q|$ et nous notons $\phi^0_\delta(X_1, \cdots, X_n, \vec{x}_{i+1})$ la formule suivante qui tient maintenant compte de la présence des k jetons posés dans le mot :

$$\bigwedge_{\eta \in \delta_{+0,0}} \forall x(\phi_\eta(x) \wedge x \in X_{\text{ent}(\eta)}) \Rightarrow x \in X_{\text{sor}(\eta)}$$
$$\wedge \bigwedge_{\eta \in \delta_{+1,0}} \forall x \forall y(\phi_\eta(x) \wedge x \in X_{\text{ent}(\eta)} \wedge S(x, y)) \Rightarrow y \in X_{\text{sor}(\eta)}$$
$$\wedge \bigwedge_{\eta \in \delta_{-1,0}} \forall x \forall y(\phi_\eta(x) \wedge x \in X_{\text{ent}(\eta)} \wedge S(y, x)) \Rightarrow y \in X_{\text{sor}(\eta)}$$

Nous voulons cette fois-ci donner une formule qui vérifie l'existence d'une exécution partant d'une 0-configuration dont l'état de contrôle est un certain état p et non d'une configuration initiale. Nous considérons ainsi la formule $\phi_{p,\delta}$ définie ainsi :

$$\phi^0_{p,\delta}(x, X_1, \cdots, X_n, \vec{x}_1) = \phi^0_\delta(X_1, \cdots, X_n, \vec{x}_1) \wedge x \in X_p$$

Cette formule est vérifiée par les $(n+k+1)$-uplets $(u, E_1, \cdots, E_n, \vec{u}_1)$ tels que pour chaque $j \in Q$, E_j contient au moins toutes les positions v du mot w pour lesquelles il existe une exécution de A sur w de la configuration $(p, u, \vec{u}_1)$ à la configuration $(j, v, \vec{u}_1)$.

On note enfin $\phi^0_{p,q}(x, y, \vec{x}_1)$ la formule suivante :

$$\forall X_1, \cdots X_n \ (\phi^0_{p,\delta}(x, X_1, \cdots, X_n, \vec{x}_1) \Rightarrow y \in X_q)$$

On peut alors vérifier que pour tout $(k+2)$-uplet $(u, v, \vec{u}_1)$ de positions, il existe une 0-exécution de la configuration $(q, u, \vec{u}_1)$ à la configuration $(q, v, \vec{u}_1)$ si et seulement si $\phi_{p,q}$ est satisfaite par $(u, v, \vec{u}_1)$. Le cas $i = 0$ est donc prouvé.

Supposons maintenant que $i > 0$ et considérons à nouveau un couple d'états $(p, q) \in Q^2$. On pose $x_i = x$ afin d'utiliser la notation $\vec{x}_i$ pour $(x_k, \cdots, x_{i+1}, x)$

La formule $\phi^i_\delta(X_1, \cdots, X_n, \vec{x}_{i+1})$ est maintenant définie en tenant compte des i-exécutions où A pose et reprend le jeton i :

$$\bigwedge_{\substack{\eta\in\delta_{\mathrm{pose},i}\\ \eta'\in\delta_{\mathrm{leve},i}}} \forall x\forall y\ (\phi_\eta(x,\vec{x}_{i+1}) \wedge x\in X_{\mathrm{ent}(\eta)} \wedge \phi^{i-1}_{\mathrm{sor}(\eta),\mathrm{ent}(\eta')}(x,y,\vec{x}_i) \wedge \phi_{\eta'}(x,\vec{x}_i)) \Rightarrow y\in X_{\mathrm{sor}(\eta')}$$

$$\wedge \bigwedge_{\eta\in\delta_{+0,i}} \forall x(\phi_\eta(x) \wedge x \in X_{\mathrm{ent}(\eta)}) \Rightarrow x \in X_{\mathrm{sor}(\eta)}$$

$$\wedge \bigwedge_{\eta\in\delta_{+1,i}} \forall x\forall y(\phi_\eta(x) \wedge x \in X_{\mathrm{ent}(\eta)} \wedge S(x,y)) \Rightarrow y \in X_{\mathrm{sor}(\eta)}$$

$$\wedge \bigwedge_{\eta\in\delta_{-1,i}} \forall x\forall y(\phi_\eta(x) \wedge x \in X_{\mathrm{ent}(\eta)} \wedge S(y,x)) \Rightarrow y \in X_{\mathrm{sor}(\eta)}$$

On considère maintenant la formule $\phi^i_{p,\delta}$ définie ainsi :

$$\phi^i_{p,\delta}(x, X_1, \cdots, X_n, x_k, \cdots, x_i) = \phi^i_\delta(X_1, \cdots, X_n, x_k, \cdots, x_i) \wedge x \in X_p$$

Cette formule est vérifiée par les $(n + k + 1 - i)$-uplets $(u, E_1, \cdots, E_n, \vec{u}_{i+1})$ tels que pour chaque état j, E_j contient toutes les positions x du mot w pour lesquelles il existe une i-exécution de A sur w de la configuration $(p, u, \vec{u}_{i+1})$ à la configuration $(j, x, \vec{u}_{i+1})$.

Enfin, on note $\phi^i_{p,q}(x, y, \vec{x}_{i+1})$ la formule suivante :

$$\forall X_1, \cdots X_n(\phi^i_{p,\delta}(x, X_1, \cdots, X_n, \vec{x}_{i+1}) \Rightarrow y \in X_q)$$

On peut alors vérifier que pour tout $(k - i + 2)$-uplet $(u, v, \vec{u}_{i+1})$ de nœuds, il existe une i-exécution de la configuration $(p, u, \vec{u}_{i+1})$ à la configuration $(q, v, \vec{u}_{i+1})$ si et seulement si $\phi_{p,q}$ est satisfaite par $(u, v, \vec{u}_{i+1})$. Pour $i = k$, la formule ci-dessous est ainsi satisfaite par les mots reconnus par A :

$$\exists x, y \bigvee_{q\in I, q'\in F} \phi^k_{q,q'}(x, y) \wedge \mathrm{ra}(x) \wedge \mathrm{ra}(y)$$

□

Comme les langages réguliers correspondent à la logique MSO sur les mots , on en déduit le théorème ci-dessous.

Théorème 2.10 *Les automates à jetons du modèle fort sur les mots reconnaissent exactement la classe des langages réguliers et les automates à jetons du modèle faible sur les mots ont le même pouvoir d'expression.*

La transformation d'un automate en formule MSO décrite dans la preuve 2.5 s'étend sans difficulté aux automates d'arbres à jetons. On en déduit la proposition suivante :

Proposition 2.6 *Les langages d'arbres reconnus par un automate à jetons du modèle fort sont réguliers.*

Considérons $k > 0$. Etant donné un automate d'arbres A à k jetons du modèle fort, il est possible de construire un automate d'arbres bottom-up qui reconnaît le même langage. En effet, la preuve de la proposition 2.5 adaptée aux arbres permet d'obtenir une formule MSO qui définit le langage reconnu par A et en utilisant la preuve du théorème 2.8, nous pouvons ensuite construire à partir de cette formule MSO un automate bottom-up équivalent. Dans le chapitre 7, nous décrirons une transformation d'un automate d'arbres à k jetons du modèle fort en un automate bottom-up avec une meilleure complexité.

CHAPITRE

3

Complémenter un automate d'arbres à jetons déterministe

Sommaire

Nous prouvons dans ce chapitre que toutes les variantes déterministes des différents modèles d'automates d'arbres séquentiels que nous avons définis en 2.3.3 et 2.4 sont fermées par complémentation. Nous montrons ainsi comment complémenter un automate cheminant déterministe et plus généralement un automate à jetons déterministe du modèle faible sans augmenter le nombre de jetons. Nous expliquons à la fin de ce chapitre comment construire à partir d'un automate à jetons déterministe du modèle fort A, un automate déterministe du modèle fort avec trois fois plus de jetons qui reconnaît le complément du langage reconnu par A. Engelfriet et Hoogeboom ont donné dans [19] une caractérisation logique du modèle fort d'automates à jetons : les langages reconnus par un automate à jetons déterministe du modèle

fort sont les langages définis par la logique de clôture transitive déterministe unaire. Une conséquence de cette caractérisation est que la classe des langages reconnus par un automate à jetons déterministe du modèle fort est fermée par complément. Cette approche ne donne cependant pas de construction du complément d'un tel automate efficace en terme de jetons. Il est cependant possible d'éviter l'augmentation du nombre de jetons dans la complémentation d'un automate déterministe du modèle fort. Nous montrons en effet dans le chapitre 5 comment transformer un automate d'arbres à jetons déterministe du modèle fort en un automate à jetons déterministe du modèle faible équivalent avec le même nombre de jetons et ce dernier automate peut être complémenté toujours le même nombre de jetons, cependant la transformation d'un automate du modèle fort en un automate du modèle faible est très coûteuse : dès que le nombre k de jetons est supérieur à 1, le nombre d'états augmente selon une tour de $(k-1)$ exponentielles. Dans la construction du complément d'un automate à jetons du modèle fort pour laquelle le nombre de jetons est multipliée par 3, le nombre d'états augmente polynomialement.

Dans tout ce chapitre, nous fixons Σ un alphabet.

3.1 Un résultat non-trivial

3.1.1 Complémenter un automate sur les mots

Rappelons tout d'abord comment on complémente un automate séquentiel sur les mots avant de nous intéresser au cas des automates d'arbres. Nous avons vu en 2.5.1 que la classes des langages réguliers sur les mots (respectivement sur les arbres) correspond à la logique MSO et est ainsi fermée par complément. Pour complémenter un automate unidirectionnel déterministe ou un automate d'arbres BU déterministe, il suffit en effet de le compléter et d'intervertir les états finaux et les états non-finaux. On peut alors complémenter un automate unidirectionnel non-déterministe ou un automate d'arbres BU non-déterministe après l'avoir déterminisé. Nous avons prouvé en 2.5 que les deux modèles d'automates à jetons sur les mots ont le même pouvoir d'expresssion : ils reconnaissent les langages réguliers. Pour complémenter un tel automate, on peut donc se ramener au cas des automates déterministes unidirectionnels. Ainsi, pour tout automate à jetons du modèle fort sur les mots A_1, il existe un automate unidirectionnel déterministe A_2 qui reconnaît le complément du langage reconnu par A_1. Pour construire A_2, d'après le théorème 2.9, il est possible de calculer une formule MSO satisfaite par les mots du langage reconnu par A_1 et d'après le théorème 2.8, on peut transformer cette formule en un automate unidirectionnel qu'on sait complémenter. Cette construction est très coûteuse car la transformation d'une formule MSO en automate est non-élémentaire. Nous expliqueront ce que signifie le coût en complexité d'une construction et ce que signifie une transformation non-élémentaire au chapitre 7.1. Il existe des méthodes beaucoup plus efficaces pour complémenter un automate à jetons sur les mots. Par exemple, Vardi a prouvé dans [51] qu'il était possible de construire en temps exponentiel un automate unidirectionnel déterministe reconnaissant le complément d'un automate bidirectionnel donné.

Nous verrons en 7.3 comment transformer un automate à jetons du modèle fort

sur les mots A_1 en un automate unidirectionnel qui reconnaît le complément du langage reconnu par A_1 sans construire de formule logique MSO intermédiaire. Nous construirons également en 7.3 un automate BU qui reconnaît le complément du langage d'arbres reconnu par un automate à jetons du modèle fort donné. Cependant cela ne prouve en aucun cas que la classe des automates à jetons sur les arbres est fermée par complément. En effet, nous verrons en 6.3 que les automates à jetons ne reconnaissent pas tous les langages d'arbres réguliers.

3.1.2 La difficulté pour complémenter un DTWA

Considérons maintenant le cas des automates cheminants déterministes. Bojańczyk et Colcombet ont prouvé dans [6] qu'on ne pouvait pas toujours déterminiser un automate cheminant donné. Le problème de la complémentation d'un automate cheminant non-déterministe est encore un problème ouvert. Rappelons la condition d'acceptation d'un de ces automates : un état final doit être atteint à la racine. Il y a ainsi deux cas dans lesquels un automate cheminant rejette un arbre :

1. son exécution se termine avant que l'automate soit dans une configuration acceptante

   ou bien

2. l'automate boucle infiniment sans jamais atteindre une configuration acceptante.

Le deuxième cas est problématique pour complémenter un automate cheminant déterministe car une exécution infinie est rejetante quelque soit l'ensemble des états finaux. Ce cas rend également difficile la construction d'un automate cheminant déterministe reconnaissant l'union de deux langages définis par des automates cheminants déterministes. Nous montrons dans un premier temps qu'il est très simple de construire le complément d'un automate cheminant déterministe qui ne boucle pas. Définissons tout d'abord un automate cheminant déterministe à exécutions finies.

Définition 3.1 *Soit A un* DTWA. *On dit que A est à exécutions finies si toute exécution de A partant de la configuration initiale est finie.*

Dans le cas des mots, nous complétons un automate unidirectionnel déterministe afin de le complémenter. De la même façon, pour complémenter un automate cheminant déterministe à exécutions finies, nous nous ramenons à un automate qui termine toujours à la racine et avec un ensemble d'états finaux réduit à un seul état à partir duquel l'automate est bloqué.

Définition 3.2 *Soit A un* DTWA. *On dit que A est à unique configuration acceptante bloquante si A a une seule configuration acceptante à partir de laquelle il ne peut appliquer aucune transition. Soit A' un* DTWA *à exécutions finies. On dit que A' termine toujours à la racine si A' est à unique configuration acceptante bloquante et si toutes les exécutions de A' se terminent à la racine.*

Le lemme ci-dessous prouve qu'on peut considérer sans perte de généralité qu'un automate cheminant déterministe est à unique configuration acceptante bloquante.

Lemme 3.1 *Soit A un* DTWA*. On peut construire un* DTWA *à unique configuration acceptante bloquante qui reconnaît le même langage.*

Preuve. On pose $A = (Q, \Sigma, q_0, F, \delta)$. On introduit le nouvel état q'_f qui sera l'unique état final de l'automate qu'on construit dans cette preuve. On supprime ensuite de δ toutes les transitions qui peuvent être appliquées à partir d'une configuration acceptante de A et, pour chaque état final q_f de A et pour chaque lettre a de Σ, on rajoute les transitions $((q_f, a, (\text{in}, \text{ra})), (q'_f, \circlearrowleft))$ et $((q_f, a, (\text{fe}, \text{ra})), (q'_f, \circlearrowleft))$. On obtient ainsi un automate équivalent à unique configuration acceptante . □

Montrons maintenant que nous pouvons supposer sans perte de généralité qu'un automate cheminant déterministe à exécutions finies termine toujours à la racine.

Lemme 3.2 *Soit A un* DTWA *à exécutions finies, on peut construire un* DTWA *qui termine toujours à la racine et qui reconnaît le même langage.*

Preuve. Soit $A = (Q, \Sigma, q_0, F, \delta)$ un DTWA à exécutions finies. D'après la construction du lemme 3.1, on peut supposer que A est à unique configuration acceptante bloquante. On introduit alors un nouvel état q'_r permettant à l'automate de remonter jusqu'à la racine quand il est bloqué. On rajoute donc pour chaque type θ et pour chaque lettre a, les transitions $((q'_r, a, \theta), (q'_r, \uparrow))$. Enfin, pour chaque état q , chaque lettre a et chaque type de nœud θ différent de la racine tels que $\delta(q, a, \theta)$ n'est pas défini, on rajoute la transition $((q, a, \theta), (q'_r, \circlearrowleft))$. L'automate équivalent ainsi obtenu termine toujours à la racine. □

Nous pouvons maintenant complémenter un automate cheminant déterministe à exécutions finies en se ramenant à un automate qui termine toujours à la racine. Nous pouvons également construire un automate cheminant déterministe qui reconnaît l'union de deux langages reconnus par des automates cheminants déterministes à exécutions finies donnés.

Proposition 3.1 *Soit A et A' des* DTWA *à exécutions finies. On peut construire un* DTWA *qui reconnaît le complément du langage reconnu par A et un* DTWA *qui reconnaît l'union des langages reconnus par A et A'.*

Preuve. Soit A un DTWA à exécutions finies. On peut supposer d'après le lemme 3.2 que A termine toujours à la racine. On introduit un nouvel état q_f qui sera l'unique état final de l'automate qu'on construit. Pour chaque état q et chaque lettre a tels que q n'est pas final et $\delta(q, a, (\text{in}, \text{ra}))$ n'est pas défini, on rajoute la transition $((q, a, (\text{in}, \text{ra})), (q_f, \circlearrowleft))$. De même, pour chaque état q et chaque lettre a tels que $\delta(q, a, (\text{fe}, \text{ra}))$ n'est pas défini, on rajoute la transition $((q, a, (\text{fe}, \text{ra})), (q_f, \circlearrowleft))$. L'automate obtenu reconnaît alors le complément du langage reconnu par A. En effet, on peut vérifier que les exécutions rejetantes de A correspondent à des exécutions acceptantes de l'automate construit dans cette preuve et inversement.

On considère maintenant A' un autre DTWA à exécutions finies. Pour reconnaître l'union des langages reconnus par A et A', on construit un automate A'' qui se comporte de la manière suivante. L'automate A'' commence par simuler l'exécution de A et accepte si A accepte. Si A rejette, à partir de la racine, A'' simule

l'exécution de A' et accepte si A' accepte. □

La difficulté pour complémenter un automate cheminant déterministe vient donc des exécutions infinies. Dans l'exemple ci-dessous un DTWA qui peut rejeter un arbre en bouclant infiniment est construit. On illustre ainsi la difficulté de complémenter certains automates cheminants sur les arbres.

Exemple 3.1 On considère L_{aco} l'ensemble des arbres t sur l'alphabet $\{a, b\}$ qui vérifient la propriété suivante : chaque sous-arbre enraciné en un fils gauche d'un nœud du chemin le plus à droite a des feuilles qui, lexicalement ordonnées, (c'est-à-dire lues de gauches à droite) forment un mot de ab^* avec leurs étiquettes. D'après la définition du chemin le plus à droite, le fils gauche d'un nœud de ce chemin est un nœud de 2^*1. L'arbre représenté figure 3.1 appartient à ce langage car les mots *abbb*, *abb* et *ab* appartiennent au langage ab^*.

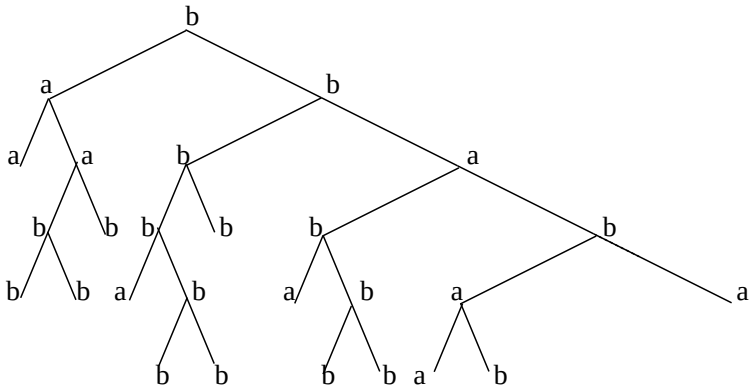

FIG. 3.1 – Un arbre de L_{aco}

Les arbres t_1 et t_2 représentés figure 3.2 appartiennent au complément de L_{aco} noté ${L_{aco}}^c$. En effet, la suite des feuilles lues de gauche à droite du sous-arbre de t_1 enraciné au nœud 21 et celle du sous-arbre de t_2 enraciné au nœud 221 constituent respectivement les mots *bbbb* et *aab* qui n'appartiennent pas au langage ab^*.

Il est très simple de construire un automate déterministe à un jeton pour le langage L_{aco} en s'inspirant de l'exemple 2.18. Nous définissons maintenant un DTWA A_{aco} qui reconnaît ce langage. La figure 3.3 décrit comment A_{aco} parcourt un arbre t du langage L_{aco}. L'automate descend le long du chemin le plus à droite comme le montrent les flèches 1, 6 et 11 de la figure 3.3. A chaque nœud de ce chemin, l'automate descend au fils gauche avec la flèche 2 ou la flèche 7 de la figure 3.3, puis il commence un parcours en profondeur de droite à gauche en vérifiant que toutes les feuilles qu'il visite sont étiquetées par b. Ce parcours en profondeur est représenté par les flèches 3 et 4 ou les flèches 8 et 9 de la figure. Dès que l'automate atteint une feuille étiquetée par a, il vérifie que cette feuille est un fils gauche puis il remonte dans l'arbre avec les flèches 5 et 10 de la figure 3.3 tant que le nœud courant a le type fils gauche. A_{aco} atteint alors un nœud qui est la racine ou un fils droit. Comme l'arbre t appartient au langage L_{aco}, ce nœud est sur le chemin le plus à droite. L'automate descend alors au fils droit et procède de même pour ce nouveau nœud du chemin le plus à droite. Si ce nœud est une feuille, A_{aco} a alors visité tous les

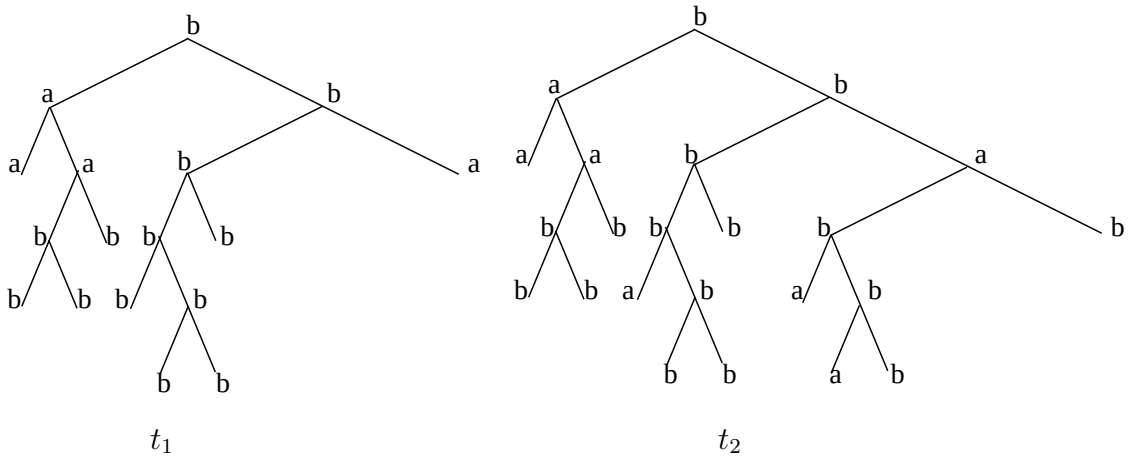

FIG. 3.2 – Les arbres t_1 et t_2 de L^c_{aco}

sous-arbres enracinés en un fils gauche du chemin le plus à droite. Dans ce cas, il remonte à la racine en vérifiant que tous les nœuds visités pendant cette remontée sont des fils droits et accepte.

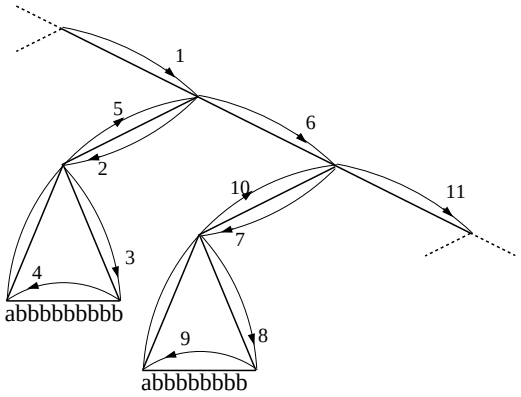

FIG. 3.3 – L'exécution de A_{aco} sur un arbre de L_{aco}

Nous décrivons maintenant les boucles de A_{aco} sur des arbres qui n'appartiennent pas à L_{aco}. La figure 3.4 représente l'exécution infinie de A_{aco} sur un arbre tel que les feuilles d'un des sous arbres enracinés en un nœud du chemin le plus à droite sont toutes étiquetées par b. L'automate traverse l'arbre comme dans la figure 3.3 pour les flèches de la figure 3.3 numérotées de 1 à 9. Mais comme A_{aco} n'a pas encore vu de feuilles étiquetées par la lettre a, il continue son parcours en profondeur , ce qui est représenté par les flèches 10, 2 et 3 de la figure 3.4. Les flèches de la figure 3.3 numérotées de 2 à 10 constituent la boucle.

La figure 3.5 montre l'exécution infinie de A_{aco} sur un arbre tel que les feuilles d'un des sous arbres enracinés en un nœud du chemin le plus à droite forme un mot avec une position de type milieu étiquetée par a. L'automate arrête son parcours en profondeur représenté par les flèches 2 et 3 de la figure 3.5 dès qu'il voit une feuille étiquetée par la lettre a. A_{aco} remonte alors selon la flèche 4 jusqu'à un fils droit qui n'appartient pas au chemin le plus à droite et descend d'un pas vers la droite puis d'un pas vers la gauche. L'automate démarre alors un nouveau parcours

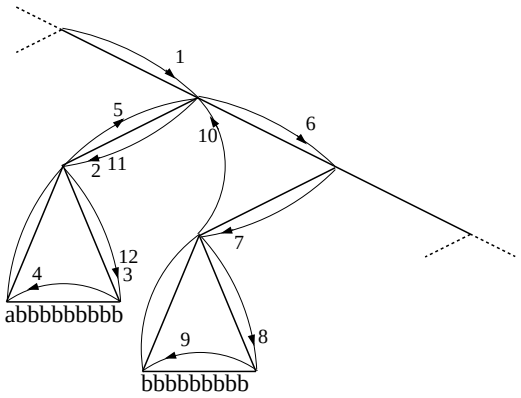

FIG. 3.4 – L'exécution de A_{aco} sur un arbre de L^c_{aco}

en profondeur de gauche à droite représenté par les flèches 6 et 7. Les flèches 4,5,6 et 7 forment ainsi la boucle.

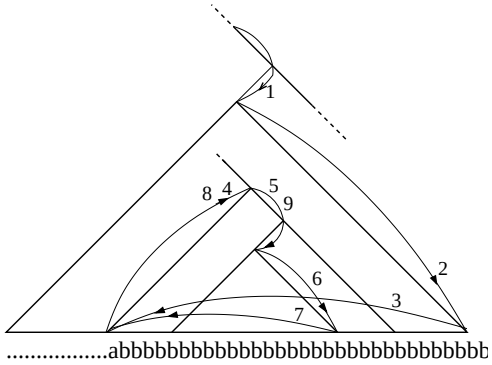

FIG. 3.5 – L'exécution de A_{aco} sur un arbre de L^c_{aco}

Nous allons maintenant donner une définition formelle de A_{aco}. Les états de A_{aco} sont :

- l'état initial q_I où A_{aco} visite les nœuds du chemin le plus à droite d'un arbre du langage L_{aco},
- les trois états q_{d2}, d_{d1}, q_m pour effectuer un parcours en profondeur d'un sous-arbre enraciné en un nœud de 21^*,
- l'état q_a dans lequel l'automate remonte tant qu'il est sur un nœud gauche après avoir vu une feuille étiquetée par la lettre a,
- l'état final q_f permettant à l'automate de remonter le long du chemin le plus à droite pour accepter un arbre de L_{aco}.

La fonction de transition de l'automate A_{aco} est décrite par le tableau suivant où le symbole noté "_" représente n'importe quel lettre de Σ :

q_I	$\_$	$(\text{in}, \overset{\text{ra}}{2})$	q_{d2}	$\swarrow$
q_I	$\_$	$(\text{fe}, \overset{\text{ra}}{2})$	q_f	$\uparrow$
q_{d2}	$\_$	$(\text{in}, \overset{\text{ra}}{\underset{2}{1}})$	q_{d2}	$\searrow$
q_{d2}	b	$(\text{fe}, \overset{1}{2})$	q_m	$\circlearrowleft$
q_m	$\_$	$(\overset{\text{in}}{\text{fe}}, 2)$	q_{d1}	$\uparrow$
q_m	$\_$	$(\overset{\text{in}}{\text{fe}}, 1)$	q_m	$\uparrow$

q_{d1}	$\_$	$(\text{in}, \overset{1}{2})$	q_{d2}	$\swarrow$
q_{d2}	a	$(\text{fe}, 1)$	q_a	$\uparrow$
q_a	$\_$	$(\overset{\text{in}}{\text{fe}}, 1)$	q_a	$\uparrow$
q_a	$\_$	$(\text{in}, \overset{\text{ra}}{2})$	q_I	$\searrow$
q_f	$\_$	$(\text{in}, 2)$	q_f	$\uparrow$

3.2 Construction du complément d'un DTWA

3.2.1 Technique de Sipser

Afin d'éliminer les exécutions infinies d'un automate cheminant déterministe, nous allons étendre à ces automates la construction de Sipser de [47] qui supprime les boucles d'une machine de Türing déterministe. Une machine de Turing est un modèle de machine qui généralise les automates bidirectionnels sur les mots et qui sera formellement défini dans le chapitre 7. L'idée de la construction de Sipser est basée sur l'observation cruciale suivante : le graphe de configurations arrière d'une machine de Turing déterministe sur une entrée w est une forêt c'est-à-dire un graphe acyclique. Le graphe de configurations arrière d'une machine de Turing M est par définition le graphe dont les sommets sont les configurations de M et tel qu'une arête relie une configuration c à une configuration c' si et seulement c est atteint à partir de la configuration c' en un pas de M. Nous définissions ci-dessous le graphe de configurations arrière d'un automate cheminant.

Définition 3.3 *Soient A un* TWA *sur un alphabet Σ et t un arbre étiqueté par cet alphabet. Une configuration est co-accessible si elle permet d'atteindre une configuration acceptante.*

Le graphe de configurations arrière $G(A,t)$ est le graphe fini dont l'ensemble des sommets correpond à l'ensemble des configurations co-accessibles et l'ensemble des arêtes vérifie la propriéré suivante : il existe une arête de la configuration c à la configuration c' si et seulement si $c' \xrightarrow{A,t} c$.

Cette notion sera généralisée dans la suite de ce chapitre en 3.3.2 au graphe des configurations co-accessibles à partir d'une configuration quelconque.

Remarque 3.1 Notons qu'un arbre est accepté par l'automate si et seulement la configuration initiale est un sommet du graphe de configurations arrière. Cette caractérisation des arbres acceptés par un automate jouera un rôle essentiel dans les constructions qui suivront.

Lemme 3.3 *Soit A un* DTWA *à unique configuration acceptante bloquante. Pour tout arbre t, le graphe de configurations arrière de A sur t est un arbre non-ordonné appelé arbre de configuration arrière de A sur t. La racine de cet arbre est la configuration acceptante.*

Preuve. D'après la définition du graphe de configurations arrière, tous les sommets de $G(A,t)$ sont liés à la configuration acceptante que l'on note (q_f, ϵ). Soit (q,v) un sommet de ce graphe. On considère deux chemins de la configuration acceptante à (q,v). Comme A est déterministe, un des chemins est inclus dans l'autre et comme la configuration acceptante est bloquante les deux chemins sont identiques. Le graphe de configurations arrière est donc un arbre non-ordonné dont la racine est la configuration acceptante. □

Remarque 3.2 On dit qu'un automate cheminant est codéterministe si pour chaque arbre t, pour chaque configuration c il existe au plus une configuration c' tel que $c' \xrightarrow{A,t} c$. Si A est un DTWA codéterministe à unique configuration acceptante, pour tout arbre, le graphe de configurations arrière est un mot.

Afin d'éliminer les exécutions infinies d'un DTWA A, nous allons construire A' un DTWA qui simule l'automate A en arrière. Sur un arbre donné t, A' va effectuer un parcours en profondeur de l'arbre de configuration arrière $G(A,t)$. Notons que l'arbre de configuration arrière est implicite dans le sens où le parcours en profondeur de cet arbre est simulé à la volée par A'. Une telle simulation est possible car l'arbre de configuration arrière est localement constructible : nous pouvons calculer à partir d'une configuration donnée tous ses fils et son père dans l'arbre de configuration arrière.

3.2.2 Le complément

Nous voulons donc construire un automate A' dont l'exécution sur un arbre t à partir de la configuration initiale est associée à l'exploration en profondeur de l'arbre de configuration arrière $G(A,t)$. Cela est possible car les informations nécessaires à cette exploration à partir d'une configuration courante $n = (q,v)$ peuvent être résumées dans une nouvelle donnée qui sera incorporée dans l'état de l'automate A'. Cette donnée permet à A' de reconstruire "à la volée" (c'est-à-dire en ligne) les fils et le père du sommet courant de l'arbre de configuration arrière. Elle permet aussi d'ordonner le graphe de configurations arrière afin de le parcourir en profondeur. Nous allons la définir formellement

Définition 3.4 *Soient $A = (Q, \Sigma, q_0, F, \delta)$ un* DTWA *à unique configuration acceptante et t un arbre de domaine $\mathcal{N}$ sur Σ.*

On note D et on appelle fonction direction la fonction partielle suivante de $\mathcal{N} \times \mathcal{N}$ à valeurs dans l'ensemble $\{\swarrow, \searrow, \circlearrowleft, \nwarrow, \nearrow\}$ définie ainsi : pour chaque nœud v, $D(v,v) = \circlearrowleft$ et pour chaque nœud interne v, $D(v, v1) = \swarrow$, $D(v, v2) = \searrow$, $D(v1, v) = \nearrow$ et $D(v2, v) = \nwarrow$.

Soit $(q,v) \in Q \times \mathcal{N}$ un sommet de l'arbre de configuration arrière qui n'est pas la configuration acceptante et qui a pour père dans $G(A,t)$ le nœud (q', v'). Le rang de (q,v) est le couple $(q, D(v,v'))$.

Remarque 3.3 Remarquons que le rang d'une configuration permet de distinguer cette configuration parmi ses frères dans l'arbre de configuration arrière. En effet, une configuration (q, v) d'un automate cheminant est définie par un état et un nœud qui peuvent être calculés à partir de son rang et de son père (q', v') dans l'arbre de configuration arrière. Ainsi, chaque sommet de $G(A, t)$ a au plus $5|Q|$ successeurs dans le graphe. Etant donné un DTWA A à unique configuration acceptante, nous fixons un ordre arbitraire sur l'ensemble des états et sur l'ensemble $\{\swarrow, \searrow, \circlearrowright, \nwarrow, \nearrow\}$. Nous obtenons alors un ordre total sur le rang en utilisant l'ordre lexicographique et pour tout arbre t, l'arbre de configuration arrière est alors ordonné. $G(A, t)$ est alors considéré comme un arbre de degré borné mais on appelle encore ses nœuds des sommets afin de ne pas les confondre avec les nœuds de t.

Pour montrer que la classe des langages d'arbres reconnus par un DTWA est fermée par complémentation, il nous suffit, d'après la proposition 3.1, de prouver le théorème ci-dessous.

Théorème 3.1 *Pour tout automate cheminant déterministe, on peut construire un automate d'arbres cheminant déterministe équivalent à exécutions finies.*

Preuve. Soit $A = (Q, \Sigma, q_0, F, \delta)$ un DTWA et t un arbre sur Σ. D'après le lemme 3.1, on peut supposer que A est à unique configuration acceptante bloquante. On construit dans cette preuve un automate A' qui effectue un parcours en profondeur de l'arbre $G(A, t)$ et qui accepte si la configuration finale (q_f, ϵ) est un nœud de $G(A, t)$. Lorsque A' visite un sommet (q, v) de $G(A, t)$, sa tête de lecture est sur le nœud v et l'état q est stocké dans l'état de A'. Si A' vient de remonter au père dans le parcours en profondeur de $G(A, t)$, c'est-à-dire s'il vient d'une configuration (q', v') tel que $(q', v') \xrightarrow{A,t} (q, v)$, son état courant contient également le rang de (q', v'). Cette dernière information lui permet de déterminer le prochain sommet de l'arbre de configuration arrière qu'il doit visiter dans le parcours en profondeur. On montre maintenant comment ces informations sont maintenues. On suppose donc que A' visite le sommet x dans le parcours en profondeur de $G(A, t)$, on pose $x = (q, v)$ et on distingue alors deux cas.

On considère dans un premier temps que le sommet x est visité pour la première fois. Il s'agit du cas où x a été atteint à partir de son père dans $G(A, t)$. La tête de A' est donc sur le nœud v de t et l'état courant de A' contient l'état q. A' calcule alors le rang du premier fils de x en testant dans l'ordre croissant chaque rang (q', d) de la manière décrite ci-dessous.

L'automate se déplace vers le nœud w dans la direction opposée à d si c'est possible. Le nœud w est, sous réserve d'existence, le nœud tel que $D(w, v) = d$. A partir de w et à l'aide de son étiquette a, de son type θ et de la fonction de transition δ, A' vérifie virtuellement si $\delta(q', a, \theta) = (q, d)$. Si c'est bien le cas, A' doit maintenant visiter la configuration (q', w). Il confirme donc son déplacement en w et stocke q' dans son état. Si ce n'est pas le cas, A' retourne en v et teste le rang suivant. q est alors toujours stocké dans son état. Si tous les rangs ont été ainsi testés sans succès, le sommet x est une feuille et A' retourne au père de x dans le graphe $G(A, t)$ en simulant A sur t d'un pas à partir de la configuration x. On obtient ainsi un nouvel état r et un déplacement d de $\{\swarrow, \searrow, \circlearrowright, \uparrow\}$ qui induit avec le type de v

une direction d' de $\{\swarrow, \searrow, \circlearrowleft, \nwarrow, \nearrow\}$. A' se déplace alors selon d et maintient dans son état les informations nécessaires : r pour l'état de la configuration courante et (q, d') pour le rang du nœud qu'il vient de visiter.

Le deuxième cas est celui où A' revisite le sommet x à partir d'un de ses fils x' de rang (q', d'). Ce rang est stocké dans l'état courant de A'. Si (q', d') est l'élément maximal dans l'ordre sur les rangs, A' a visité tous les fils de x et il retourne alors au père de x dans $G(A, t)$ en simulant A d'un pas et il maintient les informations nécessaires comme dans le cas précédent quand il visite une configuration sans fils. Si le rang (q', d') n'est pas maximal, A' visite le prochain fils de x en testant comme dans le premier toutes les possibilités pour le rang dans l'ordre croissant en partant du rang suivant.

La configuration finale acceptante de A' est constituée par l'état initial et la racine. En vertu de la remarque 3.1, A' est bien équivalent à A. De plus, toutes les exécutions de A sont finies car si A' rejette, il termine son exécution à la fin du parcours de $G(A, t)$

□

Ce théorème nous permet immédiatement de déduire le résultat ci-dessous.

Théorème 3.2 *La classe des langages d'arbres reconnu par un automate d'arbres cheminant déterministe est fermée par complémentation et par union.*

Exemple 3.2 Reprenons maintenant l'exemple 3.1. Nous savons maintenant complémenter l'automate A_{aco}. Notons L_0 l'ensemble des arbres sur l'alphabet $\{a, b\}$ dont la suite des feuilles lues de gauche à droite constituent un mot de ab^*. Le langage L^c_{aco} est l'ensemble des arbres pour lesquels il existe un sous-arbre enraciné en un nœud de 21^* qui n'appartient pas à L_0. En appliquant la construction présentée ci-dessus, on peut construire deux DTWA aux comportements distincts qui reconnaissent le complément de L_{aco} selon l'ordre que l'on pose sur l'ensemble des rangs. Le comportement du DTWA qui reconnaît le complément de L_{aco} et qui est obtenu si l'ordre défini sur l'ensemble des rangs des configurations de A_{aco} est tel que $(q_{d1}, \swarrow) < (q_I, \swarrow)$ est décrit ci-dessous.

Notons que A' simule A_{aco} en arrière. Au départ, A' descend le long du chemin le plus à droite jusqu'à la feuille la plus à droite. A' remonte ensuite le long de ce chemin et, à chaque nœud de ce chemin, A' vérifie si le sous-arbre gauche de ce nœud est dans L_0. La figure 3.3 décrit comment A' effectue cette vérification. L'automate A' commence un parcours en profondeur de gauche à droite. Il descend ainsi vers la feuille la plus à gauche du sous-arbre comme le montre la flèche 1 de la figure 3.3. Si cette feuille visitée n'est pas étiquetée par a, l'arbre est accepté. Sinon l'automate continue son parcours en profondeur jusqu'à ce qu'il visite une autre feuille étiquetée par la lettre a, ce qui est représenté par les flèches 2,3,4,5 et 6 de la figure 3.3. Si cette feuille est un fils droit, A' accepte immédiatement sinon A' remonte de fils gauche en fils gauche avec les flèches 7 et 8 de la figure jusqu'à ce qu'il atteigne un fils droit. L'automate A' remonte alors d'un pas comme le montre la flèche 9. Notons que, si l'arbre est dans L, A' se trouve alors sur le chemin le plus à droite. Si l'automate A' est ainsi remonté jusqu'à la racine, A' rejette. S'il est remonté à un fils gauche, A' accepte. S'il s'agit d'un fils droit, l'automate A' remonte au père puis descend au fils gauche et vérifie le sous-arbre enraciné au nœud courant.

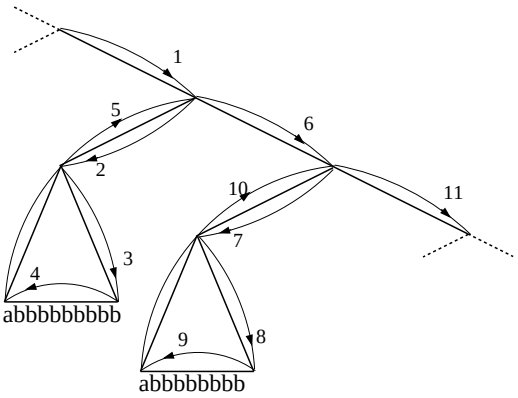

FIG. 3.6 – L'exécution de A^c_{aco} sur un arbre de L_{aco}

3.2.3 Complexité de la construction

Comme il a été dit au début de ce chapitre nous définirons dans le chapitre 7 les machines de Turing qui nous permettront d'introduire le coût en temps et en espace d'un problème et d'une construction. Dans la preuve du théorème 3.2, nous considérons un DTWA $A = (Q, \Sigma, q_0, F, \delta)$ qui reconnaît le langage d'arbres L_A et nous construisons un DTWA $A' = (Q', \Sigma, q'_0, F', \delta')$ qui reconnaît le complément de L_A. Supposons que Q est un ensemble de n états. Pour construire A', dans un premier temps, nous transformons, en utilisant le lemme 3.1, A en un automate équivalent A_1 à unique configuration acceptante bloquante qui termine toujours à la racine. A_1 a un nouvel état en plus. Ensuite, nous construisons A_2 un automate à exécutions finies équivalent. Lorsque A_2 visite un sommet de l'arbre de configuration arrière de A_1, son état contient l'état de la configuration correspondante de A_1 et le rang du dernier sommet visité si A_1 remonte au père dans son parcours de l'arbre de configuration arrière. Nous avons vu que la taille du rang est borné par $5n_1$ où $n_1 = n + 1$ est le nombre d'états de A_1 et donc l'ensemble des états de A_2 est un ensemble de $(5(n + 1) + 1)(n + 1)$ éléments. Nous transformons A_2 en un automate équivalent à exécutions finies A_3 avec un état de plus qui termine toujours à la racine à l'aide du lemme 3.2. A partir de A_3, nous utilisons enfin la proposition 3.1 pour construire A' en rajoutant seulement un état. Nous obtenons alors le théorème suivant.

Théorème 3.3 *Soient A un* DTWA*, n le nombre d'états de A et L le langage reconnu par A. On peut construire en temps polynomial un* DTWA *équivalent à exécutions finies avec $O(n^2)$ états et un* DTWA *qui reconnaît le complément de L avec $O(n^2)$ états aussi.*

Notons que la construction du complément est polynomiale. En effet, l'automate construit pour le complément a un nombre polynomial d'états et donc un nombre polynomial de transitions et chaque transition peut être calculée en temps constant. Nous définirons formellement dans le chapitre 7 les constructions polynomiales.

3.3 Complément et automates à jetons déterministes

3.3.1 Etendre la construction aux automates à jetons du modèle faible déterministe

La construction du complément pour les automates cheminants déterministes s'étend sans difficulté au modèle faible d'automates d'arbres à jetons déterministes. On étend la définition d'un automate à exécutions finies et celle d'un automate à unique configuration acceptante bloquante aux deux modèles d'automates à jetons. Pour complémenter un automate à jetons à exécutions finies on se ramène au cas où toutes ses exécutions terminent à la racine avec tous les jetons levés.

