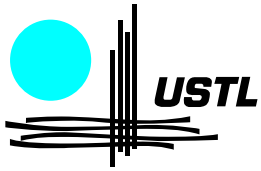


Composition de Requêtes Monadiques dans les Arbres

MÉMOIRE

présenté le 1 juillet 2005 pour le stage de

Master Recherche de l'Université des Sciences et Technologies de Lille (spécialité informatique)

par

Emmanuel Filiot

Équipe
Spécification, Test et Contraintes (STC)

Projet
INRIA MOSTRARE

Encadrants
Joachim Niehren, Jean-Marc Talbot et Sophie Tison

UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE
Laboratoire d'Informatique Fondamentale de Lille - UPRESA 8022
U.F.R d'I.E.E.A - Bât. M3 - 59655 Villeneuve d'Ascq Cedex
Tél. : +33 (0)3 20 43 47 24 - Télécopie : +33 (0)3 20 43 65 66 - Mail : direction@lifl.fr

Résumé

Un des problèmes les plus importants dans le domaine de l'extraction d'information dans les documents semi-structurés est la spécification de requêtes n-aires. Dans le contexte du Web, nous représentons les documents HTML ou XML par des arbres d'arité non bornée. Évaluer une requête n-aire dans un arbre revient alors à sélectionner un ensemble de n-uplets de noeuds de l'arbre. Nous proposons un langage de spécification de requêtes n-aires basé sur la composition de requêtes monadiques dans les arbres d'arité bornée. Nous donnons un algorithme efficace pour répondre aux requêtes de composition. Nous prouvons que le langage de composition capture toutes les requêtes régulières, ie les requêtes exprimables en logique monadique du second ordre (MSO). La preuve utilise la correspondance entre les automates d'arbres et MSO, établit par le théorème fondateur de Thatcher et Wright. Enfin, nous étendons la composition au cas des arbres d'arité non bornée, via un codage vers les arbres binaires, et prouvons que le langage capture toujours les requêtes n-aires régulières.

Remerciements

Je tiens à remercier toute l'équipe Mostrare, et plus particulièrement mes encadrants, Joachim Niehren, Jean-Marc Talbot et Sophie Tison, pour leur disponibilité et leurs conseils pertinents.

Merci aussi à Laurent Planque, pour les nombreuses discussions que nous avons eues, et pour ses nombreuses explications.

Enfin, je remercie tous ceux qui m'ont soutenu durant mon stage, et tous ceux qui ont relu mon mémoire.

Table des matières

Chapitre 1

Introduction

Un des problèmes les plus importants en *extraction d'information* et en *transformation de documents* est de pouvoir spécifier et répondre à des requêtes dans les documents *semi-structurés*, i.e. des documents structurés (sous la forme de titres, de tableaux ...) mais contenant aussi des parties non structurées (comme par exemple du texte brut).

Arbres d'arité bornée ou non bornée Dans le contexte qui nous intéresse ici, à savoir le Web, les pages HTML ou XML (*Extensible Markup Language* [2]) sont des documents semi-structurés. Nous les représentons par des arbres, dont le feuillage contient les données textuelles (voir la figure 1.1). Dans cet exemple, il apparaît que les arbres sont ordonnés (les fils d'un noeud sont ordonnés), et qu'un noeud peut avoir un nombre non borné de fils (par exemple le noeud étiqueté par LIST). On parlera d'*arbres d'arité non bornée*. À l'inverse, nous parlons d'*arbre d'arité bornée* lorsqu'il existe une borne globale fixée sur le nombre de fils d'un noeud.

Requêtes dans les arbres Dans le contexte des arbres, effectuer des requêtes revient à sélectionner des noeuds (*requêtes unaires* ou *monadiques*) ou des n-uplets de noeuds (*requêtes n-aires*) qui satisfont une certaine propriété. La figure 1.2 donne un exemple de requête monadique qui sélectionne tous les emails d'une liste. Une requête binaire sur l'arbre de la figure 1.2 pourrait être l'extraction de toutes les paires (n, m) où n est un nom, m est un email, et n et m sont frères, le résultat d'une telle requête sur l'arbre de la figure 1.2 est donné figure 1.3.

État de l'art et problématique Le cas des requêtes monadiques dans les arbres d'arité bornée ou non bornée a été très largement étudié, et il existe plusieurs formalismes pour exprimer ce type de requêtes. Un état de l'art est donné dans [15]. Citons par exemple *Datalog Monadique*[13, 14] inspiré de la programmation logique, le standard pour les documents XML, *XPATH*, et des dérivés de XPATH comme *XQuery* et *XPATH Conditionnel*[15], les *automates de requêtes* [16], les requêtes par *automates à pas* [8], des logiques modales comme *LTL*, *CTL*, *PDL*, et une logique de point fixe, le *μ-calcul* [15]. Il existe peu de formalismes pour les requêtes n-aires, nous pouvons citer les *requêtes par automates d'arbres* [17] qui admettent un algorithme de réponse en temps polynomiale, et ont une bonne expressivité (nous préciserons plus tard ce que nous entendons par «bonne») ; *XPATH Conditionnel* pour les requêtes binaires ; la *logique monadique du second ordre* (MSO), qui n'admet pas d'algorithme

```
<LIST>
  <ITEM>
    <NAME>
Eloïse
    </NAME>
    <EMAIL>
Eloïse@abélard.fr
    </EMAIL>
  </ITEM>
  <ITEM>
    <NAME>
Abélard
    </NAME>
    <EMAIL>
Abélard@éloïse.fr
    </EMAIL>
  </ITEM>
</LIST>
```

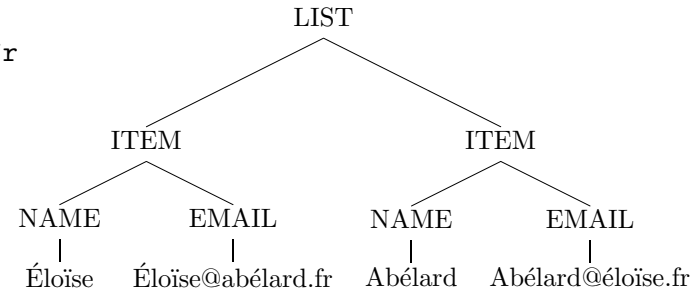


FIG. 1.1 – Un document XML et sa représentation arborescente

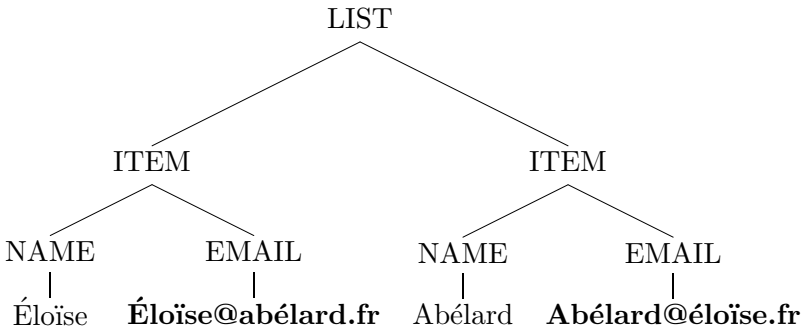
de réponse en temps polynomiale. L'étude de ces formalismes nous a conduit à soulever le problème suivant : quel formalisme adopter pour spécifier les requêtes n-aires, ayant une bonne expressivité, permettant de répondre rapidement aux requêtes, facilement spécifiable par un utilisateur, et ayant de bonnes propriétés d'apprenabilité ?

Contribution : composition de requêtes monadiques Nous proposons un formalisme de requêtes n-aires basé sur la composition de requêtes monadiques. Composer des requêtes monadiques permet d'utiliser les formalismes de requêtes monadiques déjà existants et par là-même permet d'avoir une certaine généricité dans les algorithmes, qui sont alors paramétrés par des fonctions dédiées aux requêtes monadiques.

On peut de plus composer des requêtes monadiques de différents formalismes, certains s'appuyant plutôt sur la structure de l'arbre (par exemple pour sélectionner des zones), d'autres sur le contenu textuel des noeuds.

Enfin, si on veut extraire de l'information dans les pages Web via les liens hypertextes, il peut être intéressant de pouvoir composer des requêtes. Une première requête sélectionne les liens, et une deuxième requête s'applique dans les pages pointées par les liens précédemment sélectionnés.

Nous proposons plus particulièrement un langage de composition de requêtes avec restriction dans les sous-arbres. Cette idée s'inspire d'une vision naturelle de l'extraction d'information. Lorsqu'un lecteur recherche une information dans une page de journal, il regarde globalement la structure de la page, les gros titres, et sélectionne une zone de la page qu'il juge pertinente. Ensuite, il approfondit sa recherche dans la zone sélectionnée en répétant le même processus. Si on voit la page de journal comme un arbre – l'arbre du découpage de cette page – sélectionner une zone revient à sélectionner un noeud de l'arbre par une requête monadique, et approfondir la recherche revient à faire une autre requête dans le sous-arbre



Résultat : {Éloïse@abélard.fr, Abélard@éloïse.fr}

FIG. 1.2 – Une requête monadique : extraire l'ensemble des emails (en gras), et son résultat

Résultat : {(Abélard, Éloïse@abélard.fr) , (Éloïse, Abélard@éloïse.fr)}

FIG. 1.3 – Résultat de la requête binaire sur l'arbre de la figure 1.2 : extraire tous les couples de frères (nom, email)

sélectionné précédemment.