Définition 3.5 *Soit $k > 0$ et A un* wDPTA$_k$ *à exécutions finies, on dit que A termine toujours à la racine si A a une seule configuration acceptante à partir de laquelle il ne peut appliquer aucune transition et si toutes les exécutions de A se terminent dans une configuration à la racine avec tous les jetons levés.*

On étend ensuite la définition du graphe de configurations arrière aux automates à jetons. On montre exactement comme dans le cas des automates cheminants que le graphe de configurations arrière d'un automate à jetons à unique configuration acceptante bloquante est un arbre non-ordonné. On considère maintenant le cas d'un automates à jetons du modèle faible A. Le rang d'une configuration est défini comme dans le cas des DTWA avec une fonction D qui associe à un couple (x, x') de configurations consécutives un élément de $\{\nwarrow, \nearrow, \circlearrowleft, \swarrow, \searrow, \text{pose}, \text{leve}\}$ selon la transition permettant à A d'atteindre x' à partir de x. Comme la tête de lecture de A ne se déplace pas lorsque l'automate pose ou lève un jeton, le rang d'une configuration détermine de manière unique dans le graphe $G(A, t)$ une configuration parmi ses frères. Dans l'arbre de configuration arrière d'un automate à jetons du modèle faible, un sommet a au plus $7|Q|$ fils. Pour disposer d'un ordre sur le rang, il faut donc un ordre sur Q et un ordre sur $\{\nwarrow, \nearrow, \circlearrowleft, \swarrow, \searrow, \text{pose}, \text{leve}\}$. On montre ansi le théorème suivant

Théorème 3.4 *Soit $k > 0$ Pour tout* DPTA$_k$ *du modèle faible à n états, on peut construire un* DPTA$_k$ *du modèle faible à exécutions finies qui reconnaît le même langage avec $O(n^2)$ états. La classe les langages reconnus par un* DPTA$_k$ *est donc fermée par complément et par union.*

Preuve. Soit A un DPTA$_k$ du modèle faible. On construit comme dans le cas des automates cheminants déterministes un DPTA$_k$ du modèle faible équivalent à exécutions finies A' qui simule A en arrière et effectue un parcours en profondeur du graphe de configurations arrière. A chaque fois que A' viste un sommet $(q, v, i, \vec{u}_{i+1})$ du graphe $G(A, t)$, sa tête de lecture est sur le nœud v, q est stocké dans son état et les jetons numérotés de k à i sont placés selon $\vec{u}_{i+1}$. Le rang d'une configuration est une information finie qui peut être stockée dans l'état de A'. Pour simuler en arrière une transition qui lève un jeton (respectivement qui pose un jeton), il suffit de poser le jeton sur le nœud courant (respectivement de lever le jeton qui est sur le nœud courant). On peut maintenant étendre la construction du théorème 3.1 sans difficulté. □

3.3.2 Une astuce pour complémenter le modèle fort déterministe

La construction du complément d'un automate déterministe du modèle faible ne s'étend pas directement au modèle fort. En effet, nous ne pouvons plus borner le degré des arbres de configurations arrière car, dans le modèle fort, le dernier jeton peut être levé à partir de n'importe quel nœud et un sommet de l'arbre de configurations arrière peut donc avoir un nombre arbitraire de fils de rang (q, leve). Considérons k un entier positif, A un DPTA$_k$ et t un arbre. Afin de déterminer de manière unique une configuration de rang (q, leve) parmi ses frères dans le parcours du graphe de configurations arrière, nous devons connaître la position du jeton qui est sur le point d'être levé de l'arbre t. Etant donné un sommet x de l'arbre de configuration arrière $G(A,t)$ ayant comme père le sommet x', on pose $x = (q, u, i, \vec{u}_i)$ et $x' = (q', v, j, \vec{v}_j)$ et on définit le rang étendu de x par le triplet $(q, D(x,x'), u_i)$ où D est la fonction défini comme dans le cas du modèle faible et u_i la position du jeton i dans t. Notons que nous avons besoin de u_i seulement dans le cas où $D(x,x')$ a la valeur leve. Le rang étendu définit de manière unique un sommet parmi ses frères dans $G(A,t)$ mais il ne peut plus être stocké avec une mémoire constante dans l'état de A'.

Nous ordonnons maintenant les fils d'un sommet $(q', v, i, \vec{v}_{j+1})$ dans l'arbre de configurations arrière $G(A,t)$ en fonction de leur rang étendu. L'ordre que nous définissons dépend du nœud v. Il est basé sur un ordre arbitraire sur Q, un ordre arbitraire sur $\{\nwarrow, \nearrow, \circlearrowright, \swarrow, \searrow, \text{pose}, \text{leve}\}$ où "leve" est maximal et un ordre sur les nœuds de t. Pour des raisons techniques qui deviendront claires par la suite, nous fixons l'ordre suivant sur les nœuds de t : $u <_v u'$ si dans un parcours en profondeur de gauche à droite partant du nœud v, le nœud u est visité avant le nœud u'. Un tel parcours en profondeur est constitué d'un parcours en profondeur de gauche à droite du nœud v jusqu'à la racine suivi d'un parcours en profondeur de gauche à droite de la racine au nœud v. Nous classons dans $G(A,t)$ deux sommets frères x_1 et x_2 en utilisant l'ordre lexicographique sur le rang étendu.

Dans le parcours de l'arbre de configurations arrière $G(A,t)$ décrit par la suite, nous considérons des étapes où A est simulé en avant entre deux nœuds de t distingués. Pour formaliser ceci, nous introduisons ci-dessous les arbres marqués par des jetons puis une extension de l'arbre de configurations dans laquelle la racine n'est pas forcément la configuration initiale.

Définition 3.6 *Soit $J \subseteq \mathbb{N}_{>0}$ un ensemble fini de jetons. Un arbre J-marqué est un couple (t, f) où t est un arbre et f une J-affectation des jetons sur cet arbre. Un arbre J-marqué sur l'alphabet Σ peut également être considéré comme un arbre sur l'alphabet $\Sigma \times 2^J$ tel que l'étiquette d'un nœud indique l'ensemble des jetons de J qui se trouvent sur ce nœud.*

Définition 3.7 *Considérons deux entiers i et k tels que $0 \leq i \leq k$ et un* DPTA$_k$ *A sur l'alphabet Σ. Soient $t' = (t, u_k, \cdots, u_{i+1})$ un arbre $[i+1, k]$-marqué, u un nœud de t et q un état de A. Le graphe $G^i(A, t', q, u)$ est le graphe dont l'ensemble des sommets est l'ensemble des j-configurations de A sur t' avec $j \leq i$ qui permettent d'atteindre la i-configuration $(q, u, u_k \cdots u_i)$ et tel qu'il existe une arête de la confi-*

guration x à la configuration x' si et seulement si x' est une configuration différente de $(q, u, u_k \cdots u_i)$ telle que $x' \xrightarrow{A,t} x$.

On montre que le graphe $G^i(A, t', q, u)$ défini ci-dessus est un arbre non-ordonné de racine $(q, u, u_k \cdots u_i)$ en reprenant la preuve du lemme 3.3 et nous ordonnons cet arbre avec l'ordre défini sur le rang étendu. Notons que le rang de cet arbre de configurations dépend de la taille de l'arbre t. Pour s'en rendre compte, supposons qu'une j-configuration $(p, v, u_k \cdots u_{j+1})$ a pour fils dans l'arbre $G^i(A, t', q, u)$ une $(j-1)$-configuration $(r, v, u_k u_j)$ où $j \in [1, i]$, p et r sont des états et u_j un nœud de t distinct de v. Toutes les autres configurations de la forme $(p, v, u_k \cdots u_{j+1} u')$ où u' est un nœud distinct de v sont alors des fils de la configuration x dans $G^i(A, t', q, u)$ car le jeton j peut être levé à partir de n'importe quel nœud.

Nous considérons A un DPTA$_k$. Pour $m \in [0, k]$, nous notons A_m le DPTA$_m$ sur l'alphabet $\Sigma \times 2^{[m+1,k]}$ obtenu à partir de A de la manière suivante : on supprime les transitions qui lèvent ou qui posent un jeton dont le numéro est strictement supérieur à m et on modifie les transitions qui testent la présence d'un jeton dont le numéro est strictement supérieur à m de telle sorte qu'elles effectuent ce test à l'aide de l'étiquette du nœud courant. Dans la preuve du lemme qui suit, nous construisons, pour tout entier $m \in [0, k]$, un automate à $3m$ jetons qui effectue un parcours en profondeur du graphe de configurations arrière de A_m.

Lemme 3.4 *Soient $0 \leq m \leq k$, A un* DPTA$_k$ *et (r, s) un couple d'états de A. On pose $u_{3m+1} = u_{3m+2} = \epsilon$. On peut construire A' un* DPTA$_{3m}$ *sur l'alphabet $\Sigma \times 2^{[3m+1,\cdots,3k]}$ avec deux états q'_0 et q'_f tel que pour tout arbre $[3m+1, 3k]$-marqué $t' = (t, u_{3k}, \cdots, u_{3m+1})$,*

- *l'exécution de A' sur t' partant de la configuration (q'_0, u_{3m+1}) est finie et*
- *$(q'_0, u_{3m+1}) \xrightarrow{A',t'\ *} (q'_f, u_{3m+2})$ si et seulement si il existe une m-exécution de A sur t de la configuration $(r, u_{3m+1}, u_{3k} u_{3(k-1)} \cdots u_{3(m+1)})$ à la configuration $(s, u_{3m+2}, u_{3k} u_{3(k-1)} \cdots u_{3(m+1)})$.*

Nous rappelons qu'une m-exécution est une exécution d'une m-configuration à une m-configuration durant laquelle le jeton $(m+1)$ n'est jamais levé.

Preuve. On prouve ce lemme par induction sur l'entier m.

Posons, pour tout $n \in [m+1, k]$, $v_n = u_{3n}$ et notons $t'' = (t, v_k, \cdots, v_{m+1})$ l'arbre $[m+1, k]$-marqué et G le graphe de configurations arrière $G^m(A, t', s, u_{3m+1})$ ordonné avec le rang étendu.

Le cas de base $m = 0$ est similaire à la construction du lemme 3.1. L'automate A' stocke les états r et s dans sa mémoire et commence par se placer sur le nœud u_2 qui est marqué dans t'. A partir de ce nœud et de l'état s, l'automate A' effectue un parcours en profondeur du graphe de configurations arrière G. Les nœuds u_1 et u_2 étant marqués dans t', si A' visite le sommet $(r, u_1, v_k, \cdots v_1)$ dans ce parcours il retourne au nœud u_2 et passe dans l'état q'_f.

On suppose maintenant que $m > 0$. Quand A' visite un sommet $(q, v, v_k \cdots v_{i+1})$ de G avec $i \leq m$, sa tête de lecture est en v, pour chaque $j \in [i+1, m]$, les jetons $3j$, $(3j-1)$ et $(3j-2)$ sont placés sur le nœud v_j et son état contient q et le rang du sommet de G visité précédemment qui est une information bornée contrairement au rang étendu. Ces informations sont maintenues comme il est décrit ci-dessous.

Si A ne lève pas de jeton, A se comporte comme un automate à jetons du modèle faible et il est alors facile de simuler en avant et en arrière les transitions de A qui ne lèvent pas de jeton. Notons que A' pose 3 jetons sur le nœud courant lorsqu'il simule en avant une transition de A qui pose un jeton et qu'il lève 3 jetons lorsqu'il simule en arrière une transition de A qui pose un jeton.

La difficulté dans le parcours en profondeur du graphe de configurations G est de simuler en arrière une transition qui lève un jeton. On suppose donc que A' doit visiter un sommet $x = (q, v, v_k, \cdots, v_i)$ qui est le fils dans G de $x_0 = (q_0, v, v_k \cdots v_{i+1})$ avec $i \leq m$. Son état contient alors le rang d'une configuration x' qui est le frère de x dans G dont le rang étendu est le prédécesseur du rang étendu de x.

Si $v_i = v$, d'après l'ordre défini sur le rang étendu, le rang de x' stocké dans l'état de A' n'est pas de la forme $Q \times \{\text{leve}\}$. L'automate A' pose alors les jetons $(3i+2)$, $(3i+1)$ et $3i$ sur le nœud v, retient le rang (q, leve) de x dans son état et vérifie si la configuration x permet d'atteindre la configuration x_0.

Dans le cas contraire, si $v_i \neq v$, le rang stocké dans l'état de A' est de la forme $Q \times \{\text{leve}\}$. On pose $x' = (q, v, v_k \cdots v_{i+1} v'_i)$ où v'_i est le prédécesseur de v_i dans l'ordre sur les nœuds de t relatif à v. L'automate A' doit trouver la position du nœud v_i où il doit poser 3 jetons. Pour cela, dans le parcours en profondeur de G, A' visite le sommet x directement à partir de son frère x' sans repasser par leur père. Les jetons $(3i+2)$, $(3i+1)$ et $3i$ de A' sont alors placés en v'_i et A' doit transférer ces jetons de v'_i à v_i puis revenir au nœud v qui est la position de sa tête de lecture. Remarquons que si aucune des exécutions de A sur t à partir d'une configuration de la forme $(q'', v_i, v_k, \cdots, v_i)$ ne contient la configuration x, il n'est pas nécessaire de visiter la configuration x qui ne peut pas être atteinte par A à partir d'une i-configuration. Dans cette preuve, une telle configuration est appelée une configuration inutile. A partir du nœud v, l'automate A' lève les jetons $(3i-2)$ et $(3i-1)$ puis pose le jeton $(3i-1)$ sur le nœud v pour pouvoir revenir en v si la configuration x est inutile. Notons qu'à ce stade le jeton $3i$ ne peut plus être levé. L'automate A' cherche alors le nœud v'_i où est posé le jeton $3i$ dans l'arbre t' en effectuant un parcours en profondeur de t'. Ensuite, A' se déplace vers le successeur v'_i de v_i dans l'ordre sur les nœuds de t relatif à v et pose le jeton $(3i-2)$ sur v'_i. L'automate A' détermine alors si x est une configuration inutile de la manière suivante : A' simule A en avant à partir de toutes les configurations de la forme $(q'', v_i, v_k \cdots v_i)$ avec $q'' \in Q$ et vérifie la première fois que le jeton i est levé si sa tête est sur le nœud v marqué par le jeton $(3i-1)$ et si son état courant est q.

Si x est une configuration inutile, A' teste directement le rang étendu suivant en retournant au nœud où est le jeton $(3i-2)$ et en déplaçant ce jeton vers le nœud suivant selon l'ordre sur les nœuds relatif à v. Si au contraire x n'est pas une configuration inutile, A' retourne au nœud v_i marqué par le jeton $(3i-2)$, lève les jetons $(3i-2)$, $(3i-1)$ et $3i$, les pose tous sur v_i et simule encore A en avant jusqu'à ce qu'il lève le jeton i. L'automate est alors retourné au nœud v après avoir placé les 3 jetons $(3i-2)$, $(3i-1)$ et $3i$. Il peut alors continuer son parcours de G.

La dernière difficulté qui nous a amené à procéder par induction est la suivante : quand A' vérifie si x est une configuration inutile, il est dangereux de simuler en avant l'automate A qui peut boucler. Notons alors que A doit seulement être simulé du nœud marqué par le jeton $(3i-2)$ au nœud marqué par le jeton $(3i-1)$ et

durant cette exécution A ne lève jamais le jeton i ni ceux qui ont un numéro plus grand puisque le placement des jetons obéit à une discipline de pile. A' doit donc simuler une $(i-1)$-exécution de A. On applique alors l'hypothèse d'induction pour le nœud initial v_i marqué par le jeton $(3i-2)$, le nœud terminal v marqué par le jeton $(3i-1)$ et les états q'' et q. On obtient ainsi un automate dont les exécutions partant de la configuration correspondant à $(q'', v_i, v_k \cdots v_i)$ sont finies et on utilise cet automate pour simuler A en avant.

Enfin, lorsque tous les fils de x_p ont été visités et qu'on remonte dans G avec une transition qui lève un jeton. A' retourne alors au nœud v et peut lever les 3 jetons $(3i-2)$, $(3i-1)$ et $3i$ pour simuler en avant la transition qui lève le jeton i.

Si A' visite la configuration $(r, u_{3m+1}, u_{3k}u_{3(k-1)} \cdots u_{3(m+1)})$ dans le parcours en profondeur de G, l'état r de A est retenu dans l'état de A', la tête de lecture de A' est sur le nœud u_{3m+1} qui est marqué et les $3m$ jetons de A' sont levés. L'automate A' retourne alors au nœud s qui est également marqué et passe dans l'état q'_f.

□

Nous déduisons du cas $m = k$ du lemme ci-dessus avec la racine comme nœud initial et final le théorème suivant :

Théorème 3.5 *La classe des langages d'arbres reconnus par un automate à jetons déterministe du modèle fort est fermée par complément et union.*

Soient $k \geq 0$, A un automate déterministe à k jetons du modèle fort et L_A le langage reconnu par A. On peut construire un automate déterministe à $3k$ jetons du modèle fort équivalent à exécutions finies qui reconnaît L_A et un automate déterministe à $3k$ jetons du modèle fort équivalent à exécutions finies qui reconnait le complément de L_A.

Notons que, pour chaque entier k, notre construction d'un automate déterministe à $3k$ jetons qui reconnaît le complément d'un automate déterministe à k jetons du modèle fort est polynomiale.

CHAPITRE

4

Caractérisation logique des automates à jetons

Ce chapitre porte sur la caractérisation logique des automates d'arbres à jetons. Nous avons montré avec la proposition 2.6 qu'un langage accepté par un automate à jetons est défini par une formule MSO. Nous définirons dans le chapitre 5 un langage d'arbres régulier, c'est-à-dire un langage d'arbres défini par une formule logique MSO et nous prouverons que ce langage ne peut pas être reconnu par un automate d'arbres à jetons. Le but de ce chapitre est d'identifier un fragment de la logique MSO d'expressivité équivalente aux automates d'arbres à jetons. Une telle caractérisation logique des automates d'arbres à jetons a été établie par J. Engelfriet et H.J. Hoogeboom [19]. Si leur caractérisation concerne le modèle fort des automates à jetons, elle reste tout autant valable pour le modèle faible comme le montrent les résultats du chapitre 5 établissant l'équivalence de ces deux modèles. Cette caractérisation énonce l'équivalence entre les automates d'arbres à jetons et la logique FO + posTC, une extension de la logique du premier ordre au moyen d'un opérateur de clôture transitive unaire apparaissant positivement. De manière similaire, les automates d'arbres à jetons déterministes correspondent à la la logique FO+DTC, c'est-à-dire la logique du premier ordre augmentée d'un opérateur de clôture transitive unaire déterministe.

La section 4.1 présente les logiques FO + posTC et FO+DTC qui sont respectivement des extensions des logiques FO(TC) et DTC(FO) introduites par Neven et Schwentick dans [34] pour caractériser les automates cheminants déterministes et non-déterministes. Nous énonçons ensuite les résultats de [19] concernant les automates à jetons dans la section 4.2. Nous décrivons la construction d'une formule logique du premier ordre avec clôture transitive à partir d'un automate à jetons qui est assez technique bien que sans difficulté particulière. Le lecteur peut consulter

[19] où cette construction est très bien présentée. Nous donnons par contre une nouvelle présentation de la construction d'un automate à jetons reconnaissant le langage défini par une formule FO + posTC et d'un automate à jetons déterministe reconnaissant le langage défini par une formule FO+DTC où nous utilisons les résultats du chapitre 3.

Les résultats présentés dans ce chapitre sont utilisés dans le chapitre 6 pour obtenir des conséquences logiques et séparer les logiques FO+DTC et MSO.

4.1 Logique du premier ordre et clôture transitive

Nous définissons dans cette section les logiques FO+posTC et FO+DTC sur les arbres à partir de la logique du pemier ordre. Rappelons que la logique du premier ordre est défini sur les arbres binaires étiquetés par un alphabet Σ à l'aide des connecteurs $\neg$, $\wedge$, $\vee$, des quantifications universelle et existentielle sur des variable du premier ordre et des formules atomiques suivantes :

- $P_a(x)$ pour chaque lettre a de Σ signifiant que le nœud correspondant à x est étiqueté par a,
- $S_1(x, y)$ et $S_2(x, y)$ correspondant aux relations fils gauche et fils droit,
- $x < y$ signifiant que le nœud correspondant à y est un descendant du nœud correspondant à x.

La logique du premier ordre permet d'exprimer uniquement des propriétés locales contrairement aux automates cheminant qui peuvent par exemple vérifier que la profondeur de toutes les feuilles d'un arbre est paire. Nous démontrons dans le chapitre 6 que les automates à jetons ne reconnaissent pas tous les langages réguliers. Nous présentons donc dans cette section des logiques dont le pouvoir d'expression est compris entre celui la logique du premier ordre et celui de la logique MSO qui caractérise les langages d'arbres réguliers.

Nous définissons un opérateur de clôture transitive unaire noté TC. De manière informelle, $\mathrm{TC}^{x',y'}_{x,y}\phi$ signifie qu'il existe une suite de nœuds dont le premier élément est x' et le dernier élement y' telle que chaque couple de nœuds consécutifs de cette suite satisfait la formule $\phi(x, y)$.

Définition 4.1 *Soit ϕ une formule logique avec $(m + 2)$ variables libres. On pose $\vec{z} = (z_1, \cdots, z_m)$.*

$\mathrm{TC}^{x',y'}_{x,y}\phi(x, y, \vec{z})$ est une formule ayant pour variables libres $x', y', \vec{z}$.

Etant donnés un arbre t et γ une valuation des variables $x', y', \vec{z}$ dans l'ensemble des nœuds de t, on a $t, \gamma \models \mathrm{TC}^{x',y'}_{x,y}\phi(x, y, \vec{z})$ s'il existe une suite de nœuds de l'arbre $u_1, \cdots, u_n$ telle que $u_1 = \gamma(x')$, $u_n = \gamma(y')$ et pour tout $1 \leq i < n$ on a
$t \models \phi(u_i, u_{i+1}, \gamma(z_1), \cdots, \gamma(z_m))$.

En particulier, la formule $\mathrm{TC}^{x',x'}_{x,y}\phi(x, y, \vec{z})$ est toujours vérifiée car, pour la suite de nœuds $u_1 = x', u_2 = x'$, il n'existe aucun entier i tel que $1 \leq i < 2$.

Exemple 4.1 La formule suivante exprime que la feuille la plus à droite de l'arbre considéré est à une profondeur paire, c'est-à-dire à une distance paire de la racine.

$$\exists x', y'(\mathrm{ra}(x') \wedge \mathrm{fe}(y') \wedge \mathrm{TC}^{x',y'}_{x,y}(\exists z\ S_2(x, z) \wedge S_2(z, y)))$$

Remarque 4.1 Comme toutes les clôtures transitives que nous considérons sont unaires, on dit que $\mathrm{TC}_{x,y}^{x',y'}\phi(x,y,\vec{z})$ est la clôture transitive de ϕ pour x et y de x' à y'. Les clôtures transitives n-aires sont définies de manière similaire en remplaçant dans la définition 4.1 les variables x, y, x' et y' par des n-uplets de variables. Il est montré dans [19] que la logique du premier ordre avec clôture transitive n-aire définit les langages reconnus par un automate à jetons avec n têtes de lecture. Ces langages ne sont pas tous réguliers, c'est pourquoi nous ne considérons que des clôtures transitives unaires.

Définition 4.2 *Soient ϕ une formule logique, x et y des variables libres de ϕ, on note $z_1, \cdots z_m$ les autres variables libres de ϕ. La formule $\phi(x,y,z_1,\cdots,z_m)$ est dite fonctionnelle relativement à y si, pour tout arbre t et pour tout $(m+1)$-uplet de nœuds $u, u_1, \cdots, u_m$, il existe au plus un nœud v tel que $\phi(u,v,u_1,\cdots,u_m)$ est satisfaite. La clôture transitive $\mathrm{TC}_{x,y}^{x',y'}\phi(x,y,z_1,\cdots,z_m)$ est alors dite déterministe et on peut noter cette formule ainsi :* $\mathrm{DTC}_{x,y}^{x',y'}\phi(x,y,z_1,\cdots,z_m)$.

Remarque 4.2 La définition de l'opérateur DTC est sémantique. La syntaxe de cet opérateur n'est pas effective : étant donnée une formule $\phi(x,y,z_1,\cdots,z_m)$, on ignore à priori si on peut considérer la formule $\mathrm{DTC}_{x,y}^{x',y'}\phi(x,y,z_1,\cdots z_m)$.

Exemple 4.2 Dans l'exemple 4.1, la sous-formule $\mathrm{TC}_{x,y}^{x',y'}(\exists z\ S_2(x,z)\wedge S_2(z,y))$ est une clôture transitive déterministe car, pour tout arbre t et pour tout nœud u de t, il existe au plus un nœud v tel que la formule $\exists z\ S_2(x,z)\wedge S_2(z,y)$ soit satisfaite pour $x=u$ et $y=v$. On peut alors noter cette formule $\mathrm{DTC}_{x,y}^{x',y'}(\exists z\ S_2(x,z)\wedge S_2(z,y))$

Par contre, la formule $\mathrm{TC}_{x,y}^{x',y'}(S_1(x,y)\vee S_2(x,y))$ n'est pas une clôture transitive déterministe. En effet, étant donnés un arbre t qui n'est pas réduit à une feuille et un nœud interne u de t, il existe deux nœuds v tels que $S_1(u,v)\vee S_2(u,v)$.

On remarque que la formule $\mathrm{TC}_{x,y}^{x',y'}(S_1(x,y)\vee S_2(x,y))$ est équivalente à la formule atomique $x'\leq y'$ qui peut ainsi être exprimée à l'aide d'une clôture transitive et des autres prédicats de base.

Définition 4.3 *Les formules de* FO+TC *sont définies inductivement au moyen des opérateurs du premier ordre et de la clôture transitive.*

Les formules de FO+DTC *sont définies inductivement au moyen des opérateurs du premier ordre et de la clôture transitive déterministe. Il s'agit de toutes les formules de* FO+TC *telles qu'un opérateur de clôture transitive ne peut être appliqué à une sous-formule ϕ pour des variables x et y que si ϕ est fonctionnelle relativement à y.*

La restriction qui permet de définir FO+DTC à partir de FO+TC est une restriction sémantique qui limite l'emploi d'un opérateur de clôture transitive. Nous intoduisons maintenant la logique FO + posTC à partir de FO+TC et d'une restriction syntaxique portant cette fois sur les négations. Nous considérons maintenant les formules de FO+TC dont toutes les occurrences de clôture transitive sont positives, c'est-à-dire les formules de FO+TC telles qu'il y a toujours un nombre pair de négation dans la portée d'un opérateur TC.

Par exemple, si ϕ est une formule du premier ordre sur les arbres, la formule $\neg(\exists x', y' S_1(x', y') \wedge \neg(\mathrm{TC}_{x,y}^{x',y'}(\phi)))$ est une formule de FO + posTC car il y a deux négations dans la portée de son opérateur TC. En utilisant des arguments classiques en logique, il est facile de montrer que les formules de FO+TC avec uniquement des occurrences positives de clôture transitive sont équivalentes à des formules de FO+TC où la négation ne peut être appliquée qu'aux prédicats élémentaires c'est-à-dire aux prédicats fils gauche, fils droit, ancêtre et aux prédicats associés aux lettres.

Par exemple, la formule $\neg(\exists x' \exists y' S_1(x', y') \wedge \neg(\mathrm{TC}_{x,y}^{x',y'}(\phi)))$ est équivalente à la formule $\forall x' \forall y' \neg(S_1(x', y')) \vee \mathrm{TC}_{x,y}^{x',y'}(\phi)$.

Nous utilisons ce résultat pour définir formellement la logique FO + posTC.

Définition 4.4 *Les formules de* FO + posTC *sont les formules de* FO+TC *telles que la négation n'est appliquée qu'à des formules atomiques.*

Nous verrons dans le section suivante que la logique FO + posTC caractérise les automates à jetons du modèle fort non-déterministes. La restriction portant sur les négations dans la logique FO + posTC est imposée car nous ne savons pas complémenter un automate à jetons non-déterministe.

Théorème 4.1 *Toutes les logiques avec clôture transitive ont un pouvoir d'expression inférieur ou égal à celui de la logique* MSO. *Il est toujours possible d'exprimer un opérateur de clôture transitive avec une formule* MSO.

Preuve. Il suffit de noter que la formule $\mathrm{TC}_{x,y}^{x',y'} \phi(x, y, \vec{z})$ est équivalente à la formule ci-dessous :

$\forall X \ (x' \in X \wedge (\forall x \forall y(x \in X \wedge \phi(x, y, \vec{z})) \Rightarrow y \in X) \Rightarrow y' \in X)$

On peut alors transformer une formule de FO+TC en une formule MSO équivalente. □

4.2 Jetons forts et clôture transitive

Le premier résultat porté sur la logique et les automates d'arbres séquentiels est obtenu par F. Neven et T. Schwentick dans [34] : la classe des langages reconnus par un automate cheminant est définie par la logique TC(FO). Chaque formule de TC(FO) est une clôture transitive de la racine à la racine d'une formule logique du premier ordre. Il est également montré dans [34] que les automates cheminants déterministes ont le même pouvoir d'expression que la logique DTC(FO) des clôtures transitives déterministes de la racine à la racine de formules du premier ordre.

Ces résultats sont étendus au cas des automates d'arbres à jetons par J. Engelfriet et H.J. Hoogeboom dans [19]. Dans ce qui suit, nous nous intéressons à ce cas plus général pour lequel nous avons les théorèmes suivants.

Théorème 4.2 *[19] La classe des langages d'arbres reconnus par un automate à jetons non-déterministe du modèle fort est la classe des langages définis par la logique* FO + posTC.

Théorème 4.3 *[19] La classe des langages d'arbres reconnus par un automate à jeton déterministe du modèle fort est la classe des langages définis par la logique* FO+DTC.

Notons que le théorème 4.3 implique que la classe des langages reconnus par un automate d'arbres à jetons déterministe du modèle fort est fermée par complément. Nous avons montré dans le chapitre 3 comment compléménter un automate à k jetons déterministe du modèle faible sans augmenter k le nombre de jetons. Nous généralisons ce résultat au modèle fort dans le chapitre 5 et nous montrons ainsi que, pour tout entier k, la classe des automates déterministes à k jetons est fermée par complément. Les résultats de [19] et de [32] sont liés et ont été prouvés en utilisant l'idée de [47] qui est de simuler en arrière un automate déterministe pour obtenir un automate déterministe à exécutions finies équivalent. Notons que la construction du complément d'un automate à jetons du modèle fort donnée par [19] augmente considérablement le nombre de jetons. Nous montrons au chapitre 5, que nous pouvons déduire des résultats de [8] et de [32] une construction du complément d'un automate à jetons déterministe du modèle fort avec le même nombre de jetons.

La partie 4.2.1 reprend les idées utilisées dans [19] et explique comment transformer un automate à jetons en formule logique équivalente de FO + posTC dans le cas non-déterministe et de FO+DTC dans le cas déterministe. Dans la partie 4.2.2, nous décrivons la construction réciproque en utilisant les résultats du chapitre 3 dans le cas déterministe.

4.2.1 De l'automate à la formule

La transformation d'un automate à jetons avec n têtes de lectures en formule logique du premier ordre avec clôture déterministe n-aire est décrite en détails dans [19]. Cette section présente cette construction pour $n = 1$ et montre ainsi comment exprimer le comportement séquentiel d'un automate à jetons avec les opérateurs de clôture transitive.

Lemme 4.1 *Soient $k \leq 0$ et A un* $\mathrm{PTA_k}$. *Il existe une formule close* FO + posTC *qui définit le langage reconnu par A*

Preuve. Posons $A = (Q, \Sigma, I, F, \delta)$. Rappelons que, k étant fixé, la notation $\vec{u}_i$ désigne la suite finie $u_k, \cdots, u_i$. Etant donnée $\rho = c_1 \cdots c_m$ une exécution de A, les configurations intermédiaires de ρ sont les configurations $c_2, \cdots c_{m-1}$. Notons qu'une exécution composé de deux configurations n'a donc pas de configuration intermédiaire.

Pour prouver le lemme, nous démontrons que pour tout $i \in [0, k]$ et pour tout couple d'états (p, q), il existe une formule $\psi^i_{p,q}(x, y, \vec{x}_{i+1})$ satisfaite par l'ensemble des $(k - i + 2)$-uplets $(u, v, \vec{u}_{i+1})$ telle qu'il existe une i-exécution de A, c'est à dire une exécution où A ne lève pas le jeton $(i + 1)$, de la configuration $(p, u, \vec{u}_{i+1})$ à la configuration $(q, v, \vec{u}_{i+1})$. Procédons par induction sur i.

Commençons par le cas $i = 0$ pour lequel les exécutions de A considérées ne déplacent aucun jeton.

Le passage d'une transition qui ne pose pas et ne lève pas de jetons peut s'exprimer au premier ordre et nous pouvons alors écrire pour chaque couple (p, q) d'états

de A une formule du premier ordre $\phi^0_{p,q}(u, v, \vec{u}_1)$ satisfaite par les $(k+2)$-uplets $(u, v, \vec{u}_1)$ pour lesquels la configuration $(p, u, \vec{u}_1)$ a pour successeur la configuration $(q, v, \vec{u}_1)$, c'est-à-dire pour lesquels l'automate A passe de la configuration $(p, u, \vec{u}_1)$ à la configuration $(q, v, \vec{u}_1)$ en effectuant une transition.

Nous ordonnons l'ensemble Q des états de A. Nous notons alors $Q = [1, n]$. Pour tout triplet d'états (p, q, r), nous définissons maintenant par induction sur $r \in [0, n]$ une formule $\phi^r_{p,q}(x, y, \vec{x}_1)$ satisfaite par les $(k+2)$-uplets $(u, v, \vec{u}_1)$ pour lesquels il existe une 0-exécution de la configuration $(p, u, \vec{u}_1)$ à la configuration $(q, v, \vec{u}_1)$ dont toutes les configurations intermédiaires ont un état inférieur ou égal à r.

Pour $r = 0$, nous avons déja défini la formule $\phi^0_{p,q}(x, y, \vec{x}_1)$. Pour $r > 0$, une 0-exécution de $(r, u, \vec{u}_1)$ à $(r, v, \vec{u}_1)$ dont toutes les configurations intermédiaires ont un état inférieur ou égal à r peut se décomposer en une suite d'exécutions d'une configuration de la forme $(r, u', \vec{u}_1)$ à une configuration de la forme $(r, v', \vec{u}_1)$ telle que toutes les configurations intermédiaires de ces exécutions ont un état inférieur ou égal à $(r-1)$. On définit alors inductivement la formule $\phi^r_{p,q}(x, y, \vec{x}_1)$ par la formule ci-dessous :

$$\phi^{r-1}_{p,q}(x, y, \vec{x}_1) \vee \exists z_1, z_2[(\phi^{r-1}_{p,r}(x, z_1, \vec{x}_1) \wedge \mathrm{TC}^{z_1,z_2}_{x',y'} \phi^{r-1}_{r,r}(x', y', \vec{u}_1) \wedge \phi^{r-1}_{r,q}(z_2, y, \vec{x}_1)]$$

On pose alors $\psi^0_{p,q}(x, y, \vec{x}_1) = \phi^n_{p,q}(x, y, \vec{x}_1)$ et le cas $i = 0$ est montré.

Considérons maintenant $i > 0$. Pour tout $r \in [0, n]$, on construit alors par induction sur r une formule $\Phi^r_{p,q}(x, y, \vec{x}_{i+1})$ satisfaite par les $(k-i+2)$-uplets $(u, v, \vec{u}_{i+1})$ pour lesquels il existe une i-exécution de la configuration $(p, u, \vec{u}_{i+1})$ à la configuration $(q, v, \vec{u}_{i+1})$ telle que toutes les i-configurations intermédiaires de cette i-exécution ont un état inférieur à r.

Pour $r = 0$, une telle i-exécution correspond soit à une seule transition qui ne déplace pas de jeton, soit à une transition qui pose le jeton i suivie d'une $(i-1)$-exécution et enfin d'une transition qui lève le jeton i. Nous considérons pour tout couple d'états (p, q) les formules du première ordre suivantes qui expriment le passage d'une transition de A :

- $\phi_{p,q}(x, y, \vec{x}_{i+1})$ est la formule satisfaite par les $(k-i+2)$-uplets $(u, v, \vec{u}_{i+1})$ pour lesquels la i-configuration $(p, u, \vec{u}_{i+1})$ a pour successeur la i-configuration $(q, v, \vec{u}_{i+1})$,
- $\phi^{\mathrm{pose}}_{p,q}(x, \vec{x}_{i+1})$ est la formule satisfaite par les $(k-i+1)$-uplets $(u, \vec{u}_{i+1})$ tels que la i-configuration $(p, u, \vec{u}_{i+1})$ a pour successeur la $(i-1)$-configuration $(q, u, \vec{u}_{i+1}u)$
- $\phi^{\mathrm{leve}}_{p,q}(x, \vec{x}_{i+1}, y)$ est la formule satisfaite par les $(k-i+2)$-uplets $(u, \vec{u}_{i+1}, v)$ tels que la $(i-1)$-configuration $(p, u, \vec{u}_{i+1}v)$ a pour successeur la i-configuration $(q, u, \vec{u}_{i+1})$.

La formule $\Phi^0_{p,q}(x, y, \vec{x_{i+1}})$ est alors définie par :

$$\phi_{p,q}(x, y, \vec{x}_{i+1}) \vee \bigvee_{s_1, s_2 \in Q} \phi^{\mathrm{pose}}_{p,s_1}(x, \vec{x}_{i+1}) \wedge \psi^{r-1}_{s_1,s_2}(x, y, \vec{x}_{i+1}, x) \wedge \phi^{\mathrm{leve}}_{s_2,q}(y, \vec{x}_{i+1}, x)$$

Pour $r > 0$, la formule $\Phi^r_{p,q}(x, y, \vec{x}_1)$ est donnée par :

$$\Phi^{r-1}_{p,q}(x, y, \vec{x}_1) \vee \exists z_1, z_2[(\Phi^{r-1}_{p,r}(x, z_1, \vec{x}_1) \wedge \mathrm{TC}^{z_1,z_2}_{x',y'} \Phi^{r-1}_{r,r}(x', y', \vec{u}_1) \wedge \Phi^{r-1}_{r,q}(z_2, y, \vec{x}_1)]$$

La formule $\psi^i_{p,q}$ correspond alors à la formule $\Phi^n_{p,q}$.

Pour $i = k$, la formule $\psi^k_{p,q}(x, y)$ est satisfaite par les couples (u, v) tels qu'il existe une exécution de la configuration (p, u) à la configuration (q, v). On considère enfin

$\mathrm{ra}(x)$ la formule du premier ordre satisfaite par la racine, I l'ensemble des états initial et F l'ensemble des états acceptants de A, la formule $\mathrm{ra}(x) \wedge \mathrm{ra}(y) \wedge \bigvee_{p \in I, q \in F} \psi^k_{p,q}(x,y)$ définit alors le langage reconnu par A. Notons que dans toutes les formules que nous construisons il n'y pas de négation devant un opérateur TC. La formule ψ est donc bien dans FO + posTC. □

Comme les langages réguliers sont les langages définis par la logique MSO, on obtient en utilisant le théorème 4.1 le corollaire suivant :

Corollaire 4.1 *Les langages reconnus par les automates à jetons sont réguliers.*

Remarquons que, si A est déterministe, les opérateurs TC sont appliqués dans la preuve du théorème (ci-dessus) uniquement aux formules de la forme $\phi^{r-1}_{r,r}(x', y', \vec{u}_1)$ et $\Phi^{r-1}_{r,r}(x', y', \vec{u}_i)$ qui sont fonctionnelles. En effet, pour tout $(k+1)$-uplets de nœuds $(u, \vec{u}_1)$ et pour tout $i \in [0,k]$, il y a au plus un nœud v tel qu'il existe une i-exécution ρ de la configuration $(r, u, \vec{u}_{i+1})$ à la configuration $(r, u, \vec{u}_{i+1})$ sans configuration intermédiaire d'état r. On déduit ainsi le lemme suivant pour le cas déterministe :

Lemme 4.2 *Soient $k \leq 0$ et A un* $\mathrm{DPTA_k}$. *Il existe une formule close* FO + posTC *qui définit le langage reconnu par A.*

4.2.2 De la formule à l'automate

Nous montrons dans cette partie, comment construire inductivement à partir d'une formule ϕ du premier ordre avec des clôtures transitives, un automate à jetons reconnaissant le langage défini par ϕ.

Pour le cas non-déterministe où nous considérons une formule FO+posTC, nous reprenons ci-dessous la preuve de [19] qui est assez simple.

Lemme 4.3 *Pour toute formule close* FO + posTC ψ, *il existe un* $\mathrm{PTA_k}$ *qui reconnaît le langage défini par ψ.*

Preuve.

Une valuation des variables libres d'une formule FO + posTC est fixée en posant un jeton par variable dans l'arbre donné. De manière formelle, si on considère la formule $\phi(x_1, \cdots, x_m)$, un arbre t et des nœuds $u_1, \cdots, u_m$ de t, l'automate à jetons A_ϕ construit pour ϕ doit vérifier la propriété suivante : étant donné un arbre t où les nœuds $u_1, \cdots, u_m$ sont marqués, il existe une exécution de A_ϕ à partir de la racine et de l'état initial se terminant à la racine dans un état acceptant si et seulement si $t \models \phi(u_1, \cdots, u_m)$.

Pour les formules atomiques, il est très facile de construire un automate déterministe A_ϕ. Par exemple, l'automate correspondant à la formule $x < y$ effectue un parcours en profondeur de l'arbre t donné pour chercher le nœud marqué correspondant à la variable y, remonte à partir de ce nœud jusqu'à la racine et accepte s'il croise au cours de cette remontée le nœud marqué correspondant à x.

Nous procédons maintenant par induction sur la formule.

Pour la négation d'une formule atomique ϕ, il suffit de complémenter l'automate déterministe A_ϕ en utilisant le théorème 3.5.

Pour l'union et l'intersection deux formules FO + posTC, nous utilisons la proposition 2.3.

Pour une quantification existentielle $\exists x\ \phi$, l'automate choisit de manière non-déterministe un nœud de l'arbre correspondant à la variable x, le marque avec un jeton puis simule l'automate A_ϕ.

Pour une quantification universelle $\forall x\ \phi$, l'automate effectue un parcours en profondeur de l'arbre, à chaque fois qu'il visite un nœud pour la première fois dans ce parcours, il pose un jeton dessus et il simule l'automate A_ϕ. Si A_ϕ accepte, l'automate qu'on construit lève le jeton et continue son parcours pour pouvoir tester le nœud suivant.

Il ne reste plus qu'à montrer comment construire un automate A qui correspond à la formule $\mathrm{TC}_{x,y}^{x',y'}\phi(x,y,\vec{z}_m)$. On rappelle que les nœuds correspondant aux variables libres x', y' et $\vec{z}_m$ sont marqués. L'automate A devine une suite de nœuds partant de x' et utilise ses jetons pour vérifier que ϕ est satisfaite pour chaque couple de nœuds consécutifs de cette suite de la manière suivante. L'automate A commence par poser son jeton n_x sur le nœud u_1 correspondant à la variable x' puis il choisit un nœud u_2 et pose son jeton n_y dessus. Le numéro de jeton n_y est tel que $n_x = n_y + 1$. Ensuite, A simule l'automate A_ϕ obtenu d'après ϕ par induction et vérifie ainsi si la formule ϕ est bien satisfaite par les nœuds u_1 et u_2 qui sont marqués par les jetons n_x et n_y. Si ce n'est pas le cas, l'automate A rejette et sinon A retourne sur le nœud u_2 marqué par le jeton n_y, lève les jetons n_y et n_x à partir du nœud u_2 et repose le jeton n_x sur le nœud u_2. Remarquons que le jeton n_x est levé à distance, l'automate A construit est donc un automate à jetons du modèle fort. Si u_2 est le nœud marqué qui correspond à la variable y', l'automate A lève le jeton n_x et accepte. Sinon, A choisit un nouveau nœud u_3, poser son jeton n_y dessus et vérifie si ϕ est satisfaite par les nœuds u_2 et u_3. L'automate A continue son exécution ainsi jusqu'à ce qu'il rejette ou bien qu'il choisisse le nœud correspondant à la variable y', qu'il pose son jeton n_y dessus et qu'il vérifie que ϕ est satisfaite par les nœuds marqués par les jetons n_x et n_y. Dans ce cas, l'automate a bien vérifié qu'il existe une suite de nœuds consécutifs du nœud correspondant à la variable x' au nœud correspondant à la variable y' telle que ϕ est satisfaite pour tout couple de nœuds consécutifs de cette suite.

□

La difficulté supplémentaire dans le cas déterministe de [19] est de construire inductivement des automates d'une part pour la négation et d'autre part pour la clôture transitive déterministe d'une formule FO+DTC donnée. Nous allons voir que les résultats du chapitre 3 écartent ces difficultés. D'après le lemme 3.5, nous pouvons en effet, étant donné un automate d'arbres à jetons déterministe, d'une part construire un automate à jetons déterministe qui reconnaît le complément du langage reconnu par A et d'autre part construire un automate déterministe équivalent à A qui ne boucle pas.

Lemme 4.4 *Pour toute formule close* FO+DTC ψ*, il existe un* $\mathrm{DPTA_k}$ *qui reconnaît le langage défini par* ψ*.*

Preuve. Dans le cas déterministe, on raisonne encore par induction sur la formule. On note d'abord que les automates construits pour les prédicats élémentaires dans la preuve du lemme précédent sont des automates déterministes.

Pour l'intersection de deux formules, on utilise la propriété 2.4. Pour l'union et le complément on utilise le théorème 3.5. Pour la quantification universelle, on procède comme dans le cas non-déterministe : l'automate construit dans le cas non-déterministe pour la formule $\forall x\phi$ est déterministe si l'automate construit pour la formule ϕ est déterministe Pour une quantification existentielle du premier ordre $\exists x\phi$, l'automate effectue un parcours en profondeur de l'arbre, à chaque fois qu'il visite un nœud pour la première fois dans ce parcours, il pose un jeton dessus, simule l'automate A_ϕ construit par induction. D'après le théorème 3.5, on peut supposer que l'automate à jetons déterministe A_ϕ est à exécutions finies. Si l'automate A_ϕ rejette lors de cette simulation, l'automate que nous construit pour la formule $\forall x\phi$ lève le jeton et continue son parcours en profondeur pour tester le prochain nœud visité.

Il ne reste plus qu'à considérer un opérateur de clôture transitive déterministe. On cherche donc à construire A un automate à jetons déterministe correspondant à la formule $\mathrm{DTC}^{x',y'}_{x,y}\phi(x,y,\vec{z}_m)$ où ϕ est un prédicat fonctionnel relativement à y. On considère l'automate à jetons déterministe construit pour la formule A_ϕ. D'après le théorème 3.5, on peut supposer que A_ϕ est à exécutions finies. Contrairement au cas non-déterministe, on ne peut pas deviner le chemin du nœud correspondant à x' au nœud correspondant à z', on procède de manière similaire au cas non-déterministe mais l'automate A doit tester toutes les possibilités à chaque étape où il pose le jeton n_y de la manière suivante. L'automate A pose d'abord son jeton n_x sur le nœud u_1 correspondant à la variable x' puis il effectue un parcours en profondeur de l'arbre et à chaque fois que A visite un nœud v pour la première fois dans ce parcours il pose son jeton $n_y = n_x - 1$ sur ce nœud et simule l'automate A_ϕ à exécutions finies pour vérifier si ϕ est satisfaite par u_1 et v. Si A_ϕ rejette, l'automate A lève le jeton n_y pour pouvoir le reposer sur le nœud qu'il visite juste après dans son parcours en profondeur. Si, pour tous les nœuds de l'arbre qui sont visités par A dans le parcours en profondeur, A_ϕ rejète, A rejette aussi. Dans le cas contraire A_ϕ a marqué avec son jeton n_y l'unique nœud u_2 tel que ϕ est satisfaite par u_1 et u_2. A partir de ce nœud u_2, A_ϕ lève le jeton n_y puis le jeton n_x et repose le jeton n_x sur le nœud u_2. L'automate A continue son exécution ainsi jusqu'à ce qu'il rejette ou bien qu'il pose son jeton n_y sur le nœud correspondant à la variable y' et qu'il vérifie que ϕ est satisfaite par les nœuds marqués par les jetons n_x et n_y. Dans ce cas, l'automate a bien vérifié qu'il existe une suite de nœuds consécutifs du nœud correspondant à la variable x' au nœud correspondant à la varaible y' telle que ϕ est satisfaite pour tout couple de nœuds consécutifs de cette suite, la formule est donc bien vérifiée et A accepte. □

4.2.3 Logiques avec clôture transitive pour les mots

Toutes les logiques avec clôture transitive que nous avons introduits pour les arbres dans la partie peuvent être définies de manière similaire sur les mots.