Une seconde motivation est le fait que, par exemple, dans la recherche de couples de noeuds dans un arbre, les ancêtres communs du couple constituent une information importante sur la relation qu'entretiennent les éléments du couple. Une tâche d'extraction pourrait alors être la sélection (monadique) d'un noeud intermédiaire représentant l'ancêtre du couple, puis, de manière indépendante, la sélection de la première composante, suivie de la sélection de la deuxième. En trouvant un formalisme explicitant le rôle que joue les ancêtres communs, on peut espérer trouver certaines classes de requêtes pour lesquelles l'évaluation est très efficace. Reprenons l'arbre de la figure 1.2. Pour extraire les couples de frères (nom, email), une première requête monadique pourrait être la sélection d'un noeud intermédiaire ITEM, puis, dans le sous-arbre dont la racine est ITEM, la sélection, en parallèle, du nom, et de l'email. .

Cette idée s'inspire aussi du système d'extraction LixTo [5]. Ce système permet de spécifier visuellement une requête par sélection de zones, et spécifications de sous-requêtes dans les zones précedemment sélectionnés. Par exemple, l'utilisateur charge une page web, comme la liste d'objets d'une page du site Ebay, sélectionne la zone d'un objet, et dans cette zone, sélectionne le prix. Le système généralise sur les expressions régulières de chemins allant de la racine aux zones des objets, afin de permettre la sélection de toute zone d'objet, et pas seulement celle prise en exemple par l'utilisateur pour spécifier la requête. L'expressivité du langage logique ELOG utilisé dans LixTo est étudiée dans [12]. Cependant, bien qu'étant clairement un formalisme pour les requêtes monadiques, son extension à un formalisme de spécification de requêtes n-aires n'est pas très claire.

Enfin, le processus naturel d'extraction sous-jacent au formalisme proposé peut laisser penser qu'il a de bonnes propriétés d'apprentissage. Nous détaillons ensuite ce que nous entendons par «apprentissage ».

Apprentissage de wrappers Une autre problématique liée à l'extraction d'information est l'*apprentissage de wrappers*. Un *wrapper* est un programme d'extraction d'information. Pour une classe de document donnée, l'utilisateur pourrait écrire son wrapper «à la main », ce qui n'est réservé qu'aux informaticiens, et n'est pas invulnérable aux nombreuses erreurs de codage. Une solution à ce problème est alors d'apprendre automatiquement les wrappers, à partir d'un ensemble fini d'exemples fournis par l'utilisateur. L'utilisateur sélectionne des éléments à extraire dans une page web, et le système apprend le wrapper correspondant ; ceci est l'apprentissage à partir d'*exemples positifs seuls*. Malheureusement, ce mode d'apprentissage n'est parfois pas possible, et on a besoin d'*exemples négatifs*, i.e. d'éléments de la page web dont on sait qu'ils ne doivent pas être sélectionnés. L'utilisateur devrait donc être capable de sélectionner des éléments pertinemment négatifs, ce qui suppose une connaissance préalable du système d'apprentissage. Une solution à ce problème est de faire interagir le système d'apprentissage et l'utilisateur. Le système apprend un wrapper W_0 à partir d'exemples positifs, réalise la tâche d'extraction sur un ensemble de documents donnés par l'utilisateur, et l'utilisateur indique au système quels noeuds n'auraient pas du être extraits, fournissant ainsi des exemples négatifs, le système réapprend un wrapper W_1, etc... ces interactions continuent tant que l'utilisateur n'est pas satisfait. L'apprentissage de requêtes monadiques par automates à pas via un système interactif est étudié dans [7]. Malheureusement, l'apprentissage de requêtes par automates à pas n'est pas facilement généralisable au cas n-aire, et l'idée de naturelle de la composition de requêtes monadiques fait penser que la composition est un formalisme bien adapté à l'apprentissage de requêtes n-aires, qui permettrait de plus d'utiliser les techniques d'apprentissage des requêtes monadiques. Cependant les propriétés d'apprenabilité de la composition n'ont pas été étudiées durant le stage.

Mesurer l'expressivité Les formalismes de requêtes recherchés doivent être de bons compromis entre *expressivité* et *efficacité de réponse*. Mesurer l'expressivité d'un formalisme se fait par comparaison avec des formalismes existants. On cherchera en particulier à exprimer une requête d'un formalisme dans un autre, ou à prouver qu'il existe une requête non exprimable dans un formalisme donné. Le formalisme «étalon» dans le contexte des structure arborescentes est la *logique monadique du second ordre* (MSO), car les langages d'arbres définissables en MSO sont exactement les langages d'arbres réguliers, i.e. les langages reconnus par automates d'arbres [19]. De plus, les documents XML sont souvent fournis avec une DTD, et le formalisme des *DTDs étendues* a la même expressivité que MSO [20]. Cependant, MSO n'est pas un bon formalisme pour les requêtes n-aires, car le *model checking* d'une formule ϕ par un arbre t est en temps $\mathcal{O}(2^{|\phi||t|})$, où $|\phi|$ est la taille de la formule, et $|t|$ la taille de l'arbre [11]. Malgré ce résultat, il existe des formalismes ayant la même expressivité que MSO et admettant des algorithmes efficaces pour le *model checking*, en particulier, le formalisme de composition proposé ici a la même expressivité que MSO, et nous proposons un algorithme en $\mathcal{O}(|t|^{|\phi|})$ pour le problème du model checking.

Plan Le deuxième chapitre reprend et formalise certaines notions déjà rencontrées dans cette introduction, et donne des notions préliminaires à l'introduction du formalisme de composition de requêtes, qui fait l'objet du troisième chapitre. Dans le quatrième chapitre nous donnons la preuve que ce formalisme à l'expressivité de MSO, lorsque les requêtes monadiques sont MSO-définissables, et donnons un corollaire important de la MSO-complétude. Enfin, dans le cinquième chapitre nous étendons la composition au cas des arbres d'arité non bornée, en

passant par un codage binaire et en se ramenant au cas borné. Nous donnons la preuve de MSO-complétude dans le cas des arbres d'arité non bornée.

Équipe de recherche Ce sujet s'inscrit dans le cadre du projet INRIA MOSTRARE [4] (*Modèles de structures arborescentes, apprentissage et extraction d'information*), dont l'objectif est le développement de nouvelles techniques de recherche et d'extraction d'information utilisant la structure arborescente des documents. Les deux axes de recherche du projet sont l'étude et la définition de modèles et d'algorithmes pour des structures arborescentes adaptés à la tâche d'extraction d'information, et la conception d'algorithmes d'apprentissage utilisant la structure arborescente des documents.

Chapitre 2

Requêtes dans les arbres d'arité bornée

On désigne par $\mathbb{N}_k$ l'ensemble $\{1, \ldots, k\}$ et par Σ un alphabet fini. Étant donné un n-uplet $\overrightarrow{s}$ d'éléments, on note $\mathtt{proj}_i(\overrightarrow{s})$ la i-ème composante de ce n-uplet.

Dans ce chapitre, nous définissons les arbres d'arité bornée, la notion de requêtes n-aires, et présentons deux formalismes pour les requêtes n-aires, les requêtes par formules de la logique du second ordre (MSO), et les requêtes par automates.

2.1 Arbres ordonnés d'arité bornée

Rappelons que nous représentons les pages HTML/WEB par des arbres. Comme les balises HTML sont en nombre fini, et que pour une DTD donnée, les balises XML le sont aussi, il est légitime de travailler sur un alphabet Σ fini. De plus, dans les tâches d'extraction dans les documents semi-structurés, le texte (c'est à dire les feuilles), est représenté par un symbole unique ou parfois un nombre fini de symboles (représentant la catégorie du texte, ou la police, etc...). On extrait donc une abstraction du texte, tâche qui pourra être prolongée par des techniques d'extraction dans le texte brut (*grep* par exemple), cette seconde phase ne faisant pas partie du sujet de stage.

En outre, nous avons vu que certaines balises pouvaient avoir un nombre non borné de fils dans la représentation arborescente, on parle alors d'*arbres d'arité non bornée*. Néanmoins, on peut coder – comme on le verra dans le quatrième chapitre – ce type d'arbres dans les arbres d'arité bornée définis ci-après ; c'est pourquoi notre étude commence par les arbres d'arité non bornée. Ces arbres sont un peu différents des arbres de *l'algèbre des termes* définie dans [9], où chaque symbole de l'alphabet à une arité unique. Ici les symboles qui étiquettent les noeuds peuvent avoir différentes arités, mais ces arités sont bornées par un entier fixé.