Les constructions pour prouver les caractérisations logiques des automates à jetons déterministes et non-déterministes du modèle fort s'adaptent sans difficulté au cas des mots. Nous avons aussi montré dans le chapitre 2 que les langages acceptés par un automate à jetons sur les mots sont les langages réguliers. Nous pouvons donc

déduire que les logiques MSO, FO+posTC, FO+DTC et FO+TC sont équivalentes sur les mots.

CHAPITRE

5

Equivalence du modèle faible et du modèle fort

Sommaire

Nous démontrons dans ce chapitre que le modèle fort et le modèle faible d'automates à jetons ont le même pouvoir d'expression dans le cas déterministe et dans le cas non-déterministe. Etant donné A un automate du modèle fort à k jetons, nous construisons un automate à k jetons du modèle faible équivalent qui est déterministe si A l'est. Pour donner le principe de cette construction, nous nous intéressons dans

un premier temps aux automates d'arbres non-déterministes à 1 et 2 jetons qui vérifient la propriété suivante : seul le premier jeton posé par l'automate, c'est-à-dire le jeton dont le numéro est le plus grand, peut être levé à distance. Il sera montré dans la section 5.3 que nous pouvons nous ramener par induction à des automates du modèle fort qui vérifient cette propriété. Pour traiter le cas général, nous définissons dans la section 5.2, étant donné un automate à jetons du modèle faible, son comportement pour un sous-arbre et son comportement pour un contexte. Nous montrons ensuite comment composer ces comportements de manière compatible avec la composition des sous-arbres et des contextes. La notion de comportement d'un automate à jetons est utilisée dans les constructions de ce chapitre et dans les preuves du chapitre 6. Nous décrivons enfin la transformation d'un automate à jetons du modèle fort en un automate du modèle faible équivalent avec le même nombre de jetons. Nous étudions le cas non-déterministe dans la section 5.3 et le cas déterministe dans la section 5.4.

5.1 Cas des automates non-déterministes à 1 et 2 jetons

Afin de donner dans un premier temps une idée des constructions des sections 5.3 et 5.4, nous commençons par des cas particuliers d'automates à jetons non-déterministes, le cas déterministe étant plus difficile. L'automate A du modèle fort à 2 jetons que nous considérons dans cette section a un comportement fort pour le jeton 2 uniquement. Le jeton 1 que l'automate pose après, ne peut être levé qu'à partir du nœud où il se trouve. Le cas d'un automate à 1 jeton non-déterministe décrit en 5.1.1 donne le principe de base de la construction : l'automate du modèle faible que l'on construit va simuler l'automate du modèle fort donné tout en déplaçant le jeton pas à pas du nœud où il est posé jusqu'au nœud à partir duquel il est levé par l'automate du modèle fort. Le cas d'un automate à 2 jetons montre comment résoudre la principale difficulté de la construction de 5.3 : afin de simuler l'automate A du modèle fort donné dans la partie de l'arbre où la position du jeton 2 que l'automate A' du modèle faible déplace est perdue, l'automate A' va stocker et actualiser le comportement de A dans cette partie de l'arbre.

5.1.1 Cas des automates d'arbres à un jeton

Soit A un automate à 1 jeton non-déterministe du modèle fort. Nous construisons ci-dessous un automate A' à un jeton non-déterministe équivalent du modèle faible.

Considérons une exécution de A. Tant que A ne pose pas son jeton, A se comporte comme un automate cheminant et A' le simule sans difficulté. Lorsque A pose son jeton sur un nœud u_p, A' pose aussi son jeton sur ce nœud. Notons u_ℓ le nœud où est A quand A lève son jeton placé en u_p. Pour pouvoir lever son jeton en étant en u_ℓ, l'automate A' du modèle faible va déplacer son jeton de u_p à u_ℓ tout en simulant A. Nous considérons maintenant la 0-exécution de A qui commence à la configuration où A vient de poser son jeton sur u_p et qui se termine juste avant que A lève ce jeton à partir du nœud u_ℓ. Nous décrivons ci-dessous comment A' simule cette 0-

exécution. Rappelons que le jeton n'est pas déplacé par A durant une 0-exécution de A.

Dans un arbre, deux nœuds sont reliés par un unique chemin minimal. Nous supposons ici pour simplifier que u_ℓ est un descendant de u_p pour que, dans le chemin de u_p à u_ℓ, un nœud soit toujours le père de son successeur. Ce qui suit s'adapte cependant très facilement aux autres cas. Pour simuler la 0-exécution de A considérée, A' devine pas à pas le chemin de u_p à u_ℓ et pose successivement son jeton sur tous les nœuds de ce chemin pour pouvoir enfin le lever en u_ℓ. A' doit également deviner l'état dans lequel A lève ce jeton en u_ℓ. Pour cela, A' simule A entre chaque déplacement du jeton. Durant cette simulation, la position de u_p est perdue : A' ne peut plus distinguer u_p des autres nœuds puisque son jeton n'est plus sur u_p. A partir du moment où A' déplace son jeton du nœud u_p, A' ne peut donc plus simuler A lorsque A repasse par le nœud u_p. Il faut donc interdire à A' de passer par le nœud u_p à partir du moment où le jeton de A' n'est plus sur u_p.

Posons $u_1 = u_p, \cdots, u_n = u_\ell$ avec $n > 0$ les nœuds du chemin de u_p à u_ℓ et notons que la 0-exécution de A considérée peut être décomposée de la manière suivante :

- une boucle de u_1 à u_1,
- suivie d'un déplacement de u_1 à u_2,
- suivi d'une boucle de u_2 à u_2 où A reste dans le sous-arbre enraciné en u_2
- puis pour i allant de 2 à $(n-1)$ un déplacement de u_i à u_{i+1} suivi d'une boucle dans le sous-arbre enraciné en u_{i+1}

La figure 5.1 pour laquelle $u_1 = u_p$ et $u_4 = u_\ell$ illustre cette décomposition.

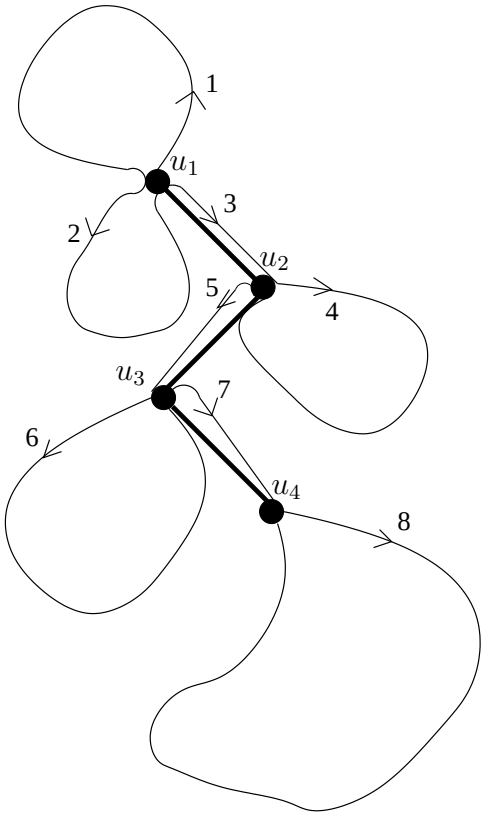

FIG. 5.1 – Décomposition d'une 0-exécution de A

Remarquons que, dans cette décomposition, pour $i \in [1, n-1]$, la boucle dans le sous-arbre enraciné en u_{i+1} ne passe pas le nœud u_1. L'automate A' va deviner cette décomposition et simuler la 0-exécution de A en déplaçant pour chaque $i \in [1, n-1]$ le jeton du nœud u_i au nœud u_{i+1} au moment où A effectue le déplacement de u_i à u_{i+1}.

L'automate A' procède alors de la manière suivante.

Au départ, A' est en u_1 et il vient de poser son jeton sur ce nœud. Il simule alors A dans l'arbre. Au cours de cette simulation, à chaque fois que A' passe par le nœud u_p, A' devine avec le non-déterminisme si la configuration correspondante de A est celle de son dernier passage sur ce nœud. Si A' décide ainsi que A ne repassera plus en u_p, A' retient l'état de cette configuration, puis choisit un de ses deux fils comme prochain nœud dans le chemin de u_p à u_ℓ et déplace son jeton sur ce nœud.

Notons que dans cette simulation, si A veut lever son jeton à distance avant que A' ait deviné son dernier passage en u_1 et déplacer son jeton, A' rejette. En effet, dans ce cas A' a fait une erreur : la dernière fois que A est passé en u_1 avant de lever son jeton, A' n'a pas deviné qu'il s'agissait du dernier passage de A sur le nœud u_1 dans la 0-exécution considérée.

Supposons maintenant que A' vient de déplacer son jeton d'un nœud u' vers un de ses fils u choisi comme son successeur dans le chemin de u_p à u_ℓ. L'automate A' a alors retenu l'état q' du dernier passage de A en u' et procède de la manière suivante :

1. A' simule A à partir de l'état q' et du nœud u' en restant dans le sous-arbre enraciné en u. Si A essaye de lever son jeton en étant sur le nœud u, A' peut simuler cette transition en levant son jeton qui est en u et termine ainsi la simulation de la 0-exécution de A.

2. Au cours de la simulation de 1, A' devine quelle est la dernière fois que A visite u au cours de la 0-exécution considérée et retient l'état de ce dernier passage de A en u. Cet état remplace alors q' dans sa mémoire et A' arrête alors la simulation décrite en 1.

   Notons que si, au cours de la simulation décrite en 1, A quitte ce sous-arbre ou lève son jeton à partir d'un nœud distinct de u, A' rejette. En effet, si A quitte le sous-arbre cela signifie que A' a fait une erreur en devinant le dernier passage de A en u' et si A lève le jeton à partir d'un nœud distinct de u, A' a fait une erreur car il n'a pas deviné le dernier passage de A en u'.

3. Enfin, A' choisit la direction de u vers son successeur dans le chemin de u_p à u_ℓ et déplace le jeton d'un pas dans cette direction.

L'automate non-déterministe à 1 jeton du modèle faible construit ainsi simule les exécutions de A où le jeton est levé à distance. Les deux automates à un jeton sont donc équivalents. Ce cas simple illustre l'idée principale de la construction : les exécutions d'un automate du modèle fort où un jeton est levé à distance sont simulées par un automate du modèle faible qui déplace le jeton en question pas à pas du nœud où il a été posé vers le nœud à partir duquel il va être levé.

5.1.2 Cas des automates d'arbres à deux jetons

Nous considérons maintenant A un automate du modèle fort à 2 jetons et nous voulons construire A' un automate à 2 jetons du modèle faible équivalent. Nous pouvons supposer sans perte de généralité que le jeton 1 de A est toujours levé quand A est sur le nœud où il est posé. En effet, lorsque le jeton 2 est posé, l'automate A se comporte comme un automate à 1 jeton du modèle fort que nous pouvons simuler par un automate à 1 jeton du modèle faible comme nous venons de le voir.

Tant que A ne pose pas son jeton 2, il peut être simulé sans aucune difficulté par un automate du modèle faible. Nous considérons donc une 1-exécution de A qui commence quand A vient de poser son jeton 2 sur un nœud u_p et qui termine juste avant que A lève ce jeton.

A' devine pas à pas le chemin du nœud u_p au nœud u_ℓ à partir duquel le jeton 2 est levé et A' déplace ce jeton le long de ce chemin pour pouvoir le lever en u_ℓ. Nous supposons toujours pour simplifier que u_ℓ est un descendant de u_p mais ce qui suit est facilement adaptable aux autres cas.

Nous expliquons maintenant pourquoi la construction est plus difficile pour un PTA_2 que pour un PTA_1. Considérons u' et u deux nœuds consécutifs du chemin de u_p à u_ℓ et supposons que A' soit sur le point de déplacer le jeton 2 de u' à son successeur u. L'automate A' ne peut plus simuler A en mémorisant l'état q du dernier passage de A en u' comme en 5.1.1. En effet, le jeton 1 de A peut être posé sur un descendant de u' quand A passe pour la dernière fois en u' comme le montre la figure 5.2. Rappelons que pour déplacer le jeton 2 de u' à u, A' doit lever le jeton 1 pour respecter la discipline de pile sur le placement des jetons et A' ne peut donc plus continuer à simuler A à partir de l'état q puisque la position du jeton 1 de A est perdue. Pour pouvoir simuler A tout en déplaçant le jeton de u' à u, A' mémorise

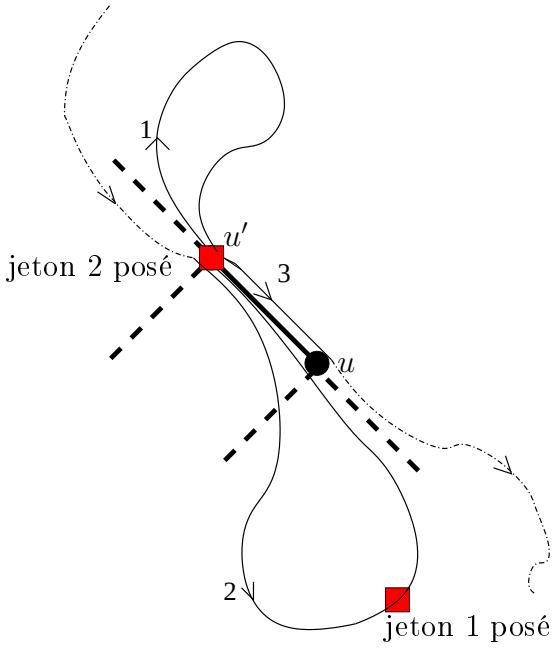

FIG. 5.2 – Le dernier passage de A en u'

donc le dernier état r où A passe en u avec le jeton 1 levé. Pour actualiser cet état

sans remonter au père de u', A' doit également mémoriser l'ensemble des couples d'états R correspondant aux boucles de A dans le contexte pointé en u telles que le jeton 1 de A est posé sur un nœud du sous-arbre enraciné en u'.

A l'aide de ces informations finies, A' simule la 1-exécution de A tout en déplaçant pas à pas le jeton 2 le long du chemin de u_p à u_ℓ comme nous le décrivons ci-dessous.

Au départ, A' est en u_p et vient de poser son jeton sur ce nœud, il stocke alors dans sa mémoire en r l'état courant q_p de la première configuration de cette 1-exécution de A. En simulant plusieurs fois de manière non-déterministe A à partir du nœud u_p, l'automate A' calcule et stocke dans R un ensemble de couples d'états (p, q) correspondant à des boucles de A dans le contexte pointé en u_p lorsque le jeton 1 de A est posé sur un descendant de u_p. L'automate A' simule ensuite A dans l'arbre à partir de l'état q_p. Au cours de cette simulation, à chaque fois que A' passe par le nœud u_p et que le jeton 1 est levé, A' devine si la configuration correspondante de A est celle de son dernier passage sur ce nœud dans une 1-configuration. Si A' devine ainsi que A ne repassera plus en u_p avec le jeton 1 levé, A' retient l'état de cette configuration, le mémorise à la place de r, choisit un de ses deux fils comme prochain nœud dans le chemin de u_p à u_ℓ et déplace le jeton 2 sur ce nœud. Notons que si, au cours de cette dernière simulation à partir de l'état q_p, l'automate A veut lever le jeton 2 à distance, l'automate A' rejette. En effet, dans ce cas, A' a fait une erreur car il n'a pas deviné le dernier passage de A en u_p dans une 1-configuration.

Considérons maintenant que A' vient de déplacer le jeton 2 d'un nœud u' vers un de ses fils u désigné comme le successeur de u' dans le chemin de u_p à u_ℓ. On note v le frère de u. La relation R mémorisée dans l'état de A' contient alors les couples d'états correspondant aux 0-exécutions de A dans le contexte pointé en u' qui sont des boucles entre deux 0-configurations pour lesquelles la tête de lecture de A est sur u', le jeton 2 sur u_p et le jeton 1 sur un descendant de u'. L'état r qui est aussi mémorisé par A' est l'état du dernier passage de A dans une 1-configuration où la tête de lecture est en u' et le jeton en u_p.

L'automate A' procède alors de la manière suivante :

1. L'automate A' actualise R. Pour cela, il simule A dans le sous-arbre enraciné en v et complète ainsi l'ensemble des couples d'états correspondant à une 0-exécution de u en u dans le contexte pointé en u.
2. L'automate A' simule ensuite A à partir de l'état r qu'il a mémorisé et de u' en restant dans le sous-arbre enraciné en u. Si, au cours de cette simulation, A choisit d'aller en u' après avoir posé le jeton 1 dans le sous-arbre enraciné en u, l'automate A' utilise R pour simuler la 0-exécution de A dans le contexte pointé en u. Si A essaye de lever ce jeton à partir de u, A' peut simuler cette transition en levant son jeton qui est en u et termine ainsi la simulation de la 0-exécution de A.
3. A' devine au cours de la simulation de 2 la dernière fois que A visite u quand le jeton 1 est levé. Il retient l'état de ce dernier passage de A dans une 1 configuration. Cet état de A est stocké en r dans l'état de A'.
4. A' choisit enfin le successeur de u dans le chemin de u_p à u_ℓ, puis il lève son jeton 2 en u et le repose sur le nouveau nœud qu'il a choisi.

Notons que si au cours de la simulation de 2, A quitte le sous-arbre enraciné en u

sans avoir posé le jeton 1 ou si A lève le jeton 2 à partir d'un nœud distinct de u, l'automate A' rejette.

L'automate non-déterministe à 2 jetons du modèle faible A' simule ainsi les exécutions de A où le jeton 2 est levé à distance et les deux automates A et A' sont équivalents. Ce cas illustre la deuxième idée principale de la construction qui est la suivante. Lorsque le jeton 2 de A' est sur un nœud u du chemin de u_p à u_ℓ, A' simule A tant que A reste dans le sous-arbre enraciné en u qui est la partie de l'arbre où le jeton 2 de A n'est pas posé. Dans l'autre partie de l'arbre, A ne peut plus être simulé directement par A' car la position du jeton 2 de A est perdue mais, avant de poser son jeton en u, A' a stocké dans son état des informations lui permettant de continuer la simulation de A sans visiter la partie de l'arbre où se trouve u_p qui est la position du jeton 2 de A.

5.2 Comportement d'un wPTA

Dans toute cette section, nous considérons un automate à k jetons du modèle faible que nous notons A et nous formalisons la notion intuitive de comportement de cet automate à jetons en introduisant la relation de simulation entre les arbres et les contextes. Intuitivement, un arbre t est simulé par un arbre s si toutes les boucles de l'automate dans t sont aussi dans s. Nous définissons une relation d'équivalence entre les arbres qui se simulent l'un l'autre et le comportement de A pour un arbre est la classe d'équivalence de cet arbre pour cette relation. En 5.2.1 nous définissons formellement les comportements et nous prouvons qu'il existe un nombre fini de comportements pour A. Nous montrons ensuite en 5.2.2 comment les comportements peuvent être composés.

5.2.1 Définition du comportement d'un wPTA

Pour décomposer une exécution d'un automate d'arbre à jetons, nous avons défini en 2.25 les i-exécutions où i est un entier inférieur ou égal au nombre de jetons de l'automate. Rappelons qu'une i-configuration est une configuration où les jetons placés dans l'arbre sont les jetons numérotés de $(i+1)$ à k et qu'une i-exécution est une exécution d'une i-configuration à une i-configuration au cours de laquelle le jeton $(i+1)$ n'est jamais levé. Nous considérons maintenant les exécutions d'un automate à jetons du modèle faible qui sont des boucles et nous définissons les i-boucles.

Définition 5.1 *Soient t un arbre et $i \in [0,k]$. Une i-boucle dans t est une i-exécution d'une configuration (q,v,f) à une configuration (p,v,f) où p et q sont des états, v est un nœud de t et f est une affectation des jetons de $[i+1,k]$. Cette i-boucle est alors une i-boucle d'arbre, respectivement une i-boucle de contexte, si toutes les i-configurations de cette boucle ont leur nœud courant dans le sous-arbre $t_{|v}$, respectivement dans le contexte $C_{t,v}$.*

Au cours d'une i-boucle d'arbre, l'automate A peut poser le jeton i dans le sous-arbre, revenir à la racine du sous arbre et démarrer une $(i-1)$-boucle de

contexte. Les i-boucles d'arbre dépendent donc des j-boucles de contexte pour $j < i$ et les j-boucles de contexte dépendent de même des h-boucles d'arbre pour $h < j$. Le comportement d'un automate pour un sous-arbre de t et le comportement de l'automate pour le contexte de t correspondant sont ainsi liés.

Nous avons défini en 2.16 la composition d'un arbre et d'un contexte et en 3.6 les arbres et les contextes marqués. Nous voulons maintenant composer un arbre et un contexte marqués.

Définition 5.2 *Soient f et g des affectations des jetons de A respectivement sur un arbre t et sur un contexte C et soit $i \in [0, k]$. On note J_1 et J_2 les domaines respectifs de f et de g. Les affectations f et g sont dites i-compatibles si*

- *$J_1 \cup J_2 = [i+1, k]$ et*
- *$\forall j \in J_1 \cap J_2$, $f(j)$ est la racine de l'arbre t et $g(j)$ le trou du contexte C.*

Un arbre marqué (t, f) et un contexte marqué (C, g) sont i-compatibles si t et C sont compatibles et si f et g sont i-compatibles. Dans ce cas, on appelle composition de (C, g) et (t, f) et on note $(C, g)[(t, f)]$ l'arbre $(C[t], f' \cup g)$ où f' est l'affectation des jetons de J_1 telle que pour tout jeton i de J_1, $f'(i)$ est le nœud de $C[t]$ qui correspond au nœud $f(i)$ de t.

Un automate à k jetons accepte un arbre si et seulement s'il existe une k-boucle de la racine à la racine commençant dans un état initial et terminant dans un état final. Nous considérons donc que le comportement d'un automate à jetons dans un arbre est caractérisé par les boucles et plus précisément par les couples d'états correspondant au premier et au dernier état de chaque boucle.

Définition 5.3 *Soient $i \in [0, k]$, $t_f = (t, f)$ et $C_g = (C, g)$ un arbre et un contexte marqués i-compatibles. On note f' l'affectation qui correspond à f dans $C_g[t_f]$ et $\beta_i(C_g, t_f)$ l'ensemble des paires (p, q) telles qu'il existe une i-boucle dans $C[t]$ de la configuration $(p, v, f' \cup g)$ à la configuration $(q, v, f' \cup g)$ où v est le trou de C. On note également $\beta_i^{arb}(C_g, t_f)$ le sous-ensemble de $\beta_i(C_g, t_f)$ qui correspond à des i-boucles d'arbre et $\beta_i^{con}(C_g, t_f)$ le sous-ensemble de $\beta_i(C_g, t_f)$ qui correspond à des i-boucles de contexte.*

Remarque 5.1 Il est clair que $\beta_i^{con}(C_g, t_f) \cup \beta_i^{arb}(C_g, t_f) \subseteq \beta_i(C_g, t_f)$. Cette inclusion est généralement stricte car dans une i-boucle, l'automate peut se rendre dans le sous-arbre et ensuite dans le contexte sans avoir encore posé le jeton i. Par contre, nous pouvons décomposer une i-boucle en une suite de i-boucles d'arbre et de i-boucles de contexte.

Notation 5.1 *A partir de maintenant, dans cette section les arbres et les contextes que nous considérons sont généralement marqués. Comme nous pouvons considérer un arbre marqué comme un arbre sur un nouvel alphabet qui code la présence des jetons, nous utilisons pour désigner les arbres et les contextes marqués les notations t et C utilisées pour les arbres et les contextes.*

Nous formalisons maintenant l'idée que, sur un arbre s, A a tous les comportements qu'il peut avoir sur un arbre t et nous définissons la simulation pour les arbres et les contextes.

Remarque 5.2 Le concept de i-simulation d'arbre ou de contexte défini ici ne doit pas être confondu avec la notion de simulation d'automate utilisée dans les sections précédentes dans le sens habituel qu'elle a en informatique.

Définition 5.4 *Soient $i \in [0,k]$, $J \subseteq [i+1,k]$, t_1 et t_2 deux arbres J-marqués ayant la même affectation de jetons à la racine. On dit que t_1 est i-simulé par t_2 si pour tout contexte marqué i-compatible C on a $\beta_i^{arb}(C,t_1) \subseteq \beta_i^{arb}(C,t_2)$. Si pour tout $j \in [0,i]$ t_1 est j-simulé par t_2, on dit que t_1 est i^*-simulé par t_2.*

On définit de manière analogue la i-simulation de contextes marqués : étant donnés C_1 et C_2 deux contextes J-marqués ayant la même affectation de jetons au trou, on dit que C_1 est i-simulé par C_2 si, pour tout arbre marqué t i-compatible, $\beta_i^{con}(C_1,t) \subseteq \beta_i^{con}(C_2,t)$ et si, pour tout $h \in [0,i]$, C_1 est h-simulé par C_2, on dit que C_1 est i^*-simulé par C_2.

Deux sous-arbres J-marqués ayant la même affectation de jetons à la racine, respectivement deux contextes J-marqués ayant la même affectation de jetons au trou, sont i-équivalents s'ils se i-simulent l'un l'autre et ils sont i^*-équivalents s'ils se i^*-simulent l'un l'autre.

Remarque 5.3 La i-équivalence et la i^*-équivalence sont comme leur nom l'indique des relations d'équivalence.

Remarque 5.4 La simulation n'étant définie que pour deux arbres ou deux contextes marqués par le même domaine d'affectation J et ayant la même affectation de jetons à la racine ou au trou, nous omettrons désormais par souci de concision ces précisions dans les énoncés de simulation ou d'équivalence. On remarque que si t i-simule s, tout contexte i-compatible avec s est i-compatible avec t et réciproquement. Il en est de même pour des contextes.

Notation 5.2 *Nous utilisons à partir de maintenant les notations τ et γ pour désigner respectivement une classe d'équivalence d'arbre et une classe d'équivalence de contexte. On note $\tau_i(t)$ la classe d'équivalence du sous-arbre marqué t pour la relation de i^*-équivalence et $\gamma_i(C)$ celle du contexte marqué C. On appelle une telle classe d'équivalence une classe de i^*-équivalence.*

Nous établissons dans les lemmes suivants des propriétés relatives aux comportements.

Lemme 5.1 *Soient $i \leq k$, t et s deux arbres marqués. Si t i^*-simule s alors pour tout contexte marqué i-compatible C, on a $\beta_i(C,s) \subseteq \beta_i(C,t)$.*

Preuve. Nous procédons par induction sur $i \in \mathbb{N}$. Une i-boucle dans $C[s]$ se décompose en une suite de i-boucles d'arbre et une suite de i-boucles de contexte. Les i-boucles d'arbres dans $C[s]$ sont associés à des i-boucles dans $C[t]$ avec les mêmes couples d'états par définition de la i-simulation. Il reste à montrer qu'une i-boucle de contexte de $C[s]$ a aussi sa correspondante dans $C[t]$. On considère alors ρ une i-boucle de contexte de $C[s]$. Le cas $i = 0$ est trivial car il n'existe pas de 0-configuration en dehors de C dans une 0-boucle de contexte de $C[s]$.

Si $i > 0$, on décompose ρ en une suite de sous-exécutions $\rho_0, \pi_1, \rho_1, \cdots, \pi_m, \rho_m$ qui vérifient les propriétés suivantes :

– pour $n \in [0,m]$, toutes les configurations de ρ_n sont telles que la tête de lecture et les jetons posés dont le numéro est inférieur ou égal à i sont dans le contexte C,
– pour $n \in [1,m]$ il existe $j \in [0,i-1]$ tel que π_n est une j-boucle d'arbre dans $C'[s]$ où C' est le contexte marqué obtenu à partir de C et des jetons $(j+1),\cdots,i$ posés et non levés au cours de $\rho_0,\cdots,\rho_{n-1}$.

Par induction, on a $\beta_j(C',s) \subseteq \beta_j(C',t)$. Ainsi, pour chaque $n \in [1,m]$, il existe une boucle π'_n dans $C'(t)$ qui correspond à π_n et comme pour chaque $n \in [0,m]$, ρ_n existe aussi dans $C[t]$, il existe une i-boucle de contexte ρ' dans $C[t]$ avec les mêmes premier et dernier états. □

Nous obtenons par dualité le lemme suivant :

Lemme 5.2 *Soient $i \leq k$, C et C' deux contextes marqués tels que C' i^*-simule C. Pour tout sous-arbre marqué t i-compatible, on a $\beta_i(C,t) \subseteq \beta_i(C',t)$.*

Notation 5.3 *Etant donné $i \in [0,k]$, nous associons maintenant à chaque classe de i^*-équivalence d'arbre τ un arbre marqué de cette classe que nous notons t_τ et nous choisissons de même pour chaque classe de i^*-équivalence de contexte γ un contexte marqué de cette classe que nous notons C_γ.*

Nous allons maintenant définir formellement le i-comportement pour un arbre t qui est une fonction permettant de calculer les i-boucles d'arbre dans un arbre $C[t]$ à partir des j-comportements pour le contexte C tels que $j < i$.

Définition 5.5 *Soient t un arbre, C un contexte et $i > 0$.*

Le i-comportement pour le contexte C noté B^i_C est une fonction qui associe à chaque classe de $(i-1)^*$-équivalence d'arbres τ l'ensemble de couples d'états $\beta_i^{con}(C,t_\tau)$.

Le i-comportement pour l'arbre t noté B^i_t est une fonction qui associe à chaque classe de $(i-1)^*$-équivalence de contexte γ l'ensemble de couples d'états $\beta_i^{arb}(C_\gamma,t)$.

Pour $i=0$, B^0_t est le triplet de fonctions $B^0_t = (B^0_{t,1}, B^0_{t,2}, B^0_{t,\epsilon})$ tel que $B^0_{t,\iota}$ est l'ensemble des 0-boucles d'arbres de t tels que la racine de t est considérée comme un fils gauche si $\iota = 1$, comme un fils droit si $\iota = 2$ ou bien comme la racine si $\iota = \epsilon$. De même, B^0_C est l'ensemble des 0-boucles de contexte de C.

Le i^*-comportement pour t noté $B^{i^*}_t$ est le $(i+1)$-uplet $(B^0_t,\cdots,B^i_t)$. De même le i^*-comportement pour C est le $(i+1)$-uplet $(C^0_t,\cdots,C^i_t)$. Si B est un i^*-comportement et $h \in [0,i]$, on note B^h la $(h+1)$-ème composante de B.

Remarque 5.5 Le i-comportement pour un arbre est bien sûr relatif à l'automate A du modèle faible considéré au début de la section. Si le choix de A est ambigü, on parle de i-comportement de A pour l'arbre t et on note $B^i_t[A]$ ce comportement.

On définit ci-dessous un ordre naturel sur les comportements.

Définition 5.6 *Etant donnés deux arbres t et s et un entier $i > 0$, $B^i_s \leq B^i_t$ si, pour toute classe de $(i-1)^*$-équivalence de contexte compatibles γ, on a $B^i_s(\gamma) \subseteq B^i_t(\gamma)$.*

$B^{i^*}_s \leq B^{i^*}_t$ signifie que pour tout $h \in [0,i]$ on a $B^h_s \leq B^h_t$.

De même, étant donnés deux contextes C et C' et un entier $i > 0$, $B^i_C \leq B^i_{C'}$ si, pour toute classe de $(i-1)^*$-équivalence d'arbre τ, on a $B^i_C(\tau) \subseteq B^i_{C'}(\tau)$ et on note $B^{i^*}_C \leq B^{i^*}_{C'}$ si, pour tout $h \in [0,i]$, on a $B^h_C \leq B^h_{C'}$.

Etant donnés deux arbres t et s, on a $B^0_s \leq B^0_t$ si pour tout $\iota \in \{1,2,\epsilon\}$,$B^0_{s,\iota} \subseteq B^0_{t,\iota}$ et étant donnés deux contextes C et C', on a $B^0_C \leq B^0_{C'}$ si $B^0_C \subseteq B^0_{C'}$.

Nous prouvons dans le lemme qui suit que les i-comportements pour des arbres, respectivement pour des contextes, que nous venons de définir correspondent aux i-classes d'équivalence d'arbre, respectivement de contexte.

Lemme 5.3 *Soient t et s deux arbres marqués et $i \in [0,k]$, s est i-simulé par t si et seulement si $B^i_s \leq B^i_t$.*

Preuve. L'implication de gauche à droite est donnée par la définition de i-simulation. Montrons maintenant l'implication de droite à gauche. Soit C un contexte marqué i-compatible avec s et donc aussi avec t. Soit γ la $(i-1)^*$-classe d'équivalence de C. On doit montrer que chaque couple d'états (p,q) qui correspond à une i-boucle d'arbre dans $C[s]$ correspond également à une i-boucle d'arbre dans $C[t]$. Par définition de B^i_s, le couple (p,q) appartient à $B^i_s(\gamma)$ et donc à $B^i_t(\gamma)$ puisque $B^i_s \leq B^i_t$. Ainsi (p,q) correspond à une i-boucle d'arbre de $C_\gamma[t]$. D'après le lemme 5.2, ce couple d'états correspond aussi à une i-boucle de $C[t]$ et on en déduit que s est i-simulé par t. □

Le lemme suivant se déduit du lemme ci-dessus par dualité.

Lemme 5.4 *Soient C et C' deux contextes marqués et $i \in [0,k]$, C est i-simulé par C' si et seulement $B^i_C \leq B^i_{C'}$.*

Nous montrons maintenant que l'ensemble des comportements de A est fini.

Lemme 5.5 *Pour tout $i \leq k$ (et quelque soit l'automate A à k jetons fixé au début de cette partie), il existe un nombre fini de i-comportements (de A) pour les arbres et un nombre fini de i-comportements pour les contextes.*

Preuve. Nous procédons par induction sur i. Pour $i = 0$, le nombre de 0-comportements pour arbres est borné par le nombre d'ensembles de couples d'états au cube. Pour $i > 0$, par induction le nombre de j-comportements pour $j \in [0, i-1]$ est fini. D'après le lemme 5.3, le nombre de classes de $(i-1)^*$-équivalence est également fini. Ainsi comme les comportements pour des arbres gauches correspondent à des fonctions d'un ensemble fini à valeurs dans l'ensemble fini des couples d'états, le nombre de i-comportements pour arbres est fini. Par dualité, on obtient qu'il existe un nombre fini de i-comportements pour contextes.

□

Remarque 5.6 On déduit immédiatement du lemme ci-dessus qu'il existe un nombre fini de i^*-comportements pour les arbres.

D'après le lemme ci-dessus, un comportement d'un automate donné pour un sous-arbre ou pour un contexte est une information finie qu'on peut stocker dans l'état d'un nouvel automate.

5.2.2 Composition des comportements

Dans la transformation d'un automate à jetons du modèle fort en un automate à jetons du modèle faible équivalent que nous décrivons en 5.3, nous utilisons certaines propriétés des comportements. Nous définissons dans cette partie des opérations de composition sur les arbres et les contextes que nous étendrons aux comportements. Nous allons aussi montrer par exemple que, pour chaque entier i, le i-comportement pour un arbre marqué t dépend seulement des i-comportements pour les sous-arbres gauche et droit de t ainsi que de l'étiquette de la racine de t et des jetons posés sur cette racine.

Nous définissons ci-dessous différentes façons de composer des arbres et des contextes marqués.

Définition 5.7 *Soit un entier $i \in [0, k]$. On considère également R un sous-ensemble de jetons disjoint de $[i+1, k]$, a une lettre de Σ, t_1 et t_2 des arbres marqués respectivement par P_1 et P_2.*

On note $\mathfrak{C}(a, R, t_1, t_2)$ l'arbre marqué constitué d'une racine étiquetée par a et marquée par les jetons de R qui a respectivement t_1 et t_2 comme sous-arbres gauche et droit. La figure 5.3 illustre cette définition.

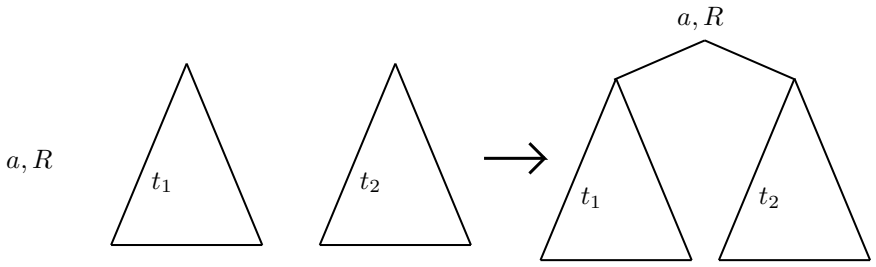

FIG. 5.3 – Représentation de la composition $\mathfrak{C}(a, R, t_1, t_2)$

On note $\mathfrak{C}(C, t_1, a, R)$ le contexte marqué obtenu à partir de C en remplaçant le trou de C par le contexte C' dont la racine est étiquetée par la lettre de Σ correspondant à l'étiquette du trou de C et dont le sous-arbre gauche est t_1 et le sous arbre droit est réduit à une feuille marquée par R et étiquetée par $(a, *)$. Cette feuille est donc le trou du contexte C' et du contexte $\mathfrak{C}(C, t_1, a, P_2)$.

La figure 5.4 montre comment on obtient $\mathfrak{C}(C, t_1, a, R)$.

Similairement, on note $\mathfrak{C}(C, a, R, t_2)$ le contexte marqué obtenu à partir de C en remplaçant le trou de C par le contexte dont la racine est étiquetée par une lettre de Σ correspondant à celle du trou de C et dont le sous-arbre droit est t_2 et le sous arbre gauche réduit à une feuille marquée par R et étiquetée par $(a, *)$. Cette feuille est donc le trou du contexte $\mathfrak{C}(C, a, R, t_2)$.

La figure 5.5 représente $\mathfrak{C}(C, a, R, t_2)$.

Nous allons monter que les compositions de contexte et d'arbre sont compatibles avec la relation de simulation. Nous définissons d'abord formellement une opération monotone.

Définition 5.8 *Etant donnés des ensembles ordonnés X, Y, Z et une fonction f de $X \times Y \to Z$, la fonction f est une opération monotone si pour tout couple d'éléments*

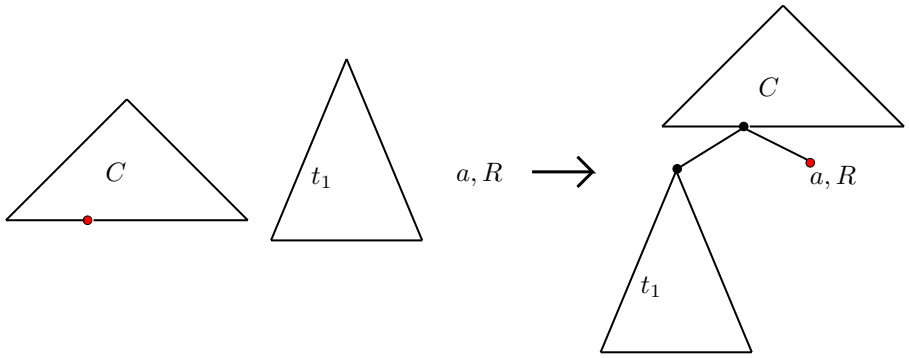

FIG. 5.4 – Représentation de la composition $\mathfrak{C}(C, t_1, a, R)$

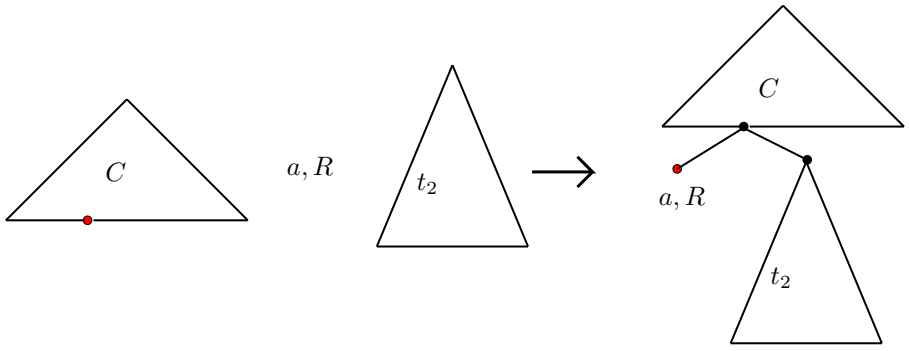

FIG. 5.5 – Représentation de la composition $\mathfrak{C}(C, a, R, t_2)$

de X (a, a') tels que $a \leq a'$ et pour tout couple d'élements de Y (b, b') tels que $b \leq b'$, on a $f(a, b) \leq f(a', b')$.

La i-simulation et la i^* simulation définies en 5.4 sont des relations d'ordre. Le lemme suivant donne un sens à la composition de comportements.

Lemme 5.6 *Les opérations de composition sont monotones par rapport à la i^*-simulation.*

Plus formellement, soient $i \in [0, k]$ R, P_1, P_2 des sous-ensembles disjoints de jetons de $[i+1, k]$, a une lettre de Σ. Si t_1 et t'_1 sont des arbres marqués par P_1 tels que $B^i_{t_1} \leq B^i_{t'_1}$, si t_2 et t'_2 sont des arbres marqués par P_2 tels que $B^i_{t_2} \leq B^i_{t'_2}$ et si C_1 et C'_1 sont des contextes marqués par R tels que $B^i_{C_1} \leq B^i_{C'_1}$, on a alors $B^i_{\mathfrak{C}(a,R,t_1,t_2)} \leq B^i_{\mathfrak{C}(a,R,t'_1,t'_2)}$, $B^i_{\mathfrak{C}(C,t_1,a,P_2)} \leq B^i_{\mathfrak{C}(C',t'_1,a,P_2)}$ et $B^i_{\mathfrak{C}(C,a,P_1,t_2)} \leq B^i_{\mathfrak{C}(C',a,P_1,t'_2)}$

Preuve. On procède par induction sur i. Pour $i = 0$, les inégalités sont évidentes car les 0-boucles se composent de manière simple : par exemple, les 0-boucles d'arbre dans $\mathfrak{C}(a, R, t_1, t_2)$ dépendent des transitions partant de la racine de ce sous-arbre et des 0-boucles dans t_1 et t_2. On considère maintenant $i > 0$. Nous allons montrer l'inégalité $B^i_{\mathfrak{C}(a,R,t_1,t_2)} \leq B^i_{\mathfrak{C}(a,R,t'_1,t'_2)}$, les autres se montrant de manière similaire. On note $t = \mathfrak{C}(a, R, t_1, t_2)$ et $t' = \mathfrak{C}(a, R, t'_1, t'_2)$. Soit γ une classe de i^*-équivalence de contexte, on doit montrer que $B^i_t(\gamma) \subseteq B^i_{t'}(\gamma)$, ce qui revient à montrer que $\beta^{arb}_i(C_\gamma, t) \subseteq \beta^{arb}_i(C_\gamma, t')$. Soit alors ρ une i-boucle d'arbre de $C_\gamma[t]$. On considère alors les trois cas suivants pour ρ :

1. le jeton i n'est pas posé,
2. le jeton i est posé sur la racine de t,
3. le jeton i est posé dans l'un des deux sous-arbres t_1 ou t_2.

Le premier cas est similaire au cas $i = 0$.

Dans le deuxième cas, ρ correspond au placement du jeton i suivi d'une suite de $(i-1)$-boucles de contexte et d'arbre dans $C_\gamma(t)$ et de la levée du jeton i. Par induction, on a $B^{i-1}_{\mathfrak{C}(a,R\cup i,t_1,t_2)} \leq B^{i-1}_{\mathfrak{C}(a,R\cup i,t'_1,t'_2)}$. Les $(i-1)$-boucles d'arbres se retrouvent ainsi dans $\mathfrak{C}(a, R\cup i, t'_1, t'_2)$ et ρ se retrouve donc dans $C_\gamma[t']$.

Dans le troisième cas, supposons par exemple que le jeton i est posé dans t_1, l'autre sous-cas étant similaire. Pour tout $j<i$, on a $B^j_{\mathfrak{C}(C_\gamma,a,R,t_2)} \leq B^j_{\mathfrak{C}(C_\gamma,a,R,t'_2)}$ par induction. Ainsi $\mathfrak{C}(C_\gamma, a, R, t_2)$ est $(i-1)^*$-simulé par $\mathfrak{C}(C_\gamma, a, R, t'_2)$ et comme $B^i_t \leq B^i_{t'}$, les i-boucles de $\mathfrak{C}(C_\gamma, a, R, t_2)[t_1]$ correspondent à des i-boucles de $\mathfrak{C}(C_\gamma, a, R, t'_2)[t'_1]$. On retrouve ainsi ρ dans $C_\gamma[t']$. □

La relation de i^*-équivalence est ainsi une congruence pour la composition. Nous pouvons déduire alors de l'opération de composition sur les contextes et les arbres une opération de composition sur les comportements pour les contextes et pour les arbres que nous notons de la même façon.

Notation 5.4 *Si B est le i-comportement pour un contexte C et B_1 et B_2 les i-comportements pour des arbres t_1 et t_2, a une lettre de l'alphabet et R un sous-ensemble des jetons de $[i+1, k]$ alors on note $\mathfrak{C}(a, R, B_1, B_2)$ le i-comportement pour l'arbre $\mathfrak{C}(a, R, t_1, t_2)$, $\mathfrak{C}(B, B_1, a, R)$ le i-comportement pour le contexte $\mathfrak{C}(C, t_1, a, P_2)$ et $\mathfrak{C}(B, a, R, B_2)$ le i-comportement pour le contexte $\mathfrak{C}(C, a, R, t_2)$. Cette définition a un sens d'après le lemme précédent.*

On peut définir la composition de i^*-comportements : par exemple, si B_1 et B_2 les i^*-comportements pour un arbre t_1 et pour un arbre t_2, a une lettre de l'alphabet et R un sous-ensemble des jetons de $[i+1, k]$ alors $\mathfrak{C}(a, R, B_1, B_2)$ est le i^*-comportement pour l'arbre $\mathfrak{C}(a, R, t_1, t_2)$.

5.3 Passage du modèle fort au modèle faible dans le cas non-déterministe

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

Théorème 5.1 *Soit A un automate à jetons du modèle fort. On peut construire un automate du modèle faible équivalent avec le même nombre de jetons. Les langages reconnus par un automate du modèle fort à k jetons sont donc aussi reconnus par un automate du modèle faible à k jetons.*

Dans la section 5.1, nous avons montré ce théorème pour les automates à 1 et 2 jetons. Nous prouvons dans cette section le théorème ci-dessus dans le cas général c'est-à-dire pour un entier k quelconque. Nous construisons d'abord en 5.3.1 un automate à jetons du modèle faible qui devine le comportement d'un automate à jetons donné. Nous expliquons ensuite en 5.3.2 comment affaiblir un jeton d'un automate du modèle fort. Affaiblir un jeton signifie transformer l'automate de manière à ce

qu'il puisse lever ce jeton seulement lorsqu'il est dessus. Finalement nous montrons le théorème 5.1 et nous comparons la taille de l'automate du modèle faible construit par rapport à la taille de l'automate du modèle fort donné.

Le corollaire ci-dessous est une conséquence immédiate des théorèmes 5.1 et 4.2

Corollaire 5.1 *La classe des langages d'arbres reconnus par un automate à jetons non-déterministe du modèle faible est la classe des langages définis par la logique* FO + posTC.

5.3.1 Deviner un comportement

Dans la section 5.1, l'automates à jetons du modèle faible à 2 que nous construisons calcule sur un arbre t et stocke dans son état un ensembles de couples d'états qui sont en fait le comportements pour le contexte $C_{t,u}$ où u est le nœud marqué par le jeton qui a le numéro le plus grand. Nous montrons dans le lemme qui suit de quelle manière un automate du modèle faible à k jetons peut deviner le comportement d'un automate A faible donné à k jetons où k est un entier quelconque.

Lemme 5.7 *Soient $i \in [0,k]$, A un automate du modèle faible à k jetons et B un i-comportement de A pour un arbre. Il existe un* wPTA$_i$ *A' qui reconnaît les arbres marqués t tels que $B^i_t[A] \geq B$. De même, si B' est un i-comportement de A pour un contexte, il existe un* wPTA$_i$ *A' qui reconnaît les contextes marqués C tels que $B^i_C[A] \geq B'$.*

Preuve. Nous montrons ce lemme pour les comportements pour des arbres. Le cas des comportements pour des contextes se démontre de manière similaire. On procède par induction sur i. Le cas $i = 0$ est facile : pour un arbre marqué t' donné, A' lance plusieurs simulations de A et vérifie ainsi l'existence d'une boucle pour chaque couple de B.

Pour $i > 0$, $B^i_t[A] \geq B$ signifie que pour chaque classe de $(i-1)^*$-équivalence de contexte γ , on a $\beta^{arb}_i(C_\gamma, t) \geq B(\gamma)$. On a vu dans la proposition 2.3 que la classe des automates du modèle faible à i jetons est fermée par intersection. Il nous suffit ainsi de prouver que pour chaque classe de $(i-1)^*$-équivalence γ et pour chaque couple (p,q) d'états de A, il existe un automate du modèle faible à i jetons A'' qui reconnaît les arbres marqués t tels que $(p,q) \in \beta^{arb}_i(C_\gamma, t)$.

On fixe donc γ une telle classe de $(i-1)^*$ équivalence, ce qui revient d'après le lemme 5.3 à se donner pour chaque $j \leq i-1$ un j-comportement B^j de A pour contexte qui est une information finie d'après le lemme 5.5. On procède maintenant de manière similaire à la preuve du lemme 5.1. Une i-boucle ρ de A pour un arbre peut être décomposée en une suite $\rho_0, \pi_1, \rho_1, \cdots, \pi_m, \rho_m$ telle que, pour tout $h \in [1,m]$, π_h est une j-boucle de contexte où $j < i$ et, pour tout $h \in [0,m]$, ρ_h est une exécution où la tête de lecture reste dans t. L'automate A'' que l'on construit se comporte exactement comme A dans t pour simuler une sous-exécution ρ_h. A chaque fois que A décide de quitter t à partir d'un état p', c'est à dire au début de chaque j-boucle π_h, A'' choisit un état q' de A tel que $(p',q') \in \beta^{arb}_j(C_\gamma, t')$ où t' est l'arbre marqué obtenu à partir de t et des jetons $(j+1), \cdots, i$ qui ont été posés par A'' pendant la simulation de $\rho_0, \pi_1, \rho_1, \cdots, \rho_{h-1}$. Un tel état q' peut être

calculé à partir de p' et de $B^j(\tau_j(t'))$. Cependant A'' ne connaît pas la classe de j^*-équivalence $\tau_j(t')$. Pour remédier à cela, A'' devine une classe de j^*-équivalence τ_j, choisit un état q' tel que (p', q') est bien dans $B^j(\tau_j)$, puis par induction A'' vérifie que pour chaque $j' \leq j$, $B^{j'}_{t'}[A] \geq B^{j'}$ où $B^{j'}$ est le j'-comportement associé à la classe de j-équivalence τ_j. D'après le lemme 5.3, cette vérification garantit que (p', q') est une j-boucle de contexte dans $C_\gamma[t']$. De plus, toutes les i-boucles d'arbres de A correspondant au couple (p, q) peuvent être ainsi simulées par A''. □

Remarque 5.7 L'automate A' a besoin d'un grand nombre d'états pour mémoriser le comportement qu'il calcule de manière non-déterministe en simulant l'automate A.

Remarque 5.8 Dans la preuve ci-dessus, si deux arbres marqués s et t sont tels que les i-comportements de A pour s et t sont égaux, les i-comportements de A' pour s et t sont aussi égaux.

Cette dernière remarque sera utilisée dans une preuve du chapitre 6.

5.3.2 Affaiblir un jeton

Dans la section 5.1, nous avons considéré des automates à jetons du modèle fort qui ont un comportement fort uniquement pour le jeton avec le numéro le plus grand. Il est en effet facile de se ramener à ce cas par induction puisqu'un automate à k jeton se comporte comme un automate à $(k-1)$ jetons une fois que le jeton k est posé.

Dans le cas général nous montrons par induction sur le nombre des jetons que l'automate peut lever à distance que le modèle faible et le modèle fort sont équivalents. Nous introduisons aussi un modèle intermédiaire d'automates à jetons qui peut lever certains jetons à distance et qui se comporte comme un automate du modèle faible pour les autres jetons : ce sont *les automates n-faibles* où n est un entier inférieur ou égal au nombre de jetons.

Définition 5.9 *Soit $n \in [0, k]$. Un automate à k jetons du modèle fort est n-faible si les jetons de $[1, n]$ peuvent être levés seulement si l'automate est dessus. Ces jetons sont dits faibles et les jetons de $[n+1, k]$ pouvant être levés à partir de n'importe quel nœud sont dits forts.*

Remarque 5.9 Un automate à k-jetons k-faible est un automate du modèle faible.

Remarque 5.10 Un automate A à k jetons n-faible se comporte comme un automate faible pour les j-exécutions avec $j \leq n$, c'est-à-dire lorsque les jetons de $[n+1, k]$ sont posés et fixés. On peut alors considérer ses j-comportements où $j \leq n$.

Lemme 5.8 *Soient $n \in [0, k-1]$ et A un automate à k jetons n-faible. On peut construire A' un automate à k jetons $(n+1)$-faible équivalent.*

Preuve. Lorsque que le jeton $(n+1)$ n'est pas posé, A se comporte comme un automate $(n+1)$-faible. On considère donc π une $(n+1)$-exécution de A entre deux placements du jeton $(n+1)$. Au début de cette exécution, A pose le jeton

$(n+1)$ sur un nœud u_p. Il est alors dans la configuration $(p, u_p, \vec{v}_{n+2}, u_p)$ où p est l'état qu'il atteint et $\vec{v}_{n+2}$ les jetons dont le placement est fixé au cours de cette $(n+1)$-exécution. A partir de là, π continue avec une n-exécution ρ qui se termine dans une configuration $(q, u_\ell, \vec{v}_{n+2}, u_p)$. Enfin, le jeton $(n+1)$ est levé à partir du nœud u_ℓ et l'exécution π s'achève. Comme le jeton $(n+1)$ de A peut être levé à distance, les nœuds u_p et u_ℓ peuvent être distincts et π ne suit pas nécessairement la restriction caractérisant l'exécution d'un automate à jetons du modèle faible. Dans un arbre, il existe un unique chemin minimal entre deux nœuds. On désigne par $u_1, \cdots, u_m$ les nœuds du chemin minimal de u_p à u_ℓ avec $u_1 = u_p$ et $u_m = u_\ell$. On décrit ci-dessous comment un automate à k jetons $(n+1)$-faible simule π. On note alors t' l'arbre $[(n+1), k]$-marqué obtenu à partir de t en plaçant les jetons numérotés de k à $(n+2)$ selon $\vec{v}_{n+2}$ et le jeton $(n+1)$ en u_p. Comme il a été vu dans la section 5.1, l'automate A' déplace le jeton $(n+1)$ le long du chemin de u_p à u_ℓ et, pour simuler A dans le contexte où la position du jeton $(n+1)$ de A est perdue, A' calcule le comportement de A pour ce contexte. On considère maintenant uniquement le cas où u_p est un ancêtre de u_ℓ, les autres cas étant similaires. On remarque tout d'abord que dans ρ le jeton $(n+1)$ n'est pas levé. Au cours de cette exécution, A se comporte donc comme un automate du modèle faible.

On décompose ρ ainsi : $\rho = \pi_1, \rho_2, \pi_2, \cdots \rho_m, \pi_m$ tel que
- π_1 est une n-exécution de u_p à son successeur u_2 dans le chemin de u_p à u_ℓ,
- pour $j \in [2, m]$, ρ_j est une n-boucle de contexte dans $C_{t',u_j}[t'|u_j]$,
- pour $j \in [2, (m-1)]$, π_j est un simple déplacement de u_j à u_{j+1},
- π_m est une n-boucle d'arbre dans $C_{t',u_\ell}[t'|u_\ell]$.

Ainsi, pour $j \in [2, m]$, la dernière n-configuration dans ρ dont le nœud courant est u_j est la dernière configuration de ρ_j. On note r_j l'état courant de cette configuration de ρ_j. A' simule alors ρ de la manière suivante.

Tout d'abord, A' devine un $(n-1)^*$-comportement de A pour contexte que l'on note B, vérifie que $B \leq B^{n-1}_{C_{t',u_p}}$ comme dans le lemme 5.7 et rejette si ce n'est pas le cas. Ensuite, A' simule A jusqu'à ce qu'il devine r_1 l'état du dernier passage de A dans une n-configuration où la tête de lecture est en $u_1 (= u_p)$. Par la suite, A' n'a aucune difficulté à simuler les sous-exécutions π_j pour $j \in [1, m]$. La difficulté est de simuler pour chaque $j \in [2, m]$ les sous-exécutions ρ_j. En effet, la position u_p du jeton $(n+1)$ de A qui est dans le contexte C_{t',u_j} est perdue lorsque le jeton $(n+1)$ de A' est posé sur un nœud u_j tel que $j > 1$.

On montre aussi comment A' simule chaque ρ_j par induction sur $j \in [1, m]$. Le cas $j = 1$ est décrit ci-dessus. On suppose maintenant qu'un $(n-1)^*$-comportement B de A pour contexte tel que $B \leq B^{n-1^*}_{C_{t',u_{j-1}}}$ a été calculé et stocké avec l'état r_{j-1} de A dans l'état de A'. On admet également que la tête de lecture de A' et son jeton $(n+1)$ sont sur le nœud u_{j-1}.

L'automate A' devine tout d'abord si u_j est dans le sous-arbre gauche ou le sous-arbre droit puisqu'on a supposé que u_ℓ est un ancêtre de u_p. On suppose par exemple que A' a deviné que u_ℓ est dans le sous-arbre gauche, l'autre sous-cas étant similaire. Le nœud u_j est alors le fils gauche de u_{j-1}. On note a_j l'étiquette de u_j, R_j l'ensemble de jetons posés sur ce nœud, t_j le sous-arbre $t'|u_j$ et t'_j le sous-arbre droit de u_{j-1}. L'automate A' devine un $(n-1)^*$-comportement B' de A et vérifie comme dans le lemme 5.7 si le $(n-1)^*$-comportement de A pour t'_j est plus grand que B'.

Si c'est bien le cas, A' retient dans son état le $(n-1)^*$-comportement B'' défini par $\mathfrak{C}(B, a_i, R_i, B')$ à la place de B. D'après le lemme 5.6, on a bien $B'' \leq B^{n-1^*}_{C_{t,u_j}}$. Ensuite, à partir du nœud u_{j-1} et de l'état r_{j-1}, l'automate A' simule une transition de A qui l'amène sur le nœud u_j, puis il continue la simulation de A jusqu'à ce qu'il devine le dernier passage de A en u_j dans une n-configuration. Il mémorise alors dans son état à la place de r_{j-1} l'état r_j de A qui correspond à ce dernier passage . Au cours de cette simulation, si A veut retourner sur le nœud u_{j-1} sans avoir posé le jeton n, A' rejette car il s'est trompé en supposant que l'état r_{j-1} correspondait à celui du dernier passage de A dans une n-configuration en u_{j-1}. Par contre, si, au cours de cette simulation, A veut repasser par le nœud u_{j-1} à partir d'un état p_j après avoir posé les jetons de n à n' avec $n' \leq n$, A' va calculer le dernier état q_j de la $(n'-1)$-boucle de contexte de A dans le contexte C_{t',u_j} de la manière suivante. L'automate A' devine une classe de $(n-2)^*$-équivalence τ, vérifie comme dans le lemme 5.7 si l'arbre marqué obtenu à partir de t_j et du placement des jetons numérotés de n à n' est dans une classe d'équivalence plus grande que τ. Si ce n'est pas le cas, A' rejette. Si c'est bien le cas, il choisit alors un état q_j tel que $(p_j, q_j) \in B''^{n'-1}(\tau)$ pour calculer la boucle de contexte de A qu'il ne peut plus simuler.

Lorsque le jeton $(n+1)$ de A' est en u_ℓ, A' simule A à partir de l'état r_m jusqu'à ce que A lève son jeton $(n+1)$ à partir de ce nœud. □

En affaiblissant un à un tous les jetons d'un automate du modèle fort à l'aide du lemme 5.8, nous transformons un automate du modèle fort en un automate non-déterministe du modèle faible équivalent avec le même nombre de jetons.

5.3.3 Complexité du passage du modèle fort au modèle faible

La construction décrite ci-dessus est cependant très coûteuse dès que le nombre de jeton est strictement supérieur à 1. Le cas d'un automate à 1 jeton est décrit en 5.1.1. L'automate construit a une taille linéaire par rapport à celle de l'automate de départ A car il stocke un seul état de A au cours de la simulation. Dans le cas d'un automate A à 2 jetons du modèle fort, on peut affaiblir le premier jeton et construire un automate A' avec un nombre linéaire d'états par rapport à A tel que le jeton 1 est faible et le jeton 2 est fort. Pour affaiblir le jeton 2, nous avons vu en 5.1.2 que l'automate construit doit stocker dans son état un ensemble de couples d'états de A. Le nombre d'états de l'automate du modèle faible construit est donc exponentiel par rapport à la taille de l'automate de départ. D'après la définition inductive des comportements, pour affaiblir le jeton n d'un automate nous construisons un automate dont la taille est une tour de $(n-1)$ exponentielles par rapport à la taille de A. Notre construction est donc non-élémentaire.

5.4 Passage du modèle fort au modèle faible dans le cas déterministe

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

Théorème 5.2 *Soit A un automate à jetons déterministe du modèle fort. On peut construire un automate déterministe du modèle faible équivalent avec le même nombre de jetons. Les langages reconnus par un automate déterministe du modèle fort à k jetons sont donc aussi reconnus par un automate déterministe du modèle faible à k jetons.*

Dans la section précédente, nous avons montré ce théorème pour les automates non-déterministes. Nous adapatons maintenant la preuve du cas non-déterministe au cas déterministe. Dans cette section nous considérons donc A un automate déterministe à k jetons du modèle fort. Nous présentons tout d'abord en 5.4.1 la construction d'un automate du modèle faible équivalent pour le cas d'un automate avec un unique jeton, c'est-à-dire le cas $k = 1$. Nous montrons ensuite en 5.4.2 comment un automate à jetons déterministe peut calculer de manière exacte les comportements de A. Nous nous servons alors des résulats du chapitre 3 pour résoudre les problèmes causés par les exécutions infinies. Nous montrons enfin comment adapter la construction que nous avons décrite au lemme 5.8 au cas déterministe et nous présentons une conséquence de ce théorème concernant la complémentation du modèle fort d'automate à jetons.

5.4.1 Calculer et non deviner

Dans le cas déterministe, l'automate que nous construisons doit calculer tout ce qui est deviné dans la construction du cas non-déterministe. Le non-déterminisme dans la preuve est utilisé à plusieurs reprises dans la construction de la preuve du lemme 5.8 qui montre comment affaiblir un jeton :

- pour deviner pas à pas le chemin du nœud sur lequel le jeton à affaiblir est posé u_p vers un nœud u_ℓ à partir duquel ce jeton peut être levé,
- pour deviner le dernier passage de A sur un nœud marqué de ce chemin dans une configuration où tous les jetons qui ont un numéro plus petit que celui du jeton qu'on veut affaiblir sont levés
- pour deviner des comportements et vérifier qu'ils sont inclus dans des comportements pour des sous-arbres et des contextes calculés partiellement.

Notons que dans le cas non-déterministe, l'automate A' du modèle faible que nous avons construit ne peut pas calculer de manière exacte un comportement de A pour un arbre t. En effet, A' peut simuler A un nombre arbitraire de fois pour calculer des boucles de A mais A' n'a aucun moyen de savoir quand il les a toutes calculées.

Dans cette section, l'automate déterministe A' que nous construisons calcule pas à pas le chemin de u_p le nœud sur lequel le jeton est posé au cours de l'exécution partant de la configuration initiale vers l'unique nœud u_ℓ à partir duquel le jeton posé en u_p est levé s'il existe. Il calcule également l'état du dernier passage de A dans une $(k-1)$-configuration où k est le numéro du jeton à affaiblir. La construction que nous décrivons ci-dessous pour transformer un automate à 1 jeton déterministe du modèle fort en un automate à 1 jeton déterministe du modèle faible illustre ces deux premiers points. Notons que dans le cas non-déterministe l'automate A' ne peut pas calculer le chemin de u_p à u_ℓ car ce chemin n'est pas unique.

Nous considérons donc maintenant que A est un automate à un jeton déterministe du modèle fort. Nous construisons ci-dessous un automate A' à un jeton déterministe

équivalent du modèle faible. Tant que A ne pose pas son jeton, il se comporte comme un automate cheminant déterministe et A' le simule sans difficulté. Lorsque A pose son jeton sur un nœud u_p, A' pose aussi son jeton sur ce nœud. Quand le jeton de A est fixé, A se comporte comme un automate cheminant. Nous pouvons alors considérer d'après le théorème 3.1 que lorsque le jeton 1 de A est fixé toutes les exécutions de A sont finies.

Ensuite, A' simule A jusqu'à ce qu'il lève le jeton à partir d'un nœud u_ℓ. Si u_ℓ n'existe pas c'est à dire si A ne lève pas son jeton alors A' rejette. Durant cette simulation de A à chaque fois que A' passe sur son jeton, il stocke dans son état la direction qu'il a prise ensuite et l'état correspondant à ce dernier passage. A la fin de cette simulation, lorsque A veut lever le jeton, A' a alors déterminé le successeur u_2 de u_p dans le chemin de u_p à u_ℓ et l'état de son dernier passage en u_p. A' retourne alors sur son jeton le déplace en u_2 et reprend la simulation de A à partir de u_2.

L'automate A' calcule ainsi pas à pas le chemin de u_p à u_ℓ et pose successivement son jeton sur tous les nœuds de ce chemin pour pouvoir enfin le lever en u_ℓ. A chaque étape, A' calcule comme ci-dessus l'état du dernier passage de A sur le nœud du chemin où est posé le jeton de A' et la direction qu'il a prise ensuite. Ainsi lorsque le jeton de A' est en u_ℓ et que A' a calculé l'état du dernier passage de A en u_ℓ il peut simuler la transition de A qui lève le jeton à partir de cet état et de ce nœud.

Etant donné un automate déterministe à 1 jeton du modèle fort à n états, nous pouvons ainsi construire un automate déterministe à 1 jeton du modèle faible équivalent à $O(n^2)$ états. En effet l'état de l'automate A' que nous construisons contient l'état courant de la simulation de A, la direction vers le nœud u_ℓ et l'état du dernier passage de A sur le jeton de A'.

Dans le cas d'un automate à plusieurs jetons l'automate déterministe du modèle faible que nous construisons doit également calculer de manière exacte les comportements de A dans des sous-arbres et des contextes et les exécutions infinies de A rendent ce calcul plus difficile.

5.4.2 Gérer les exécutions infinies dans le calcul du comportement

Nous montrons dans le lemme qui suit comment construire un automate qui calcule de manière exacte le comportement pour un arbre ou pour un contexte donné. L'automate A' que nous construisons va simuler l'automate déterministe A afin de stocker dans sa mémoire des couples d'états qui correspondent à des boucles de A. Nous utilisons les résultats du chapitre 3 pour que A' termine toujours son calcul même dans le cas où A admet des exécutions infinies.

Lemme 5.9 *Soient A un automate déterministe du modèle faible à k jetons, $i \in [0,k]$ et B^i un i-comportement de A pour un arbre. Il existe un automate déterministe du modèle faible à i jetons A' qui reconnaît les arbres marqués t tels que $B^i_t[A] = B^i$. De même, si B^i est un i-comportement de A pour un contexte il existe un automate déterministe du modèle faible à i jetons A'' qui reconnaît les contextes marqués C tels que $B^i_C[A] = B^i$*

Preuve. On procède par induction sur i. Le cas $i = 0$ est facile : A' simule A sur l'arbre pris en entrée et vérifie pour chaque couple d'états qu'une boucle correspondant à ce couple existe dans l'arbre si et seulement si ce couple appartient à B^0. Cette vérification peut être effectuée car on a vu dans le chapitre 3 qu'on peut considérer sans perte de généralité que A' est à exécutions finies.

Pour $i > 0$ vérifier que $B^i_t = B^i$ revient à vérifier que $\beta^{arb}(C_\gamma, t) = B^i(\gamma)$ pour chaque classe γ de $(i-1)^*$-équivalence. Comme la classe des langages reconnus par un automate à i-jetons est fermée par les opérateurs booléens il suffit de vérifier la proposition suivante :

Soient $i \in [1, k]$, γ une classe de $(i-1)^*$-équivalence de contexte et (p, q) un couple d'états de A, il existe un automate déterministe à i jetons du modèle faible qui reconnait les arbres marqués t tels que $(p, q) \in \beta^{arb}_i(C_\gamma[t])$

La preuve de cette propriété s'inspire de celle de la propriété correspondante du cas non-déterministe dans le lemme 5.7. On considère donc une i-boucle d'arbre ρ dans $C_\gamma[t]$ que l'on décompose en $\rho_0, \pi_1, \cdots, \pi_m, \rho_m$ où les π_h sont des j-boucles de contexte avec $j < i$ et les ρ_h sont des exécutions à l'intérieur de t. A' se comporte comme A dans t pour simuler une exécution π_h. Lorsque A essaye de quitter t à partir d'un état p' au début d'une j-boucle de contexte π_k, il doit calculer l'état q' s'il existe tel que $(p', q') = B^j(\tau_j(t'))$ où t' est l'arbre marqué obtenu à partir de t et des jetons de $[j+1, i]$ placés durant la sous-exécution $\rho_0, \pi_1, \cdots, \rho_{h-1}$ et $\tau_j(t')$ sa classe de j^*-équivalence. Si un tel état q' n'existe pas A' doit rejeter. La difficulté est alors de calculer $\tau_j(t')$.

Pour cela, A' considère successivement toutes les classes de j^*-équivalence possibles et vérifie pour chacune d'elles de la manière suivante s'il s'agit de celle de t'. Soit τ une j^*-classe d'équivalence d'arbre. D'après le lemme 5.3 τ est un ensemble de r-comportements B_r où r décrit l'ensemble $[0, j]$. Par induction, il existe un automate déterministe à r jetons du modèle faible A^r_β qui reconnaît les arbres marqués t tels que $B^r_t[A] = B_r$. A' peut maintenant simuler séquentiellement $A^0_\beta, \cdots, A^j_\beta$ pour déterminer si τ est la j^*-classe d'équivalence de t'. Le dernier problème est que A' peut boucler infiniment s'il rejette, c'est à dire si t' n'est pas dans la classe d'équivalence c. Ce problème peut être résolu en utilisant le résultat du chapitre 3 qui permet de supposer sans perte de généralité qu'un automate déterministe du modèle faible est à exécutions finies. □

5.4.3 Equivalence des deux modèles déterministes et conséquence

Nous avons vu en 5.4.1 comment calculer le chemin de u_p à u_ℓ pas à pas et déterminer pour chaque nœud u de ce chemin l'état du dernier passage de A en u dans une $(k-1)$-configuration. Il a été montré en 5.4.2 comment calculer un comportement de A à l'aide d'un automate déterministe. Nous pouvons maintenant adapter au cas déterministe la construction de 5.8 qui affaiblit un jeton .

Lemme 5.10 *Soient $0 \leq n < k$ et A un* DPTA$_k$ *n-faible. On peut construire A' un* DPTA$_k$ *$(n+1)$-faible équivalent.*

Preuve. Cette preuve suit les même lignes que la preuve du lemme 5.8 pour le cas non-déterministe en remplaçant le lemme 5.7 par le lemme 5.9. Dans la preuve du lemme 5.8, le non-déterminisme était utilisé d'une part pour deviner le chemin de u_p à u_ℓ et d'autre part pour deviner la classe de k^*-équivalence des sous-arbres rencontrés. Dans le cas déterministe le chemin de u_p à u_ℓ peut être calculé de la manière suivante. A' simule A jusqu'à ce qu'il atteigne u_ℓ le nœud à partir duquel A lève le jeton $k+1$ et, durant cette simulation, A' stocke dans son état la direction qu'il a prise la dernière fois qu'il est passé par le jeton $k+1$ et l'état correspondant au dernier passage de A sur ce nœud dans une n-configuration. Pour calculer la classe de k^*-équivalence d'un sous-arbre rencontré, A' considère successivement toutes les classes de k^*-équivalence et vérifie pour chacune d'entre elle s'il s'agit de celle du sous-arbre considéré de la manière suivante. A' simule l'automate du lemme 5.9 que l'on peut supposer à exécutions finies d'après le lemme 3.4.

□

En utilisant le lemme ci-dessus, il est possible d'affaiblir un à un tous les jetons d'un automate à jetons du modèle fort et nous construisons ainsi un automate du modèle faible équivalent avec le même nombre de jetons. Nous avons ainsi démontré le théorème 5.2 comme nous l'avions annoncé au début de la section. Comme dans le cas non-déterministe cette construction est non-élémentaire dès que le nombre de jetons est supérieur à 1.

Nous avons montré dans le chapitre 3 qu'on pouvait complémenter un automate déterministe du modèle fort en multipliant le nombre de jetons par 3. La construction de ce chapitre qui est beaucoup plus coûteuse en terme d'états que celle du lemme 3.5 montre que l'augmentation du nombre de jetons n'est pas nécessaire.

Corollaire 5.2 *La classe des langages reconnus par un automate à k jetons déterministe du modele fort est fermée par complément et par union.*

Preuve. Ce corollaire est une conséquence directe des théorème 5.2 et 3.4. □

Corollaire 5.3 *La classe des langages d'arbres reconnus par un automate à jeton déterministe du modèle fort est la classe des langages définis par la logique* FO+DTC.

Preuve. Ce corollaire est une conséquence directe des théorème 5.2 et 4.3. □

CHAPITRE

6

Hiérarchie des automates d'arbres à jetons

Sommaire

Dans le chapitre 2, nous avons montré avec le théorème 2.10 que, pour tout entier k, les automates à k jetons sur les mots reconnaissent la classe des langages réguliers sur les mots. Sur les mots, tous les modèles d'automates séquentiels que nous avons présentés dans le chapitre 2 ont donc le même pouvoir d'expression. Nous pouvons ainsi construire à partir d'un automate à jetons non-déterministe sur les mots un automate unidirectionnel déterministe qui reconnaît le même langage. Nous nous intéressons dans ce chapitre au pouvoir d'expression des automates d'arbres à jetons. Nous avons prouvé dans le chapitre 2 avec le théorème 2.6 que les langages reconnus par un automate d'arbres à jetons sont réguliers. Nous voyons à la fin de ce chapitre que la réciproque de cette proposition est fausse : nous présentons en effet en 6.3.2 un langage d'arbres régulier $\mathcal{L}^r$ qui ne peut pas être reconnu par un automate

d'arbres à jetons. Nous généralisons ainsi le résultat suivant montré dans [7] par M. Bojańczyk et T. Colcombet : les automates d'arbres cheminants ne reconnaissent pas tous les langages réguliers. Dans [7], un langage régulier $\mathcal{L}^p$ est défini et il est prouvé que $\mathcal{L}^p$ ne peut pas être reconnu par un automate d'arbres cheminant. Le langage $\mathcal{L}^p$ est cependant reconnu par un automate d'arbres à un jeton. En 6.1.2 nous construisons inductivement à partir de ce langage une famille de langages $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ qui sépare la classe des langages reconnus par un automate déterministe à n jetons de la classe des langages reconnus par un automate à $(n-1)$ jetons. Nous prouvons ainsi que le pouvoir d'expression des automates à jetons augmente avec le nombre de jetons dans le cas déterministe comme dans le cas non-déterministe. Dans ce chapitre, un autre résultat de M. Bojańczyk et T. Colcombet sur les automates cheminants est également étendu aux automates d'arbres à jetons : dans [6], il est prouvé que les automates cheminants ne peuvent pas tous être déterminisés. Ils définissent pour cela un langage $\mathcal{L}^{3g}$ reconnu par un automate d'arbres cheminant mais qui ne peut pas être reconnu par un automate d'arbres cheminant déterministe. A partir de ce langage, nous construisons en 6.1.3 une famille de langages d'arbres $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ qui sépare la classe des langages reconnus par un automate cheminant non-déterministe de la classe des langages reconnus par un automate déterministe à $(n-1)$ jetons. Nous prouvons ainsi qu'on ne peut pas déterminiser un automate d'arbres cheminant même si on s'autorise à ajouter un nombre fixé de jetons.

Nous définissons tout d'abord dans la section 6.1 les familles de langages $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ et $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ par induction à l'aide des langages définis en [7] et en [6]. Pour chaque entier n strictement positif, nous présentons un automate déterministe à n jetons qui reconnaît le langage $\mathcal{L}_n$ et un automate cheminant non-déterministe qui reconnaît le langage $\mathcal{M}_n$. La section 6.2 est la partie technique de ce chapitre dans laquelle nous prouvons que, pour chaque entier n strictement positif, le langage $\mathcal{L}_n$ ne peut pas être reconnu par un automate à $(n-1)$ jetons. Nous prouvons de manière similaire que le langage $\mathcal{M}_n$ ne peut pas être reconnu par un automate déterministe à $(n-1)$ jetons. Enfin dans la dernière section, en adaptant la construction de la famille de langage $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$, nous définissons un langage $\mathcal{L}^r$ qui sépare la classe des langages d'arbres réguliers de la classe des langages reconnus par un automate d'arbres à jetons. D'après la caractérisation logique des automates d'arbres à jetons de [19] que nous rappelons dans le chapitre 4, ce langage sépare également la logique monadique du second ordre sur les arbres de la logique FO + posTC sur les arbres.

6.1 Deux familles de langages d'arbres

Dans cette section, nous définissons les familles de langages $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ et $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ inductivement à l'aide des langages $\mathcal{L}^p$ et $\mathcal{L}^{3g}$ définis par M. Bojańczyk et T. Colcombet en [7] et en [6]. Les arbres des langages $\mathcal{L}^p$ et $\mathcal{L}^{3g}$ ont une forme sympathique : ce sont des arbres sur un alphabet à deux lettres $\{a, b\}$ tels que seules les feuilles peuvent être étiquetées par la lettre a. Nous définissons dans la notation 6.2 $\mathcal{L}^{p,p}$ une variante du langage $\mathcal{L}^p$.

Pour un entier n strictement positif, un arbre du langage $\mathcal{L}_n$ est un arbre obtenu par induction à partir d'un arbre du langage $\mathcal{L}^{p,p}$ en remplaçant chaque feuille dont l'étiquette est la lettre a par un arbre de L_{n-1} et chaque feuille dont l'étiquette est

la lettre b par un arbre qui n'appartiennent pas au langage $\mathcal{L}_{n-1}$. Ainsi un arbre de $\mathcal{L}_n$ a un niveau de plus qu'un arbre de $\mathcal{L}_{n-1}$. La famille de langage $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ est définie par induction de manière similaire en remplaçant le langage $\mathcal{L}^{p,p}$ par le langage $\mathcal{L}^{3g}$. Nous définissons dans un premier temps de manière formelle les arbres à niveaux et la notion de repliage d'un arbre en 6.1.1. La notion de K-repliage sert à définir inductivement les deux familles de langages $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ et $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ que nous présentons en 6.1.2 et 6.1.3. Nous prouvons enfin dans cette section que, pour chaque entier n, le langage L_n est reconnu par un automate déterministe à n jetons et le langage $\mathcal{M}_n$ est reconnu par un automate cheminant non-déterministe.

6.1.1 Arbres et langages à niveaux

Afin de définir formellement les langages de séparation utilisés dans la preuve des théorèmes 6.1 et 6.2, nous définissons tout d'abord quelques notions préliminaires : les arbres quasiment blancs, leur structure de branchement, les arbres à niveaux et le repliage d'un arbre à niveaux selon un langage d'arbre donné.

Pour prouver qu'un langage d'arbres ne peut pas être reconnu par un automate d'arbres cheminant déterministe ou bien par un automate d'arbres cheminant non-déterministe, le principe des preuves des résultats de [7] et de [6] que nous généralisons dans ce chapitre est de construire, pour chaque automate de la classe considérée, des arbres de taille suffisamment grande et dont la forme est suffisamment monotone pour que l'automate soit perdu en les parcourant. Cette idée de noyer l'automate dans un arbre où un certain motif est répété vient du lemme de l'étoile aussi appelé lemme de pompage qui donne une caractérisation des langages de mots réguliers. M. Bojańczyk et T. Colcombet ont ainsi introduit en [6] *les arbres quasiment blancs* et leur *structure de branchement*.

Définition 6.1 *Un arbre quasiment blanc est un arbre sur l'alphabet $\{a, b\}$ tel que tous les nœuds internes sont étiquetés par la lettre b.*

La structure de branchement $\sigma(t)$ d'un arbre quasiment blanc t est l'arbre sur un alphabet unaire défini inductivement sur la structure de t de la manière suivante.

- *si t est vide ou réduit à une feuille étiquetée par b, l'arbre $\sigma(t)$ est l'arbre vide.*
- *si t est réduit à une feuille étiquetée par a, l'arbre $\sigma(t)$ est l'arbre t.*
- *si t a pour sous-arbre gauche t_1 et pour sous arbre droit t_2 et si $\sigma(t_1)$ est l'arbre vide, l'arbre $\sigma(t)$ est l'arbre $\sigma(t_2)$.*
- *si t a pour sous-arbre gauche t_1 et pour sous arbre droit t_2 et si $\sigma(t_2)$ est l'arbre vide, l'arbre $\sigma(t)$ est l'arbre $\sigma(t_1)$.*
- *si t a pour sous-arbre gauche t_1 et pour sous arbre droit t_2 et si $\sigma(t_1)$ et $\sigma(t_2)$ ne sont pas vides, l'arbre $\sigma(t)$ a comme sous-arbre droit $\sigma(t_1)$ et comme sous-arbre gauche $\sigma(t_2)$.*

La structure de branchement $\sigma(t)$ d'un arbre quasiment blanc t est donc la structure d'arbre dont les feuilles correspondent aux feuilles de t étiquetées par a et les nœuds internes correspondent aux nœuds internes de t dont le sous-arbre gauche et le sous-arbre droit contiennent chacun au moins une feuille étiquetée par a.

Exemple 6.1 On représente dans la figure 6.1 un arbre quasiment blanc t et sa

structure de branchement $\sigma(t)$. Les nœuds de t marqués par un petit disque sont les nœuds de sa structure de branchement.

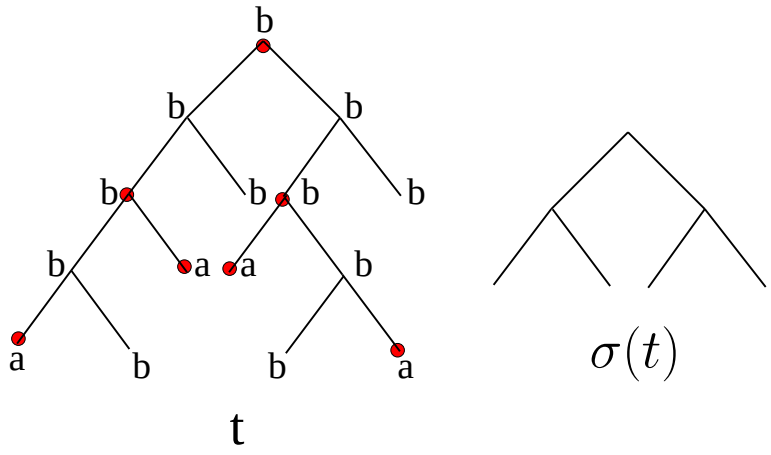

FIG. 6.1 – Un arbre quasiment blanc t et sa structure de branchement $\sigma(t)$

Les arbres des langages de séparation que nous définissons en 6.1.2 et en 6.1.3 sont des arbres sur un alphabet de trois lettres $\{a, b, c\}$ tels que les nœuds étiquetés par la lettre c constituent des coupes de l'arbre que nous appelons des *niveaux*.

Définition 6.2 *Soit $k > 0$. Un arbre à k niveaux t est un arbre sur l'alphabet $\{a, b, c\}$ tel que tout chemin de la racine à une feuille est étiqueté par un mot du langage $(cb^*)^k(a+b)$. Soit $i \in [1, k]$, un nœud de t est sur le niveau i si le sous-arbre enraciné en t est un arbre à i niveaux.*

Remarque 6.1 Un arbre à 1 niveau est obtenu à partir d'un arbre quasiment blanc, en remplaçant l'étiquette de sa racine par la lettre c. Par abus de langage, nous considérons aussi qu'un arbre à 1 niveau est un arbre quasiment blanc.

Remarque 6.2 Un arbre à k niveaux est un arbre dont la racine est étiquetée par c qui contient exactement k antichaines (ou coupes) étiquetées par la lettre c, des feuilles étiquetées par a et tel que tous les autres nœuds sont étiquetés par b.

Les arbres que nous définissons ci-dessus ont un nombre fini de niveaux et nous pouvons construire pour chaque entier k, un automate cheminant qui vérifie qu'un arbre donné est à k niveaux.

Lemme 6.1 *Etant donné $k \in \mathbb{N}$, il existe un automate cheminant qui reconnaît l'ensemble des arbres à k niveaux.*

Preuve. Soit t un arbre. L'automate A que nous construisons dans cette preuve vérifie tout d'abord que la racine de t est étiquetée par la lettre c puis il effectue un parcours en profondeur de t. Pour chaque nœud u, on note n_u^c le nombre de nœuds étiquetés par c sur le chemin de la racine de t à u. Quand A visite un nœud u, son état contient le nombre n_u^c tant que ce nombre est inférieur ou égal à k. Pour la racine ce nombre est 1 à condition que la racine soit étiquetée par c bien sûr.

Si la racine n'est pas étiquetée par c, A rejette. Par la suite, dans le parcours en profondeur de t, à chaque fois que A visite un nœud étiqueté par c à partir de son père, A incrémente de 1 le nombre stocké dans son état et si à partir d'un nœud étiqueté par c, l'automate A remonte au père il décrémente de 1 ce nombre. Dès que ce nombre dépasse k ou dès que ce nombre est différent de k au moment où A visite une feuille, il rejette. Au cours de ce parcours A vérifie aussi que toutes les feuilles sont étiquetées par les lettres a ou b et tous les nœuds internes par les lettres b ou c. Si A termine son parcours en profondeur de t, il a bien vérifié que t était à k niveaux et il accepte. □

Nous définissons maintenant le *repliage* d'un arbre à k niveaux selon un langage d'arbres à $(k-1)$ niveaux. Cette notion de repliage permet de construire les familles de langages $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ et $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ respectivement en 6.1.2 et en 6.1.3.

Définition 6.3 *Soient $k > 0$ et K un langage d'arbres à $(k-1)$ niveaux. Le K-repliage d'un arbre t à k niveaux est l'arbre quasiment blanc obtenu à partir de t en supprimant tous les nœuds en dessous du niveau $(k-1)$ et en réétiquetant chaque nœud v de ce niveau par a si $t_{|v}$ est dans K et par b sinon.*

La figure 6.2 illustre cette définition et montre comment nous obtenons le K-repliage d'un arbre à k niveaux. Dans cette figure, l'arbre à k niveaux donné est représenté et le K-repliage à droite.

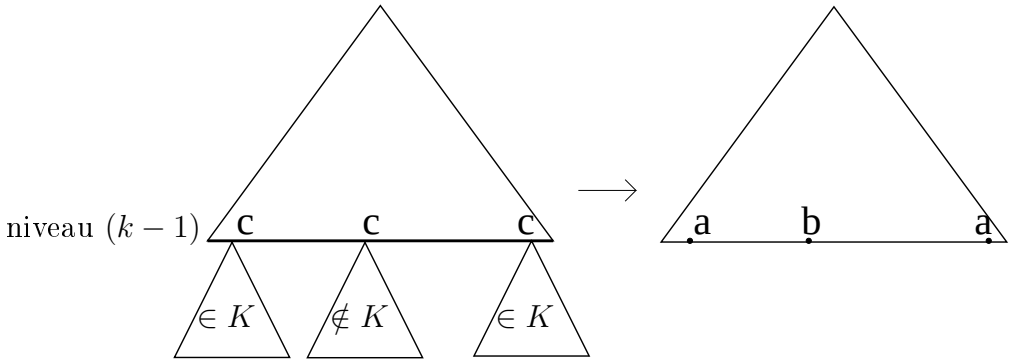

FIG. 6.2 – Construction du K-repliage d'un arbre à n niveaux

6.1.2 La famille de langages $\mathcal{L}_n$

Dans cette partie, nous définissons la famille de langage $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ et nous montrons que chaque langage $\mathcal{L}_n$ est reconnu par un automate déterministe à n jetons. Cette famille de langages est définie inductivement à partir du langage $\mathcal{L}^{p,p}$ qui lui même construit en s'inspirant du langage $\mathcal{L}^p$ utilisé par M. Bojańczyk et T. Colcombet pour séparer les langages d'arbres réguliers des langages reconnus par un automate d'arbres cheminant.

Notation 6.1 *On note $\mathcal{L}^p$ l'ensemble des arbres quasiment blancs t tels que, dans la structure de branchement $\sigma(t)$, tous les chemins de la racine à une feuille ont une longueur paire.*

Exemple 6.2 L'arbre quasiment blanc représenté à gauche de la figure 6.1 appartient à $\mathcal{L}^p$ car dans sa structure de branchement représentée à droite de la figure, tous les chemins de la racine à une feuille sont de longueur 2.

Le lemme ci-dessous a été démontré dans [7]. Sa preuve est technique et difficile et ce lemme permet de déduire que $\mathcal{L}^p$ ne peut pas être reconnu par un automate cheminant.

Lemme 6.2 *Pour tout automate cheminant A, il existe un arbre s' dans $\mathcal{L}^p$ et un arbre t' qui n'appartient pas à $\mathcal{L}^p$ tels que s' est 0-simulé par t'.*

Rappelons qu'étant donné un automate cheminant A, un arbre s' est 0-simulé par un arbre t' si, d'après la définition 5.4, les couples d'états correspondant à des boucles de A dans s' correspondent également à des boucles de A dans t'.

Nous montrons dans la proposition 6.2 que le langage $\mathcal{L}^p$ est reconnu par un automate d'arbres déterministe à un jeton. Pour construire inductivement la famille de langages $\mathcal{L}_n$, nous allons cependant définir à partir de $\mathcal{L}^p$ un nouveau langage d'arbres $\mathcal{L}^{p,p}$ qui possède une propriété plus forte que celle du langage $\mathcal{L}^p$ donnée par le lemme 6.2.

Nous avons en effet besoin d'un langage d'arbres $\mathcal{L}$ satisfaisant la propriété suivante : pour tout automate cheminant il existe deux arbres 0-équivalents tels que l'un est dans le langage $\mathcal{L}$ l'autre non. En utilisant le lemme 6.2, on obtient deux arbres $s' \in \mathcal{L}^p$ et $t' \notin \mathcal{L}^p$ tels que s' est 0-simulé par t. Nous voulons en plus que t' soit 0-simulé par s'.

Rappelons encore que, d'après la définition 5.4, étant donné un automate cheminant, deux arbres 0-équivalents sont deux arbres ayant exactement les mêmes couples d'états correspondant à des boucles de la racine à la racine.

Notation 6.2 *On considère $\mathcal{L}^{p,p}$ l'ensemble des arbres quasiment blancs t pour lesquels la structure de branchement $\sigma(t)$ contient un nombre pair de nœuds v de 1^*2 tels que, dans le sous-arbre $\sigma(t)_{|v}$, tous les chemins d'une feuille à la racine sont de longueur paire.*

Nous avons vu dans l'exemple 2.18 que les nœuds d'un arbre de la forme 1^* sont appelés les nœuds du chemin le plus à gauche.