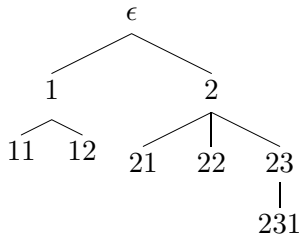
Définition 1 (Domaine d'arbre d'arité k). Un *domaine d'arbre d'arité k* est un sous-ensemble fini D de $\mathbb{N}_k^*$ (monoïde libre sur $\mathbb{N}_k$) tel que :

1. si $\pi \in D$ et π' est un préfixe de π, alors $\pi' \in D$
2. si $\pi i \in D$ pour $i \in \mathbb{N}_k$, alors pour tout $0 < j < i,\ \pi j \in D$

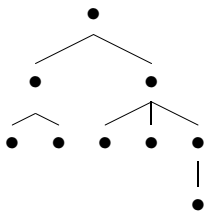
Sans ambiguité sur l'arité, on parlera de *domaine d'arbre*. Les éléments de D sont appelés *noeuds* ou *positions*. On parlera donc aussi d'ensemble de noeuds pour désigner D. Les préfixes

d'un noeud π sont appelés *ancêtres* et un noeud πi est appelé *i-ième fils* de π. La condition 1 impose que tous les ancêtres d'un noeud $\pi \in D$ soient dans D, et la condition 2 dit intuitivement que l'existence du i-ème fils d'un noeud est conditionnée par l'existence du j-ième fils de ce noeud pour tout j plus petit que i. Enfin, le noeud ϵ est appelé *racine* et les noeuds $\pi \in D$ tel que $\pi 1 \notin D$ (i.e. des noeuds sans fils) sont appelés *feuilles*.

Exemple 1. Voici une représentation arborescente du domaine d'arbre $D = \{\epsilon, 1, 11, 12, 2, 21, 22, 23, 231\}$:



Les positions dans cette représentation constituent une information redondante, D pouvant simplement être représenté par :



Un domaine d'arbre décrit donc une structure d'arbre. Un arbre n'est alors rien d'autre que la donnée d'un domaine, et d'un étiquetage des noeuds dans un alphabet Σ. Plus formellement :

Définition 2 (Arbres ordonnés d'arité k sur Σ). Un *arbre ordonné t d'arité k* sur un alphabet Σ est une application d'un domaine d'arbre D d'arité k vers Σ. L'ensemble D des noeuds de l'arbre sera aussi noté $\texttt{dom}(t)$.

Enfin, l'ensemble des arbres ordonnés d'arité bornée k sera noté T_Σ^k, et on note $T_\Sigma = \bigcup_{k \in \mathbb{N}} T_\Sigma^k$, et on notera parfois $\texttt{label}^t(\pi)$ pour $t(\pi)$, voire $\texttt{label}(\pi)$ lorsqu'il n'y aura pas d'ambiguité.

On parle d'arbres ordonnés car les fils d'un noeud sont ordonnés. Il faut remarquer que l'arbre n'est pas nécessairement complet, un noeud pouvant avoir moins de k fils. Définissons maintenant la notion de sous-arbre :

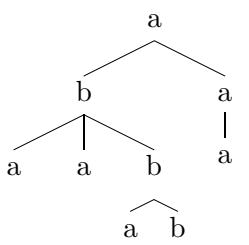
Définition 3 (Sous-arbre). Étant donné un arbre t sur un alphabet Σ, et un noeud $\pi \in \texttt{dom}(t)$, le *sous-arbre induit* par π (ou *sous-arbre en position* π) est l'arbre $t|_\pi$ tel que $\texttt{dom}(t|_\pi)$ est le domaine $\texttt{dom}(t)$ *relativisé* au noeud π défini par :

$$\texttt{dom}(t|_\pi) \;=\; \{\pi' \mid \pi\pi' \in \texttt{dom}(t)\}$$

et $t|_\pi$ est défini par :

$$\begin{array}{rcccc} t|_\pi & : & \mathtt{dom}(t|_\pi) & \to & \Sigma \\ & & \pi' & \mapsto & t(\pi\pi') \end{array}$$

Exemple 2. Sur $\Sigma = \{a, b\}$ voici la représentation d'un arbre t d'arité 3 et du sous-arbre $t|_{13}$ en position 13 :

$t =$ 

$t|_{13} =$ 

On définit aussi un ordre partiel $\lhd^*$ sur les noeuds d'un arbre qui décrit intuitivement la relation «être au-dessus de », qu'on appelle *ordre préfixe*.

Définition 4 (ordre préfixe, hauteur, taille). Étant donné un domaine d'arbre D, l'*ordre préfixe* $\lhd^*$ sur D est défini par :

$$\pi \lhd^* \pi' \text{ si et seulement si } \pi \text{ est un préfixe de } \pi'$$

La *taille* $|t|$ d'un arbre $t : D \to \Sigma$ est définie comme le cardinal de D.
La *hauteur* $h(t)$ d'un arbre $t : D \to \Sigma$ est définie par $h(t) = max_{\pi \in D} l(\pi)$, où $l(\pi)$ est la longueur du mot π.

Arbres et grammaires On définira parfois une classes d'arbres d'arité bornée k sur un alphabet Σ par une grammaire. Par exemple, l'ensemble T^k_Σ peut être vu comme l'ensemble des arbres générés par la grammaire :

$$\begin{array}{rcll} t & ::= & a & a \in \Sigma \\ & & |f(t_1, \ldots, t_n) & f \in \Sigma, n \leq k \end{array}$$

Les arbres $t_1, \ldots, t_n$ sont alors les sous-arbres de $f(t_1, \ldots, t_n)$ en position $1, 2, \ldots, n$ respectivement. Il sera alors parfois commode de faire des preuves par induction sur la structure de l'arbre, le cas de base étant la feuille, ou de définir des fonctions sur les arbres inductivement sur leurs structures. Par exemple, la taille d'un arbre peut être définie inductivement par : $|a| = 1$ pour $a \in \Sigma$, et $|f(t_1, \ldots, t_n)| = \Sigma_{i=1}^n |t_i| + 1$ pour $f \in \Sigma$. De même, le domaine d'un arbre peut aussi être défini inductivement sur la structure de l'arbre par $\mathtt{dom}(a) = \{\epsilon\}$ et $\mathtt{dom}(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \bigcup_{i=1}^n \{i.\pi \mid \pi \in \mathtt{dom}(t_i)\}$. Comme l'ensemble des arbres générés par la grammaire donnée ci-dessus est en bijection avec l'ensemble des arbres T^k_Σ, nous confondons les notions d'arbre (une fonction d'un domaine vers l'alphabet est vu comme un élément généré cette grammaire et réciproquement).

2.2 Requêtes n-aires dans les arbres

Définissons formellement la notion de requête dans les arbres d'arité bornée.

Définition 5 (requête sur les arbres d'arité k). Une *requête n-aire* q sur les arbres d'arité k est une application qui à tout arbre $t \in T_\Sigma^k$ associe un sous-ensemble de $\mathtt{dom}(t)^n$:

$$\forall t \in T_\Sigma^k \; q(t) \subseteq \mathtt{dom}(t)^n$$

Une *requête monadique* est une requête 1-aire.

Définition 6 (union, intersection, produit, complémentaire, projection). Étant données trois requêtes n-aires q_1,q_2 et q sur et une requête m-aire q' sur T_Σ^k, on définit l'*union* $q_1 \cup q_2$ de q_1 et q_2, l'*intersection* $q_1 \cap q_2$ de q_1 et q_2,le *produit cartésien* $q \times q'$ de q et q', la *i-ème projection* $\mathtt{proj}_i(q)$ de q, le *complémentaire* q^c de q par :

$$\forall t \in T_\Sigma^k \quad \begin{cases} (q_1 \cup q_2)(t) & = & q_1(t) \cup q_2(t) \\ (q_1 \cap q_2)(t) & = & q_1(t) \cap q_2(t) \\ (\mathtt{proj}_i(q))(t) & = & \{(\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n) \; : \; (\pi_1, \ldots, \pi_n) \in q(t)\} \\ q^c(t) & = & \mathtt{dom}(t)^n - q(t) \\ (q \times q')(t) & = & q(t) \times q'(t) \text{ modulo associativité} \end{cases}$$

Le produit cartésien de deux requêtes est pris modulo associativité, c'est à dire que $(\pi_1, \ldots, \pi_n, \pi'_1, \ldots, \pi'_n) \in (q \times q')(t)$ ssi $(\pi_1, \ldots, \pi_n) \in q(t)$ et $(\pi'_1, \ldots, \pi'_m) \in q'(t)$.

2.3 Requêtes par formules MSO

Nous définissons un premier formalisme de requêtes, qui utilise des formules de la logique du premier order (FO), ou de la logique du second ordre (MSO). Avant de voir comment les formules peuvent être utilisées pour spécifier des requêtes, nous définissons les logiques utilisées, FO et MSO.

2.3.1 MSO [10, 15]

Certains formalismes logiques permettent de décrire des propriétés des noeuds d'un arbre, d'axiomatiser une classe d'arbres, ou de spécifier des requêtes. C'est le cas de la *logique du premier ordre*, appelée FO (*First Order*), et de la *logique monadique du second ordre*, plus expressive, appelée MSO (*Monadic Second Order*). Comme nous avons dit dans le chapitre *Introduction*, MSO est le formalisme de référence pour comparer l'expressivité de tout formalisme de requête, car les langages d'arbres MSO-définissables sont exactement les langages d'arbres réguliers [19], ie les langages d'arbres reconnaissables par automates d'arbres, comme nous verrons dans la section 2.4. Une logique est donnée par un ensemble de *formules* sur un *vocabulaire* fixé et une *classe d'interprétation des formules*, i.e. une classe de *structures* sur lesquelles ces formules sont interprétées. Dans le contexte des arbres d'arité bornée, la classe d'interprétations sera l'ensemble des structures induites par les arbres. Nous donnons ici les règles de formation des formules de MSO, le vocabulaire et les structures propres aux arbres d'arité bornée, la relation de satisfiabilité des formules dans ces structures, et définissons le problème du *model-checking*.