Exemple 6.3 La figure 6.3 représente un arbre t de $\mathcal{L}^{p,p}$, nous avons marqué les nœuds du chemin le plus à gauche de la structure de branchement. Nous pouvons ainsi voir que dans la structure de branchement de t, il existe deux nœuds du chemin le plus à gauche dont le sous-arbre droit est tel que tous les chemins de la racine à une feuille sont de longueur paire. L'arbre représenté sur la figure appartient donc bien au langage $\mathcal{L}^{p,p}$.

Nous montrons maintenant que le langage $\mathcal{L}^{p,p}$ satisfait la propriété voulue donnée par la proposition suivante.

Proposition 6.1 *Pour tout automate cheminant A, il existe un arbre s dans $\mathcal{L}^{p,p}$ et un arbre t qui n'appartient pas à $\mathcal{L}^{p,p}$ tels que s et t sont 0-équivalents.*

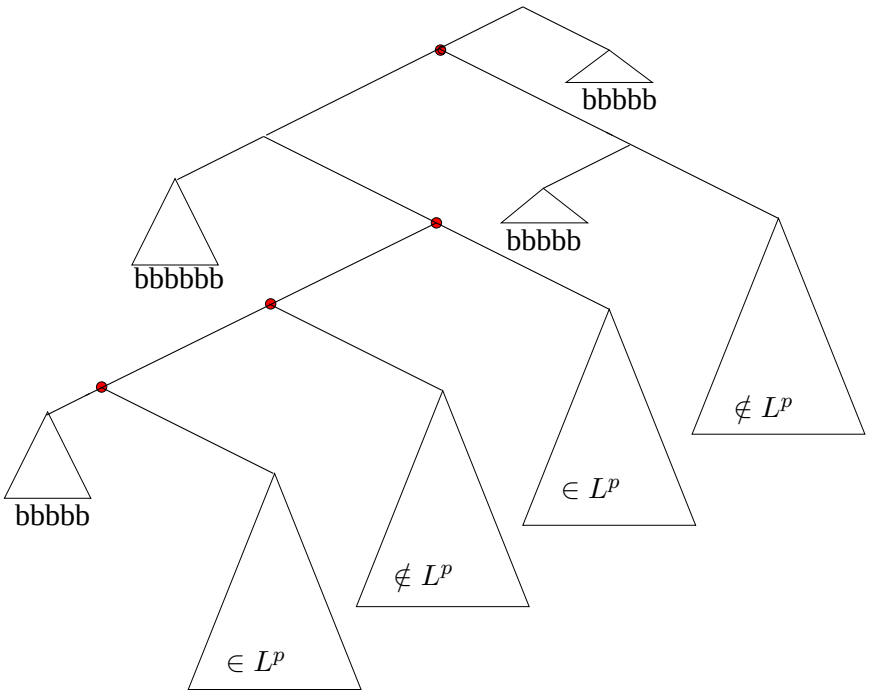

FIG. 6.3 – Représentation d'un arbre de $\mathcal{L}^{p,p}$

Preuve.

On considère A un automate cheminant. Soient s' et t' les arbres donnés par le lemme 6.2. On note m le nombre de 0-comportements de A poour des arbres. Etant donné un entier i strictement positif, on note t_i l'arbre dont le chemin le plus à gauche est de longueur $(m+1)$ et tel que un sous-arbre enraciné sur un nœud de la forme 1^j2 avec $j \in [0,m]$ est s' si $j < i$ et t' sinon. On observe que t_i est dans $\mathcal{L}^{p,p}$ si et seulement si i est pair. On remarque ensuite que t_{i+1} est obtenu à partir de t_i en remplaçant un des sous-arbres s' par t'. La suite des 0-comportements pour t_i avec $i \in [0,m]$ est donc croissante et comme m est le nombre de 0-comportements pour des arbres, il existe un entier $\ell \in [0,m]$ tel que A a le même 0-comportement pour t_ℓ et $t_{\ell+1}$. Comme un seul de ces deux arbres appartient à $\mathcal{L}^{p,p}$, on a bien démontré le lemme. □

Les langages $\mathcal{L}^p$ et $\mathcal{L}^{p,p}$ ne peuvent donc pas être reconnus par un automate cheminant. La proposition ci-dessous permet alors de séparer la classe des langages reconnus par un automate d'arbres à 1 jeton de la classe des langages reconnus par un automate d'arbres cheminant.

Proposition 6.2 *Le langage $\mathcal{L}^p$, respectivement $\mathcal{L}^{p,p}$, est reconnu par un automate d'arbres déterministe à 1 jeton.*

Preuve.

On commence par montrer cette proposition pour le langage $\mathcal{L}^p$. On considère un arbre quasiment blanc t et on construit ci-dessous un automate déterministe à un jeton A^p qui reconnaît ce langage. Cet automate commence par effectuer un parcours en profondeur de t. A chaque fois qu'il visite un nœud dans ce parcours, A^p pose un jeton sur ce nœud puis effectue un simple parcours en profondeur du sous-arbre

droit et du sous-arbre gauche de ce nœud marqué pour vérifier si ces sous-arbres contiennent une feuille étiquetée par a et donc si ce nœud est dans la structure de branchement de t et, enfin, il lève le jeton pour pouvoir le reposer sur le prochain nœud qu'il visite. L'automate A^p peut alors calculer la parité du nombre de fois qu'il passe par un nœud de la structure de branchement au cours de son parcours en profondeur de t. Il vérifie que ce nombre est impair à chaque fois qu'il visite une feuille étiquetée par a et reconnaît ainsi le langage $\mathcal{L}^p$.

On construit maintenant $A^{p,p}$ un automate déterministe à 1 jeton qui reconnaît $\mathcal{L}^{p,p}$. On considère à nouveau un arbre quasiment blanc t. L'automate $A^{p,p}$ effectue un parcours en profondeur de t. A chaque fois qu'il visite un nœud u, il détermine comme A^p si ce nœud est dans la structure de branchement de t. Si c'est le cas, avant de lever le jeton de u, il effectue un parcours en profondeur de droite à gauche à partir de u et vérifie si, après avoir visité le sous-arbre enraciné en u dans ce parcours, l'automate remonte jusqu'à la racine sans visiter de feuilles étiquetées par a. Il détermine ainsi si le nœud u appartient au chemin le plus à gauche de la structure de branchement $\sigma(t)$. L'automate $A^{p,p}$ peut ainsi vérifier à chaque étape de son parcours s'il est sur un nœud du chemin le plus à gauche de la structure de branchement de t. L'automate vérifie alors pour chaque nœud du chemin le plus à gauche de t si le sous-arbre droit de ce nœud est dans $\mathcal{L}^p$ en procédant comme A^p mais en vérifiant aussi à chaque fois qu'il viste un nœud de la structure de branchement s'il est revenu sur le chemin le plus à gauche de la structure de branchement. Il compte aussi la parité du nombre de sous-arbres de $\mathcal{L}^p$ enracinés en un fils droit du chemin le plus à a gauche de $\sigma(t)$ qu'il a visités. L'automate $A^{p,p}$ reconnaît ainsi le langage $\mathcal{L}^{p,p}$. □

Nous construisons maintenant une famille de langages $\mathcal{L}_n$ à partir du langage de base $\mathcal{L}_1 = \mathcal{L}^{p,p}$ qui satisfait le lemme 6.1.

Notation 6.3 *On note $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ la famille de langages d'arbres définis inductivement ainsi :*

- *$\mathcal{L}_1$ est le langage $\mathcal{L}^{p,p}$.*
- *pour $n > 0$, le langage $\mathcal{L}_n$ est l'ensemble des arbres à n niveaux dont le $\mathcal{L}_{n-1}$-repliage est dans $\mathcal{L}^{p,p}$.*

Nous étendons maintenant la proposition 6.2 à tous les langages de la famille $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$.

Lemme 6.3 *Pour tout $k \in \mathbb{N}_{>0}$, $\mathcal{L}_k$ est reconnu par un automate déterministe à k jetons.*

Preuve. On procède par induction sur k. Le cas $k = 1$ est résolu par la proposition 6.2.

On suppose donc que $k > 1$. On veut construire un automate à k jetons A qui reconnaît $\mathcal{L}_k$. On considère un arbre t sur l'alphabet $\{a, b, c\}$. L'automate à k jetons qu'on construit peut vérifier si un nœud est sur le chemin le plus à droite de la structure de branchement du $\mathcal{L}_{k-1}$-repliage de t de la manière suivante. Il place son jeton k sur ce nœud puis il effectue un parcours en profondeur des sous-arbres droit et gauche de ce nœud marqué. Pendant ce parcours, à chaque fois que A visite un nœud du niveau $(k-1)$ il simule l'automate à $(k-1)$ jetons obtenu par induction pour le langage L_{k-1} et vérifie si le sous-arbre enraciné en ce nœud du

niveau $(k-1)$ appartient à L_{k-1} ou non. Enfin il retourne au nœud marqué par le jeton k et effectue un parcours en profondeur de droite à gauche et vérifie qu'après avoir parcouru le sous-arbre enraciné au nœud marqué par le jeton k il ne visite plus que des feuilles étiquetées par b. Nous décrivons maintenant comment A procède sur l'arbre t. L'automate A commence par vérifier comme il est décrit dans le lemme 6.1 que t est un arbre à k niveaux. Il effectue ensuite un parcours en profondeur de t et visite ainsi tous les sous-arbres enracinés en un fils droit du chemin le plus à gauche de la structure de branchement du $\mathcal{L}_{k-1}$-repliage de t. On considère t' un de ces sous-arbres de t et on note V l'ensemble des nœuds v de t' du niveau $(k-1)$ tels que le sous-arbre enraciné en v appartient à $\mathcal{L}_{k-1}$ et W l'ensemble des nœuds de t' dont le sous-arbre gauche et le sous-arbre droit contiennent chacun au moins un nœud de V. Pour vérifier si les chemins de la racine de t' à un nœud de V contiennent chacun un nombre pair de nœuds de W, l'automate A procède de la manière suivante. Il effectue un parcours en profondeur de t', à chaque fois qu'il visite un nœud v, il pose son jeton k sur ce nœud et vérifie si ce nœud est un fils droit du chemin le plus à gauche pour savoir s'il a fini de visiter t'. Il détermine ensuite si le v nœud qui est toujours marqué par le jeton k est un nœud de W. Pour cela il vérifie si les sous-arbres droit et gauche de v contiennent chacun un nœud de V. On remarque que comme le niveau k de t est fixé l'automate peut compter le niveau du dernier nœud de t étiqueté par c qu'il a croisé. Par induction, l'automate peut donc vérifier si un nœud du niveau $(k-1)$ appartient à V en utilisant ses $(k-1)$ jetons restants. L'automate compte ainsi la parité du nombre de nœud de W qu'il croise dans son parcours de t' et vérifie que ce nombre est pair à chaque fois qu'il est sur le niveau $(k-1)$. L'automate A vérifie ainsi si le $\mathcal{L}_{k-1}$-repliage de t' est dans $L^{p,p}$. Il peut alors compter la parité des sous-arbres enracinés en un fils droit du chemin le plus à gauche de la structure de branchement du $\mathcal{L}_{k-1}$-repliage de t tels que le $\mathcal{L}_{k-1}$-repliage est dans $\mathcal{L}^p$. L'automate A reconnaît ainsi le langage $\mathcal{L}_k$. □

Nous avons ainsi défini une famille de langages $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ telle que chaque langage $\mathcal{L}_k$ est reconnu par un automate déterministe à k jetons et telle que $\mathcal{L}_1$ vérifie la propriété donnée par la proposition 6.1.

6.1.3 La famille de langages $\mathcal{M}_n$

Nous construisons dans cette partie la famille de langages à niveaux $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ et nous décrivons pour chaque entier k strictement positif un automate cheminant qui reconnaît le langage $\mathcal{M}_k$. Le langage de base utilisé pour construire cette famille a été introduit par M. Bojańczyk et T. Colcombet dans [6] pour prouver qu'on ne pouvait pas déterminiser un automate d'arbres cheminant. La construction de la famille de langages $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ à partir de ce langage de base que nous appellerons $\mathcal{L}^{3g}$ est similaire à celle de la famille de langage $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ à partir du langage $\mathcal{L}^{p,p}$. Les arbres de $\mathcal{L}^{3g}$ sont des arbres quasiment blancs avec trois feuilles étiquetées par a.

Remarque 6.3 Si on considère l'ensemble des arbres quasiment blancs ayant exactement trois feuilles étiquetées par la lettre a, il existe pour ces arbres deux structures de branchement possible que nous représentons figure 6.4.

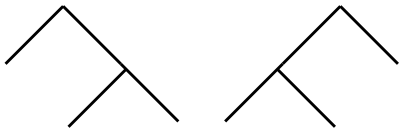

FIG. 6.4 – Les structures de branchement possibles d'un arbre avec 3 a

Notation 6.4 *On note $\mathcal{L}^{3g}$ l'ensemble des arbres quasiment blancs t dont la structure de branchement $\sigma(t)$ est la structure d'arbre représentée figure 6.5. On considère*

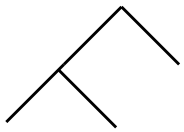

FIG. 6.5 – La structure de branchement possible d'un arbre de $\mathcal{L}^{3g}$

que la racine des arbres de $\mathcal{L}^{3g}$ est étiquetée par la lettre c afin d'avoir un langage d'arbres à 1 niveau.

Exemple 6.4 L'arbre quasiment blanc t représenté à gauche de la figure 6.6 appartient à $\mathcal{L}^p$ comme le montre sa structure de branchement représentée à droite de la figure. Dans cette figure, nous avons noté uniquement les feuilles de l'arbre t étiquetées par a.

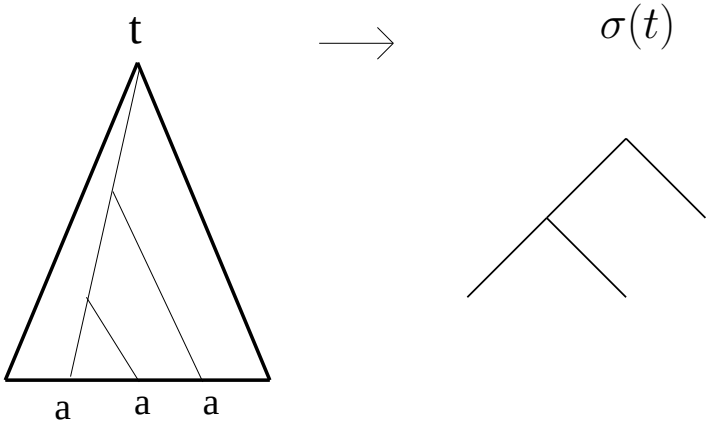

FIG. 6.6 – Représentation d'un arbre de $\mathcal{L}^{3g}$

Le langage $\mathcal{L}^{3g}$ satisfait la propriété décrite dans le lemme ci-dessous qui a été démontré dans [6]. Ce lemme a pour conséquence immédiate que $\mathcal{L}^{3g}$ ne peut pas être reconnu par un automate cheminant déterministe et il sera utilisé dans la sous-section 6.2.3.

Lemme 6.4 *Pour tout automate cheminant déterministe A, il existe un arbre s' dans $\mathcal{L}^{3g}$ et un arbre t' qui n'appartient pas à $\mathcal{L}^{3g}$ tels que s' et t' sont 0-équivalents.*

Le langage $\mathcal{L}^{3g}$ satisfait ainsi directement la propriété souhaitée contrairement au langage $\mathcal{L}^p$ à partir duquel nous avons du définir $\mathcal{L}^{p,p}$. Le langage $\mathcal{L}^{3g}$ est donc

le langage de base de la famille de langages $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ que nous définissons par la suite.

La proposition ci-dessous implique que le langage $\mathcal{L}^{3g}$ sépare la classe des langages reconnus par un automate cheminant non-déterministe de la classe des langages reconnus par un automate déterministe à 1 jeton.

Proposition 6.3 *Le langage $\mathcal{L}^{3g}$ est reconnu par un automate cheminant non-déterministe.*

Preuve. On décrit dans cette preuve un automate cheminant non-déterministe A^{3g} qui reconnaît le langage $\mathcal{L}^{3g}$. Etant donné un arbre t, l'automate A^{3g} commence par effectuer un parcours en profondeur et vérifie ainsi de manière déterministe que l'arbre a exactement trois feuilles étiquetées par la lettre a et que tous les autres nœuds sont étiquetés par la lettre b. L'automate A^{3g} doit ensuite trouver lequel des deux arbres de la figure 6.4 est la structure de branchement de t. Pour cela, il se rend sur la feuille étiquetée par a la plus à gauche de t en effectuant un nouveau parcours en profondeur de gauche à droite jusqu'à ce qu'il atteigne une feuille étiquetée par la lettre a. A partir de cette feuille, A^{3g} remonte de manière non-déterministe jusqu'à un ancêtre u de cette feuille puis il effectue à partir de ce nœud un parcours en profondeur de droite à gauche et accepte s'il croise exactement deux feuilles étiquetées par la lettre a au cours de ce dernier parcours. Si l'arbre t est dans $\mathcal{L}^{3g}$ il existe toujours une exécution acceptante de A^{3g} : par exemple celle pour laquelle A^{3g} remonte de manière non-déterministe jusqu'au nœud dont le sous-arbre gauche contient la feuille étiquetée par a la plus à gauche de t et dont le sous-arbre droit contient la deuxième feuille de t étiquetée par a. Si l'arbre t a la structure de branchement représenté à droite dans la figure 6.4, l'automate A^{3g} croise toujours 1 ou bien 3 feuilles étiquetées par la lettre a dans son parcours en profondeur de droite à gauche à partir d'un ancêtre u de la feuille étiquetée par a la plus à gauche. □

Nous construisons maintenant une famille de langages $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ à partir du langage de base $\mathcal{M}_1 = \mathcal{L}^{3g}$. Cette construction est similaire à celle de la famille de langages $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ à partir du langage de base $\mathcal{L}_{3g}$.

Définition 6.4 *La famille de langages d'arbres $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ est définie inductivement ainsi :*

- *$\mathcal{M}_1$ est le langage $\mathcal{L}^{3g}$.*
- *pour $n > 0$, le langage $\mathcal{M}_n$ est l'ensemble des arbres à n-niveaux dont le $\mathcal{L}_{n-1}$-repliage est dans $\mathcal{L}^{3g}$.*

Lemme 6.5 *Pour tout $k \in \mathbb{N}_{>0}$, $\mathcal{M}_k$ et son complément sont reconnus par un automate cheminant non-déterministe.*

Preuve. On procède par induction sur k. Le cas $k = 1$ est résolu partiellement par la proposition 6.3.

On suppose que $k \geq 1$. Soit t un arbre. L'automate que l'on construit pour $\mathcal{M}_k$ vérifie dans un premier temps que t est un arbre k niveaux puis visite tous les nœuds du niveau $(k-1)$. Pour chacun de ces nœuds u, il devine si le sous-arbre enraciné en u est dans $\mathcal{M}_{k-1}$ et vérifie si ce choix est correct en simulant les automates donnés par

l'hypothèse d'induction pour le langage $\mathcal{M}_{k-1}$ et le complément de $\mathcal{M}_{k-1}$. Il rejète l'arbre si celui ci ne contient pas exactement trois sous-arbres de $\mathcal{M}_{k-1}$ enracinés en un nœud du niveau $(k-1)$. La suite de la construction de l'automate pour $\mathcal{M}_k$ suit le même principe que celle de l'automate A^{3g} de la preuve de la proposition 6.3. On note u_1, u_2, u_3 les trois nœuds ordonnés de gauche à droite du niveau $(k-1)$ dont le sous-arbre est dans $\mathcal{M}_{k-1}$. L'automate construit pour $\mathcal{M}_k$ se rend en u_1, remonte dans l'arbre vers un ancêtre u de u_1 de manière non-déterministe puis effectue un parcours en profondeur de droite à gauche en vérifiant si chaque nœud du niveau $(k-1)$ qu'il visite est un des trois nœuds u_1, u_2 ou u_3. L'automate pour $\mathcal{M}_k$ accepte si durant le dernier parcours en profondeur à partir du nœud u, il croise exactement deux de ses trois nœuds.

L'automate qu'on construit pour reconnaître le complément de $\mathcal{M}_k$ vérifie d'abord de manière déterministe si l'arbre t est à k niveaux. Si ce n'est pas le cas il accepte. Il vérifie ensuite si l'arbre t contient exactement trois sous-arbres de $\mathcal{M}_{k-1}$ enracinés en un nœud du niveau $(k-1)$ et accepte si ce n'est pas le cas. Enfin l'automate procède comme l'automate pour $\mathcal{M}_k$ pour vérifier si le $\mathcal{M}_{k-1}$-repliage de t a la structure de branchement représenté à gauche de la figure 6.4. □

Nous avons ainsi défini une famille de langages $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ tels que chaque langage $\mathcal{M}_k$ est reconnu par un automate cheminant et tel que M_1 vérifie la propriété donnée par le lemme 6.4.

6.2 Nombre de jetons, déterminisme et pouvoir d'expression

Cette section est la partie technique de ce chapitre dans laquelle la proposition 6.1 et le lemme 6.4 concernant les automates cheminants et les langages $L^{p,p}$ et L^{3g} sont généralisés aux automates d'arbres à jetons et aux familles de langages $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ et $(\mathbb{M}_n)_{n\in\mathbb{N}_{>0}}$. Nous démontrons ainsi que, pour chaque entier k, le langage $\mathcal{L}_{k+1}$ n'est pas reconnu par un automate à k jetons et que le langage $\mathcal{M}_{k+1}$ n'est pas reconnu par un automate déterministe à k jetons. Le principe de la preuve est de montrer par induction qu'un automate d'arbres à jetons qui reconnaît $\mathcal{L}_{k+1}$ a besoin, au cours d'une exécution sur un arbre t, de placer k jetons dans un sous-arbre enraciné en un nœud u du niveau k pour déterminer si ce sous-arbre appartient à $\mathcal{L}_k$ et donc quelle sera l'étiquette du nœud qui le remplace dans le $\mathcal{L}_k$-repliage de t et d'utiliser en même temps un autre jeton au dessus du niveau k pour déterminer si le $\mathcal{L}_k$-repliage de l'arbre t est dans $\mathcal{L}^{p,p}$. L'idée est la même pour la preuve concernant le langage $\mathcal{M}_{k+1}$: un automate à jetons déterministe qui reconnaît ce langage se sert à la fois de k jetons au dessous d'un nœud u du niveau k pour savoir si le sous-arbre enraciné en u appartient à $\mathcal{M}_k$ et d'un autre jeton au dessus du niveau k pour vérifier si le $\mathcal{M}_k$-repliage de l'arbre t est dans $\mathcal{L}^{p,p}$. Nous devons ainsi montrer qu'étant donné un automate d'arbres à k jetons dans un arbre t à $(k+1)$ niveaux l'action de poser un jeton sur un nœud u situé au dessus du niveau k aveugle l'automate sur les propriétés des nœuds de ce niveau. Pour formaliser cette intuition, nous introduisons en 6.2.1 un nouveau type d'automates d'arbres qui étend le modèle des automates d'arbres cheminants : ce sont les automates d'arbres cheminants

avec oracle. Nous généralisons ensuite les propositions 6.2 et 6.3 à ce nouveau type d'automates. Enfin, étant donné A un automate à k jetons, nous construisons en 6.2.2, deux arbres de part et d'autre de $\mathcal{L}_{k+1}$ pour lesquels A a le même comportement. L'automate A ne peut donc les distinguer et reconnaître le langage $\mathcal{L}_{k+1}$. Nous procédons de même en 6.2.3 pour prouver que le langage $\mathcal{M}_k$ n'est pas reconnu par un automate déterministe à k jetons.

Grace aux résultats du chaptre 5, nous pouvons supposer dans tout ce chapitre que les automates d'arbres à jetons que nous considérons dans cette partie sont du modèle faible. Cette hypothèse nous permet de gérer plus facilement l'alternance des jetons de part et d'autre du niveau k dans un arbre de L_{k+1} ou de $\mathcal{M}_{k+1}$.

6.2.1 Automates cheminants avec oracle

Dans cette partie nous présentons des concepts préliminaires qui sont utilisés dans la suite du chapitre : nous définissons ainsi les motifs, les oracles de structure et les automates cheminants avec oracle. Nous démontrons ensuite le lemme qui permet d'étendre les propriétés des langages $\mathcal{L}^{p,p} = \mathcal{L}_1$ et $\mathcal{L}^{3g} = \mathcal{M}_1$ aux automates d'arbres cheminants avec oracle.

Nous introduisons tout d'abord *les motifs* qui sont une généralisation des contextes où le symbole $*$ peut apparaître sur plus d'une seule feuille.

Définition 6.5 *Un motif sur l'alphabet Σ est un arbre sur $\Sigma \cup (\Sigma \times \{*\})$ tel que tous les nœuds internes sont étiquetés par une lettre de Σ. Le symbole $*$ peut donc apparaître uniquement sur les feuilles. Les feuille dont l'étiquette contient le symbole spécial $*$ sont appelées les ports du motifs. Les ports sont ordonnés en utilisant l'ordre de gauche à droite sur les feuilles. Le nombre de ports d'un motif C est appelé l'arité de ce motif. Un motif ayant n ports est aussi appelé un motif n-aire.*

Les motifs se composent de la même façon que les contextes et les sous-arbres.

Définition 6.6 *Soient C un motif n-aire et $C_1, \cdots, C_n$ des motifs tels que, pour tout $i \in [1, n]$, l'étiquette de la racine de C_i correspond à la première composante de l'étiquette du i-ème port de C. On note $C[C_1, \cdots, C_n]$ le motif construit à partir de C en remplaçant le i-ème port de C par le motif C_i.*

Nous définissons maintenant un *oracle de structure*. Il s'agit d'un automate d'arbres BU déterministe complet et invariant par étiquette, c'est à dire qui a la même exécution sur deux arbres ayant le même domaine.

Définition 6.7 *Un oracle de structure O est un triplet (Q, s_0, δ) tel que Q est un ensemble fini d'états, $s_0 \in Q$ est l'état initial, Σ est un alphabet et δ est une fonction partout définie de $(Q \times Q)$ à valeurs dans Q appelée fonction de transition.*

Le calcul de O sur un arbre t est défini comme en 2.3.2 : il s'agit une fonction ρ de l'ensemble des nœuds de t à valeurs dans Q telle que, pour chaque nœud x, on a
- $\rho(x) = \delta(q_0, q_0)$ si x est une feuille et
- $\rho(x) = \delta(\rho(x_1), \rho(x_2))$ si x a pour fils droit x_1 et pour fils gauche x_2.

Définition 6.8 *Etant donnés un motif n-aire C et un oracle de structure O dont l'ensemble d'états est noté Q, on définit une fonction notée C^O de Q^n à valeurs dans Q telle que $C^O(q_1, \cdots, q_n)$ est l'état atteint par O à la racine de C s'il démarre son exécution en affectant au i-ème port l'état q_i pour chaque $i \in [1, n]$ et en affectant l'état s_0 à toutes les autres feuilles de C. Cette fonction est totalement définie car O correspond à un automate complet.*

Etant donné un arbre t un nœud v de t et un oracle de structure O, la O-information structurelle sur (t, v) est le couple $((C_{t,v})^O, (t_{|v})^O) \in Q^Q \times Q$.

La fonction C^O définie ci-dessus ne dépend pas des étiquettes des nœuds du contexte C. Comme les seules oracles que nous utilisons dans cette thèse sont des oracles de structure, un oracle désigne à partir de maintenant un oracle de structure.

Remarque 6.4 Etant donné C un motif n-aire, des motifs $C_1, \cdots, C_n$ et O un oracle de structure on a l'égalité de fonctions suivantes $(C[C_1, \cdots, C_n])^O = C^O({C_1}^O, \cdots, {C_n}^O)$.

Nous introduisons un nouveau type d'automates qui étend le modèle des automates d'arbres cheminants. Il s'agit des *automates cheminants avec oracle*. Un automate cheminant avec un oracle O se comporte comme un automate cheminant qui a accès à chaque étape de son exécution sur un arbre t à la O-information structurelle sur (t, v) où v est le nœud sur lequel est sa tête de lecture.

Définition 6.9 *Un automate cheminant avec oracle est un couple (A, O) tel que O est un oracle (P, s_0, δ_O) et A est un quintuplet $(Q, \Sigma, I, F, \delta)$ où Q est un ensemble fini d'états, Σ est un alphabet, $I \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états initiaux, $F \subseteq Q$ est un sous-ensemble de Q appelé ensemble des états finaux ou ensemble des états acceptants et δ est un sous-ensembe de $(Q \times \Sigma \times (P^P \times P)) \times (Q \times \{\uparrow, \swarrow, \searrow, \circlearrowleft\})$ appelé ensemble des transitions. L'automate cheminant avec oracle est déterministe si δ correspond à une fonction de $Q \times \Sigma \times (P^P \times P)$ à valeurs dans $(Q \times \{\uparrow, \swarrow, \searrow, \circlearrowleft\})$. La taille d'un automate cheminant avec oracle (A, O) est la somme du nombre d'états de A et du nombre d'états de O.*

Un automate cheminant avec oracle (A, O) sur un arbre t applique une transition en fonction de l'état courant de A, de l'étiquette du nœud courant u et de la O-information structurelle sur (t, u). Plus formellement, une configuration de (A, O) sur un arbre t est un couple (q, u) où q est un état de Q et u un nœud de l'arbre t. Une configuration initiale de A sur t est une configuration de $I \times \{\epsilon\}$, c'est-à-dire une configuration telle que la tête de lecture de l'automate est sur la racine et telle que l'état de contrôle est un état initial de l'automate. Une configuration acceptante de A sur t est une configuration de $F \times \{\epsilon\}$. Une configuration (r, v) est un successeur d'une configuration (q, u) ce que l'on note $(q, u) \xrightarrow{(A,O)} (r, v)$ s'il existe une transition $((q, a, (f, p)), (\mu, r))$ telle que (f, p) est la O-information structurelle sur (t, u) et

- $v = u$ si $\mu = \circlearrowleft$,
- v est le père de u si $\mu = \uparrow$,
- v est le fils gauche de u si $\mu = \swarrow$ et
- v est le fils droit de u si $\mu = \searrow$.

Remarque 6.5 Les automates cheminants avec oracle sont une extension des automates cheminants car le type d'un nœud peut être considéré comme une information structurelle.

Remarque 6.6 Les notions de 0-équivalence et de 0-comportements pour un sous-arbre et d'un contexte définies en 5.4 et en 5.5 pour les automates à jetons et donc pour les automates cheminants s'étendent sans difficultés au modèle des automates cheminants avec oracles. Le 0-comportement de l'automate cheminant avec oracle (A, O) correspond alors à un couple (p, f) où $p \in P$ est un état de O et f une fonction de P^P à valeurs dans $2^{Q\times Q}$ faisant correspondre à une fonction C^O où C est un contexte, l'ensemble des boucles d'arbres de (A, O) dans $C[t]$. Nous nous intéresserons dans cette section aux boucles de la racine à la racine d'un arbre pour un automate cheminant avec oracle. Ces boucles de la racine à la racine sont simplement données par un ensemble de couples d'états de A.

Rappelons qu'un morphisme d'arbres est une application de l'ensemble des arbres à valeurs dans l'ensemble des arbres qui respectent la structure d'arbres c'est à dire qui est compatible avec l'opération de composition. Nous définissons ci-dessous des morphismes d'arbres quasiment blancs à l'aide d'un motif binaire et de deux arbres. Ces morphismes sont appelés *les morphismes simples d'arbres.*

Définition 6.10 *On considére l'alphabet $\Sigma = \{a, b\}$. Un morphisme simple d'arbres h est généré par*

- *un motif binaire C tel que les deux ports sont étiquetés par $(b, *)$ et tous les autres nœuds sont étiquetés par la lettre b,*
- *un arbre t_b dont tous les nœuds sont étiquetés par b et*
- *un arbre quasiment blanc t_a avec exactement une feuille étiquetée par a qui n'est pas la racine.*

On définit $h(t)$ inductivement sur t de la manière suivante.

- *Si t est réduit à une feuille étiquetée par b, l'arbre $h(t)$ est t_b.*
- *Si t est réduit à une feuille étiquetée par a, l'arbre $h(t)$ est l'arbre t_a.*
- *Si t a pour sous-arbre gauche t_1 et pour sous arbre droit t_2, l'arbre $h(t)$ est l'arbre $C[h(t_1), h(t_2)]$.*

Etant donné un morphisme simple h et un arbre t, le morphisme de nœuds associé à h noté h_N fait correspondre à chaque nœud u de t la racine du motif ou de l'arbre par lequel u est remplacé dans $h(t)$.

Etant donné t un arbre quasiment blanc, l'arbre $h(t)$ est obtenu par composition en remplaçant tous les nœuds internes par C, toutes les feuilles dont l'étiquette est la lettre a par t_a et toutes les feuilles dont l'étiquette est la lettre b par t_b.

Remarque 6.7 Etant donné un morphisme simple h et un arbre quasiment blanc t, les arbres t et $h(t)$ ont la même structure de branchement.

La proposition suivante nous permet par la suite de remplacer la notion d'automates cheminants par la notion d'automates cheminants avec oracle dans la proposition 6.1 et le lemme 6.4.

Proposition 6.4 *Soit (A, O) un automate cheminant avec oracle. Il existe un morphisme simple d'arbres h et un automate cheminant A' qui accepte l'ensemble des arbres quasiment blancs t dont l'image $h(t)$ est acceptée par (A, O). De plus, deux arbres t et t' ont les mêmes couples d'états correspondant à des boucles de la racine à la racine si et seulement si $h(t)$ et $h(t')$ ont les mêmes couples d'états correspondant à des boucles de la racine à la racine.*

Si (A, O) est déterministe, A' peut être choisi déterministe également.

Pour prouver cette proposition nous utilisons le lemme suivant dont la preuve technique est similaire à celle du lemme 9 de [6].

Lemme 6.6 *Pour chaque oracle O, il existe un motif binaire C sur l'alphabet $\{b, (b, *)\}$ et un arbre s sur l'alphabet unaire $\{b\}$ tels que si on note $t_{(b,*)}$ le contexte (ou le motif unaire) réduit au nœud étiqueté par $(b, *)$ et si on pose $f = (C[t_{(b,*)}, s])^O$ et $g = (C[s, t_{(b,*)}])^O$, on a alors $f = g$, $f \circ f = f$ et $f(s^O) = s^O$*

Nous donnons maintenant la preuve de la proposition 6.4.

Preuve. Soit (A, O) un automate cheminant avec oracle. On considère le motif binaire C et l'arbre s donnés par le lemme 6.6. On note alors s_a l'arbre quasiment blanc obtenu en remplaçant l'étiquette de la feuille la plus à gauche de s par a et h le morphisme simple défini par le motif C, l'arbre s dont tous les nœuds sont étiquetés par b et l'arbre quasiment blanc s_a.

On montre d'abord par induction que pour chaque arbre t et chaque nœud v de t on a :$(h(t)_{|h_N(v)})^O = (h(t_{|v})^O = s^O$ et si $v \neq \epsilon$ alors $(C_{h(t),h_N(v)})^O = f$ où $f = (C[t_{(b,*)}, s])^O$.

En effet, si v est une feuille, l'arbre $h(t)_{|h_N(v)} = h(t_{|v})$ est soit l'arbre s soit l'arbre s_a et on a bien $(h(t)_{|h_N(v)})^O = (h(t_{|v})^O = s^O$. Sinon, v est un nœud interne, $v1$ est son fils droit et $v2$ son fils gauche. On a alors l'égalité suivante par hypothèse d'induction $(h(t)_{|h_N(vi)})^O = (h(t_{|vi})^O = s^O$ pour $i \in \{1, 2\}$.

Ainsi $(h(t)_{|h_N(v)})^O = (h(t_{|v})^O = (C[h(t_{|v1}, t_{|v2}])^O = f(s^O) = s^O$

Pour l'autre égalité, on suppose maintenant que v est le fils droit de la racine. Le cas où v est le fils gauche de la racine est similaire. On note t_1 le sous-arbre gauche de t et on a $(C_{h(t),h_N(v)})^O = (C[h(t_1), (b, *)])^O r = (C[(s, t_{(b,*)}])^O$ car $(h(t_1))^O = s^O$. On a donc bien $(C_{h(t),h_N(v)})^O = f$. Supposons enfin que v est le fils droit d'un nœud interne w. Le cas où v est le fils gauche d'un nœud interne est similaire. On note t_1 le sous-arbre gauche de w. Comme $(h(t_1))^O = s^O$ et $(C[s, t_{(b,w)}])^O = f$, on a bien $(C_{h(t),h_N(v)})^O = (C_{h(t),h_N(w)}[C[h(t_1), t_{(b,w)}]])^O = f \circ f = f$.

Ainsi chaque nœud différent de la racine qui est l'image d'un nœud par h_N a pour O-information structurelle (f, s^O) et la racine a pour image structurelle (Id, s^0) où Id est la fonction identité.

On construit maintenant un automate cheminant A' tel que pour chaque arbre t et pour chaque nœuds v et w les deux propositions suivantes sont équivalentes :

- dans $h(t)$, l'automate (A, O) peut atteindre la configuration $(q, h_N(w))$ à partir de la configuration $(p, h_N(v))$
- dans t, l'automate A' peut atteindre la configuration (q, w) à partir de la configuration (p, v)

La construction d'un tel automate A' permet de conclure cette preuve en posant $v = w = \epsilon$ dans la propriété ci-dessus. Intuitivement, l'automate cheminant A' se

comporte sur le nœud v de l'arbre t de la même façon que l'automate cheminant avec oracle (A, O) se comporte sur le nœud $h_N(v)$ dans l'arbre $h(t)$ et comme l'information structurelle sur un de ces nœuds est toujours (f, s^O) sauf à la racine, l'automate cheminant A' n'a pas besoin d'oracle.

On définit maintenant formellement A'. Soient t un arbre et v un nœud de t. On considère r une exécution de (A, O) sur $h(t)$ partant de la configuration $(q, h_N(v))$. Soit $(p, h_N(w))$ la deuxième configuration de r dont le nœud courant est l'image d'un nœud de t par le morphisme h_N, la première configuration de r de ce type étant $(q, h_N(v))$. On a au plus 5 possibilités pour le nœud w : il peut être le père de v, un des deux fils de v, v lui même ou bien le frère de v. La dernière possibilité est due au fait qu'un chemin dans le motif C d'un port vers l'autre port ne passe pas nécessairement par la racine de C. Cette observation permet de définir formellement A' à partir des exécutions de A dans s, s_a et C. Par exemple, s'il existe une exécution de (A, O) dans $C[s_a, s_a]$ de la configuration $(q, h_N(\epsilon 1))$ à la configuration $(p, h_N(\epsilon 2))$ qui ne passe par aucun autre nœud de la forme $h_N(x)$ où x est un nœud de l'arbre quasiment blanc dont les sous-arbres gauche et droit sont réduits à une feuille étiquetée par a alors on ajoute à A' les transitions $(q, a, (\mathrm{fe}, 1), (q', \uparrow))$ et $(q', a, (\mathrm{in}, \mathrm{ra}), (p, \searrow))$ où q' est un nouvel état intermédiaire qui apparaît uniquement dans ces deux transitions de A'.

Si (A, O) est déterministe, étant donnée r une exécution de (A, O) sur $h(t)$ partant de la configuration $(q, h_N(v))$, la deuxième configuration de r dont le nœud courant est l'image d'un nœud de t par le morphisme h_N est définie de manière unique et l'automate A' est alors déterministe. □

Le lemme ci-dessus nous permet d'éliminer l'oracle et de nous ramener à la proposition 6.1 ou au lemme 6.4 dans la preuve des deux corollaires suivants.

Corollaire 6.1 *Pour tout automate cheminant avec oracle, il existe des arbres s et t tels que $s \in \mathcal{L}^{p,p}$ et $t \notin \mathcal{L}^{p,p}$ qui ont les mêmes couples d'états correspondant aux boucles de la racine à la racine. En particulier, aucun automate cheminant avec oracle ne reconnaît le langage $\mathcal{L}^{p,p}$.*

Preuve. Ce corollaire est une conséquence des propositions 6.1 et 6.4. On considère (A, O) un automate cheminant avec oracle. Soient h le morphisme simple et A' l'automate cheminant donnés par la proposition 6.4. D'après la proposition 6.1, il existe s' et t' qui ont les mêmes boucles de la racine à la racine pour A tel que $s' \in \mathcal{L}^{p,p}$ et $t' \notin \mathcal{L}^{p,p}$. On pose $s = h(s')$ et $t = h(t')$. D'après la remarque 6.7, on a aussi $s \in \mathcal{L}^{p,p}$ et $t \notin \mathcal{L}^{p,p}$ et s et t ont les mêmes boucles de la racine à la racine pour l'automate cheminant avec oracle (A, O). □

Corollaire 6.2 *Pour tout automate cheminant avec oracle déterministe (A, O), il existe des arbres s et t tels que $s \in \mathcal{L}^{3g}$ et $t \notin \mathcal{L}^{3g}$ et tels que (A, O) a le même 0-comportement pour s et t. En particulier, aucun automate cheminant avec oracle déterministe ne reconnaît le langage $\mathcal{L}^{3g}$.*

Preuve. Ce corollaire est une conséquence de la proposition 6.4 et du lemme 6.4. On considère (A, O) un automate cheminant déterministe avec oracle. Soient h le morphisme simple et A' l'automate cheminant déterministe donnés par la proposition 6.4. D'après le lemme 6.4, il existe s' et t' qui ont les mêmes boucles de la

racine à la racine pour A tel que $s' \in \mathcal{L}^{3g}$ et $t' \notin \mathcal{L}^{3g}$. On pose $s = h(s')$ et $t = h(t')$. D'après la remarque 6.7, on a aussi $s \in \mathcal{L}^{3g}$ et $t \notin \mathcal{L}^{3g}$ et s et t ont les mêmes boucles de la racine à la racine pour l'automate cheminant avec oracle (A, O).

□

6.2.2 $\mathrm{PTA_k} \subsetneq \mathrm{PTA_{k+1}}$

Le but de cette partie est de montrer pour tout entier k strictement positif que le langage $\mathcal{L}_{k+1}$ n'est pas reconnu par un automate à k jetons. Nous allons pour cela considérer un automate à k jetons A et construire inductivement des arbres de part et d'autres du langage $\mathcal{L}_{k+1}$ que A n'arrive pas à distinguer.

Nous rappelons que tous les automates à jetons utilisés dans les preuves de ce chapitre sont des automates du modèle faible. D'après les résultats du chapitre 5, les résultats des théorèmes que nous prouvons dans ce chapitre s'étendent au modèle fort d'automates d'arbres à jetons.

Nous avons introduit le pliage d'un arbre à niveaux selon un langage d'arbres pour construire les familles de langages de séparation $(\mathcal{L}_n)_{n\in\mathbb{N}_{>0}}$ et $(M_n)_{n\in\mathbb{N}_{>0}}$. Nous commençons par définir dans cette partie les *pliages par comportements* d'un arbre qui seront utilisés dans les preuves par induction qui vont suivre.

Définition 6.11 *Soient $i \in \mathbb{N}$, A un automate d'arbres à i jetons et t un arbre sur un alphabet Σ. On note $\mathcal{B}$ l'ensemble des i^*-comportements de A pour des arbres et pour $j \in [0, i]$ on note $\mathcal{B}_j$ l'ensemble des j-comportements de A pour des arbres. On considère que $\mathcal{B} = \mathcal{B}_0 \times \cdots \times \mathcal{B}_i$.*

Un pliage de t par comportements de A est un arbre sur l'alphabet $\Sigma \cup (\Sigma \times \mathcal{B})$ obtenu à partir de t en remplaçant pour certains nœuds v de t le sous-arbre t_v par un seul nœud étiqueté par (σ, B) où σ est l'étiquette de v et B le i^*-comportement pour t_v.

Le lemme suivant montre comment construire à partir de l'automate A un automate à jetons qui simule sur un pliage de t par comportements de A une exécution de l'automate à jetons A sur t.

Lemme 6.7 *Soit $i \in \mathbb{N}$. Pour tout automate à i jetons A, il existe un automate à i jetons A' qui reconnaît l'ensemble des pliages par comportements de A d'arbres acceptés par A. Si A est déterministe, A' peut être choisi déterministe.*

Preuve. On considère t un arbre et t' un pliage de t par comportements de A. On construit ci-dessous un automate à i jetons A' qui se comporte comme A tant qu'il ne visite pas de feuilles étiquetées par un comportement. Supposons alors que A' atteint une feuille v de t' étiquetée par un i^*-comportement $B = (B_0, \cdots, B_i)$ après avoir posé les jetons numérotés de i à $(j+1)$ avec $j \in [0, i]$. On a $j = 0$ si tous les jetons ont été posés et $j = i$ si aucun jeton n'a été posé. L'automate A' simule alors la j-boucle de A dans le sous-arbre en calculant un couple d'états (q, q') de $B^j(\gamma)$ où γ est la $(j-1)^*$-classe d'équivalence du contexte $C_{t,v}$ marqué par les jetons numérotés de i à $(j+1)$. La classe d'équivalence γ ne peut pas être calculée de manière exacte dans le cas non-déterministe. On utilise alors le lemme 5.7 : A' devine une classe de $(j-1)^*$-équivalence de contexte γ' correspondant à un $(j-1)^*$-comportement pour

contexte, puis A' vérifie que $\gamma' \leq \gamma$ en simulant l'automate non-déterministe obtenu avec le lemme 5.7. Si c'est bien le cas il choisit le couple d'états (q, q') dans $B_j(\gamma')$ et sinon il rejette.

On adapte la construction ci-dessus au cas déterministe : pour calculer la j-boucle effectuée par A dans le sous-arbre t_v, A' calcule de manière exacte et déterministe le $(j-1)^*$-comportement pour le contexte $C_{t,v}$ en utilisant le lemme 5.9. □

Remarque 6.8 Dans la preuve du lemme 6.7 ci-dessus, pour simuler dans l'arbre t' une j-boucle de A dans le sous-arbre t_v, l'automate A' a seulement besoin de l'information concernant le j-comportement de A : il utilise uniquement la composante B_j de B dans l'étiquette de la feuille v de t'.

La suite de cette partie consiste à prouver le lemme ci-dessous qui nous permet de déduire avec le lemme 6.3 que le pouvoir d'expression des automates à k jetons augmente quand l'entier k augmente dans le cas déterministe et non-déterministe.

Lemme 6.8 *Pour tout entier k strictement positif, le langage $\mathcal{L}_k$ n'est pas reconnu par un automate à $(k-1)$ jetons.*

Nous considérons $k \geq 1$ et A un automate à $(k-1)$ jetons du modèle faible avec m états. Nous construisons maintenant inductivement pour $i \in [1, k]$ des arbres s_i et t_i à i niveaux $(i-1)^*$-équivalents pour A tels que $s_i \in \mathcal{L}_i$ et $t_i \notin \mathcal{L}_i$. Les arbres s_1 et t_1 seront définis en utilisant le lemme suivant.

Lemme 6.9 *Soit $n > 0$, il existe des arbres à 1 niveau $s_1 \in \mathcal{L}_1$ et $t_1 \notin \mathcal{L}_1$ tels que pour tout automate cheminant avec oracle de taille inférieure à n, les arbres s_1 et t_1 ont les mêmes couples d'états correspondant à des boucles de la racine à la racine.*

Preuve. On considère l'ensemble fini $\mathcal{E}_n$ des automates cheminants avec oracle de taille inférieure à n. On construit un automate cheminant avec oracle $(A_{\mathcal{E}_n}, O_{\mathcal{E}_n})$ qui choisit de manière non-déterministe un automate cheminant avec oracle de $\mathcal{E}_n$ et qui le simule. D'après le corollaire 6.1, il existe $s_1 \in \mathcal{L}_1$ et $t_1 \notin \mathcal{L}_1$ qui ont les mêmes couples d'états correspondant à des boucles de la racine à la racine pour $A_{\mathcal{E}_n}$. D'après la construction de $A_{\mathcal{E}_n}$, les arbres s_1 et t_1 ont les mêmes couples d'états correspondant à des boucles de la racine à la racine pour tous les automates de $\mathcal{E}_n$. □

Nous revenons maintenant à la preuve du lemme 6.8 et nous considérons des arbres s_1 et t_1 donnés par le lemme 6.9 pour un certain entier n que nous définirons par la suite. Notons que cet entier n dépendra seulement de l'automate à $(k-1)$ jetons A.

Nous construisons inductivement à partir de ces deux arbres deux familles d'arbres s_i et t_i pour $i \in [1, n]$ de la manière suivante. Pour $i > 1$, l'arbre s_i, respectivement l'arbre t_i, est obtenu à partir de s_1, respectivement à partir de t_1, en remplaçant toutes les feuilles dont l'étiquette est la lettre a par le sous-arbre s_{i-1} et toutes les feuilles dont l'étiquette est la lettre b par le sous-arbre t_{i-1}.

La figure 6.7 montre comment nous construisons inductivement l'arbre s_i pour un entier $i > 1$.

Pour $i > 0$, nous disons qu'un arbre à $(i+1)$ niveaux est **difficile** si tous ses sous-arbres de niveaux i (c'est-à-dire les sous-arbres enracinés en un nœud du niveau 2) sont soit s_i soit t_i. Par exemple, les arbres s_{i+1} et t_{i+1} sont difficiles.

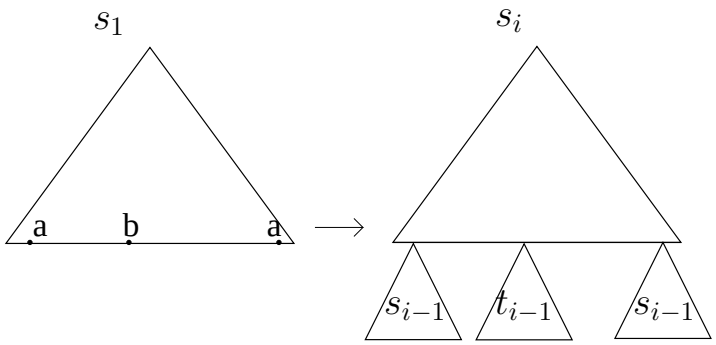

FIG. 6.7 – Construction de l'arbre s_i à partir de l'arbre s_1

Le lemme 6.10 ci-dessous implique que A accepte s_k si et seulement si A accepte t_k. Comme s_k appartient au langage $\mathcal{L}_k$ et que t_k n'appartient pas à ce langage, on montre ainsi que A ne reconnaît pas ce langage et comme A est un automate à $(k-1)$ jetons choisis de manière quelconque , il nous reste pas qu'à prouver le lemme 6.10 pour conclure la preuve du lemme 6.8.

Lemme 6.10 *Pour chaque $i \in [0, k-1]$ les arbres s_{i+1} et t_{i+1} sont i^*-équivalents.*

Preuve. On procède par induction sur i. Pour le cas $i = 0$ il suffit de choisir l'entier n utilisé pour construire les arbres s_1 et t_1 avec le lemme 6.9 tel que n soit supérieur à la taille de l'automate cheminant avec oracle correspondant à l'automate cheminant défini par A lorsque tous ses jetons sont posés. On rappelle qu'un automate cheminant est un cas particulier d'automate cheminant avec oracle.

Soit maintenant $i > 0$. On considère B un i^*-comportement de A pour un arbre et on note A_B l'automate à i-jetons construit à l'aide du lemme 5.7 qui accepte les arbres pour lesquels le i^*-comportement de A est supérieur ou égal à B. A l'aide du lemme 6.7, on construit A'_B l'automate à i-jetons qui simule A_B sur les pliages d'arbres par comportement de A_B. Il suffit maintenant de prouver qu'il existe des pliages des arbres s_{i+1} et de t_{i+1} par i^*-comportement de A_B qui ne peuvent être distingués par A'_B. Ceci implique en effet d'après le lemme 6.7 que A_B ne peut pas différencier s_{i+1} de t_{i+1}. Comme on a choisi arbitrairement B parmi tous les i^*-comportement de A pour des arbres, on pourra alors conclure que s_{i+1} et t_{i+1} sont équivalents.

On note respectivement B_s et B_t les i^*-comportements de l'automate A_B pour s_i et de t_i.

Dans cette preuve, étant donné le i^*-comportement pour arbre B de l'automate A, on appelle le B-repliage $\bar{t}$ d'un arbre difficile t à $(i+1)$ niveaux le pliage de t par comportement de A_B dans lequel chaque sous-arbre s_i enraciné au niveau 2 est remplacé par un nœud dont l'étiquette contient le comportement B_s et chaque sous-arbre t_i enraciné au niveau 2 est remplacé par un nœud dont l'étiquette contient le comportement B_t. On note que le B-repliage de s_{i+1} (respectivement de t_{i+1}) est obtenu partir de s_1 (respectivement t_1) en remplaçant les feuilles dont l'étiquette est la lettre a par $B - s$ et les feuilles dont l'étiquette est la lettre b par B_t.

Il nous reste plus qu'à prouver que A'_B a les mêmes couples d'états correspondant à des i-boucles de la racine à la racine dans s_{i+1} et dans t_{i+1}

L'idée générale de la preuve de la proposition ci-dessus est que l'automate à i jetons A'_B peut être simulé par un automate cheminant avec oracle sur les B-repliages d'arbres difficiles. Nous allons donc construire un automate cheminant avec oracle (A_0, O) qui accepte exactement les mêmes B-repliages d'arbres difficiles que A'_B. La taille de (A_0, O) va dépendre de celle de A'_B qui dépend elle même de la taille de A. On considérera que l'entier k qui a permis de définir s_1 et t_1 a été choisi suffisamment grand pour que (A_0, O) ait les mêmes couples d'états correspondant à des boucles de la racine à la racine dans les B-repliages de s_{i+1} et de t_{i+1}.

On définit maintenant l'automate cheminant avec oracle (A_0, O). Rappelons qu'une i-boucle de A'_B peut être décomposée en une suite de i-exécutions qui sont chacune de l'une des deux formes suivantes

- une i-exécution (c, c') constituée de deux i-configurations telles que c' est obtenue à partir de c avec une simple transition qui ne pose pas et ne lève pas de jeton.
- une i exécution qui commence avec une transition qui pose le jeton i suivie d'une $(i-1)$ boucle et enfin d'une transition qui lève le jeton i

Une i-exécution de la première forme peut être simulée sans difficulté par un automate cheminant (même sans oracle). Il reste donc pour finir cette preuve à montrer comment simuler une i-exécution de la deuxième forme.

On doit donc prouver qu'étant donné le B-repliage $\bar{t}$ d'un arbre difficile marqué et un nœud u de $\bar{t}$, l'existence d'une $(i-1)$-boucle partant du nœud u et d'un état p et terminant dans un état q ne dépend pas des étiquettes des nœuds de $\bar{t}$. On rappelle que tous les nœuds internes de $\bar{t}$ sont étiquetés par la lettre b et les feuilles de $\bar{t}$ par B_s ou B_t les i^*-comportements de A_B. D'après la remarque 6.8, les $(i-1)$-boucles de A'_B ne dépendent pas de toute l'information des i^*-comportements pour s_i et pour t_i mais seulement de l'information sur le $(i-1)^*$ comportement pour s_i et pour t_i. D'après notre hypothèse d'induction, l'automate A a le même $(i-1)^*$-comportement pour les arbres s_i et t_i. A_B a donc aussi le même $(i-1)^*$-comportement pour ces arbres d'après la remarque . Les $(i-1)$-boucles de A'_B peuvent ainsi être simulées par un automate à $(i-1)$ jetons dont les transitions ne dépendent pas de l'étiquette du nœud courant. Les preuves des propositions de la section 2.5 montrent qu'on peut remplacer cet automate à $(i-1)$ jetons par un automate d'arbres bottom up en construisant pour chaque couples d'états (p, q) de A'_B une formule MSO intermédiaire à une variable libre satisfaite en u si et seulement si (p, q) correspond à une $(i-1)$-boucle de A'_B partant du nœud u ; on construit ainsi un oracle O tel que, pour chaque nœud v, la O information structurelle sur $(\bar{(t)}, v)$ donne l'ensemble des pairs d'états correspondant à une $(i-1)$-boucle de v à v. Nous verrons dans le chapitre 7 comment construire directement l'automate d'arbres bottom-up déterministe O qui calcule les i-boucles de A_B. □

Nous avons ainsi prouvé dans cette section que le langage d'arbres L_k n'est pas reconnu par un automate d'arbres à $(k-1)$ jetons. Nous avons également montré dans la proposition 6.3 que le langage L_k est reconnu par un automate d'arbres déterministe à k jetons. Nous pouvons conclure que les automates à $(k+1)$ jetons reconnaissent plus de langages que les automates à k jetons dans le cas déterministe comme dans le cas non-déterministe.

Théorème 6.1 *Il existe un langage d'arbres reconnu par un automate d'arbres à $(k+1)$ jetons qui ne peut pas être reconnu par un automate d'arbres à k jetons.*

Il existe un langage d'arbres reconnu par un automate d'arbres déterministe à $(k+1)$ jetons qui ne peut pas être reconnu par un automate d'arbres déterministe à k jetons.

6.2.3 TWA $\not\subseteq$ DPTA$_k$

Toutes les preuves de la partie 6.2.2 s'adaptent au cas déterministe et à la famille de langages d'arbres $(\mathcal{M}_n)_{n\in\mathbb{N}_{>0}}$ construite dans la partie 6.1.3 pour prouver le lemme suivant :

Lemme 6.11 *Pour tout entier k strictement positif, le langage $\mathcal{M}_k$ n'est pas reconnu par un automate d'arbres déterministe à $(k-1)$ jetons.*

D'après le théorème 3.4, on peut se restreindre au cas des automates d'arbres à jetons à exécutions finies.

On démontre d'abord la variante suivante pour le cas déterministe du lemme 6.9.

Lemme 6.12 *Soit $n > 0$, il existe des arbres à 1 niveau $s_1 \in \mathcal{M}_1$ et $t_1 \notin \mathcal{M}_1$ tels que pour tout automate cheminant déterministe avec oracle à exécutions finies de taille inférieure à n les arbres s_1 et t_1 ont les mêmes couples d'états correspondant à des boucles de la racine à la racine.*

Preuve. On considère l'ensemble fini $\mathcal{E}_k$ des automates cheminants avec oracle de taille inférieur à k et on construit un automate cheminant déterministe avec oracle qui simule tous les automates de $\mathcal{E}_k$ l'un après l'autre et qui mémorise dans son état pour chacun de ses automates tous les couples d'états correspondant à des boucles de la racine à la racine. D'après le corollaire 6.2, il existe $s_1 \in \mathcal{M}_1$ et $t_1 \notin \mathcal{M}_1$ qui ont les mêmes couples d'états correspondant à des boucles de de la racine à la racine. Ces arbres ont ainsi les mes couples d'états correspondant à des boucles de la racine à la racine pour tous les automates de $\mathcal{E}_k$. □

On considère ensuite un automate d'arbres déterministe à $(k-1)$ jetons et on construit comme dans la preuve du lemme 6.8 des arbres s_k et t_k qui ne peuvent être distingués par A et tels que $s_k \in \mathcal{M}_k$ et $t_k \notin \mathcal{M}_k$.

Le théorème suivant est une conséquence directe des lemmes 6.5 et 6.11.

Théorème 6.2 *Pour tout entier $k > 0$, il existe un langage d'arbres reconnu par un automate d'arbres cheminant non-déterministe qui ne peut pas être reconnu par un automate d'arbres déterministe à k jetons.*

M. Bojańczyk et T. Colcombet ont prouvé dans [6] que les automates cheminants ne peuvent pas être déterminisés. Le théorème 6.2 est une généralisation de ce résultat : on ne peut pas déterminiser un automate cheminant même avec un nombre fixé de jetons. Si on ne fixe pas le nombre k de jetons, le problème consistant à transformer un automate cheminant en un automate d'arbres à jetons déterministe est un problème ouvert.

6.3 Automates à jetons et langages réguliers

Le résultat que nous présentons dans cette section est le plus important du chapitre. M. Bojańczyk et T. Colcombet ont démontré dans [7] que les automates d'arbres cheminants ne reconnaissent pas tous les langages d'arbres réguliers. Nous généralisons ce résultat en prouvant que les automates d'arbres à jetons ont moins de pouvoir d'expression que les automates d'arbres bottom-up qui effectuent des calculs parallèles. Nous construisons pour cela un langage $\mathcal{L}^r$ qui contient l'union des langages de la famille $(\mathcal{L}_n)_n$ et tels que pour tout entier k les arbres à k niveaux qui n'appartiennent pas à $\mathcal{L}_k$ n'appartiennent pas à $\mathcal{L}^r$. La preuve du lemme 6.8 montre que $\mathcal{L}^r$ ne peut pas être reconnu par un automate d'arbres à jetons et il nous reste donc plus qu'à prouver que $\mathcal{L}^r$ est un langage régulier. Enfin à l'aide de la caractérisation logique des automates d'arbres à jetons que nous avons vu dans le chapitre 4 nous déduisons de ce qui précède que la logique du premier ordre avec clôture transitive positive sur les arbres est strictement moins expressive que la logique monadique du second ordre sur les arbres.

6.3.1 A propos de l'union des $\mathcal{L}_n$

Notons $\mathcal{L}_\cup$ l'union des langages de la famille $(\mathcal{L}_n)_{n\in\mathbb{N}}$. Ce langage ne peut pas être reconnu par un automate à jetons. En effet, nous avons prouvé avec le lemme 6.10 que pour chaque entier k strictement positif et pour chaque automate A à k jetons, il existe un arbre s appartenant à $\mathcal{L}_k$ et un arbre t à k niveaux qui n'appartient pas à $\mathcal{L}_k$ (et qui n'appartient pas à $L_\cup$) tels que A ne peut distinguer s et t. Ainsi, aucun automate d'arbres à jetons ne peut reconnaître le langage $\mathcal{L}_\cup$. Cependant, ce langage n'est pas régulier. En effet tous les arbres de $\mathcal{L}_\cup$ sont tels que tous les chemins de la racine à une feuille ont le même nombre de nœuds étiquetés par la lettre c et ce nombre peut être arbitrairement grand et un automate d'arbres bottom-up ne peut pas vérifier avec sa mémoire bornée qu'un arbre donné est d'un certain niveau k où k est un entier quelconque. Nous allons donc transformer $\mathcal{L}_\cup$ en un langage régulier tels que les arbres s_k du lemme 6.10 appartiennent à ce langage régulier et que les arbres t_k de ce lemme n'appartiennent pas à ce langage.

6.3.2 Le langage régulier $\mathcal{L}^r$

Nous allons introduire un nouveau langage $\mathcal{L}^r$ semblable à $\mathcal{L}_\cup$ mais tel que les arbres de $\mathcal{L}^r$ ne suivent pas la restriction imposant que tous les chemins de la racine à une feuille aient le même nombre de nœuds étiquetés par la lettre c.

Le langage $\mathcal{L}^r$ que nous définissons dans cette partie est un langage d'arbres à niveaux dans lequel les chemins de la racine à une feuille dans un arbre n'ont pas tous les même nombre de c.

Définition 6.12 *Un arbre à niveaux est un arbre sur l'alphabet $\{a, b, c\}$ tel que tout chemin de la racine à une feuille est étiqueté par un mot du langage $(cb^*)^*(a + b)$.*

Un arbre à niveaux est un arbre dont la racine est étiquetée par la lettre c, les nœuds internes par les lettres b et c et les feuilles par les lettres a et b.

Nous devons maintenant adapter la définition 6.3 pour introduire le L-repliage d'un arbre à niveaux où L est un langage d'arbres à niveaux.

Définition 6.13 *Soit K un langage d'arbre à niveaux et t un arbre à niveaux. Le K-repliage de t est l'arbre quasiment blanc obtenu à partir de t en remplaçant chaque sous-arbre enraciné en un nœud u étiqueté par la lettre c et dont le seul ancêtre étiqueté par la lettre c est la racine par une feuille étiquetée par la lettre a si le sous-arbre en question est dans K et par une feuille étiquetée par b sinon.*

Nous pouvons maintenant définir formellement le langage $\mathcal{L}^r$.

Définition 6.14 *Le langage d'arbres $\mathcal{L}^r$ est l'union des langages $\mathcal{L}^r_k$ pour $k \in \mathbb{N}$ tels que*

- $\mathcal{L}^r_0 = \bigcup_{k \in \mathbb{N}_{>0}} \mathcal{L}_k$,
- *pour $k > 0$, le langage $\mathcal{L}^r_k$ est l'union de $\mathcal{L}^r_{k-1}$ et de l'ensemble des arbres à niveaux dont le $\mathcal{L}^r_{k-1}$-repliage est dans $\mathcal{L}^{p,p}$.*

Le langage $\mathcal{L}^r$ est en fait le plus petit langage d'arbres qui contient tous les langages $\mathcal{L}_k$ pour $k > 0$ et tel que le $\mathcal{L}^r$-repliage d'un arbre de $\mathcal{L}^r$ est dans $\mathcal{L}^r$.

Proposition 6.5 *Le langage d'arbres $\mathcal{L}^r$ est régulier.*

Preuve. La proposition 6.2 montre qu'il existe $\mathcal{B}$ un automate d'arbres bottom-up déterministe qui reconnaît le langage $\mathcal{L}^{p,p}$. Nous allons construire à partir de cet automate un automate d'arbres bottom-up $\mathcal{B}'$ qui reconnaît le langage $\mathcal{L}^{p,p}$. Notons q_a et q_b les états respectivement attribués par $\mathcal{B}$ à une feuille étiquetée par a et à une feuille étiquetée par b. L'automate $\mathcal{B}'$ se comporte comme $\mathcal{B}$ tant qu'il est sur une nœud étiquetée par a ou b. Lorsque B' rencontre un nœud u étiqueté par c, il simule la transition appliquée par l'automate $\mathcal{B}$. Si $\mathcal{B}$ attribue au nœud u un état final, $\mathcal{B}'$ attribue à ce nœud l'état q_a sinon $\mathcal{B}'$ attribue à u l'état q_b. □

Nous pouvons maintenant établir le théorème suivant.

Théorème 6.3 *Le langage d'arbres $\mathcal{L}^r$ est un langage d'arbres régulier qui ne peut pas être reconnu par un automate d'arbres à jetons.*

Preuve. On a vu avec la proposition 6.5 que $\mathcal{L}^r$ est un langage d'arbres régulier. Pour chaque automate d'arbres à k jetons A, on construit comme dans la preuve du lemme 6.8 des arbres s_k et t_k que A ne peut pas distinguer et d'après la construction de ces deux arbres, $s_k \in \mathcal{L}^r$ et $t^k \notin \mathcal{L}^r$. Il n'existe donc pas d'automate d'arbres à jetons qui reconnaît le langage $\mathcal{L}^r$. □

Nous avons ainsi montré que les automates d'arbres à jetons reconnaissent plus de langages que les automates cheminants et moins de langage que les automates bottom-up. Cette classe d'automates étudiée dans cette thèse a donc un pouvoir d'expression intéressant.

6.3.3 Une conséquence logique sur les arbres

Nous avons vu une caractérisation logique des automates d'arbres à jetons dans le chapitre 4 : les langages d'arbres reconnus par un automate d'arbres à jetons sont

les langages définis par la logique FO + posTC sur les arbres. Comme les langages d'arbres réguliers correspondent à la logique MSO sur les arbres, le théorème 6.3 permet de séparer la logique FO + posTC de la logique MSO sur les arbres.

Théorème 6.4 *La logique* MSO *sur les arbres est (strictement) plus expressive que la logique* FO + posTC *sur les arbres.*

Ce résultat est intéressant car les logiques FO + posTC et MSO sont équivalentes sur les mots. Le problème de savoir si la logique FO+TC sur les arbres est moins expressive que la logique MSO sur les arbres est un problème ouvert lié au problème ouvert de la complémentation d'un automate d'arbres à jetons : si la classe des automates d'arbres était fermée par complément, la logique FO + posTC sur les arbres et la logique FO+TC sur les arbres seraient équivalentes et ainsi nous pourrions déduire du théorème 6.4 que la logique FO+TC est moins expressive que la logique MSO sur les arbres.

Remarque 6.9 Le problème de savoir si la logique FO+DTC est équivalente à la logique FO + posTC sur les arbres restent un problème ouvert car nous ne savons pas s'il est possible de déterminiser un automate d'arbres à jetons en s'autorisant à ajouter un nombre fixé (mais non borné) de jetons.

CHAPITRE

7

Complexité du vide et de l'inclusion pour les automates à jetons

Dans ce chapitre, nous nous intéressons aux deux problèmes suivants :

1. Etant donné un automate d'arbres à k jetons, cet automate accepte-t-il au moins un arbre ?
2. Etant donnés A_1 et A_2 deux automates à k jetons, les arbres acceptés par A_1 sont ils tous acceptés par A_2 ?

Nous avons montré avec la proposition 2.6 comment construire, pour un automate d'arbres à jetons donné, une formule MSO équivalente. Cette transformation d'un automate d'arbres à jetons en formule logique permet d'obtenir un test non-élémentaire pour les problèmes d'inclusion et du vide. Dans [31], T. Milo, D. Suciu et V. Vianu ont prouvé que cette complexité non-élémentaire est inévitable lorsque le nombre de jetons k fait partie de l'entrée du problème, c'est-à-dire lorsque k n'est pas fixé pour l'ensemble des données du problème. Nous étudions dans ce chapitre les complexités des problèmes du vide et de l'inclusion pour les variantes déterministes et non-déterministes des automates d'arbres à k jetons des modèles forts et faibles. Le nombre de jetons ne fait alors pas partie de l'entrée : il est fixé pour l'ensemble des données d'un problème considéré. Nous introduisons d'abord dans la section 7.1 les notions et les théorèmes classiques de complexité qui nous serviront par la suite. Nous énonçons les résultats de ce chapitre à la fin de la section la section 7.2. Le problèmes du vide pour les automates d'arbres bottom-up est polynomial. Nous rappelons ce résultat dans la section 7.2 et nous montrons une borne supérieure des complexités du vide et de l'inclusion pour les automates d'arbres cheminants en se ramenant au cas des automates d'arbres bottom-up à l'aide d'une construction obtenue avec S.S. Cosmadakis, H. Gaifman, P.C. Kanellakis et M.Y. Vardi dans [14]

et [51]. Nous présentons à la fin de la section 7.2 le résultat donnant les complexités du vide et de l'inclusion pour tous les modèles d'automates d'arbres à jetons. Les sections 7.3 et 7.4 démontrent respectivement les bornes supérieure et inférieure de la complexité de ces problèmes.

7.1 Complexité et machines de Turing

La complexité est une discipline qui consiste à classifier les problèmes décidables en fonction de l'efficacité des algorithmes qui les résolvent. Nous présentons dans cette section des concepts et des théorèmes qui nous serons utiles dans la suite de ce chapitre. Nous définissons dans un premier temps dans la partie 7.1.1 les problèmes de décisions et les machines de Turing. Nous introduisons ensuite dans la partie 7.1.2 les complexités en temps et en espace d'un problème et nous énonçons le théorème de Savitch. Enfin, nous définissons en 7.1.3 les machines de Turing alternantes et nous donnons le lien entre l'espace alternant et le temps.

7.1.1 Machines de Turing et problèmes de décision

Nous définissons tout d'abord ce que nous appelons un problème ou un *problème de décision.*

Définition 7.1 *Un problème de décision est l'énoncé d'une question à laquelle on peut répondre par oui ou par non. Pour chaque problème, on fixe un ensemble d'instances et un sous-ensemble de ces instances pour lesquelles la réponse est oui.*

Les instances d'un problèmes sont souvent appelées les entrées d'un problème. Résoudre un problème consiste à déterminer si une instance pour ce problème est positive c'est-à-dire à associer à chaque entrée de ce problème la sortie "oui" pour une instance positive et "non" pour une autre instance.

A titre d'exemple, nous définissons maintenant *les problèmes du vide et de l'inclusion* auxquels nous nous intéresserons par la suite.

Définition 7.2 *Soit ς une classe d'automates d'arbres, respectivement une classe d'automates sur les mots.*

Le problème du vide pour ς est le suivant : étant donné A un automate de ς, le langage accepté par A est-il vide ?

Le problème de l'inclusion pour C est le suivant : étant donnés A_1 et A_2 deux automates de ς, le langage reconnu par A_1 est-il inclus dans le langage reconnu par A_2 ?

Les *machines de Turing* ont été introduites comme un modèle permettant de modéliser n'importe quel algorithme. Informellement, une machine de Turing travaille sur un ruban infini à droite avec un contrôle fini et une tête de lecture/écriture qui se déplace sur le ruban comme la tête de lecture d'un automate bidirectionnel. Sur ce ruban est écrit un mot fini suivi d'une infinité de symboles blancs. Selon l'état de contrôle et la lettre sous la tête de lecture, la machine choisit un nouvel état et peut se déplacer d'une case et réécrire le symbole courant. Elle peut également s'arrêter pour accepter ou bien rejeter le mot pris en entrée.

Définition 7.3 *Une machine de Turing est un quadruplet $\mathcal{M} = (Q, \Sigma, q_{in}, F, \delta)$, où Q est un ensemble fini d'états, q_{in} est l'état initial, F l'ensemble des états finaux, Σ est un alphabet disjoint de Q et contenant toujours deux symboles particuliers qui sont le marqueur de début noté $\triangleright$ et le caractère blanc noté $\square$ et δ est un sous-ensemble de $(Q \times \Sigma) \times (Q \times \Sigma \times \{\rightarrow, \leftarrow, \circlearrowleft\})$ appelé relation de transition. Une machine de Turing est déterministe si δ correspond à une fonction de $Q \times \Sigma$ à valeurs dans $Q \times \Sigma \times \{\rightarrow, \leftarrow, \circlearrowleft\}$.*

Nous imposons les restrictions suivantes à la relation de transition : le marqueur de début n'est jamais effacé. Une transition ne peut donc pas être de la forme $((q, \triangleright), (q', b, D))$ avec $b \neq \triangleright$ ou avec $D = \leftarrow$.

Une transition de la forme $((q, a), (q', b, D))$ signifie que si l'état de contrôle de la machine est q et si sa tête de lecture est sur le symbole a, $\mathcal{M}$ peut choisir d'aller dans l'état q' puis de réécrire le symbole b à la place du symbole a sur lequel est la tête de lecture/écriture de $\mathcal{M}$ et enfin d'effectuer le déplacement D.

Plus formellement, une *configuration* de $\mathcal{M}$ est un triplet (q, u, v) où $q \in Q$ est l'état de contrôle et où le mot uv du langage $\triangleright\Sigma^*$ suivi d'une infinité de symboles blancs est le contenu du ruban. Pour cette configuration, la tête de lecture/écriture de la machine pointe sur le premier symbole de v. Nous notons a la première lettre de $v = av''$.

Supposons qu'il existe une transition de la forme $((q, a), (q', a', D))$, la configuration (q, u, v) a alors pour successeur la configuration (q', u', v'), ce que nous notons $(q, u, v) \xrightarrow{\mathcal{M}} (q', u', v')$, dans les cas suivants :

- $D = \rightarrow$, $u' = ua'$ et $v' = v''$,
- $D = \leftarrow$, $u = u'b$ avec $b \in \Sigma$ et $v = ba'v''$,
- $D = \circlearrowleft$, $u' = u$ et $v' = a'v''$.

Nous notons $\xrightarrow{\mathcal{M}\,*}$ la clôture transitive de la relation $\xrightarrow{\mathcal{M}}$. Si $(q, u, v) \xrightarrow{\mathcal{M}} (q', u', v')$ on dit qu'il existe un *calcul* ou une exécution de $\mathcal{M}$ de la configuration (q, u, v) à la configuration (q', u', v'). Un calcul sur un mot w est un calcul partant de la *configuration initiale* $(q_{in}, \epsilon, \triangleright w)$. Un calcul est *acceptant* s'il termine dans une configuration où l'état de contrôle est un état final. Une machine de Turing est *à exécutions finies* si pour toute entrée w, tous les calculs de $\mathcal{M}$ sur w sont finis. Tout comme les automates, les machines de Turing définissent des langages.

Définition 7.4 *Le langage accepté par $\mathcal{M}$ est l'ensemble des mots w de Σ^* tels qu'il existe un calcul acceptant sur w. Un langage est décidé par une machine de Turing déterministe $\mathcal{M}$ si $\mathcal{M}$ est à exécutions finies et si $\mathcal{M}$ accepte ce langage.*

Si nous prenons en entrée les codages des instances d'un problème, une machine de Turing déterministe à exécutions finies peut être considérée comme un algorithme résolvant un problème, c'est-à-dire acceptant ou rejetant l'entrée. Une machine de Turing peut également modéliser un algorithme de construction. En effet, une construction est un algorithme produisant une sortie et étant donné un mot pris en entrée, nous considérons que la machine arrête son calcul dès qu'elle atteint un état acceptant et que le codage de la sortie produit par la machine est alors le contenu de son ruban.

7.1.2 Le temps et l'espace

Nous nous intéressons maintenant aux ressources en temps et en espace utilisées par une machine de Turing pour décider un problème .

Définition 7.5 *Soit une fonction f de $\mathbb{N}$ à valeurs dans $\mathbb{N}$.*

Un langage L est décidé en temps $f(n)$ par une machine de Turing déterministe $\mathcal{M}$ si $\mathcal{M}$ décide L et si pour toute instance w, la machine accepte ou rejette w en $f(|w|)$ étapes de calcul au plus. Dans ce cas, on note que $L \in$ TIME $(f(n))$.

Un langage est décidé en temps $f(n)$ par une machine de Turing non-déterministe $\mathcal{M}$ si $\mathcal{M}$ accepte L et si pour toute entrée w, tout calcul de $\mathcal{M}$ sur w prend au plus $f(|w|)$ étapes. Dans ce cas, on note que $L \in$ NTIME $(f(n))$.

Un langage L est décidé en espace $f(n)$ par une machine de Turing déterministe $\mathcal{M}$ si $\mathcal{M}$ décide L et si pour toute entrée w, le nombre de symboles non-blancs écrits sur le ruban lors du calcul de la machine sur w n'excède pas $f(|x|)$. Dans ce cas on note que $L \in$ SPACE $(f(n))$.

Un langage L est décidé en espace $f(n)$ par une machine de Turing non-déterministe $\mathcal{M}$ si $\mathcal{M}$ accepte L et si pour toute entrée w et pour tout calcul de $\mathcal{M}$ sur w, le nombre de symboles non-blancs écrits sur le ruban n'excède pas $f(|x|)$. Dans ce cas, on note que $L \in$ NSPACE $(f(n))$.

Les ensembles de langages TIME $f(n)$, NTIME $f(n)$, SPACE $f(n)$ et NSPACE $f(n)$ sont des classes de complexité.

Les constructions en temps ou en espace $f(n)$ se définissent formellement de manière similaire. Lorsque nous décrivons un algorithme en temps $f(n)$ nous ne définissons pas la machine de Turing et le codage associé de l'entrée mais nous vérifierons que les étapes élémentaires de l'algorithme sur une entrée e sont bornées par $f(|e|)$ où $|e|$ est la taille de l'entrée. Par exemple, lorsque dans le théorème 3.3 nous montrons comment construire en temps polynomial le complément d'un automate d'arbres cheminant nous ne considérons pas les codages des automates cheminants et une machine de Turing permettant de définir formellement la construction. De même, pour montrer qu'un algorithme s'effectue en espace $f(n)$ nous vérifions simplement que pour chaque entrée e la mémoire utilisée par l'algorithme est bornée par $f(|e|)$.

Notation 7.1 *On donne ci-dessous quelques classes de complexité classiques :*

- P $= \bigcup_{j>0}$ TIME(n^j),
- PSPACE $= \bigcup_{j>0}$ SPACE(n^j),
- NPSPACE $= \bigcup_{j>0}$ NSPACE(n^j),
- EXPTIME $= \bigcup_{j>0}$ TIME(2^{n^j}),
- EXPSPACE $= \bigcup_{j>0}$ SPACE(2^{n^j}).

On note enfin $\mathrm{tow}(k,n)$ *la fonction tour d'exponentielle de hauteur k, définie récursivement sur k par* $\mathrm{tow}(0,n) = n$ *et* $\mathrm{tow}(k+1,n) = 2^{\mathrm{tow}(k,n)}$.

*On définit alors la classe de complexité k-*EXPTIME *de la manière suivante :*

k-EXPTIME $= \cup_{j>0}$TIME $(\mathrm{tow}(k, n^j))$

Nous utilisons dans la section 7.4 une conséquence immédiate du théorème de Savitch prouvé dans [43] reliant les classes de complexités en espace déterministe et non-déterministe.

Proposition 7.1 (Savitch) PSPACE =NPSPACE

Nous allons chercher dans les autres sections de ce chapitre, à déterminer dans quelles classes de complexité se trouvent les langages des instances positives pour les problèmes du vide et de l'inclusion sur les différentes classes d'automates d'arbres à k jetons pour un entier k fixé. Les classes de complexité que nous avons introduites donnent une borne supérieure pour la complexité d'un problème ou d'une construction. Si nous prouvons qu'un algorithme résout un problème en temps exponentiel, ce problème peut aussi être résolu par un autre algorithme en temps polynomial. La notion de réduction polynomiale permet d'introduire un ordre entre certaines classes de complexité et la notion de problème dur permet d'introduire une borne inférieure pour la complexité d'un problème.

Définition 7.6 *Soient deux langages L_1 et L_2. On dit que L_1 se réduit polynomialement à L_2 s'il existe une fonction f calculable en temps polynomial telle que pour tout mot w, $w \in L_1$ si et seulement si $f(w) \in L_2$.*

Soit une classe de complexité $\mathcal{C}$, le langage L est $\mathcal{C}$-dur si et seulement si tout problème de $\mathcal{C}$ se réduit polynomialement à L. De plus si $L \in \mathcal{C}$, L est dit $\mathcal{C}$-complet.

7.1.3 Machines de Turing alternantes

Une *machine de Turing alternante* est une extension des machines de Turing. Une telle machine a la possibilité à chaque étape de créer des copies supplémentaires d'elle même qui continueront indépendamment le calcul dans de nouveaux états de contrôle. La notion d'alternance a été introduite par A.K. Chandra, D.C. Kozen et L.J. Stockmeyer dans [10].

Nous définissons formellement ci-dessous les machines de Turing alternantes.

Définition 7.7 *Une machine de Turing alternante est un quadruplet*

$\mathcal{M} = (Q, \Sigma, Q_\exists, Q_\forall, q_{in}, F, \delta)$, où Q est un ensemble fini d'états, $Q_\exists \subseteq Q$ est l'ensemble des états existentiels, Σ est un alphabet qui contient le caractère blanc et le marqueur de début, $Q_\forall \subseteq Q$ est l'ensemble des états universels, q_{in} est l'état initial, F l'ensemble des états finaux, Σ est un alphabet disjoint de Q et contenant toujours deux symboles particuliers qui sont le sélectionneur de début noté $\triangleright$ et le blanc noté $\square$ et enfin $\delta_\exists$ est un sous-ensemble de $(Q_\exists \times \Sigma) \times (Q \times \Sigma \times \{\rightarrow, \leftarrow, \circlearrowleft\})$ appelée ensemble des transitions existentielles, et $\delta_\forall$ est un sous-ensemble de $((Q_\forall \times \Sigma) \times (Q \times \Sigma \times \{\rightarrow, \leftarrow, \circlearrowleft\})^2$ appelé ensemble des transitions universelles.

Une configuration de $\mathcal{M}$ est un triplet (q, u, v) où $q \in Q$, et où $uv \in \triangleright\Sigma^*$ est le contenu du ruban. La configuration (q, u, v) est existentielle si $q \in Q_\exists$, universelle si $q \in Q_\forall$ et acceptante si $q \in F$.

Un calcul ou une exécution d'une machine de Turing alternante $\mathcal{M}$ sur un mot w est un arbre dont les nœuds sont des configurations et les branches sont des calculs

au sens des machines de Turing non-alternantes. La racine de cet arbre est la configuration initiale $(q_{in}, \epsilon, \triangleright w)$. Une configuration universelle (q, u, v) a deux fils, ces deux configurations sont spécifiées par une transition universelle effectuées à partir de (q, u, v). Une configuration existentielle a un unique fils donné par une transition existentielle. Un calcul est acceptant si toutes ses feuilles sont des configurations acceptantes. Une entrée w est acceptée s'il existe un calcul acceptant sur cette entrée.

Nous définissions à nouveau les notions de temps et d'espace pour les machines de Turing alternantes.

Si f est une fonction de $\mathbb{N}$ à valeurs dans $\mathbb{N}$, une machine de Turing alternante décide un langage L en temps $f(n)$ si pour toute entrée w toute branche dans un calcul sur w ne contient pas plus de $f(|w|)+1$ configurations. De même, une machine de Turing alternante décide un langage L en espace $f(n)$ si pour toute entrée w et pour tout calcul de $\mathcal{M}$ sur w le nombre de symboles écrit sur le ruban pour toutes les branches de ce calcul ne dépasse pas $f(|w|)$.

Nous notons ATime $(f(n))$ et ASpace $(f(n))$ les classes de complexité associées.

La propriété suivante relie les classes de complexité alternante en espace et les classes de complexité déterministe en temps. Il s'agit d'une conséquence des résultats de [10].

Proposition 7.2 *[10] On a les égalités suivantes :*
- ExpTime $=$ ASpace(n),
- *pour* $k > 0$, k-ExpTime $= \bigcup_{j>0}$ ASpace$(\mathrm{tow}(k-1, n^j))$

La définition 7.6 de problème ς-dur s'étend aux classes de complexité définies par des machines de Turing alternantes. Pour montrer la borne inférieure du problème du vide des automates d'arbres à k jetons, nous utilisons le résultat suivant qui est une conséquence immédiate de la définition d'un problème ASpace $(f(n))$-complet :

Proposition 7.3 *Le langage des descriptions des machines de Turing, respectivement des machines de Turing alternantes, utilisant un espace* Space $(f(n))$*, respectivement* ASpace $(f(n))$ *et acceptant l'entrée vide est* Space $(f(n))$*-complet, respectivement* ASpace $(f(n))$*-complet.*

7.2 Les problèmes du vide et de l'inclusion pour les automates d'arbres

Cette section présente les résultats du chapitre : nous énonçons les théorèmes 7.2 et 7.3 donnant la complexité du vide et de l'inclusion pour les automates à k jetons où k est un entier fixé sur les mots et sur les arbres, dans les cas déterministe et non-déterministe, pour les modèles forts et faibles. Les bornes supérieures et inférieures de la complexité de ces problèmes théorèmes sont prouvées respectivement dans les sections 7.3 et 7.4. Pour prouver la borne supérieure pour les automates d'arbres à jetons, nous réduisons k-exponentiellement le problème considéré au problème du vide pour les automates d'arbres bottom-up. Il est en effet possible de décider un

temps linéaire si le langage reconnu par un automate d'arbres bottom-up est vide, ce qui implique la proposition suivante :

Proposition 7.4 *Le problème du vide pour les automates d'arbres bottom-up est polynomial.*

Pour montrer que le vide d'un automate à k jetons donné est décidable en temps k-exponentiel, il suffit alors de transformer en temps k-exponentiel un automate à k jetons en un automate d'arbres bottom-up qui reconnaît le même langage. Pour réduire le problème de l'inclusion pour les automates d'arbres à jetons au problème du vide pour les automates d'arbres bottom-up, il suffit également de transformer un automate d'arbres à jetons A en un automate d'arbres bottom-up qui reconnaît le complément du langage reconnu par A.

M.Y. Vardi montre dans [51] comment, étant donné un automate bidirectionnel sur les mots, construire en temps exponentiel deux automates déterministes unidirectionnels qui reconnaissent le même langage et son complément. Il décrit également avec S.S. Cosmadakis, H. Gaifman et P.C. Kanellakis dans [14] comment transformer en temps exponentiel un automate d'arbres cheminant alternant donné en un automate d'arbres bottom-up qui reconnaît le même langage ou en un automate d'arbres bottom-up qui reconnaît le complément de ce langage. Nous décrivons dans la partie 7.2.1 cette construction pour les automates cheminants que nous étendrons ensuite au cas des automates à jetons dans la section 7.3.

7.2.1 Complexité du vide et l'inclusion pour les automates cheminants

Pour prouver que les langages reconnus par un automate cheminant sont réguliers, nous avons vu dans la partie 2.5.2 comment transformer un tel automate en une formule logique MSO. Cette transformation est non-élémentaire. Nous allons prouver ci-dessous qu'il est possible de transformer un automate cheminant en un automate d'arbres bottom-up équivalent ce qui implique que les langages reconnus par les automates d'arbres cheminants sont réguliers. Le lemme ci-dessous a été prouvé dans [14] pour un modèle qui généralise les automates cheminants en les dotant d'un comportement alternant.

Lemme 7.1 ([14]) *Pour tout automate d'arbres cheminant A, on peut construire en temps exponentiel un automate d'arbres bottom-up qui reconnaît le même langage que A et un automate d'arbre bottom-up qui reconnaît le complément du langage reconnu par A.*

Preuve. On considère A un automate d'arbres cheminant. On construit dans cette preuve un automate d'arbres bottom-up B qui calcule les boucles d'arbres de A. L'automate B a donc pour ensemble d'états l'ensemble $2^{Q_A \times Q_A} \times \text{TYPE}$. Rappelons que $\text{TYPE} = \{in, fe\} \times (\{ra, 1, 2\})$ est l'ensemble des types d'un nœud. Dans une exécution acceptante de B, un nœud u est étiqueté par un état dont la première composante est l'ensemble de tous les couples (q, q') d'états de A pour lesquels il existe une boucle de A partant de l'état q et du nœud u, restant dans le sous-arbre enraciné en u et se terminant dans l'état q'. Pour calculer cet ensemble de

couples d'états de A qui correspond aux boucles d'arbres, B devine le type de u en utilisant son non-déterminisme et vérifie cette information en remontant dans l'arbre. L'automate A étant fixé, ces boucles se calculent alors très facilement. En effet, si u est une feuille, elles ne dépendent que de l'étiquette et du type de u et si u est un nœud interne, elles dépendent des types et des étiquettes des nœuds u, u_1 et u_2 ainsi que des boucles d'arbres pour ses fils $u1$ et $u2$. Ainsi, si on veut construire un automate d'arbres bottom-up B qui reconnaît le même langage que A, on choisit comme ensemble d'états acceptants les états tels que

- la première composante contient un couple (q, q') où q est un état initial de A et q' un état acceptant de A et
- la deuxième composante est un des deux types possibles pour la racine, c'est-à-dire un élément de $\{(in, ra), (fe, ra)\}$.

Si on veut construire un automate d'arbres bottom-up B qui reconnaît le complément du langage reconnu par A on choisit comme ensemble d'états acceptants les états tels que

- la première composante ne contient pas de couple (q, q') tel que q est un état initial de A et q' un état acceptant de A et
- la deuxième composante est le type racine.

□

Une conséquence du lemme ci-dessus est que les problèmes du vide et de l'inclusion pour les automates cheminants sont dans ExpTime. En effet, pour décider le vide d'un automate d'arbres cheminant, il suffit de le transformer un temps exponentiel en un automate bottom-up B puis de décider le vide pour B. De même, étant donnés deux automates cheminants A_1 et A_2, pour vérifier si tous les arbres acceptés par A_1 sont aussi acceptés par A_2, il suffit de construire en temps exponentiel un automate d'arbres bottom-up B_1 qui reconnaît le même langage que A_1, un automate d'arbres bottom-up B_2 qui reconnaît le complément du langage de A_2 et l'automate d'arbre bottom-up produit B qui reconnaît l'intersection des langages de B_1 et de B_2 et il ne reste plus qu'à décider le vide pour l'automate $\mathcal{B}$. F. Neven montre implicitement dans [38] la borne inférieure du problème du vide que nous redémontrons dans la section 7.4 : le problème du vide pour les automates d'arbres cheminants est ExpTime-dur. Comme décider le vide d'un automate revient à vérifier si son langage est inclus dans le langage reconnu par un automate trivial sans etat acceptant, nous obtenons alors le théorème suivant.

Théorème 7.1 *Les problèmes du vide et de l'inclusion pour les automates d'arbres cheminants sont* ExpTime*-complets.*

7.2.2 Complexité du vide et de l'inclusion pour les automates d'arbres à jetons

Nous énonçant maintenant les théorèmes donnant les résultats de ce chapitre.