On appelle *vocabulaire* un ensemble fini V de symboles de *relations* (ou *prédicats*). Chaque symbole $P \in V$ est muni d'un entier $\mathrm{ar}(P)$, son *arité*. On notera $P(.,.)$ pour dire que le symbole P est d'arité 2.

Structure Une *V-structure* S (finie) sur un vocabulaire V est la donnée d'un ensemble fini $\mathtt{dom}(S)$ appelée *domaine*, et d'une relation $R^S \subseteq \mathtt{dom}(S)^n$ d'arité n pour chaque symbole $R \in V$ tel que $\mathrm{ar}(R) = n$. La relation R^S est encore appelé *interprétation* de R dans S. On écrira plutôt $R^S(e_1, \ldots, e_n)$ au lieu de $(e_1, \ldots, e_n) \in R^S$, et S sera notée $< \mathtt{dom}(S), R^S, P^S, ... >$.

Arbre comme structure Tout arbre $t \in T_\Sigma^k$ peut être vu comme une structure sur le vocabulaire $\mathbb{T} = \{\mathtt{label}_a(.), S_i(.,.), =\ (.,.)\}_{a\in\Sigma, 1\leq i\leq k}$, dont le domaine est précisément le domaine de l'arbre, et dont les symboles de prédicats sont interprétés par :

1. Pour tout i tel que $0 < i \leq k$, $S_i^t = \{(\pi, \pi i) \mid \pi i \in \mathtt{dom}(t)\}$
2. Pour tout $a \in \Sigma$, $\mathtt{label}_a^t = \{\pi \mid t(\pi) = a\}$
3. $=$ est interprété comme l'égalité.

Dans la suite nous confondrons un arbre t et sa structure associée
$< \mathtt{dom}(t), (S_i^t)_{1\leq i\leq k}, (\mathtt{label}_a^t)_{a\in\Sigma} >$, et omettrons d'indiquer l'égalité, que nous supposerons toujours présente. De plus, lorsqu'il n'y aura pas d'ambiguïté sur la structure d'interprétation, on confondra les symboles de prédicats et leur interprétation.

MSO

Soit $\mathtt{Var} = \{x, y, z, x_1, x_2, \ldots\}$ un ensemble dénombrable de variables du premier ordre, et un ensemble dénombrable $\{X, Y, Z, \ldots\}$ de variables du second-ordre (notées en majuscule).

Définition 7 (syntaxe des formules de MSO). Une *formule* de la *logique monadique du second ordre* sur un vocabulaire V est définie inductivement par la grammaire :

$$\phi \ ::= \ P(\overrightarrow{x}) \mid x \in X \mid \exists x\ \phi \mid \exists X\ \phi \mid \phi \vee \phi \mid \neg\phi$$

pour tout $P \in V$ d'arité n, avec $\overrightarrow{x} \in \mathtt{Var}^n$.
L'ensemble des formules de MSO sur le vocabulaire V est noté MSO$[V]$.
On définit $\phi_1 \wedge \phi_2$ par $\neg(\neg\phi_1 \vee \neg\phi_2)$, $\forall x\phi$ par $\neg\exists x\neg\phi$, $\forall X\phi$ par $\neg\exists X\neg\phi$, et $\phi_1 \to \phi_2$ par $\neg\phi_1 \vee \phi_2$. Enfin, $\equiv$ désignera l'équivalence syntaxique des formules.

L'ensemble VL(ϕ) des *variables libres* d'une formule MSO est défini inductivement par :
– VL$(P(\overrightarrow{x}))$ est l'ensemble des variables apparaissant dans $\overrightarrow{x}$
– VL$(\phi_1 \vee \phi_2) =$ VL$(\phi_1) \cup$ VL(ϕ_2)
– VL$(\neg\phi) =$ VL(ϕ)
– VL$(\exists x\phi) =$ VL$(\phi) - x$ (dans ce cas la variable x est dite liée dans $\exists x\phi$)
– VL$(\exists X\phi) =$ VL$(\phi) - X$ (dans ce cas la variable X est dite liée dans $\exists X\phi$)
Par exemple, VL$(\forall x S_1(x, y) \wedge \forall y(S_2(y, z) \wedge y \in X)) \ = \ \{y, z, X\}$

On note $\phi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ pour indiquer que VL$(\phi) \subseteq \{x_1, \ldots, x_n, X_1, \ldots, X_m\}$. Une formule sans variable libre est appelée *formule close*. Étant donnée une formule ϕ de MSO$[V]$ sur un vocabulaire V, et une V-structure S, une *valuation* ν des variables libres de V est de domaine de définition inclu dans VL(ϕ), qui à toute variable libre du premier ordre associe un élément de $\mathtt{dom}(S)$, et à toute variable libre du second ordre associe un sous-ensemble de $\mathtt{dom}(S)$. On notera $\nu[x \leftarrow \pi]$ la valuation ν' égale à ν sur VL$(\phi) - x$, et qui affecte

π à x. On définit de même la valuation $\nu[X \leftarrow \Pi]$.

Dans la suite du rapport, nous désignons par MSO l'ensemble MSO[$\mathbb{T}$], ie l'ensemble des formules MSO sur le vocabulaire des arbres d'arité bornée. Voici un exemple de formule MSO sur les arbres d'arité bornée :

$$\forall x \exists y \ S_1(x,y) \rightarrow \texttt{label}_a(x)$$

Cette formule est satisfaite par un arbre t si et seulement si tout noeud de t est étiqueté par a. La notion de satisfiabilité des formules de MSO est définie par :

Définition 8 (satisfiabilité d'une formule). Étant donné un arbre $t \in T_\Sigma^k$ et une formule $\phi(x_1, \ldots, x_n, X_1, \ldots, X_n)$, une $\mathbb{T}$-structure t, et une valuation ν des variables libres de ϕ, la relation de satisfiabilité $\models$ est définie inductivement par :

$$\begin{array}{lll}
t, \nu \models S_i(x,y) & \text{ssi} & S_i^t(\nu(x), \nu(y)) \\
t, \nu \models \texttt{label}_a(x) & \text{ssi} & \texttt{label}_a^t(\nu(x)) \\
t, \nu \models x \in X & \text{ssi} & \nu(x) \in \nu(X) \\
t, \nu \models \phi_1 \vee \phi_2 & \text{ssi} & t, \nu \models \phi_1 \text{ ou } t, \nu \models \phi_2 \\
t, \nu \models \neg\phi & \text{ssi} & \text{non } t, \nu \models \phi \\
t, \nu \models \exists x \ \phi & \text{ssi} & \text{il existe } \pi \in \texttt{dom}(t) \text{ tq } t, \nu[x \leftarrow \pi] \models \phi \\
t, \nu \models \exists X \ \phi & \text{ssi} & \text{il existe } \Pi \in 2^{\texttt{dom}(t)} \text{ tq } t, \nu[X \leftarrow \Pi] \models \phi
\end{array}$$

On dit alors que (t, ν) est un *modèle* de ϕ, ou (t, ν) *satisfait* ϕ. On notera souvent $t, (\pi_1, \ldots, \pi_n, \Pi_1, \ldots, \Pi_m) \models \phi(x_1, \ldots, x_n, X_1, \ldots, X_m)$, ou $t \models \phi(\pi_1, \ldots, \pi_n, \Pi_1, \ldots, \Pi_m)$ à la place de $t, \nu \models \phi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ où ν est définie par $\nu(x_i) = \pi_i \ \forall 1 \leq i \leq n$ et $\nu(X_i) = \Pi_i \ \forall 1 \leq i \leq m$.

Si nécessaire, la relation de satisfiabilité sera aussi noté $\models_{MSO}$.

La *logique du premier ordre* (FO), est définie comme MSO, **sans** la quantification ensembliste, et le test d'appartenance ($\exists X \ \phi$ et $x \in X$).

La figure 2.1 donne quelques exemples (importants) de formules de MSO. On a alors :
- $t \models \texttt{egal}(\pi_1, \pi_2)$ si et seulement si $\pi_1 = \pi_2$. L'égalité n'est alors pas nécessaire dans le vocabulaire $\mathbb{T}$.
- $t \models \texttt{child}(\pi_1, \pi_2)$ si et seulement si π_2 est un fils de π_1 dans t ;
- $t \models \texttt{leaf}(\pi)$ si et seulement si π est une feuille de t ;
- $t \models \texttt{root}(\pi)$ ssi π est une racine ;
- pour tout arbre t, $t \models \texttt{True}$ (le domaine d'un arbre est toujours supposé non vide)
- enfin, la formule $\lhd^*(x,y)$ représente l'ordre préfixe

Un problème fondamental en logique est le problème du *model-checking* : étant donnés un arbre t, une valuation ν, et une formule ϕ, est-ce que $t, \nu \models \phi$?

Proposition 1 (model checking de MSO(resp.FO) [11]). *Le problème du model-checking de* MSO[$\mathbb{T}$] *(resp.* FO[$\mathbb{T}$]*) est PSPACE-Complet.*

De plus, il existe un autre théorème sur la complexité du model-checking, qui établit que si on veut avoir une complexité fixée en la taille de l'arbre, alors la complexité en la taille de la formule est non élémentaire (non bornée par une tour d'exponentielles) :

1. $\texttt{egal}(x, y) \equiv x \lhd^* y \wedge y \lhd^* x$
2. $\texttt{child}(x, y) = \bigvee_{i=1}^{k} S_i(x, y)$
3. $\texttt{leaf}(x) = \neg\exists y \texttt{child}(x, y)$
4. $\texttt{root}(x) = \neg\exists y\ \texttt{child}(y, x)$
5. $\texttt{True} = \exists x\ \texttt{root}(x)$
6. $\lhd^*(x, y) \equiv \forall X\ (x \in X \wedge \forall z_1 z_2 (z_1 \in X \wedge \texttt{child}(z_1, z_2) \rightarrow z_2 \in X) \rightarrow y \in X)$

FIG. 2.1 – FO-formules sur les arbres d'arité bornée

Théorème 1 (Frick et Grohe [11]). *Il n'existe pas d'algorithme pour le model-checking de FO et MSO, pour une formule ϕ de FO et MSO, et un arbre $t \in T_\Sigma$, en temps $\mathcal{O}(f(|\phi|)p(|t|))$ où f est une fonction élémentaire (i.e. une fonction bornée par une tour d'exponentielles), p un polynôme, $|\phi|$ étant la taille de la formule et $|t|$ la taille de l'arbre.*

Enfin, on peut définir un langage d'arbres par une formule close de MSO, c'est l'ensemble des arbres qui satisfont cette formule. Plus formellement :

Définition 9 (MSO-définissabilité). On dit qu'un langage d'arbre $L \subseteq T_\Sigma^k$ est MSO-définissable si et seulement si il existe une formule close ϕ de MSO tel que :

$$t \in L \text{ ssi } t \models \phi$$

Voyons maintenant comment définir des requêtes par des formules MSO.

2.3.2 Requêtes par formules

Une formule de FO (resp. MSO) à n variables libres du premier ordre peut définir une requête n-aire. Formellement, étant donnée une formule $\phi(x_1, \ldots, x_n)$ de FO(resp. MSO) à n variables libres du premier ordre, la requête n-aire $\texttt{query}_\phi$ est définie par :

$$\forall t \in T_\Sigma^k, \texttt{query}_\phi(t) = \{(\pi_1, \ldots, \pi_n) \in \texttt{dom}(t)^n \mid t \models \phi(\pi_1, \ldots, \pi_n)\}$$

La requête $\texttt{query}_\phi$ est alors dite FO(resp. MSO)-définissable.

Par exemple, si $\phi(x) = \exists y\ \texttt{child}(x, y)$ alors la requête $\texttt{query}_{\phi(x)}$ définit la requête monadique qui sélectionne tous les noeuds ayant un père.

2.4 Requêtes par automates

Nous définissons la notion d'automates d'arbres dans la section 2.4.1, et nous montrons comment ils peuvent être utilisés comme formalisme de requêtes n-aires dans la section 2.4.2.

2.4.1 Automates d'arbres

Rappelons brièvement la définition et quelques propriétés des automates d'arbres définis dans [9].

Définition 10 (Automates d'arbres). Un *automate d'arbres d'arité bornée k* (ou *automate d'arbres* lorsqu'il n'y aura pas d'ambiguïté) sur les arbres d'arité bornée k, et un quadruplet $\mathcal{A} = (\Sigma, \texttt{states}(\mathcal{A}), \texttt{finals}(\mathcal{A}), \texttt{rules}(\mathcal{A}))$ où :
- Σ est un alphabet fini
- $\texttt{states}(\mathcal{A})$ est un ensemble fini d'états
- $\texttt{finals}(\mathcal{A})$ est un ensemble d'états finaux tel que $F \subseteq Q$
- $\texttt{rules}(\mathcal{A})$ est un ensemble fini de règles de la forme :

$$\begin{array}{rcll} a & \rightarrow & p & \text{où } a \in \Sigma, p \in \texttt{states}(aaa) \\ f(p_1, \ldots, p_n) & \rightarrow & p & \text{où } f \in \Sigma, n \leq k, p, p_1 \ldots, p_n \in \texttt{states}(\mathcal{A}) \end{array}$$

La *taille* $|\mathcal{A}|$ de $\mathcal{A}$ est définie par $|\Sigma| + |\texttt{states}(\mathcal{A})| + |\texttt{finals}(\mathcal{A})| + |\texttt{rules}(\mathcal{A})|$.

Remarquons que la borne k sur l'arité intervient dans l'arité des règles de la forme $f(p_1, \ldots, p_n) \rightarrow p$. Définissons maintenant le fonctionnement d'un automate sur un arbre.

Définition 11 (run). Étant donné un automate d'arbre $\mathcal{A}$ sur les arbres d'arité bornée k sur Σ, et un arbre $t \in T_\Sigma^k$, un *run* $r_\mathcal{A}^t$ sur t, est un arbre sur l'alphabet $\texttt{states}(\mathcal{A})$ tel que $\texttt{dom}(r_\mathcal{A}^t) = \texttt{dom}(t)$, et tel que pour tout noeud $\pi \in \texttt{dom}(t)$:
- si π est une feuille étiquetée par a et $r_\mathcal{A}^t(\pi) = p$, alors $(a \rightarrow p) \in \texttt{rules}(\mathcal{A})$
- si π est un noeud d'arité n étiqueté par f, $r_\mathcal{A}^t(\pi) = p$, et pour tout i tel que $1 \leq i \leq n$, il existe $g \in \Sigma$ tel que $r_\mathcal{A}^t(\pi i) = p_i$, alors $(f(p_1, \ldots, p_n) \rightarrow p) \in \texttt{rules}(\mathcal{A})$

Comme dans le cas des automates de mots, on peut maintenant définir la notion de reconnaissabilité et de régularité.

Définition 12 (reconnaissance, régularité). Étant donné un automate d'arbres $\mathcal{A}$, on dit qu'un arbre $t \in T_\Sigma$ est *accepté* (ou *reconnu*) par $\mathcal{A}$ s'il existe un run $r_\mathcal{A}^t$ sur t tel que $r_\mathcal{A}^t(\epsilon) \in \texttt{finals}(\mathcal{A})$. On dit aussi que le *run est acceptant*. L'ensemble des arbres reconnus par $\mathcal{A}$ est appelé *langage reconnu par* $\mathcal{A}$ et est noté $\mathcal{L}(\mathcal{A})$. On dit qu'un langage d'arbres L est *régulier* s'il existe un automate d'arbres $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = L$.

Proposition 2 (opérations Booléennes [9]). *La classe des langages d'arbres réguliers est close par intersection, union et complémentaire.*

D'ailleurs, ces opérations de clôture sont effectives, on peut construire un automate qui reconnaît l'intersection et l'union de deux langages réguliers, et le complémentaire d'un langage régulier.

Définition 13 (déterminisme). Un automate d'arbres $\mathcal{A}$ est dit *déterministe* si et seulement si pour tout membre gauche d'une règle de l'automate, il n'existe qu'un seul membre droit :

$$\begin{array}{ll} \forall a \in \Sigma & |\{a \rightarrow p \in \texttt{rules}(\mathcal{A})\}| \leq 1 \\ \forall f \in \Sigma, \forall p_1, \ldots, p_n \in \texttt{states}(\mathcal{A}) & |\{f(p_1, \ldots, p_n) \rightarrow p \in \texttt{rules}(\mathcal{A})\}| \leq 1 \end{array}$$

Comme dans le cas des automates de mots, pour tout automate d'arbre, il existe un automate d'arbre déterministe qui reconnaît le même langage, mais dont la taille peut être exponentielle en la taille de l'automate de départ [9]. Enfin, un automate est dit *complet* si et seulement si il existe un run de cet automate sur tout arbre de T_Σ^k. On peut toujours compléter un automate pour qu'il devienne complet, tout en reconnaissant le même langage.

Énonçons maintenant un théorème fondateur dans le contexte des arbres :

Théorème 2 (J.W. Thatcher et J.B. Wright [19]). *Un langage d'arbre est MSO-définissable si et seulement si il est régulier.*

En particulier, à toute formule MSO correspond un automate d'arbre équivalent (en terme de langage), mais la taille de l'automate est non élémentaire en la taille de la formule. Nous ne donnons pas la construction de l'automate, mais nous pouvons dire que c'est la quantification ensembliste combinée à la négation qui fait exploser la taille de l'automate.

Donnons tout de même la réciproque, car elle nous servira pour la preuve de MSO-Complétude du *chapitre 4*. Étant donnée un automate d'arbre $\mathcal{A}$, il s'agit de trouver une formule qui reconnaît le même langage. Une solution est de simuler un run de l'automate sur un arbre t. L'étiquetage de l'arbre par des états de l'automate correspond à partionner le domaine $\texttt{dom}(t)$ en sous-ensemble X_p pour tout $p \in \texttt{states}(\mathcal{A})$. Ensuite, on simule naturellement le fonctionnement des règles de l'automate et on vérifie que la racine de l'arbre est bien étiquetée par un état final. Supposons que $\texttt{states}(\mathcal{A}) = \{p_1, \ldots, p_n\}$, la formule s'écrit alors :

$$
\begin{aligned}
\phi \equiv \quad & \exists X_{p_1} \ldots X_{p_n} \\
& \forall x \ \textstyle\bigvee_i (x \in X_{p_i}) \wedge \bigwedge_i \forall x (x \in X_{p_i} \rightarrow \bigwedge_{j \neq i} x \notin X_{p_j}) \\
& \wedge \exists x \ \texttt{root}(x) \wedge \textstyle\bigvee_{p \in \texttt{finals}(\mathcal{A})} x \in X_p \\
& \wedge \forall x \bigwedge_{\substack{f \in \Sigma \\ p \in \texttt{states}(\mathcal{A})}} ((\texttt{label}_f(x) \wedge x \in X_p) \rightarrow \\
& \qquad \Big(\bigvee_{\substack{f(p_1, \ldots, p_n) \rightarrow p \\ \in \texttt{rules}(\mathcal{A})}} \bigwedge_{i=1}^{n} \exists y \ S_i(x, y) \wedge y \in Y_{p_i} \Big))
\end{aligned}
$$

La deuxième ligne vérifie que $(X_{p_1}, \ldots, X_{p_n})$ forme une partition du domaine, la troisième ligne vérifie que la racine de l'arbre est étiquetée par un état final, et le reste simule le fonctionnement des règles.