Théorème 7.2 *Soit $k > 0$. Les problèmes du vide et de l'inclusion sont k-*ExpTime*-complets pour les classes d'automates d'arbres suivantes :*

- *les automates d'arbres à k jetons du modèle faible,*
- *les automates d'arbres à k jetons du modèle fort,*

– *les automates d'arbres déterministes à k jetons du modèle faible et*
– *les automates d'arbres déterministes à k jetons du modèle fort.*

Ce théorème sera prouvé dans les deux sections suivantes. Nous prouvons la borne supérieure de la complexité des problèmes considérés en réduisant ces problèmes au problème du vide pour les automates d'arbres bottom-up. Nous montrons alors dans la section 7.3 comment transformer en temps k-exponentiel un automate d'arbres à k jetons A en un automate d'arbre bottom-up qui calcule toutes les boucles de A de la racine à la racine. Pour prouver la borne inférieure, nous utilisons la propriété 7.3 et nous montrons dans la section 7.4 comment simuler une machine de Turing utilisant un espace $(k-1)$-exponentiel avec un automate à k jetons dont la taille est polynomiale par rapport à la taille de l'entrée de la machine de Turing. La proposition 7.2 nous permet alors de conclure.

Remarque 7.1 On remarque tout d'abord que la complexité des problèmes du vide et de l'inclusion est la même pour les automates d'arbres à 1 jeton et pour les automates d'arbres cheminants alors que les pouvoirs d'expression de ces deux classes d'automates sont différents comme nous l'avons vu dans le chapitre 6.

On note ensuite que le problème du vide a la même complexité pour les modèles forts et faibles d'automates à k jetons. Nous avons montré que les deux modèles ont aussi le même pouvoir d'expression mais notre transformation d'un automate à k jetons du modèle fort en un automate équivalent à k jetons du modèle faible est $(k-1)$-exponentielle en supposant que seul le jeton k peut être levé à distance dans l'automate du modèle fort. Il est naturel de se demander si notre transformation $(k-1)$-exponentielle est optimale. Il serait alors intéressant de savoir si le modèle fort est plus concis que le modèle faible.

Nous montrons également qu'en se restreignant aux cas des mots et en adaptant les preuves de la borne supérieure et de la borne inférieure de la complexité des problèmes du vide et de l'inclusion des automates d'arbres à k jetons, nous obtenons le théorème suivant :

Théorème 7.3 *Soit $k > 0$ Les problèmes du vide et de l'inclusion sont $(k-1)$-*ExpSpace*-complet pour les classes d'automates sur les mots suivantes :*
– *les automates à k jetons du modèle faible sur les mots,*
– *les automates à k jetons du modèle fort sur les mots,*
– *les automates déterministes à k jetons du modèle faible sur les mots et*
– *les automates déterministes à k jetons du modèle fort sur les mots.*

7.3 Borne supérieure

Le but de cette section est de montrer comment transformer en temps k-exponentiel un automate d'arbres à k jetons du modèle fort sur un alphabet Σ en un automate d'arbres bottom-up. Nous allons ainsi étendre la construction donnée en [14] pour les automates d'arbres cheminants ou en [51] pour les automates bidirectionnels sur les mots au modèles faible et forts des automates d'arbres à jetons en procédant par induction sur le nombre de jetons.

L'idée de base de l'induction est la suivante : un automate à k jetons A se comporte comme un automate à $(k-1)$ jetons à partir du moment où il pose son jeton k et jusqu'à ce qu'il le lève.

Dans le cas du modèle faible, nous transformons par induction cet automate à $(k-1)$ jetons correspondant à l'automate à k jetons A lorsque son jeton k est posé en un automate d'arbres bottom-up qui calcule toutes ses boucles à partir de la configuration où le jeton k de A vient d'être posé jusqu'à la configuration où ce jeton est sur le point d'être levé. Nous obtenons ainsi un nouveau modèle d'automate d'arbres à jetons qui simule un automate bottom-up quand il pose un jeton. Nous introduisons ce modèle que nous appelons les automates d'arbres faibles bottom-up à k jetons avec la définition 7.8 et nous étendons la construction de [14] à ce modèle.

Dans le cas du modèle fort, pour transformer un automate d'arbres à k jetons du modèle fort en un automate bottom-up nous ne construisons pas d'automate d'arbres à k jetons du modèle faible équivalent car le passage du modèle fort au modèle faible est trop coûteux dès que le nombre de jetons est supérieur à 1. La difficulté pour étendre la construction de [14] au modèle fort d'automates à jetons est que les exécutions d'un automates fort entre le dépot et la levée d'un jeton ne sont pas des boucles comme dans le modèle faible. Nous introduisons aussi deux nouveaux modèles d'automates d'arbres pour gérer le comportement fort puis nous étendons la construction de [14] au modèle fort.

7.3.1 De nouveaux modèles d'automates d'arbres

Nous présentons dans cette partie des nouveaux modèles d'automates que nous définissons pour simuler un automate d'arbres à jetons. Pour transformer un automate d'arbres à jetons en automate bottom-up, nous utilisons en effet des automates intermédiaires qui calculent des informations, par exemple un ensemble de couples d'états correspondant aux exécutions de l'automate d'arbres à jetons considéré.

Si nous pouvons construire, étant donné un automate d'arbres faible à $(k-1)$ jetons, un automate d'arbres bottom-up qui calcule toutes ses boucles, nous obtenons à partir d'un automate faible à k jetons un nouveau modèle d'automate d'arbres qui combine le comportement d'un automate à jetons du modèle faible et celui d'un automate d'arbres bottom-up. Informellement un automate d'arbres faible bottom-up à k jetons est un automate d'arbres à k jetons du modèle faible qui peut simuler un automate bottom-up quand il place son dernier jeton sur le nœud courant.

Définition 7.8 *Un automate d'arbres faible bottom-up à k jetons est un couple (A, B) tels que*

- *B est un automate d'arbres bottom-up sur un alphabet de la forme $\Sigma \times 2^{\{1,\cdots,k\}}$,*
- *A est un* wPTA$_k$ *sauf que les transitions qui lèvent et posent le jeton 1 sont de la forme $(Q_A \times \Sigma \times \mathrm{TYPE} \times \{1\} \times 2^{[k]} \times Q_B) \times Q_A$ où Q_A et Q_B sont les ensembles d'états respectifs de A et B.*

La taille de (A, B) est la somme des tailles de A et de B c'est à dire la somme de leurs nombres d'états.

Un automate d'arbres faible bottom-up à k jetons se comporte comme un automate d'arbres faible à k jetons jusqu'à ce qu'il choisisse de poser le jeton 1 sur un nœud u.

L'automate (A, B) pose alors le jeton 1 en u, simule l'automate bottom-up B sur l'arbre marqué par le jeton 1 puis lève le jeton 1 et reprend son comportement cheminant à partir du nœud u et d'un état qui dépend de l'état atteint par B à la racine de l'arbre marqué lors de sa simulation.

Plus formellement, pour $i \in [1, k]$, une i-configuration de (A, B) sur un arbre t est un triplet (q, u, f) où q est un état de A, u est un nœud de l'arbre t et f est une affectation des jetons de $\{i+1, \cdots, k\}$. Soient maintenant $c = (q, u, f)$ et $c' = (q', u', f')$ deux configurations de (A, B) où u un nœud de t de type θ et d'étiquette a. La configuration c' est un successeur de la configuration c, ce que l'on note $c \xrightarrow{(A,B),t} c'$, dans les deux cas suivants :

1. $c \xrightarrow{A,t} c'$ et l'automate ne pose pas le jeton 1 durant cette étape,
2. $u' = u$, $f' = f$ et $((q, a, \theta, 2, f^{-1}(u), q_b), q')$ est une transition de A où q_b est l'état atteint par B à la racine sur l'arbre marqué $(t, f \cup \{(1, u)\})$.

Dans le premier cas, (A, B) se comporte ainsi comme un automate d'arbres à k jetons du modèle faible en effectuant la transition de c à c'. Dans le second cas, (A, B) pose le jeton 1, effectue une évaluation de B sur l'arbre marqué par les k jetons, puis retourne sur le nœud marqué par le jeton 1 et lève ce jeton. Remarquons que, pour une configuration d'une exécution de (A, B), le jeton 1 n'est jamais posé. Nous notons $\xrightarrow{(A,B),t\ *}$ la cloture transitive de la relation $\xrightarrow{(A,B),t}$ et nous remarquons que les définitions de k boucles d'arbre et de contexte s'étendent sans difficultés au modèle des automates d'arbres faibles bottom-up à jetons.

Pour traiter le modèle fort par induction, nous définissons une version forte du modèle des automates faibles bottom-up à k jetons. Nous introduisons d'abord un nouveau type d'automate bottom-up pouvant sélectionner un nœud avec un état spécial au cours d'un calcul.

Définition 7.9 *Un automate bottom-up sélectionneur est un quintuplet $(Q, q_0, Q_f, Q', \delta)$ où $Q' \subseteq Q$ et (Q, q_0, Q_f, δ) est un automate d'arbres bottom-up vérifiant la propriété suivante : étant donné un arbre t, pour chaque calcul ρ acceptant sur t, il existe un unique nœud u dans l'arbre t avec $\rho(u) \in Q'$. On dit alors que le nœud u est sélectionné par B dans le calcul ρ.*

Ces automates permettent de sélectionner dans un arbre t un nœud u_ℓ et de calculer les couples d'états correspondant à des exécutions d'un automate fort à k jetons à partir de la configuration où le jeton k vient d'être posé jusqu'à celle où le jeton k est sur le point d'être levé à partir du nœud u_ℓ.

Nous pouvons maintenant définir un nouveau modèle d'automates d'arbres qui combine le comportement d'un automate à jetons du modèle fort et celui d'un automate bottom-up sélectionneur. Intuitivement, un automate d'arbres fort bottom-up sélectionneur à k jetons se comporte comme un automate d'arbres fort à k jetons jusqu'à ce qu'il choisisse de poser le jeton 1. L'automate d'arbres fort bottom-up sélectionneur à k jetons simule alors l'automate bottom-up sélectionneur et reprend son comportement cheminant à partir du nœud sélectionné par l'automate bottom-up sélectionneur.

Définition 7.10 *Un automate d'arbres fort bottom-up sélectionneur à k jetons est un couple (A, B) tels que*

- *B est un automate bottom-up sélectionneur sur un alphabet de la forme $\Sigma \times 2^{\{1,\cdots,k\}}$*
- *A est un PTA_k sauf que les transitions qui lèvent et posent le jeton 1 sont remplacées par des transitions de la forme $(Q_A \times \Sigma \times \mathrm{TYPE} \times \{1\} \times 2^{[k]} \times Q_B) \times Q_A$ où Q_A et Q_B sont les ensembles d'états respectifs de A et B.*

Formellement, pour $i \in [1, k]$, une i-configuration de (A, B) sur un arbre t est un triplet (q, u, f) où q est un état de A, u un nœud de t et f une affectation des jetons de $\{i+1, \cdots, k\}$. Soit u un nœud de t de type θ et d'étiquette a et soient $c = (q, u, f)$ et $c' = (q', u', f')$ deux configurations de (A, B). La configuration c' est un successeur de la configuration c ce que l'on note $c \xrightarrow{(A,B),t} c'$ dans les deux cas suivants :

- $c \xrightarrow{A} c'$ et A ne pose pas le jeton 1 en effectuant la transition de c à c'.
- $f' = f$, x' est le nœud sélectionné par B sur l'arbre marqué $(t, f \cup \{(1, x)\})$ et $((q, a, \theta, 2, f^{-1}(x), q_b), q')$ est une transition de (A, B) où q_b est l'état atteint par B à la racine sur l'arbre marqué $(t, f \cup \{(1, x)\})$.

Dans le premier cas, (A, B) se comporte ainsi comme un automate d'arbres à k jetons du modèle fort en effectuant la transition de c à c'. Dans le second cas, (A, B) pose le jeton 1, effectue une évaluation de B sur l'arbre marqué par les k jetons, puis retourne sur le nœud sélectionné par B et lève le jeton 1 à distance. Remarquons que, pour une configuration d'une exécution de (A, B), le jeton 1 n'est jamais posé.

On note $\xrightarrow{(A,B),t\ *}$ la clôture transitive de la relation $\xrightarrow{(A,B),t}$.

Pour étendre la construction de [14] aux automates à jetons du modèle fort par induction sur le nombre de jetons, nous allons simuler avec un automate d'arbre bottom-up sélectionneur les exécutions d'un automate d'arbres à k jetons à partir du moment où il pose le jeton k jusqu'à ce qu'il le lève. L'automate bottom-up sélectionneur que nous construisons effectue son calcul sur un arbre marqué par des jetons considéré comme un arbre sur un alphabet de la forme $\Sigma \times 2^{[1,k]}$ et il doit simuler les exécutions de l'automate d'arbres à jetons à partir d'un nœud distingué par le jeton k jusqu'au nœud qu'il sélectionne. Nous définissons ci-dessous ce qu'est un *nœud distingué*.

Définition 7.11 *Soit (t, f) un arbre J-marqué où J est un ensemble de jetons et soit u un nœud de cet arbre. Le nœud u est un nœud distingué si u est la racine de t ou si un jeton $i \in J$ est posé sur u, c'est-à-dire si $f^{-1}(u) \neq \emptyset$. L'arbre (t, f) est alors marqué en u. On appelle ensemble de distinctions de l'arbre (t, f) l'ensemble $J \cup \{\mathrm{ra}\}$. Si μ est une distinction de l'arbre marqué (t, f), on dit que (t, f) est marqué par μ. Dans le premier cas, u est le* ra*-nœud de t ou de (t, f) et dans le second cas, u est le i-nœud de (t, f).*

Remarque 7.2 On a vu dans la définition 3.6 qu'un arbre J-marqué sur un alphabet Σ pouvait être considéré comme un arbre sur l'alphabet $\Sigma \times 2^J$ tel que l'étiquette d'un nœud indique l'ensemble des jetons qui se trouvent sur ce nœud. Un nœud distingué est alors la racine ou un nœud dont l'étiquette sur $\Sigma \times 2^J$ est de la forme (a, J') avec $J' \neq \emptyset$.

Nous pouvons maintenant définir comment l'automate bottom-up sélectionneur que nous construisons par la suite doit simuler l'automate d'arbres à jetons considéré.

Définition 7.12 *Soient (A, B) un automate d'arbres fort bottom-up sélectionneur à k jetons, respectivement un automate d'arbres faible bottom-up à k jetons, C un automate d'arbres bottom-up sélectionneur et μ l'élément* ra *ou bien un entier strictement supérieur ou égal à k. L'automate C simule (A, B) de manière exacte à partir de μ sur les arbres marqués par μ s'il vérifie les deux conditions suivantes :*

1. *les ensembles de couples d'états de A sont les états acceptants de C*
2. *pour tout arbre marqué (t, f), si le nœud u est le nœud μ-marqué de (t, f) on a l'équivalence suivante :*

   C sélectionne un nœud v et atteint un état acceptant q_f à la racine si et seulement si $q_f = \{(q, q') \mid (q, u) \xrightarrow{(A,B),t\ *} (q', v), q'$ est acceptant$\}$.

Remarque 7.3 Dans cette définition, l'automate C calcule des couples d'états correspondant à des exécutions de (A, B) se terminant dans un état acceptant mais pas nécessairement à la racine. On s'intéressera en effet aux exécutions entre le dépot et la levée d'un jeton et on parlera donc d'exécutions terminales et non d'exécutions acceptantes. Ainsi l'état atteint par C à la racine contient exactement les couples constitués des premiers et des derniers états de tous les exécutions terminales de C entre les nœuds u et v.

7.3.2 Simuler de manière exacte un automate fort à jetons avec un automate d'arbres bottom-up

Nous prouvons dans cette partie la borne supérieure de la complexité du vide et de l'inclusion pour les automates d'arbres du modèle fort à jetons.

Le lemme 7.2 ci-dessous étend la construction de [51] ou de [14] que nous avons décrite dans la preuve du lemme 7.1 : il permet de construire à partir d'un automate d'arbres faible bottom-up à 1 jeton (A, B) un automate d'arbres bottom-up sélectionneur qui calcule les 1-exécutions de (A, B). Pour calculer les boucles de contexte, nous utilisons le non-déterminisme de l'automate bottom-up sélectionneur et il nous faut également gérer le comportement bottom-up de l'automate (A, B). Nous introduisons d'abord l'*évaluation* et la *relation d'évaluation* qui donne le comportement d'un automate bottom-up sur un arbre et sur un contexte respectivement.

Définition 7.13 *Soient B un automate bottom-up sur un alphabet Σ, t un arbre et C un contexte sur un alphabet Σ. On note Q_B l'ensemble des états de B.*

L'évaluation de B sur t est l'état atteint par B à la racine de t et la relation d'évaluation de B sur un contexte C est la relation de $Q_B \times Q_B$ telle qu'un couple d'états (q, q') est dans cette relation si et seulement si B atteint l'état q' à la racine de C dans un calcul qui démarre en affectant l'état q au port de C.

On définit des opérations de composition sur les évaluations et les relations d'évaluation de manière similaire à la composition des comportements sur les arbres et les contextes définie dans la notation 5.4. Par exemple, si γ est la relation d'évaluation

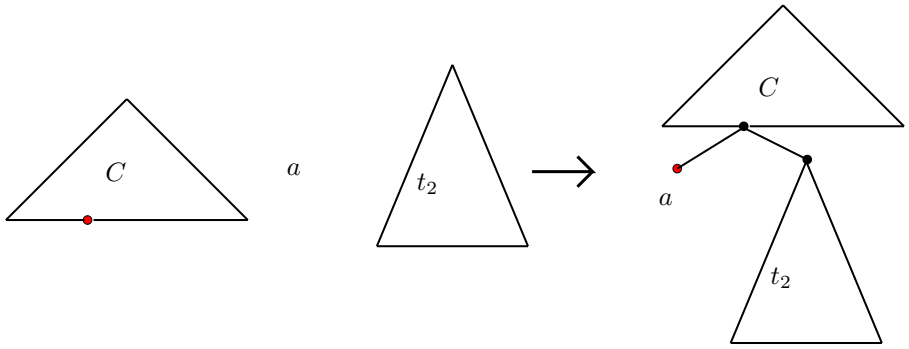

FIG. 7.1 – Rappel : représentation de la composition $\mathfrak{C}(C, a, R, t_2)$

d'un contexte C et si τ est l'évaluation d'un arbre t_2 alors $\mathfrak{C}(\gamma, a, \tau)$ est la relation d'évaluation du contexte $\mathfrak{C}(C, a, \emptyset, t_2)$ représenté figure 7.1.

Lemme 7.2 *Soit (A, B) un automate d'arbres faible bottom-up à 1 jeton sur un alphabet marqué de la forme $\Sigma \times 2^J$ où $1 \notin J$ et μ un élément de $\{ra\} \cup J$. On peut construire en temps exponentiel en la taille de (A, B) un automate bottom-up sélectionneur C tel que C simule de manière exacte (A, B) à partir de μ sur les arbres marqués par μ.*

Preuve. On note Q_A l'ensemble des états de A, Q_{f_A} l'ensembles des états acceptants de A et Q_B l'ensemble des états de B. De plus, dans cette preuve, étant donnés deux nœuds u_1 et u_2, la *position de u_1 relativement à u_2* est une information finie qui indique si u_1 est un descendant de u_2, le nœud u_2 lui même ou bien un autre nœud. Soit t un arbre marqué par μ et u le μ-nœud de t. L'automate C construit dans cette preuve choisit un nœud v et simule toutes les exécutions de A de u à v. Pour cela, C stocke dans l'état affecté à un nœud x de t une information finie sur les positions de u et v relativement à x ainsi que toutes les boucles de (A, B) dans le sous-arbre enraciné en x. De plus, l'automate C devine toutes les boucles de (A, B) dans le contexte pointé en x et vérifie les informations qu'il a devinées en remontant dans l'arbre jusqu'à la racine. A partir de toutes ces informations qu'il a calculées et devinées, C peut déterminer à la fin d'une exécution acceptante les couples d'états correspondant à des exécutions de (A, B) de u à v. En effet, une fois qu'il a atteint la racine à la fin de son exécution, C a vérifié qu'il n'a pas fait d'erreur dans ses choix pour les boucles de contexte et, dans ce cas, il a bien calculé exactement toutes les exécutions de (A, B) de u à v. Si C détecte une erreur, il ne passe pas dans un état acceptant.

Plus formellement, sur un nœud x, l'état C contient les informations suivantes :
- la position de u relativement à x,
- l'évaluation $q(x)$ de B sur $t_{|x}$,
- l'évaluation $q'(x)$ de B sur le sous-arbre marqué que l'on note t_x^1 et qui est obtenu à partir de $t_{|x}$ lorsque le jeton de A est posé sur le nœud x,
- l'ensemble $R(x) \subseteq Q_{\mathcal{A}} \times Q_A$ des boucles d'arbre de (A, B) dans t_x.

De plus, l'automate $\mathcal{C}$ devine et retient dans son état sur le nœud x les informations suivantes :
- le type de x et la position de v relativement à x,

– la relation d'évaluation $c(x) \subseteq Q_B \times Q_B$ de B sur le contexte $C_{t,x}$,
– l'ensemble $S(x) \subseteq Q_A \times Q_A$ des boucles de contexte de (A, B) dans $C_{t,x}$.

Toutes ces informations devinées par C sont vérifiées lorsque C remonte dans l'arbre. A la racine, C a choisi une position pour v et sait si tous les choix qu'il a fait en devinant des informations sont corrects.

A l'aide des relations $R(x)$ et $S(x)$, C calcule et retient dans son état la relation $T(x) \subseteq Q_A \times Q_A$ telle que :

– si u et v ne sont pas des descendants de x, $T(x)$ est vide,
– si u est un descendant de x et v n'en est pas un, $T(x)$ est l'ensemble des 1-exécutions de (A, B) de u à x.
– si v est un descendant de x et si u n'en est pas un, $T(x)$ est l'ensemble des 1-exécutions de (A, B) de x à v.
– si u et v sont tout deux des descendants de x, $T(x)$ est l'ensemble des 1-exécutions de u à v.

Ainsi à la fin du calcul de C, c'est à dire quand x est confondu avec la racine, la restriction de $T(x)$ à $Q_A \times Q_{f_A}$ donne le résultat souhaité.

L'automate C calcule toutes les informations ci-dessus en remontant des feuilles à la racine de la manière suivante. Le cas où x est une feuille est trivial. Considérons maintenant que x est un nœud interne d'étiquette a, on note x_1 son fils gauche d'étiquette a_1 et x_2 son fils droit d'étiquette a_2. Les étiquettes a_1 et a_2 sont aussi stockés dans l'état de C. Les informations pour x_1 et x_2 ayant été calculées et stockées dans les états affectés à ces nœuds, l'automate C calcule les informations pour x ainsi :

– $q(x)$ et $q'(x)$ sont calculés à l'aide de $q(x_1)$ et de $q(x_2)$ en simulant B,
– $c(x)$ est deviné et C vérifie que ce choix est cohérent avec la relation $c(x_1)$ et l'état $q(x_2)$ ainsi qu'avec $c(x_2)$ et $q(x_1)$. Par exemple, C vérifie que $c(x_1) = \mathfrak{C}(a_1, \emptyset, c(x), q(x_2))$
– $R(x)$ est calculé en combinant d'une part les 1-boucles d'arbre de $R(x_1)$ et les 1-boucles d'arbre de $R(x_2)$ et d'autre part une transition qui simule B et qui dépend donc de $q'(x)$ et de $c(x)$.
– $S(x)$ est deviné et C vérifie qu'il est cohérent avec $S(x_1)$, $S(x_2)$ et toutes les autres relations. Par exemple, C vérifie que $S(x_1)$ correspond à la combinaison des 1-boucles de contexte de $S(x)$, des 1-boucles d'arbres de $R(x_2)$ et enfin d'une transition qui simule B et qui dépend donc de $q'(x_1)$ et de $c(x_1)$.
– $T(x)$ est facile à actualiser à l'aide des informations ci-dessus une fois que les positions de u et de v relativement à x sont connues.

La taille de C est bien exponentielle en celle de (A, B) et C simule bien (A, B) de manière exacte à partir de μ. □

Remarque 7.4 La construction de la preuve du lemme ci-dessus est la même que la construction de [51] et de [14], c'est pourquoi nous obtenons la même borne supérieure pour la complexité des problèmes du vide pour les automates d'arbres cheminants et pour les automates d'arbres à 1 jeton.

Pour simuler de manière exacte un automate d'arbres fort bottom-up sélectionneur noté (A, B) avec un automate d'arbres bottom-up sélectionneur, nous construisons

un automate d'arbres faible bottom-up intermédiaire qui simule (A, B) et nous utilisons le lemme 7.2.

Définition 7.14 *Soient (A, B) un automate d'arbres fort bottom-up sélectionneur à k jetons et (A', B') un automate d'arbres faible bottom-up à k jetons. L'automate (A', B') simule (A, B) de manière exacte s'il vérifie les deux conditions suivantes :*

1. *l'ensemble des états de A est inclus dans celui de A'.*
2. *pour tout arbre t, pour tout couple (u, v) de nœuds de t et pour tout couple (q, q') d'états de A, on a l'équivalence suivante : $(q, u) \xrightarrow{(A,B),t\ *} (q', v)$ si et seulement si on a $(q, u) \xrightarrow{(A',B'),t\ *} (q', v)$.*

Lemme 7.3 *Soit (A, B) un automate d'arbres fort bottom-up sélectionneur à 1 jeton. On peut construire en temps polynomial (A', B') un automate d'arbres faible bottom-up à 1 jeton qui simule (A, B) de manière exacte.*

Preuve. L'automate A' simule A jusqu'à ce que A soit sur un nœud u_p et lance un calcul de B qui sélectionne un nœud u_ℓ. A partir de u_p, l'automate A' devine un chemin dans l'arbre de u_p à u_ℓ et se déplace pas à pas le long de ce chemin en simulant B' à chaque étape jusqu'à ce qu'il rejette ou qu'il atteigne le nœud u_ℓ. La difficulté supplémentaire par rapport à la transformation d'un automate du modèle fort à 1 jeton en un automate du modèle faible à 1 jeton décrite dans la partie 5.1.1 est de gérer le comportement bottom-up de B.

On montre maintenant comment (A', B') simule la transition de A qui lance un calcul de B. A partir de u_p, A' se déplace pas à pas le long du chemin de u_p à u_ℓ et simule B' à chaque étape. A chaque étape, si A' est sur un nœud u de ce chemin, il retient dans son état :

- l'état q_r atteint par B à la racine de l'arbre dans la simulation de B lancée par A à partir du nœud u_p,
- l'état q_u atteint par B sur le nœud courant u toujours dans la simulation de B lancée à partir du nœud u_p et
- un élément de l'ensemble $\{\textsc{Init}, \swarrow, \searrow, \nwarrow, \nearrow, \circlearrowleft\}$ qui est $\textsc{Init}$ au départ lorsque A' est en u_p et qui est ensuite la direction de u à son successeur dans le chemin de u_p à u_ℓ.

Ces informations sont calculées et actualisées en utilisant la partie bottom-up B' de (A', B').

L'automate B' procède aussi de la manière suivante. Il devine d'abord un état q_u^0 de Q_B et un élément Δ^0 de l'ensemble $\{\textsc{Init}, \swarrow, \searrow, \nwarrow, \nearrow, \circlearrowleft\}$, qui sont supposés être l'état q_u et l'élément Δ stockés dans l'état courant de A'. L'automate A' vérifie à chaque fois qu'il simule B' que B' ne s'est pas trompé dans le choix de cet état et cette direction. On distingue ensuite trois cas :

1. L'élément Δ^0 est $\textsc{Init}$. L'automate B' simule alors B et termine dans un état qui contient l'état q_u atteint par B sur le nœud u où est A', la direction $\Delta \in \{\swarrow, \searrow, \nwarrow, \nearrow, \circlearrowleft\}$ entre le nœud u et le nœud sélectionné u_ℓ par B et l'état q_r atteint par B à la racine. On remarque que Δ peut être $\circlearrowleft$ si le nœud sélectionné par B est u.

2. L'élément Δ^0 est $\nearrow$, le cas où $\Delta^0 = \nwarrow$ est similaire.
   L'automate B' simule alors B jusqu'à ce qu'il atteigne le nœud u distingué par le jeton posé par A. Quand ce nœud est atteint, si B a déjà sélectionné lors de cette simulation un nœud dans le sous-arbre gauche de u alors B' rejette. Dans le cas contraire, l'état courant est ignoré et B' recalcule l'état courant q_u en supposant qu'il a attribué l'état q_u^0 au fils gauche de u. Il retient alors dans son état le nouvel état q_u qu'il a recalculé et la direction Δ de u vers le nœud sélectionné, puis il reprend sa simulation de B. A la racine B' termine dans un état qui contient q_u^0, Δ^0 et l'état atteint par B à la racine dans la simulation courante avec q_u et Δ.
3. L'élément Δ^0 est $\searrow$, le cas où l'élement $\Delta^0 = \swarrow$ est similaire.
   Dans ce cas B' simule B jusqu'à ce qu'il atteigne u. Quand ce nœud est atteint, B' sait si un nœud a déjà été sélectionné dans le sous-arbre droit de u. Si ce n'est pas le cas ou si l'état courant n'est pas q_u^0, B' rejette. Sinon, B' retient l'état q_u que B a atteint au fils droit $u2$ de u et la direction Δ de u_2 à u_ℓ. A la racine, B' accepte dans un état contenant q_u^0 et Δ^0 ainsi que l'état q_r atteint par B à la racine durant la simulation courante avec q_u et Δ.

Nous pouvons maintenant définir A'. L'automate A' simule A jusqu'à ce que A lance un calcul de B à partir d'un nœud u_p. A partir de ce nœud, A' lance un calcul de B', vérifie que B' a bien deviné l'élément INIT qui correspond au premier cas et stocke dans son état les nouvelles valeurs q_u et Δ calculées par B' à la racine dans son état. L'automate A' procède alors de la manière suivante jusqu'à ce que le cas $\circlearrowleft$ soit atteint. Il se déplace d'un pas dans la direction donnée par Δ, simule B', vérifie que les informations q_u^0 et Δ^0 devinées par B' correspondent bien à celles qu'il a retenues dans son état courant. Si ce n'est pas le cas, il rejette et si c'est bien le cas, il actualise ces valeurs avec les nouvelles valeurs calculées par B'. Quand la valeur de Δ calculée par B' est $\circlearrowleft$, l'automate A' reprend sa simulation de A en utilisant l'état q_r. □

Nous montrons maintenant comment simuler de manière exacte un automate d'arbres fort bottom-up sélectionneur à k jetons avec un un automate bottom-up sélectionneur lorsque $k > 0$.

Lemme 7.4 *Soient (A, B) un automate d'arbres fort bottom-up sélectionneur à k jetons et μ une distinction. Nous pouvons construire en temps k-exponentiel en la taille de (A, B) un automate bottom-up sélectionneur C qui simule (A, B) de manière exacte.*

Preuve. On procède par induction sur k. Le cas initial $k = 1$ est une conséquence directe des lemmes 7.3 et 7.2. On suppose maintenant que le lemme est prouvé pour un entier k et on veut le prouver pour l'entier $(k + 1)$. On construit (A', B) un automate fort bottom-up sélectionneur à un jeton tel que A' est défini à partir de A de la manière suivante :

- les états de A' sont les états de A,
- les transitions de A' sont les transitions de A correspondant à des configurations où tous les jetons de $[2, k]$ sont posés,
- les états acceptants de A' sont les états à partir duquel le jeton 2 peut être levé.

Les automates forts bottom-up sélectionneurs (A', B) et (A, B) ont exactement les mêmes exécutions à partir d'une configuration où le jeton 2 de A est posé sur le nœud courant u_p jusqu'à une configuration où ce jeton est sur le point d'être levé par (A, B) à partir d'un nœud u_ℓ. En utilisant le lemme 7.3 on construit en temps polynomial (A'', B') un automate faible bottom-up à 1 jeton qui simule de manière exacte (A', B) sur un arbre $[2, k]$-marqué. On utilise maintenant le lemme 7.2 et on construit en temps exponentiel un automate bottom-up sélectionneur C' qui simule (A'', B') de manière exacte à partir du jeton 2 sur des arbres $[2, k]$-marqués. On considère maintenant l'automate fort bottom-up sélectionneur à $(k-1)$ jetons (A''', C') défini de la manière suivante :

- les états de A''' sont les états de A ,
- les transitions de A''' sont les transitions de A correspondant à des configurations où le jeton 2 n'est pas posé et où toutes les transitions qui posent le jeton 2 sont remplacées par une simulation de C',
- les états acceptants de A''' sont les états acceptants de A.

Il est facile de vérifier que (A''', C') simule (A, B) de manière exacte à partir de μ et on obtient ainsi l'automate C en appliquant l'hypothèse d'induction à l'automate (A''', C'). □

Proposition 7.5 *Soit A un automate d'arbres du modèle fort à k jetons. On peut construire en temps polynomial en la taille de A un automate d'arbres du modèle fort à k jetons A' qui accepte le même langage que A et tel que le jeton 1 de A' est faible, c'est-à-dire tel que la fonction de transition de A' ne lui permet pas de lever le jeton 1 à distance.*

Preuve. La construction de A' à partir de A est similaire à la transformation d'un automate du modèle fort à 1 jeton en un automate du modèle faible à 1 jeton qui est décrite en 5.1.1 et qui est polynomiale. □

Nous étendons la construction de [51] aux automates d'arbres à jetons du modèle fort en prouvant le théorème suivant.

Théorème 7.4 *Soit A un automate d'arbres du modèle fort à k jetons. On peut construire en temps k-exponentiel en la taille de A un automate d'arbres bottom-up C qui accepte le même langage que A et un automate d'arbres bottom-up C' qui accepte le complément du langage accepté par A.*

Preuve.

Soit A un automate fort à k jetons. On note A' l'automate fort équivalent à k jetons tel que le jeton 1 est faible obtenu en temps polynomial à l'aide de la proposition 7.5. On définit maintenant (A'', B) un automate fort bottom-up sélectionneur à 1 jeton. L'automate A'' est obtenu à partir de A' de la manière suivante :

- les états de A'' sont les états de A''
- les transitions de A'' sont les transitions de A' correspondant à des configurations où les jetons de $[2, k]$ sont posés,
- les états acceptants de A'' sont les états à partir desquels le jeton 2 peut être levé.

Soit B l'automate bottom-up sélectionneur trivial qui se contente de sélectionner le nœud où le jeton 1 est posé c'est-à-dire le nœud à partir duquel A lance sa

simulation. L'automate (A'', B) a les mêmes exécutions que A' d'une configuration où le jeton 2 de A' vient d'être posé jusqu'à une configuration où ce jeton est sur le point d'être levé à partir d'un nœud u_ℓ. En utilisant le lemme 7.2, on construit en temps exponentiel un automate bottom-up sélectionneur C' qui simule (A'', B) de manière exacte à partir du jeton 2. On considère maintenant (A''', C') l'automate d'arbres fort bottom-up sélectionneur à $(k-1)$ tel que A''' est défini à partir de A de la manière suivante :

- les états de A''' sont les états de A,
- les états acceptants de A''' sont les états acceptants de A et
- les transitions de A''' sont les transitions de A correspondant aux configurations où le jeton 2 n'est pas posé telles que les transitions posant le jeton 2 sont remplacées par des transitions effectuant une simulation de C'.

Il est alors facile de vérifier que (A''', C') simule A de manière exacte. On considère maintenant C l'automate bottom-up sélectionneur construit en utilisant le lemme 7.3 qui simule (A''', C') à partir de la racine et qui sélectionne toujours la racine. Par construction de C, l'état de C atteint à la racine contient exactement tous les couples d'états (q, q') tels que si A démarre à la racine dans une configuration sans jeton dont l'état est q, il retourne à la racine dans une configuration sans jeton dont l'état est q'. Il est maintenant immédiat de définir C_1 and C_2 à partir de C en choisissant de manière appropriée l'ensemble des états acceptants. □

Remarque 7.5 Ce théorème permet de construire à partir d'un automate à jetons du modèle fort un automate bottom-up qui reconnaît le même langage et cette construction est optimale. Nous avons donc une nouvelle preuve de la proposition 2.6 qui établit que les langages reconnus par un automate d'arbres à jetons sont réguliers.

Comme le problème du vide est polynomiale pour les automates d'arbres bottom-up et qu'étant donnés deux automates d'arbres bottom-up, nous pouvons construire en temps polynomial un automate d'arbres bottom-up qui reconnaît l'intersection des langages reconnus par les deux automates donnés, nous déduisons du théorème 7.4 le théorème suivant.

Théorème 7.5 *Soit $k > 0$. Les problèmes du vide et de l'inclusion pour les automates d'arbres à jetons du modèle fort sont dans k-*ExpTime*.*

7.3.3 Borne supérieure pour les automates à jetons sur les mots

Si nous adaptons les preuves de la partie précédente au cas des mots nous obtenons le théorème suivant donnant la borne supérieure de la complexité des problème du vide et de l'inclusion pour les automates à jetons sur les mots.

Théorème 7.6 *Soit $k > 0$. Les problèmes du vide et de l'inclusion pour les automates à jetons du modèle fort sur les mots sont dans $(k-1)$-*ExpSpace*.*

Preuve. Supposons d'abord que $k = 1$. On considère donc A un automate du modèle fort à 1 jeton sur les mots. On peut supposer que A est du modèle faible car la transformation d'un automate du modèle fort à 1 jeton en un automate du modèle

faible à 1 jeton est polynomiale. On considère C l'automate unidirectionnel qui calcule toutes les boucles de A et qui est équivalent à A qu'on pourrait contruire comme dans le lemme 7.2. Au lieu de construire explicitement cet automate et de vérifier l'existence d'un chemin de l'état initial à l'état acceptant, on simule cet automate à la volée en mémorisant l'état courant de C et en calculant un de ces successeurs à la volée. Comme chaque état de C a une taille polynomiale, on peut ainsi construire une machine de Turing non déterministe utilisant un espace polynomial qui résout ce problème. Ainsi, d'après le théorème de Savitch énoncé dans la proposition 7.1, le problème du vide pour les automates à 1 jeton sur les mots est PSPACE. Comme on peut construire en temps polynomial un automate unidirectionnel qui reconnaît l'intersection des langages reconnus par deux automates unidirectionnels donnés et que l'automate unidirectionnel C qu'on construit peut facilement reconnaître le complément du langage reconnu par A, le problème de l'inclusion est aussi dans PSPACE pour les automates à 1 jetons sur les mots.

On suppose maintenant $k > 1$ et on considère un automate du modèle fort à k jetons sur les mots. On construit en temps $(k-1)$ exponentiel comme dans la preuve du théorème 7.4 un automate C équivalent à A qui correspond à un automate faible bottom-up à 1 jeton sur les mots. L'automate C est en fait un automate à 1 jeton sur les mots car sur les mots il n'y a pas de différence entre un comportement cheminant et un comportement bottom-up. Comme les problèmes du vide et de l'inclusion sont dans PSPACE pour C, on peut conclure.

□

7.4 Borne inférieure

Nous prouvons dans cette section la borne inférieure de la complexité des problèmes du vide et de l'inclusion pour tous les modèles d'automates d'arbres à k jetons : nous montrons que tous ces problèmes sont k-EXPTIME-durs dans le cas des arbres et $(k-1)$-EXPSPACE-durs dans le cas des mots. Pour le cas des arbres par exemple, il nous suffit de prouver la borne inférieure pour le problème du vide pour les automates d'arbres à k jetons déterministes du modèle faible puisque tous les autres problèmes sont plus difficiles que celui-là. L'idée de la preuve dans le cas des arbres est d'utiliser les propositions 7.2 et 7.3. Il nous suffit alors de simuler une machine de Turing alternante $\mathcal{M}$ utilisant un espace $\text{tow}(k-1, n)$ sur une entrée de taille n en construisant un automate déterministe du modèle faible à k jetons ayant un nombre d'états polynomial en n et qui reconnaît un langage non-vide si et seulement si $\mathcal{M}$ accepte son entrée.

Nous définissons tout d'abord un codage des configurations d'une machine de Turing utilisant un espace $(k-1)$-exponentiel. Nous montrons ensuite comment utiliser les jetons d'un automate d'arbres pour comparer les codages de deux configurations et nous construisons enfin l'automate déterministe à k jetons du modèle faible de taille polynomiale en n qui accepte uniquement l'arbre correspondant à l'exécution de $\mathcal{M}$ sur une entrée de taille n si cette exécution est acceptante.

Les machines de Turing que nous considérons dans cette section travaillent sur un alphabet binaire.

7.4.1 Codage d'une configuration par un k-nombre

Nous définissons tout d'abord de manière inductive les k-nombres que nous utiliserons pour coder les configurations d'une machine de Turing.

Définition 7.15 *Soit $n > 0$. Un k-nombre de taille n est défini récursivement de la manière suivante.*

Un 1-nombre de taille n est un arbre sur l'alphabet $\{0, 1, \#\}$ vérifiant les propriétés suivantes :

- *la racine est étiquetée par le symbole $\#$,*
- *le chemin le plus à droite, c'est à dire le chemin de la racine à la feuille la plus à droite est constitué de $(n+1)$ nœuds en comptant la racine et*
- *chacun des n nœuds du chemin le plus à droite différent de la racine est étiqueté par un bit, c'est-à-dire un élément de $\{0, 1\}$.*

Un 1-nombre définit ainsi un entier de $[0, 2^n - 1]$.

Si $k > 1$, un k-nombre de taille n est un arbre tel que

- *la racine est étiquetée par le symbole $\#$,*
- *le chemin le plus à droite est constitué de $(\mathrm{tow}(k-1, n)+1)$ nœuds en comptant la racine*
- *chacun des $\mathrm{tow}(k-1, n)$ nœuds du chemin le plus à droite différents de la racine u est étiqueté par un bit et a pour sous-arbre gauche un $k-1$-nombre qui code la distance entre le nœud u et le fils droit de la racine.*

L'entier défini par un k-nombre de taille n est l'entier dont l'écriture binaire est donnée par la suite des bits étiquetant les nœuds du chemin le plus à droite et lus de la racine à la feuille la plus à droite.

La figure 7.2 représente un 1-nombre de taille n et la figure 7.3 représente un k-nombre de taille n.

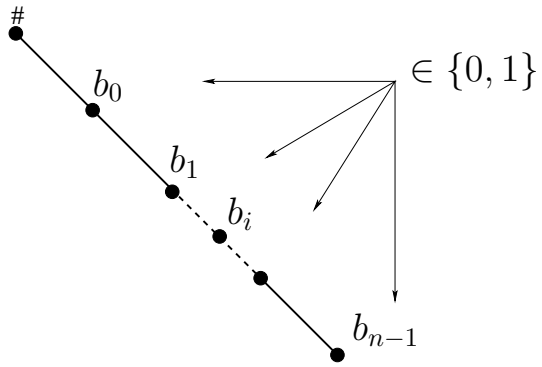

FIG. 7.2 – Représentation d'un 1-nombre de taille n

Remarque 7.6 Un 1-nombre est en fait un mot qu'on a transformé en arbre en rajoutant des nœuds inutiles que nous n'avons pas représentés dans la figure 7.2.

Notation 7.2 *Si x est un k-nombre, les nœuds du chemin le plus à droite de x distincts de la racine sont appelés les bits de x. Pour chacun de ces bits u, le $(k-1)$-nombre qui est le sous-arbre gauche de u est appelé la position de u.*

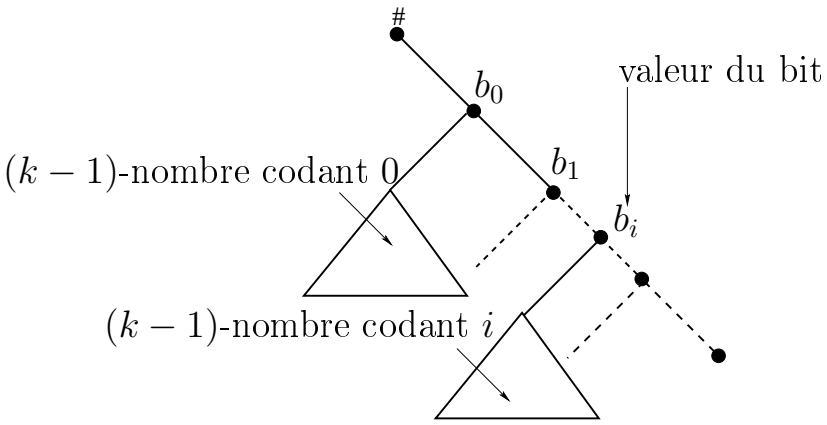

FIG. 7.3 – Représentation d'un k-nombre de taille n

Une configuration d'une machine de Turing $\mathcal{M}$ utilisant un espace $(k-1)$-exponentiel est codée de la manière suivante : chaque bit du k-nombre code le bit de même position sur la bande de la machine. La position de la tête de la machine et l'état courant de $\mathcal{M}$ sont codés en étendant la notion de k-nombre : on rajoute l'état à l'étiquette du bit de la position de la tête de lecture/écriture. Un k-nombre est ainsi un arbre sur l'alphabet $\{\sharp, 0, 1\} \cup (\{0,1\} \times Q_{\mathcal{M}})$ où $Q_{\mathcal{M}}$ est l'ensemble des états de $\mathcal{M}$.

Définition 7.16 *Soit f une fonction faisant correspondre à chaque nœud u d'un arbre t un ensemble des nœud de t. Une telle fopnction f est dite déterminée s'il existe A_f un automate cheminant déterministe avec un état spécifique q_S vérifiant la propriété suivante : pour arbre t et pour tout couple (u,v) de nœuds de t, on a $v \in f(u)$ si et seulement si $(q_I, u) \xrightarrow{A,t\ *} (q_S, v)$ où q_I est l'état initial de A.*

Remarque 7.7 Si f est fonctions déterminée, l'automate A_f passe successivement par tous les nœuds de $f(u)$ à partir de la configuration (q_I, u).

Les fonctions déterminées que nous utiliserons sont les ensembles de bits d'un k-nombre situés juste en dessous ou bien juste au-dessus d'un nœud u.

Nous utilisons ainsi des fonctions déterminées très simples telle que les tailles des automates cheminants qui les définissent ne dépendent pas des paramètres k et n.