2.4.2 Requêtes par automates

Voyons maintenant comment on peut utiliser les automates d'arbres pour spécifier des requêtes.

Définition 14 (Requêtes par automates [17]). Étant donné un automate d'arbres $\mathcal{A}$ d'arité k, un entier n, et un ensemble $S \subseteq \texttt{states}(\mathcal{A})^n$ appelé *ensemble de sélection*, la *requête* $\texttt{query}_{\mathcal{A},S}$ *par l'automate* $\mathcal{A}$ *et l'ensemble de sélection* S est définie par :

$$
\forall t \in T_\Sigma^k \quad \texttt{query}_{\mathcal{A},S}(t) = \{(\pi_1, \ldots, \pi_n) \mid \begin{array}{l} \text{il existe un run acceptant } r \text{ sur } t \\ \text{tel que } (r(\pi_1), \ldots, r(\pi_n)) \in S\} \end{array}
$$

L'exemple de la figure 2.2 représente la requête qui sélectionne, dans les arbres binaires sur l'alphabet $\{a, b, f\}$, toutes les paires de feuilles soeurs étiquetées par (a, b). La partie gauche représente l'automate avec l'ensemble de sélection S, et la partie droite représente les deux runs acceptant sur l'arbre $f(f(a, b), f(a, b))$.

On peut prouver qu'on ne peut définir cette requête comme une requête par automate qu'avec un automate non déterministe (voir [17]). En effet, chaque run acceptant ne doit

$\texttt{states}(\mathcal{A}) = \{p_a, p_b, p, p_s\}$
$\texttt{finals}(\mathcal{A}) = \{p_s\}$
$S = \{(p_a, p_b)\}$
$\texttt{rules}(\mathcal{A}) =$

$a \to p_a$	$a \to p$
$b \to p_b$	$b \to p$
$f(p,p) \to p$	$f(p_a,p_b) \to p_s$
$f(p_s,p) \to p_s$	$f(p,p_s) \to p_s$

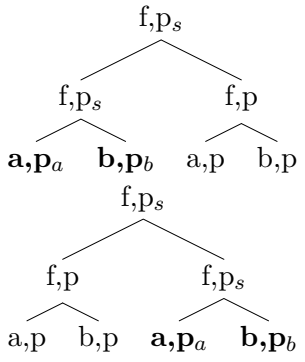


FIG. 2.2 – Requête binaire par automate et les deux runs associés sur l'arbre $f(f(a,b), f(a,b))$. Les noeuds sélectionnés sont les paires de feuilles soeurs étiquetées par (a,b), elles sont marquées en gras.

étiqueter qu'au plus deux feuilles soeurs par des états de sélection, car sinon, on sélectionnerait forcément deux noeuds non frères. L'état p_s indique qu'une sélection a été faite, et l'état p indique le contraire. Comme on a pas de règle de la forme $f(p_s, p_s) \to \_$, jamais on ne pourra sélectionner plus de deux couples dans un run acceptant. L'automate doit donc choisir le couple qui sera sélectionné par un run acceptant, d'où le non-déterminisme. Ainsi, l'équivalence entre deux automates n'implique pas forcément leur équivalence en tant que formalisme de spécification de requêtes.

Ayant maintenant défini des automates pour effectuer des requêtes, on peut définir la notion de requête régulière.

Définition 15 (requête régulière). Une requête q est dite *régulière* si et seulement si il existe un automate d'arbres $\mathcal{A}$, et un ensemble de sélection S tel que $q = \texttt{query}_{\mathcal{A},S}$.

Comme dans le cas des langages d'arbres, la notion de régularité correspond exactement avec les requêtes MSO-définissables.

Théorème 3 ([19, 17]). *Une requête est régulière si et seulement si elle est MSO-définissable.*

La preuve de ce théorème est similaire à celle du théorème 2, dans le sens direct il suffit de simuler un run acceptant de l'automate, comme on l'a fait dans la section **??** par la formule ϕ, en vérifiant que chaque noeud sélectionné est étiquetté par une composante d'un n-uplet de sélection de S, la formule MSO est alors $\phi'(x_1, \ldots, x_n) \equiv \phi \wedge \bigvee_{(p_1,\ldots,p_n)\in S} \bigwedge_{i=1}^{n} x_i \in X_{p_i}$, et la requête équivalente est $\texttt{query}_{\phi(x_1,\ldots,x_n}$.

La réciproque est donnée dans [17].

De plus, la classe des requêtes régulières est close par opérations Booléennes :

Théorème 4 (Clôture par opérations booléennes [17]). *L'union et l'intersection de deux requêtes n-aires régulières est une requête n-aire régulière.*
Le complémentaire d'une requête n-aire régulière est une requête n-aire régulière.
Le produit cartésien de requêtes régulières est une requête régulière
Le complémentaire d'une requête régulière est une requête régulière.

Chapitre 3

Langage de composition de requêtes monadiques

Rappelons que le sujet du stage était de trouver un formalisme pour composer les requêtes monadiques, afin d'exploiter les formalismes de requêtes monadiques existants, d'étudier les liens entre les requêtes monadiques et les requêtes n-aires, capturant toutes les requêtes régulières, admettant un algorithme de réponse efficace, ayant de bonnes propriétés d'apprenabilité, et permettant à un utilisateur de spécifier facilement ses requêtes.

Avant de présenter le formalisme de composition qui a été retenu, dans la section *Langage de composition*, une section *Introduction* décrit les différentes approches du problème posé.

3.1 Introduction

3.1.1 Une première proposition naïve : le produit cartésien de requêtes monadiques régulières

Une solution naïve pour composer des requêtes monadiques régulières, est de les composer de manière indépendante. Une première requête monadique sélectionne les éléments de la première composante, une deuxième sélectionne les éléments de la deuxième composante, etc... le résultat final étant le produit cartésien de tous les ensembles obtenus. Par exemple, la requête ternaire q de la figure 3.1 sélectionne tous les triplets (π_1, π_2, π_3) tel que π_1 est une feuille, π_2 est étiqueté par a et π_3 n'est pas une feuille. Cette requête peut se définir comme le produit cartésien des trois requêtes monadiques régulières q_1, q_2 et q_3 :

$$\forall t \in T_\Sigma^k, \quad q(t) = q_1(t) \times q_2(t) \times q_3(t)$$

Un produit cartésien de requêtes est encore noté $q_1 \times q_2 \times q_3$, on a donc $q = q_1 \times q_2 \times q_3$.

Ce type de requêtes n-aires (produit cartésien de requêtes monadiques régulières) admet un algorithme de réponse efficace, en la somme des temps pour répondre aux requêtes monadiques. Malheureusement il existe de nombreuses requêtes régulières qui ne peuvent s'écrire comme produit cartésien de requêtes monadiques, et même comme union finie de produits cartésiens de requêtes monadiques. Une preuve de ce fait est donnée dans [17], basée sur les formules MSO, nous en donnons ici une preuve directe.

Requête ternaire	Requêtes monadiques
$\phi(x,y,z) \equiv \texttt{leaf}(x) \wedge \texttt{label}_a(y) \wedge \neg\texttt{leaf}(z)$ $q = \texttt{query}_{\phi,(x,y,z)}$	$q_1 = \texttt{query}_{(\texttt{leaf}(x)),x}$ $q_2 = \texttt{query}_{(\texttt{label}_a(x)),x}$ $q_3 = \texttt{query}_{(\neg\texttt{leaf}(x)),x}$

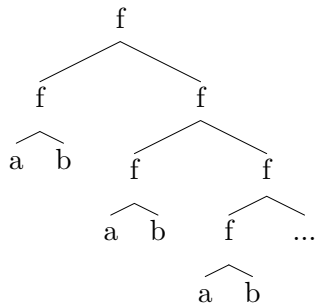
$$\forall t \in T^k_\Sigma \qquad q(t) = q_1(t) \times q_2(t) \times q_3(t)$$

FIG. 3.1 – Requête ternaire régulière close par produit cartésien sur des arbres binaires sur Σ

Lemme 1. *La classe $\mathcal{C}_\times$ des requêtes n-aires de la forme $\bigcup_{1\leq i\leq m} q_1^i \times \ldots q_n^i$ où les q_i^j sont des requêtes monadiques régulières est une sous-classe stricte de la classe des requêtes n-aires régulières.*

Démonstration. Par le théorème 4, toute requête de $\mathcal{C}_\times$ est régulière.

Pour prouver que c'est une sous-classe stricte, montrons que la requête définie figure 2.2, où l'on veut sélectionner toutes les paires de feuilles soeurs étiquetées par (a, b), n'appartient pas à $C_\times$. Supposons que cette requête, appelons-la q, puisse s'écrire comme une union finie de produits cartésiens, i.e. qu'il existe un entier n, tel que pour tout arbre $t \in T^2_\Sigma$, on ait $q(t) = \bigcup_{1\leq i\leq n} q_{1i}(t) \times q_{2i}(t)$. Prenons alors un arbre t ayant $n+1$ paires de feuilles sélectionnables étiquetées par (a, b) :



L'ensemble P des feuilles sélectionnées est donc $P = \{(2^i 11, 2^i 12) \mid 0 \leq i \leq n\}$, et il peut s'écrire de la forme $P = \bigcup_{1\leq i\leq n} P_i$ avec $P_i = q_{1i}(t) \times q_{2i}(t)$. Pour des raisons de cardinalité (lemme des tiroirs), il existe nécessairement un entier i tel que $|P_i| \leq 2$, ce qui implique $|q_{1i}(t)| \geq 2$ et $q_{2i}(t) \neq \emptyset$. Deux paires de feuilles ayant le même noeud en deuxième composante peuvent alors être sélectionnées, ce qui est absurde. □

Remarque 1. La classe $\mathcal{C}_X$ correspond à la classe des *requêtes non ambiguës*, ie les requêtes exprimables par *automates non ambiguës*, ie les automates pour lesquels il existe au plus un run acceptant par arbre [17].

3.1.2 Seconde proposition : produits cartésiens, union et complémentaire

Au formalisme précédent, ajoutons la possibilité de prendre le complémentaire d'une requête. Définissons formellement la classe $\mathcal{C}$ des requêtes obtenues. Soit $\mathcal{C}_n$ la sous-classe des requête n-aires de $\mathcal{C}$, la famille $(\mathcal{C}_n)_{n\geq 1}$ est définie par :

- pour toute requête monadique régulière q, $q \in \mathcal{C}_1$;
- pour tout $m \geq 1$,pour tout $q_1, \ldots, q_m \in \mathcal{C}_n$, $(\bigcup_{1\leq i\leq m} q_i) \in \mathcal{C}_n$
- pour tout $q \in \mathcal{C}_n$, $q^c \in \mathcal{C}_n$
- pour tout $q_1 \in \mathcal{C}_{i_1}, \ldots, q_m \in \mathcal{C}_{i_m}$, $(q_1 \times \ldots \times q_m) \in \mathcal{C}_n$ où $n = \Sigma_{j=1}^m i_j$

et $\mathcal{C} = \bigcup_{n\geq 1} \mathcal{C}_n$.

On peut remarquer qu'ajouter la possibilité de faire des intersections serait redondant, l'intersection étant exprimable à partir de l'union et du complémentaire. On a malheureusement le résultat suivant :

Lemme 2. *$\mathcal{C} = \mathcal{C}_\times$, et par conséquent, $\mathcal{C}$ est une sous-classe stricte de la classe des requêtes régulières.*

Démonstration. L'inclusion $\supseteq$ est immédiate. L'autre inclusion se montre en remarquant que pour tout $q_1, q_2 \in \mathcal{C}_n$, et pour tout $q \in \mathcal{C}_m$, $(q_1 \cup q_2) \times q = (q_1 \times q) \cup (q_2 \times q)$ (*). On peut donc faire «remonter» dans le terme les opérateurs d'unions. De même, pour tout $q_1, q_2 \in \mathcal{C}$, $(q_1 \times q_2)^c = (q_1^c \times q_2^c) \cup (q_1^c \times q_2) \cup (q_1 \times q_2^c)$ (**). On peut donc faire «descendre » dans le terme les opérateurs de complémentaire. On termine la preuve en montrant par induction sur la structure des requêtes que toute requête s'écrit comme une union finie de produits cartésiens de requêtes monadiques régulières.

- c'est immédiat pour le cas monadique
- si $q = (\bigcup_{1\leq i\leq n} q_i)$ est une requête p-aire, alors par hypothèse, chaque q_i s'écrit de la forme $\bigcup_{1\leq j\leq m_i} q^j_{1,i} \times \ldots \times q^j_{p,i}$, donc $q = \bigcup_{1\leq i\leq n} \bigcup_{1\leq j\leq m_i} q^j_{1,i} \times \ldots \times q^j_{p,i}$
  donc $q = \bigcup_{\substack{1\leq i\leq n \\ 1\leq j\leq m_i}} q^j_{1,i} \times \ldots \times q^j_{p,i}$
- si $q = q_1 \times \ldots \times q_n$, alors par hypothèse, chaque q_i s'écrit de la forme $\bigcup_{1\leq j\leq m_i} q^j_{1,i} \times \ldots \times q^j_{p,i}$, et par (*) , en distribuant l'union, on se ramène à une union finie de produits cartésiens
- si $q = q_1^c$, alors par hypothèse, q_1 est de la forme $\bigcup_{1\leq j\leq m} q_1^j \times \ldots \times q_p^j$,
  donc $q = (\bigcup_{1\leq j\leq m} q_1^j \times \ldots \times q_p^j)^c = (\bigcap_{1\leq j\leq m} (q_1^j \times \ldots \times q_p^j)^c)$, alors par (**),le complémentaire de chaque produit cartésien se ramène à une union finie de produits cartésiens de requêtes régulières monadiques. La requête q est alors de la forme $(\bigcap_{1\leq j\leq m} \bigcup_{i\in I_j} (q^j_{1i} \times \ldots \times q^j_{pi}))$ qui se ramène par les lois de De Morgan à une expression de la forme $(\bigcup_{j\in J} \bigcap_{i\in I_j} (q^j_{1i} \times \ldots \times q^j_{pi}))$, où J et I_j sont finis pour tout j. Enfin, comme l'intersection de produits cartésiens est égale au produit cartésien des intersections, q est de la forme $(\bigcup_{j\in J} (\bigcap_{i\in I_j} q^j_{1i}) \times \ldots \times (\bigcap_{i\in I_j} q^j_{pi})))$. Enfin, on peut conclure en disant que toute intersection de requêtes monadiques régulières est régulière.

□

Remarque 2. La classe des requêtes non ambiguës est donc close par complémentaire, par intersection, par union, et par produit cartésien.

3.1.3 Troisième proposition : requêtes paramétrées

Le lemme 2 montre que pour espérer obtenir l'expressivité de MSO, il est nécessaire que le formalisme recherché explicite un lien entre les requêtes qui sont composées. L'idée qui vient alors est de composer des requêtes *paramétrées* par un certain noeud, i.e. dont le résultat dépend d'un certain noeud. Intuitivement, pour sélectionner la i-ème composante d'un n-uplet, on se souvient de la $(i-1)$-ème composante sélectionnée. Nous allons montrer que ce

formalisme n'est toujours pas assez puissant pour obtenir l'expressivité de MSO, mais nous l'étendrons dans la prochaine sous-section, afin d'obtenir l'expressivité de MSO. Formellement

Définition 16 (Requête paramétrée). Une *requête paramétrée* q est une fonction qui à tout arbre $t \in T_\Sigma^k$ et tout noeud $\pi \in \texttt{dom}(t)$ associe un sous-ensemble de $\texttt{dom}(t)$.

$$\forall t \in T_\Sigma^k \; \forall \pi \in \texttt{dom}(t) \qquad q(t,\pi) \subseteq \texttt{dom}(t)$$

Étant données une requête monadique q_1 (qui peut être vu comme une requête paramétrée dont le noeud argument est inutile), et $n-1$ requêtes paramétrées $q_2, \ldots, q_n$, la *composition* $q_1.q_2.q_3....q_n$ de ces requêtes est la requête n-aire définie par :

$$(q_1.q_2...q_n)(t) = \{(\pi_1, \ldots, \pi_n) \mid \pi_1 \in q_1(t), \pi_2 \in q_2(t,\pi_1), \ldots, \pi_n \in q_n(t,\pi_{n-1})\}$$

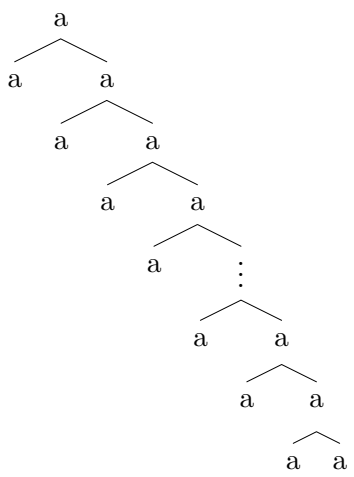
De plus, nous ajoutons à la composition de requêtes paramétrées une notion de mémoire finie représentée par une fonction (*historique*) μ associant à chaque rang de composante déjà sélectionnée un état p d'un ensemble fini d'états Q (formellement une requête paramétrée sera alors une fonction de $T_\Sigma^k \times \texttt{dom}(T_\Sigma^k) \times (\{1,\ldots,i\} \to Q)$ dans $2^{\texttt{dom}(T_\Sigma^k)}$ où i est le nombre de composantes déjà sélectionnées). Par exemple, la requête ternaire r telle que $r(t) = \{(\pi i, \pi(i+1), \pi(i+2)) \mid \texttt{label}(\pi i) = a \wedge \texttt{label}(\pi(i+2)) = a \;\vee\; \texttt{label}(\pi i) = b \wedge \texttt{label}(\pi(i+2)) = b\}$ n'est pas exprimable comme composition de requêtes paramétrées (car la dernière composante sélectionnée dépend uniquement de la première sélectionnée), mais l'est comme composition de requêtes paramétrées avec mémoire : il suffit de garder en mémoire l'étiquette de la première composante sélectionnée. Cependant, ce formalisme ne capture pas toutes les requêtes régulières :