7.4.2 Comparer deux k-nombres avec des jetons

Le lemme technique qui suit et que nous utilisons dans la partie suivante pour simuler une machine de Turing alternante utilisant un espace $(k-1)$-exponentiel avec un automate déterministe à jetons s'inspire du résultat de [24] montrant qu'un automate déterministe à $k+1$ jetons sur les mots est exponentiellement plus succint qu'un automate déterministe à k jetons sur les mots.

Lemme 7.5 *Soient $n > 0$ et $k > 0$.*

1. *Il existe un automate déterministe du modèle faible à $(k-1)$ jetons de taille polynomiale en n qui, à partir d'un nœud u distingué d'un arbre t et de son*

état initial, retourne en u dans un état qui est acceptant si et seulement si le sous-arbre enraciné en u forme un k-nombre de taille n.

2. *Pour chaque fonction déterminée f, il existe un automate déterministe du modèle faible à k jetons de taille polynomiale en n qui, à partir d'un nœud u d'un arbre t et de son état initial, retourne en u dans un état acceptant si et seulement s'il existe un nœud $v \in f(u)$ tels que u et v sont les racines du même k-nombre de taille n .*

3. *Il existe un automate déterministe du modèle faible à k jetons de taille polynomiale en n qui à partir d'un nœud u (et de son état initial) retourne en u dans un état acceptant si et seulement si le k-nombre de taille n enraciné en $u1$ le fils gauche de u est le successeur du k-nombre enraciné en $u21$ le fils gauche du fils droit de u.*

Preuve. On fixe $n > 0$. Les trois points sont prouvés ensemble par induction sur k.

On note A_f l'automate cheminant qui permet de définir f dans le point 2.

Si $k = 1$, le point 1 est facile à prouver. L'automate utilise un nombre linéaire d'états pour vérifier que l'arbre a bien $(n+1)$ nœuds sur le chemin le plus à droite, que le premier nœud est étiqueté par le symbole $\sharp$, que tous les autres nœuds sont étiquetés par un bit sauf un qui contient également un état dans son étiquette, l'automate retrourne ensuite en u et accepte si l'arbre a bien la forme souhaitée. Pour le point 2, l'automate pose successivement le jeton sur chaque nœud v de l'ensemble de nœud $f(u)$ qu'il identifie en simulant A_f puis il simule l'automate construit pour le point 1 et vérifie ainsi que les sous-arbres enracinés en u et v sont bien des 1-nombres. Ensuite l'automate compare bit par bit les sous-arbres enracinés en u et en v en faisant des aller-retour entre les nœuds u et v qui sont distingués. On rappelle que le nœud u est distingué par hypothèse et que le nœud v est marqué par le jeton. La position courante du bit que l'automate compare est mémorisée dans l'état et il suffit ainsi d'avoir un nombre linéaire d'états. Pour le point 3, le jeton est posé sur le nœud $u21$ et l'automate procède comme pour le point 2 en simulant l'incrémentation de 1 au lieu de vérifier l'égalité.

On suppose maintenant que $k > 1$. On considère d'abord le point 1. Par induction, il est facile de vérifier avec un automate déterministe du modèle faible à $(k-2)$ jetons que le sous-arbre enraciné en u à la bonne forme, c'est à dire que sa racine est étiquetée par le symbole $\sharp$ et qu'elle est suivie d'une suite de $(k-1)$-nombres. Il reste ensuite à vérifier que cette suite correspond à l'ensemble des $(k-1)$-nombres dans l'ordre croissant (de 0 à $\mathrm{tow}(k-1,n)-1$). Pour chaque nœud v du chemin le plus à droite du sous-arbre enraciné en u, l'automate pose le jeton k sur v et simule l'automate à $(k-1)$ jetons obtenu avec le point 3 par induction afin de vérifier que les positions de v et du fils droit de v sont des $(k-1)$-nombres successifs. Une fois que cette vérification a été effectuée pour chaque bit v de u, l'automate retourne en u en remontant dans l'arbre jusqu'à ce qu'il visite un nœud étiqueté par le symbole $\sharp$.

On démontre maintenant le point 2. L'automate vérifie d'abord par induction que u est la racine d'un k-nombre. Ensuite, pour chaque nœud $v \in f(u)$ il procède de la manière suivante. Il vérifie d'abord si v est la racine d'un k-nombre et si c'est le cas, il pose le jeton k successivement sur chaque bit z du k-nombre enraciné en v. Soit g la fonction faisant correspondre à un de ces bits z l'ensemble des bits du k-nombre

enraciné en u. La fonction g est déterminée par l'automate cheminant déterministe qui à partir de z retourne en u qui est distingué et visite ensuite successivement tous les bits du k-nombre enraciné en u en descendant à droite. Par induction, d'après le point 2, l'automate peut trouver avec ses $(k-1)$ jetons restants parmi les bits de u celui avec la même position que z et vérifie que les valeurs des bits correspondants sont égaux. Si ces valeurs sont différentes, l'automate retourne en z, lève le jeton k et retourne en u en remontant jusqu'à ce qu'il viste un nœud étiqueté par le symbole $\sharp$. Il procède alors de même pour le prochain nœud de $f(u)$. Si les valeurs des bits correspondent, l'automate retourne en z, lève le jeton k et recommence pour le bit suivant de v. Une fois que tous les bits de v ont ainsi été traités successivement, l'automate retourne en u et accepte.

Le point 3 est montré comme le point 2 avec les différences suivantes. L'ensemble des nœuds $f(u)$ est un singleton et le nœud u n'a plus besoin d'être marqué car il peut être retrouvé avec la position du jeton k. De plus, au lieu de vérifier l'égalité de deux $(k-1)$-nombres, l'automate simule l'incrémentation de 1. □

Ce lemme est toujours valide pour la légère extension de la définition de k-nombre que nous utilisons pour coder une configuration d'une machine de Turing.

7.4.3 Borne inférieure pour les automates à jetons

Théorème 7.7 *Soit $k \geq 1$. Le problème du vide (et donc le problème de l'inclusion) pour les automates déterministes du modèle faible à k jetons est k-*ExpTime*-dur.*

Preuve. Pour les automates cheminants déterministe et donc pour les automates déterministes à 1 jeton du modèle faible, la borne inférieure ExpTime-dur est implicite dans [38] mais elle peut être aussi obtenue en adaptant facilement la preuve ci-dessous.

Nous supposons donc maintenant que $k \geq 2$.

Pour prouver le théorème, nous simulons une machine de Turing alternante utilisant un espace $\mathrm{tow}(k-1,n)$ sur une entrée de taille n avec un automate déterministe du modèle faible à k jetons de taille polynomiale en n.

Plus précisément, étant donné une machine de Turing alternante $\mathcal{M}$ utilisant un espace $\mathrm{tow}(k-1,n)$ sur une entrée w de taille n, nous construisons $A_{\mathcal{M}}$ un automate déterministe du modèle faible à k jetons de taille polynomiale en n tel que les arbres acceptés par $A_{\mathcal{M}}$ sont les codages des exécutions acceptantes de $\mathcal{M}$. En effet, on a vu comment coder une configuration de $\mathcal{M}$ avec un k-nombre et une exécution de $\mathcal{M}$ est codée en concaténant les k-nombres codant les configurations d'une étapes donnée par une transition existentielle de l'exécution de $\mathcal{M}$ et en utilisant la structure de branchement d'un arbre pour coder les étapes de l'exécution donné par des transitions universelles.

Nous construisons maintenant $A_{\mathcal{M}}$ qui accepte exactement les exécutions acceptantes de $\mathcal{M}$ sur l'entrée w. Dans un premier temps $A_{\mathcal{M}}$ vérifie que l'arbre a la forme suivante :

1. il est construit en assemblant des k-nombres ayant chacun un unique nœud dont l'étiquette contient un état,
2. chaque configuration qui contient un état existentiel est suivi par une unique configuration,

3. chaque configuration qui contient un état universel est suivi de deux configurations,
4. les configurations sans successeur sont acceptantes et le k-nombre à la racine de l'arbre code la configuration initiale de la machine sur l'entrée w.

Le premier point peut être vérifié avec $(k-1)$ jetons comme il est montré dans le lemme 7.5. Les autres points sont immédiats à vérifier.

Il reste ensuite à vérifier que deux k-nombres successifs correspondent à des configurations successives pour la machine de Turing. L'automate procède alors de la manière suivante. $A_{\mathcal{M}}$ considère successivement chaque configuration en effectuant un parcours en profondeur de l'arbre. Pour chaque configuration il se comporte de la manière suivante. Il marque successivement chaque position de la bande en posant le jeton k sur le $(k-1)$-nombre correspondant. On considère f la fonction qui fait correspondre à x l'ensemble des positions de la configuration juste au-dessus de x. Cette fonction est déterminée par l'automate cheminant déterministe qui remonte dans l'arbre jusqu'à ce qu'il viste un nœud étiqueté par le symbole $\sharp$ puis qui passe dans l'état q_S et qui continue alors de remonter jusqu'à ce qu'il visite le prochain nœud étiqueté par le symbole $\sharp$. L'automate utilise alors ses $(k-1)$ jetons restants pour trouver la position correspondante dans la configuration précédente comme dans la partie 2 du lemme 7.5. Il vérifie ensuite que les bits marqués par les deux $(k-1)$-nombres sont égaux si aucun ne contient un état dans son étiquette et vérifie sinon que ces bits correspondent à une transition de $\mathcal{M}$. □

7.4.4 Borne inférieure pour les automates à jetons sur les mots

Pour adapter les preuves précédentes au cas des mots, il suffit d'utiliser la version des k-nombres de taille n sur les mots définis dans [24] par N. Globerman et D. Harel.

Définition 7.17 *Soit $n > 0$ et $k > 0$. Un mot de degré k sur n est un mot sur l'alphabet $\{0, 1, \#\}$ défini récursivement sur k de la manière suivante.*

Un mot de degré 1 sur n est un mot du langage $\#\{0,1\}^n$.

Un mot de degré 1 sur n définit ainsi un entier de $[0, 2^n - 1]$.

Si $k > 1$, on pose $m = tow(k-1, n) - 1$, un mot de degré k sur n est alors un mot de la forme $w_0 b_0 \# w_1 b_1 \# \cdots w_m b_m \#$ où chaque b_i est un bit et chaque w_i est un mot de degré $(k-1)$ sur n qui code l'entier i.

L'entier défini par $w_0 b_0 \# w_1 b_1 \# \cdots w_m b_m \#$ est l'entier de $[0, tow(k,n) - 1]$ dont le codage binaire est $b_0 \cdots b_m$

La structure de branchement de l'arbre était nécessaire pour coder l'alternance de la machine de Turing alternante. Sur les mots, nous pouvons encore coder un comportement non-déterministe et simuler une machine de Turing utilisant un espace $(k-1)$-exponentiel sur une entrée de taille n avec un automate à k jetons dont la taille est polynomial par rapport à n.

CHAPITRE

8

Perspectives

Ce travail porte sur deux modèles séquentiels d'automates d'arbres à jetons pour lesquels le placement des jetons suit une hiérarchie de pile. Dans le modèle fort le dernier jeton posé peut être levé à distance et dans le modèle faible le dernier jeton posé peut être levé uniquement si la tête de lecture de l'automate est sur le nœud où il est posé.

Dans ce dernier chapitre de perspectives, nous dressons d'abord en 8.1 un bilan des principaux résultats originaux démontrés. Nous montrons ensuite dans 8.2 comment certains résultats peuvent s'étendre aux modèles d'automates cheminants sur des arbres non bornés dont on a rappelé l'importance pour les applications pratiques au début de ce travail. Enfin, dans la section 8.3, nous énonçons quelques problèmes ouverts qui se situent dans le prolongement de ce travail.

8.1 Bilan

Nous démontrons dans le chapitre 5 que les deux modèles ont le même pouvoir d'expression ([8]). Nous construisons en effet, étant donné un automate d'arbres à jetons du modèle fort, un automate d'arbres du modèle faible équivalent avec le même nombre de jetons et nous adaptons cette construction pour le cas déterministe. Dans cette transformation, le nombre d'états augmente selon une tour de $(k-1)^2$ exponentielles où k est le nombre de jetons. La construction décrite dans le chapitre 5 est donc polynomiale si $k = 1$.

Dans le chapitre 3, nous prouvons que la classe des langages d'arbres reconnus par un automate à jetons déterministe est fermée par complément ([32]). Nous décrivons une construction de complexité quadratique du complément d'un automate d'arbres à jetons déterministe du modèle faible qui préserve le nombre de jetons. La classe

des langages reconnus par un automate cheminant déterministe est donc fermée par complément. Pour le modèle fort, nous complémentons un automate déterministe en multipliant par 3 le nombre de jetons et toujours avec une augmentation polynomiale du nombre d'états.

Le chapitre 6 établit une hiérarchie des classes d'automates d'arbres à jetons selon leur pouvoir d'expression. Nous montrons que le pouvoir d'expression des automates d'arbres à k jetons augmente strictement avec le nombre k de jetons pour les variantes déterministe et non-déterministe. Pour cela, nous définissons pour chaque entier k strictement positif un langage reconnu par un automate déterministe à k jetons mais par aucun automate non-déterministe à $(k-1)$ jetons. Nous prouvons aussi qu'on ne peut pas toujours déterminiser un automate cheminant même si on s'autorise à ajouter un nombre fixé de jetons en construisant, pour chaque entier k, un langage reconnu par un automate cheminant mais par aucun automate déterministe à k jetons. Nous démontrons enfin que les automates d'arbres à jetons ne reconnaissent pas tous les langages réguliers ([8]). En utilisant la caractérisation logique des automates d'arbres à jetons de J. Engelfriet et H.J. Hoogeboom présentée dans le chapitre 4, nous déduisons de ce dernier résultat que la logique $\mathrm{FO}+\mathrm{posTC}$ est strictement moins expressive que la logique MSO sur les arbres. Notons que ces deux logiques sont équivalentes sur les mots.

Enfin, dans le chapitre 7, nous prouvons que pour tout entier k strictement positif, les problèmes de décision du vide et de l'inclusion sont k-ExpTime complets pour les automates d'arbres à k jetons ([42]). Nous établissons la borne supérieure pour les automates non-déterministes à k jetons du modèle fort en réduisant ces problèmes au problème du vide pour les automates d'arbres bottom-up qui est polynomial : étant donné un automate à k jetons, nous construisons en temps k-exponentiel un automate bottom-up qui reconnaît le même langage et un automate bottom-up qui reconnaît le complément de ce langage. Nous prouvons ensuite la borne inférieure pour les automates déterministes du modèle faible en simulant une machine de Turing alternante utilisant un espace $(k-1)$-exponentiel où $k > 0$ avec un automate déterministe à k jetons du modèle faible. Nous montrons aussi en adaptant les preuves précédentes que les problèmes du vide et de l'inclusion sont tous deux $(k-1)$-ExpSpace complets pour les automates à k jetons sur les mots.

8.2 Des arbres binaires aux arbres de rang non-borné

Les chapitres précédents présentent les propriétés de deux modèles d'automates à jetons sur les arbres binaires finis étiquetés par un alphabet fini. Pour les arbres de rang borné, nous avons défini le type d'un nœud en 2.20. Il est alors facile de définir les automates à jetons qui reconnaissent des langages d'arbres de rang borné. L'étude de ces automates sur des arbres de rang borné est similaire à celle sur les arbres binaires et nous pouvons adapter sans difficulté les preuves de tous les théorèmes relatifs aux automates à jetons sur les arbres binaires que nous avons prouvés au cas des automates à jetons sur des arbres de rang borné.

Rappelons que les automates d'arbres à jetons sont utilisés dans la modélisation

des transformations de documents XML et qu'un document XML a une structure d'arbre fini de rang non-borné. Nous expliquons dans cette section comment étendre nos résultats aux automates à jetons sur des arbres de rang non-borné.

Les arbres de rang non-borné sont définis en 2.1.1 et, comme le précise la remarque 2.14, il nous faut adapter la définition du type d'un nœud afin de définir les automates cheminants et les automates à jetons sur les arbres de rang non-borné étiquetés par un alphabet fini. Etant donné un arbre de rang non-borné, le type d'un nœud est une information finie qui indique si le nœud considéré est la racine, une feuille, le fils d'un nœud qui est le plus à gauche parmi ses frères ou le fils d'un nœud qui est le plus à droite parmi ses frères. Le modèle des automates cheminants dont l'unique tête de lecture se déplace de nœud en nœud se généralise alors aux arbres de rang non-borné de la manière suivante. L'ensemble des déplacements possibles pour les automates cheminants sur les arbres de rang non borné est $\{\uparrow, \swarrow, \rightarrow, \leftarrow\}$. Ainsi, à partir d'un nœud v d'un arbre de rang non-borné, un automate cheminant peut se rendre au père de v, au premier fils de v, au frère juste à droite de v (right sibling) ou au frère juste à gauche de v (left sibling) si ces nœuds existent. Un automate cheminant choisit un nouvel état et un déplacement en fonction de l'état courant, du type et de l'étiquette du nœud courant.

Notons qu'il est difficile, voir impossible, pour un automate cheminant sur un arbre de rang non-borné de trouver, à partir d'un nœud quelconque de la forme ui avec $i > 0$, l'étiquette de u le père de ce nœud puis de retourner sur le nœud ui. Pour avoir un modèle d'automate séquentiel avec un pouvoir d'expression intéressant, nous pouvons considérer un nouveau modèle d'automates cheminants dans lequel l'automate a accès à l'étiquette du père du nœud courant sans avoir à se déplacer au risque de perdre la position du nœud courant. En effet, sur l'alphabet $\{0, 1, \wedge, \vee, \neg\}$, le langage des arbres qui représentent un circuit booléen évalué à vrai amène à conjecturer qu'on augmente le pouvoir d'expression du modèle des automates d'arbres cheminants sur les arbres de rang non-borné en permettant à l'automate de connaître l'étiquette du père du nœud courant.

Les automates à jetons des modèles fort et faible sont définis sur les arbres de rang non-borné en adaptant la définition de ces automates sur les arbres binaires : il suffit de modifier l'ensemble des types d'un nœud et l'ensemble des déplacements de l'automate comme pour les automates cheminants sur les arbres de rang non-borné. Notons qu'on peut facilement simuler un automate cheminant A qui a accès à l'étiquette du père du nœud courant avec un automate à 1 jeton A' : à chaque fois que A' visite un nœud u, A' pose son jeton sur u, remonte au père de u, retient l'étiquette du père de u puis A' parcourt tous les fils de u et s'arrête quand il est retourné sur son jeton.

Les résultats du chapitre 6 sont toujours valables pour les automates à jetons sur les arbres de rang non-borné. En effet, comme les langages d'arbres binaires reconnus par un automate du modèle faible à k jetons sur les arbres de rang non-borné sont exactement les langages d'arbres reconnus par un automate du modèle faible à k jetons sur les arbres binaires, pour tout entier k strictement positif, le langage d'arbres $\mathcal{L}_k$ défini en 6.3 sépare la classe des langages reconnus par un automate déterministe du modèle faible à k jetons sur les arbres de rang non-borné de la classe des langages reconnus par un automate non-déterministe du modèle faible à $(k-1)$

jetons sur les arbres de rang non-borné. De même, pour tout entier k strictement positif, le langage d'arbres $\mathcal{M}_k$ défini en 6.4 sépare la classe des langages d'arbres reconnus par un automate cheminant non-déterministe sur les arbres de rang non-borné de la classe des langages d'arbres reconnus par un automate déterministe du modèle faible à k jetons sur les arbres de rang non-borné.

Afin de complémenter un automate d'arbres à jetons A sur les arbres de rang non-borné, on construit un automate à jetons A' qui reconnaît le même langage et tel que A' peut remonter au père v à partir d'un nœud u seulement si u est le fils de v le plus à gauche : au lieu de remonter directement au père à partir d'un nœud interne quelconque, A' se déplace vers le frère juste à gauche tant que c'est possible et remonte ensuite au père. L'arbre de configuration arrière de A' est de rang borné et on peut alors adapter facilement la construction du complément de A' décrite dans le chapitre 3.

Le passage du modèle fort au modèle faible décrit dans le chapitre 5 s'étend ensuite sans difficulté dans le cas non-déterministe et dans le cas déterministe aux automates à jetons sur des arbres de rang non-borné.

Notons maintenant que les arbres de rang non-borné peuvent être encodés par des arbres binaires. La figure 8.1 illustre un des codages possibles.

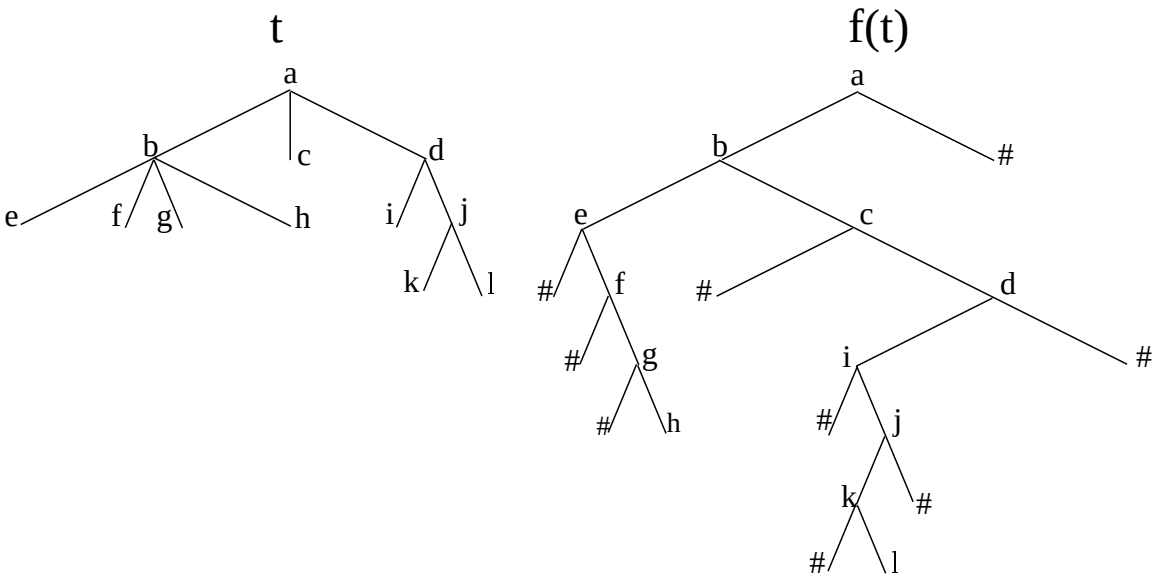

FIG. 8.1 – L'arbre de rang non-borné t et l'arbre binaire qui le code $f(t)$

L'arbre de rang non-borné t est codé par l'arbre binaire $f(t)$. Intuitivement, la relation "fils gauche" pour l'arbre binaire $f(t)$ correspond à la relation "premier fils" pour l'arbre t, la relation "fils droit" pour l'arbre binaire $f(t)$ correspond à la relation "frère suivant" pour l'arbre t et des feuilles étiquetées par le symbole $\sharp$ sont insérées pour que chaque nœud de $f(t)$ ait toujours 0 ou 2 fils. Etant donné un automate à jetons A sur les arbres de rang non-borné, on peut construire en temps linéaire un automate à jetons A' sur les arbres binaires qui reconnaît les codages des arbres reconnus par A de la manière suivante :

- si A pose ou lève un jeton, A' effectue la même action,
- si A descend au premier fils, A' descend au fils gauche,

– si A se déplace vers le frère juste à droite, A' descend vers le fils droit,
– si A se déplace vers le frère juste à gauche, A' remonte au père
– si A remonte au père, A' remonte dans l'arbre jusqu'à ce qu'il atteigne un fils gauche u et enfin A' remonte au père de u.

Etant donné un entier k strictement positif, les problèmes de décision du vide et de l'inclusion pour les automates à k jetons sur les arbres de rang non-borné se ramènent alors à ces deux problèmes pour les automates à k jetons sur les arbres binaires. Les résultats du chapitre 7 sont donc toujours valables pour les automates d'arbres à jetons sur les arbres de rang non-borné et, pour tout entier k strictement positif, les problèmes de décision du vide et de l'inclusion pour les automates à k jetons sur les arbres de rang non-borné sont k-ExpTime complets.

De plus, étant donné un langage d'arbres de rang non-borné L et un automate à jetons A sur les arbres binaires reconnaissant l'ensemble des codages des arbres de L, on peut construire en temps linéaire un automate à jetons A' sur les arbres de rang non-borné qui reconnaît L :
– si A pose ou lève un jeton, A' effectue la même action,
– si A descend au fils gauche, A' descend au premier fils,
– si A descend au fils droit, A' se déplace vers le frère juste à droite,
– si A remonte au père à partir d'un fils gauche, A' remonte au père,
– si A remonte au père à partir d'un fils droit, A' se déplace vers le frère juste à gauche.

On peut alors montrer que la caractérisation logique des automates d'arbres à jetons sur les arbres de rang borné de J Engelfriet et de H.J. Hoogeboom que nous présentons dans le chapitre 4 se généralise aux automates d'arbres à jetons sur les arbres de rang non-borné ([44]).

8.3 Problèmes ouverts

8.3.1 Concision du modèle fort ?

Dans ce travail, nous étudions deux modèles séquentiels d'automates d'arbres à jetons et nous montrons qu'ils ont le même pouvoir d'expression et la même complexité pour les problèmes du vide et de l'inclusion. Le passage du modèle fort au modèle faible est cependant très coûteux dès que le nombre de jetons est strictement supérieur à 1 : étant donné un automate du modèle fort à k jetons, la construction d'un automate à k jetons du modèle faible équivalent que nous décrivons dans le chapitre 5 augmente le nombre d'états selon une tour de $(k-1)^2$ exponentielles. Il serait alors intéressant de montrer que le modèle fort est plus concis que le modèle faible quand le nombre de jetons est strictement supérieur à 1. L'exemple suivant nous suggère que la construction du chapitre 5 est optimale pour $k = 2$.

Exemple 8.1 Considérons un entier n et l'ensemble des graphes à 2^n sommets numérotés de 0 à $(2^n - 1)$. Nous codons chaque sommet par un mot de n bits. Un mot $a_1 \cdots a_n \diamond b_1 \cdots b_n \sharp$ du langage $(0 \cup 1)^n \diamond (0 \cup 1)^n \sharp$ sur l'alphabet $\{0, 1, \diamond, \sharp\}$ code l'arc (u, v) où u est codé par le mot $a_1 \cdots a_n$ et v par le mot $b_1 \cdots b_n$. Un mot w du langage $((0 \cup 1)^n \diamond (0 \cup 1)^n \sharp)^*$ représente alors le graphe G_w tel que l'ensemble

des arcs de G correspond à l'ensemble des facteurs de w de la forme $(0 \cup 1)^n \diamond (0 \cup 1)^n \sharp$. Notons $\mathcal{E}_1$ l'ensemble des graphes tels qu'il existe un chemin du sommet 0 au sommet $(2^n - 1)$, c'est à dire tels que le sommet $(2^n - 1)$ est accessible à partir du sommet 0. Nous notons L_1 le langage des mots qui représentent des graphes de $\mathcal{E}_1$.

Nous décrivons maintenant un automate A du modèle fort à 2 jetons avec un nombre d'états polynomial en n qui reconnaît L_1. Etant donné un mot w, A vérifie d'abord en utilisant un nombre d'états linéaire en n que w représente un graphe, c'est-à-dire que w est de la forme $((0 \cup 1)^n \diamond (0 \cup 1)^n \sharp)^*$. Si ce n'est pas le cas, A rejette sinon A devine un à un les arcs d'un chemin dans G_w du sommet 0 au sommet $(2^n - 1)$ de la manière suivante. L'automate A parcourt le mot w, choisit un facteur de w codant un arc et vérifie avec un nombre linéaire d'états que ce facteur est de la forme $0^n \diamond v_1 \sharp$ où v_1 code un sommet. Ensuite, A pose le jeton 2 sur le symbole $\diamond$ de ce facteur, choisit un autre facteur $u_2 \diamond v_2 \sharp$ de w correspondant à un arc et pose le jeton 1 sur le symbole $\diamond$ de ce facteur. L'automate A vérifie alors bit par bit en faisant des aller-retours entre ses deux jetons que les mots u_2 et v_1 sont égaux. Pour effectuer ces n comparaisons, A mémorise dans son état la position des bits qu'il est en train de comparer dans les mots u_2 et v_1, il utilise donc un nombre linéaire d'états. Si u_2 et v_1 sont distincts, A rejette. Dans le cas contraire, A retourne sur la position i_1 où est posé le jeton 1 et vérifie avec un nombre linéaire d'états si le nœud v_2 est le nœud $(2^n - 1)$, c'est-à-dire si le nœud v_2 est codé par une suite de n symboles "1". Si c'est bien le cas, A lève ses deux jetons et accepte. Sinon, A retourne sur la position i_1, lève le jeton 1, puis le jeton 2 "de manière forte" en restant sur i_1. Enfin, A repose son jeton 2 sur i_1 et continue ainsi pour deviner la suite du chemin du sommet 0 au sommet $(2^n - 1)$.

Nous pouvons également construire un automate bidirectionnel A' qui reconnaît le langage L_1 avec un nombre d'états exponentiel en n. L'automate A' devine arc par arc le chemin du sommet 0 au sommet $(2^n - 1)$ comme l'automate A mais il ne dispose pas des jetons pour comparer bit par bit deux sommets. Supposons que A' ait deviné un chemin du sommet 0 à un sommet v_i. L'automate A' stocke dans son état les n bits qui codent le sommet v_i, parcourt ensuite le mot w et choisit un sommet correspondant à un arc (u_{i+1}, v_{i+1}). L'automate A' vérifie ensuite si les bits du mot u_{i+1} correspondent à ceux du mot v_i stockés dans la mémoire de A'. Si ce n'est pas le cas, A' rejette. Par contre, si les mots u_{i+1} et v_i sont identiques, A' mémorise les n bits de v_{i+1} à la place des n bits de v_i. L'automate A' devine ainsi avec un nombre exponentiel d'états un chemin du sommet 0 au sommet $(2^n - 1)$.

On peut montrer que tout automate bidirectionnel qui reconnaît L_1 a un nombre d'états qui est au moins exponentiel en n.

En effet, considérons A_0 un automate cheminant avec N états qui reconnaît L_1. Remarquons que le comportement d'un automate bidirectionnel pour un suffixe w, est déterminé par l'ensemble des couples d'états correspondant aux boucles de A sur w qui partent et terminent au début de w et par l'ensemble des couples d'états correspondant aux exécutions de A sur w qui partent au début de w et qui terminent à la fin de w. Le nombre de 0-comportements de A_1 pour un suffixe est donc $(2^N)^2 = 2^{2N}$.

Pour tout sous-ensemble V des sommets de $[1, (2^n - 2)]$, nous considérons le mot w_V qui code la suite d'arêtes $(u_1, (2^n - 1)), \cdots, (u_m, (2^n - 1))$ où $u_1, \cdots, u_m$

sont les sommets de V rangés dans l'ordre croissant. Comme il existe 2^{2^n-2} sous-ensembles de $[1,(2^n-2)]$, si $2N < 2^n-2$, il existe deux sous-ensembles distincts de sommets V_1 et V_2 tels que A a le même 0-comportement pour w_{V_1} et w_{V_2}. Dans ce cas, il existe un sommet u tel que $u \in V_1$ et $u \notin V_2$ ou bien tel que $u \in V_1$ et $u \notin V_2$. L'automate A a alors les mêmes exécutions sur les mots $0^n \diamond w_u \sharp w_{V_1}$ et $0^n \diamond w_u \sharp w_{V_2}$ où w_u est le mot qui code le sommet u. Comme un seul de ces mots appartient au langage L, l'automate A ne peut reconnaître ce langage si son nombre d'états est inférieur à $(2^{n-1}-1)$.

Si on parvient à prouver que tout automate à 2 jetons du modèle faible qui reconnaît L a au moins un nombre d'états exponentiel en n, on déduirait alors que le modèle fort d'automate à 2 jetons est exponentiellement plus concis que le modèle faible d'automates à 2 jetons. Le codage des sommets en utilisant les k-nombres définis dans le chapitre 7 pourrait ensuite permettre de généraliser la conjecture et d'établir un résultat de concision sur les automates à k jetons.

8.3.2 Complémentation d'un automate à jetons ?

Rappelons que M. Bojańczyk et T. Colcombet ont prouvé dans [6] que les automates cheminants ne peuvent pas tous être déterminisés et dans [7] que la classe des langages reconnus par un automate cheminant est strictement incluse dans la classe des langages réguliers. Il reste encore un problème ouvert fondamental sur les automates cheminants : la complémentation. Peut-on toujours complémenter un automate cheminant non-déterministe ?

Nous avons décrit dans le chapitre 3 une construction du complément d'un automate cheminant déterministe mais comme nous ne pouvons pas toujours déterminiser un automate cheminant, la complémentation des automates cheminants non-déterministes reste un problème ouvert.

On peut alors étudier les deux questions suivantes :

- Etant donné un entier k, peut on complémenter un automate cheminant non-déterministe en s'autorisant à ajouter k jetons ?
- Peut-on complémenter un automate cheminant en s'autorisant à ajouter un nombre non-fixé de jetons ?

Les mêmes problèmes se posent également pour les automates d'arbres à jetons :

- Peut-on complémenter un automate d'arbres à jetons non-déterministe sans ajouter de jetons ?
- Peut-on complémenter un automate d'arbres à jetons non-déterministe en ajoutant un nombre fixé de jetons ?
- Peut-on complémenter un automate d'arbres à jetons non-déterministe en ajoutant un nombre quelconque de jetons ?

La caractérisation logique de J. Engelfriet et H.J. Hoogeboom montre que le dernier de ces problèmes est équivalent au problème suivant : la logique FO + posTC est-elle fermée par complément ? Si c'était le cas, on pourrait caractériser les automates d'arbres à jetons par la logique FO+TC et sinon la logique FO + posTC serait strictement moins expressive que la logique FO+TC sur les arbres.

8.3.3 Déterminiser un automate à jetons en rajoutant des jetons ?

Nous avons prouvé dans le chapitre 6 qu'on ne pouvait pas toujours déterminiser un automate d'arbres cheminant même en s'autorisant à ajouter un nombre fixé de jetons. Il est alors naturel de se demander s'il est possible de déterminiser un automate d'arbres à jetons en s'autorisant à ajouter un nombre quelconque de jetons. Si c'était le cas, notons qu'on pourrait complémenter un automate d'arbres à jetons en augmentant le nombre de jetons et que les logiques FO+DTC, FO + posTC et FO+TC seraient alors équivalentes sur les arbres.

TAB. 8.1 – Index des notations

$t_f = (t, f)$	arbre marqué	Définition 5.2
$C_g = (C, g)$	contexte marqué	Définition 5.2
$C_g[t_f]$	composition contexte-arbre	Définition 5.2
$\beta_i(C_g, t_f)$	paire d'états sur i-boucle	Définition 5.3
$\beta_i^{arb}(C_g, t_f)$	paire d'états sur i-boucle de sous-arbre	Définition 5.3
$\beta_i^{con}(C_g, t_f)$	paire d'états sur i-boucle de contexte	Définition 5.3
τ, τ_i	classe de i^*-équivalence d'arbre	Notation 5.3
t_τ	arbre choisi dans la classe τ	Notation 5.3
γ, γ_i	classe de i^*-équivalence de contexte	Notation 5.3
C_γ	contexte choisi dans la classe γ	Notation 5.3
B_C^i	i-comportement de contexte défini par : $B_c^i(\tau) = \beta_i^{con}(C, t_\tau)$	Définition 5.5
B_t^i	i-comportement d'arbre défini par : $B_t^i(\gamma) = \beta_i^{arb}(C_\gamma, t)$	Définition 5.5
$B_C^{i^*}$	i^*-comportement de contexte défini par : $B_c^{i^*}(\tau) = \beta_{i^*}^{con}(C, t_\tau)$	Définition 5.5
$B_t^{i^*}$	i^*-comportement d'arbre défini par : $B_t^{i^*}(\gamma) = \beta_{i^*}^{arb}(C_\gamma, t)$	Définition 5.5
$\mathcal{C}(a, R, t_1, t_2)$	composition d'arbres	Définition 5.7
$\mathcal{C}(C, t_1, a, P_2), \mathcal{C}(C, a, P_1, t_2)$	compositon d'arbre et de contexte	Définition 5.7
$\mathcal{C}(a, R, B_1, B_2)$	composition de i^*-comportements d'arbre	Notation 5.4
$\mathcal{C}(B, B_1, a, P_2), \mathcal{C}(B, a, P_1, B_2)$	composition de i^*-comportements d'arbre et de contexte	Notation 5.4
$\rho = \rho_0, \pi_1, , ..., \rho_m, \pi_m$	décomposition d'une n-exécution d'arbres en exécution d'arbres et de boucles de contexte	Preuve des lemmes 5.7,5.8,5.9

TABLE DES FIGURES

INDEX

BIBLIOGRAPHIE

[1] A.V. Aho and J.D. Ullman. Translations on a context-free grammar. In *Information and Control*, 19(5), pages 439–475, 1971.

[2] N. Alon, T. Milo, F. Neven, D. Suciu and V. Vianu. XML with data values : typechecking revisited. In *PODS'01 : Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 138–149, 2001.

[3] M. Benedikt, L. Libkin and F. Neven. Logical definability and query languages over ranked and unranked trees. In *ACM Transactions on Computational Logic*, 8(2), pages 138–149, 2007.

[4] J.-C. Birget. Two-Way Automata and Length-Preserving Homomorphisms. In *Mathematical Systems Theory* 29(3), pages 191–226, 1996.

[5] M. Blum and C. Hewitt. Automata on a 2-Dimensional Tape. In *SWAT'67 : Conference Record of 1967 Eighth Annual Symposium on Switching and Automata Theory*, IEEE, pages 155–160, 1967.

[6] M. Bojańczyk and T. Colcombet. Tree-Walking Automata Cannot Be Determinized. In *ICALP'04 : Proceedings of the 31st International Colloquium on Automata, Languages and Programming*, pages 246–256, 2004.

[7] M. Bojańczyk and T. Colcombet. Tree-walking automata do not recognize all regular languages. In *STOC'05 : Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 234–243, 2005.

[8] M. Bojańczyk, M. Samuelides, T. Schwentick, and L. Segoufin. Expressive power of pebble automata. In *ICALP'06 : Proceeding of the 33rd International Colloquium on Automata, Languages and Programming*, pages 157–168, 2006.

[9] J.R. Büchi. Weak second-order arithmetic and finite automata. Z. Math. Logik Grundl. Math, 6, pages 66–92, 1960.

[10] A.K. Chandra, D.C. Kozen and L.J. Stockmeyer. Alternation. In *Journal of the ACM*, 28, pages 114–133, 1981.

[11] T. Cachat. Two-Way Tree Automata Solving Pushdown Games. In *Automata, Logics and Infinite Games : a guide to current research*, Springer, pages 303–317, 2002.

[12] B. ten Cate. The expressivity of XPath with transitive closure. In *PODS'06 : Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 328–337, 2006.

[13] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison and M. Tommasi Tree Automata Techniques and Applications. Available at `http://www.grappa.univ-lille3.fr/tata`.

[14] S.S. Cosmadakis, H. Gaifman, P.C. Kanellakis, and M.Y. Vardi. Decidable Optimization Problems for Database Logic Programs. In *STOC'88 : Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 477–490, 1988.

[15] J. Doner. Decidability of the weak second-order theory of two successors. In *Notices of the American Mathematical Society*, 12, pages 365–468, 1965.

[16] J. Doner. Tree acceptors and some of their applications. In *Journal of Computer and System Sciences*, 4(5), pages 406–451, Elsevier, 1970.

[17] C.C. Elgot. Decision problems of finite automata design and related arithmetics. In *Transactions of the American Mathematical Society*, 98(1), pages 21-51, 1961.

[18] J. Engelfriet and H.J. Hoogeboom. Tree-walking pebble automata. In *Jewels are forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 72–83, Springer, 1999.

[19] J. Engelfriet and H.J. Hoogeboom. Nested Pebbles and Transitive Closure. In *STACS'06 : 23rd International Symposium on Theoretical Aspects of Computer Science*, pages 477–488, 2006.

[20] J. Engelfriet, H.-J. Hoogeboom and J.-P. Van Best. Trips on Trees. In *Acta Cybernetica* 14(1), pages 51–64, 1999.

[21] J. Engelfriet and S. Maneth. A comparison of pebble tree transducers with macro tree transducers. In *Acta Informatica* 39(9) : pages 613–698, 2003.

[22] J. Engelfriet, H.J. Hoogeboom and B. Samwel. XML transformation by tree-walking transducers with invisible pebbles. In *PODS'07 : Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 63–72, 2007.

[23] T. Frühwirth, E. Shapiro, M.Y. Vardi and E. Yardeni. Logic Programs as Types for Logic Programs. In *LICS'91 : Proceedings oh the 6th IEEE Symposium on Logic in Computer Science*, pages 300–309, 1991.

[24] N. Globerman and D. Harel. Complexity Results for Two-Way and Multi-Pebble Automata and their Logics. In *Theoretical Computer Science*, 169(2), pages 161–184, Elsevier, 1996.

[25] Y. Gurevich and L. Harrington. Trees, Automata and Games. In *STOC'82 : Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pages 60–65, 1982.

[26] P. Habermehl, A. Muscholl, T. Schwentick and H. Seidl. Counting in trees for free. In *ICALP'04 : Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 1136–1149, 2004.

[27] T. Kaminura and G. Slutzki. Parallel two-way automata on directed ordered acyclic graphs. In *Information and Control*, 49(1), pages 10–51, 1981.

[28] V Kumar, P. Madhusudan and M. Viswanathan. Visibly Pushdown Automata for Streaming XML. In *WWW'07 : Proceedings of the 16th international Conference on Word Wide Web*, pages 1053–1062, 2007.

[29] M. Marx. First Order Paths in Ordered Trees. In *ICDT'05 : Proceedings of the 10th International Conference on Database Theory*, pages 114–128, 2005.

[30] W. Martens and J. Niehren. Minimizing Tree Automata for Unranked Trees. In *DBPL'05 : Proceedings of the 10th International Symposium on Database Programming Languages*, pages 232–246, 2005.

[31] T. Milo, D. Suciu and V. Vianu. Typechecking for XML Transformers. In *Journal of Computer and System Sciences*, 66(1), pages 66–97, Elsevier, 2003.

[32] A. Muscholl, M. Samuelides and L. Segoufin. Complementing deterministic tree-walking automata. In *Information Processing Letters*, 99(1), pages 33–39, Elsevier, 2006.

[33] A. Neumann and H. Seidl. Locating Matches of Tree Patterns in Forests. In *FSTTCS'98 : Proceedings of the 18th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 134–145, 1998.

[34] F. Neven and T. Schwentick. On the power of tree-walking automata. In *ICALP'00 : Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 547–560, 2000.

[35] F. Neven and T. Schwentick. XPath containment in the presence of disjunctions, DTDs, and variables. In *ICDT'03 : Proceedings of the 9th International Conference on Database Theory*, pages 315–329, 2003.

[36] F. Neven and T. Schwentick. Query Automata. In *PODS'99 : Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 205–214, 1999.

[37] F. Neven. On the Power of Walking for Querying Tree-Structured Data. In *PODS'02 : Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 77–84, 2002.

[38] F. Neven. Extensions of Attribute Grammars for Structured Documents Queries. In *DBPL'99 : Proceedings of the 7th International Symposium on Database Programming Languages*, pages 99–116, 1999.

[39] C.H. Papadimitriou. Complexity Theory. Addison Wesley, 1994.

[40] J.-P. Pécuchet. Automates Boustrophédon et Mots Infinis. In *Theoretical Computer Science*, 35, pages 115–122, Elsevier, 1985.

[41] M.O. Rabin and D. Scott. Finite Automata and their decision problems. In *IBM Journal of Research and Development*, 3, pages 114–125, 1959.

[42] M. Samuelides and L. Segoufin. Complexity In *FCT'07 : Proceedings of the 16th International Symposium on Fundamentals of Computation Theory*, pages 458–469, 2007.

[43] W.J. Savitch. Relationship between non-deterministic and deterministic tape classes. In *Journal of the ACM*, 4, pages 172–192, 1970.

[44] T. Schwentick. Automata for XML– A survey. In *Journal of Computer and System Sciences*, 73(3), pages 289–315, Elsevier, 2007.

[45] L. Segoufin and V. Vianu. Validating Streaming XML Documents. In *PODS'02 : Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 53–64, 2002.

[46] H. Seidl. Deciding equivalence of finite tree automata. In *STACS'89 : Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science*, pages 480–492, 1989.

[47] M. Sipser. Halting space-bounded computations. In *FOCS'78 : Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 73–74, 1978.

[48] D. Suciu. The XML typechecking problem. In *ACM SIGMOD Record*, 31(1), pages 89–96, 2002.

[49] J.W. Thatcher and J.B. Wright. Generalized finite automata. In *Notices of the American Mathematical Society*,12, pages 649–820, 1965.

[50] J.W. Thatcher and J.B. Wright. Generalized finite automata with an application to a decision problem of second order logic. In *Theory of Computing Systems*, 2, pages 57–82, Springer, 1968.

[51] M.Y. Vardi. A note on the reduction of two-way automata to one-way automata. In *Information Processing Letters*, 30(5), pages 261–264, 1989.

[52] M. Veanes. On computational complexity of basic decision problems of finite tree automata. in *UPMAIL Technical Report 133, Uppsala University, Computing Science Department*, 1997.

[53] W. Thomas. Languages, automata, and logic. in *Handbook of formal languages*, 3, pages 389–455, Springer, 1997.

[54] World Wide Web Consortium. Extensible Markup Language (XML). Available at `http://www.w3.org/XML/`.

[55] World Wide Web Consortium. XML Path Language. Available at `http://www.w3.org/TR/xpath`.

[56] World Wide Web Consortium. Overview of SGML Ressources. Available at `http://www.w3.org/Markup/SGML`.

[57] World Wide Web Consortium. XSL Transformations (XSLT). Available at `http://www.w3.org/TR/xslt`.

[58] Word Wide Web Consortium. HTML. Available at `http://www.w3.org/html/`.

[59] Word Wide Web Consortium. XML Query (XQuery). Available at `http://www.w3.org/XML/Query`.