Lemme 3. *La composition de requêtes paramétrées avec mémoire ne capture pas toutes les requêtes régulières*

Démonstration. Considérons la requête régulière ternaire q telle que $q(t) = (\pi_1, \pi_2, \pi_3)$ ssi π_3 est le plus petit ancêtre commun de π_1 et π_2 dans t. Cette requête est régulière car elle peut être exprimée par la formule MSO à trois variables libres :

$$\phi(x,y,z) \;\equiv\; y \lhd^* z \wedge x \lhd^* z \wedge \forall z'(y \lhd^* z' \wedge x \lhd^* z') \to z \lhd^* z'$$

Supposons qu'elle soit exprimable comme une composition de requêtes avec une mémoire finie d'états Q. Soit $n = |Q|$, soit $t \in T_\Sigma^k$ de la forme :



tel que $h(t) > n^2 + 1$. Soit l'ensemble $P_1 = \{2^k 1 \mid 0 \leq k \leq n^2\} \subseteq \texttt{dom}(t)$ des feuilles filles gauches d'un noeud. Supposons maintenant que la première requête sélectionne un noeud de P_1 qui donne un historique $\mu_1 = 1 \mapsto m_1$ où $m_1 \in Q$. Comme $|P_1| > n^2$ il existe deux noeuds de P_1 qui conduisent au même historique. Supposons maintenant que la seconde requête, qui dépend de la première composante sélectionnée, et de l'historique μ_1, sélectionne le noeud $2^{depth(t)}$ et conduise à l'historique $\mu_2 = \mu_1[2 \leftarrow m_2]$ où $m_2 \in Q$. Il y a au plus n^2 historiques μ_2 possibles pour sélectionner les deux premières composantes (la première dans P_1 et la seconde égale à $2^{depth(t)}$). Par conséquent il existe deux paires de noeuds différentes $(2^{k_1}1, 2^{h(t)})$ et $(2^{k_2}1, 2^{h(t)})$ $(k_1 \neq k_2)$ sélectionnées par les deux premières requêtes, et conduisant au même historique. La dernière requête sélectionne donc le même ancêtre commun pour ces deux paires, puisque les deuxièmes composantes sont les mêmes, égales à $2^{h(t)}$, ce qui est absurde, puisque $\texttt{ppac}(2^{k_1}1, 2^{h(t)}) = 2^{k_1} \neq 2^{k_2} = \texttt{ppac}(2^{k_2}1, 2^{h(t)})$, où $\texttt{ppac}(.,.)$ représente le plus petit ancêtre commun. □

3.1.4 Un premier formalisme MSO-complet

Nous étendons le formalisme de composition développé dans la section 3.1.3 avec la possibilité de faire des unions, des intersections, des projections, et des permutations des composantes sélectionnées. Ce formalisme est MSO-complet. On l'appelera *composition de requête paramétrées*. Le formalisme développé dans la section 3.1.3 ne sera alors qu'un cas particulier de ce nouveau formalisme.

Nous donnons brièvement la syntaxe des formules de composition, et la sémantique associée.

Étant donnés un ensemble Q de requêtes paramétrées, et un ensemble $\texttt{Var}$ de variables, les *formules de composition* sur les requêtes paramétrées Q sont définies par la syntaxe suivante :

$$
\begin{array}{lll}
\phi \quad ::= & & \\
 & q(x) & q \in Q \text{ et } x \in \texttt{Var} \\
\mid & q(x).\phi & \textit{composition} \\
\mid & \phi \wedge \phi & \textit{conjonction} \\
\mid & \phi \vee \phi & \textit{disjonction}
\end{array}
$$

L'ensemble $\mathrm{VL}(\phi)$ des variables libres d'une formule de composition ϕ est défini comme

l'ensemble des variables apparaissant dans ϕ. Étant donné un arbre $t \in T_\Sigma^k$, et une formule de composition ϕ, une *valuation* des variables de ϕ dans $\texttt{dom}(t)$ est une application de $\mathrm{VL}(\phi)$ dans $\texttt{dom}(t)$. Étant donnés une formule de composition ϕ, un arbre $t \in T_\Sigma^k$, une valuation ν des variables de ϕ, et un noeud $\pi \in \texttt{dom}(t)$ dit *noeud de relativisation*, la *relation de satisfiabilité* $t, \nu, \pi \models \phi$ d'une formule ϕ est inductivement définie par :

$$\begin{array}{lll} t,\nu,\pi \models q(x) & \text{ssi} & \nu(x) \in q(t,\pi) \\ t,\nu,\pi \models q(x).\phi & \text{ssi} & \nu(x) \in q(t,\pi) \text{ et } t,\nu,\nu(x) \models \phi \\ t,\nu,\pi \models \phi_1 \wedge \phi_2 & \text{ssi} & t,\nu,\pi \models \phi_1 \text{ et } t,\nu,\pi \models \phi_2 \\ t,\nu,\pi \models \phi_1 \vee \phi_2 & \text{ssi} & t,\nu,\pi \models \phi_1 \text{ ou } t,\nu,\pi \models \phi_2 \end{array}$$

Enfin, étant donnée une formule de composition ϕ, et un n-uplet $\overrightarrow{x}$ de variables de $\mathrm{VL}(\phi)$, la requêtes $\texttt{query}_{\phi,\overrightarrow{x}}$ est définie par :

$$\forall t \in T_\Sigma^k \qquad \texttt{query}_{\phi,\overrightarrow{x}}(t) = \{\nu(\overrightarrow{x}) \mid t,\nu,\epsilon \models \phi\}$$

Voyons tout de suite un exemple :

Exemple 3. Nous allons définir la requête binaire q dans les arbres binaire, telle qu'un couple de noeuds est solution de la requête si et seulement si ils sont des feuilles, ils sont frères, le premier est à gauche du deuxième, il est étiqueté par a, et le deuxième est étiqueté par b. Une solution serait de sélectionner la première composante, c'est-à-dire une feuille étiquetée par a, et, partant de ce noeud, sélectionner son frère droit si et seulement si c'est une feuille étiquetée par b. La requête s'écrit donc $\texttt{query}_{q_1(x).q_2(y),(x,y)}$, où q_1 et q_2 sont définies par :

$$\begin{array}{lll} q_1(t,\pi) = \texttt{query}_{\psi_1^\pi(x)}(t) & \text{où} & \psi_1^\pi(x) \equiv \texttt{leaf}(x) \wedge \texttt{label}_a(x) \\ q_2(t,\pi) = \texttt{query}_{\psi_2^\pi(x)}(t) & \text{où} & \psi_2^\pi(x) \equiv \exists y\ S_1(y,\pi) \wedge S_2(y,x) \wedge \texttt{leaf}(x) \wedge \texttt{label}_b(x) \end{array}$$

Ajoutons que la notation $S_1(y,\pi)$ dénote la formule MSO $\gamma^\pi(y)$ suivante : si on note $\pi = i_1 \dots i_m$, $(\forall j\ i_j \in \{1,2\})$ alors $\gamma^\pi(y) \equiv \exists r\ \texttt{root}(r) \wedge \exists x_1 \dots \exists x_m\ S_{i_1}(r,x_1) \wedge S_{i_2}(x_1,x_2) \wedge \dots \wedge S_{i_m}(x_{m-1},x_m) \wedge S_1(y,x_m)$.

Nous verrons plus loin que ce formalisme capture toutes les requêtes régulières, et donc toutes les requêtes MSO-définissables. Cependant, nous nous sommes intéressés – c'est l'objet de la section suivante – à une restriction de ce formalisme, sur des requêtes paramétrées q ayant la *propriété de restriction aux sous-arbres*, i.e. :

$$\forall t \in T_\Sigma^k\ \forall \pi, \pi' \in \texttt{dom}(t) \qquad \pi' \in q(t,\pi) \text{ ssi } \exists \pi''\ \pi' = \pi\pi''\ \wedge \pi'' \in q(t|_\pi, \epsilon)$$

Intuitivement, cela signifie qu'il suffit simplement de restreindre le domaine d'interprétation au sous-arbre induit par le paramètre. Dès lors, les requêtes paramétrées peuvent être écrites dans un formalisme de requêtes monadiques (c'est pour cette raison qu'on parle de composition de requêtes monadiques et non de requêtes paramétrées). La restriction aux sous-arbres est alors fixée dans la relation de satisfiabilité.

Nous nous sommes intéressés à cette restriction, qui capture néanmoins toutes les requêtes régulières, car elle explicite le rôle joué par les ancêtres communs des composantes des n-uplets à extraire, et par là-même semble avoir de bonnes propriétés d'apprenabilité. Elle correspond à une vision naturelle de l'extraction d'information ; comme il a été dit dans l'introduction, la sélection d'une information commence par la sélection d'une zone qui la contient. De plus,

dans [18] est étudié un formalisme de requêtes (ETL) avec restriction dans les sous-arbres. Il est MSO-complet mais n'admet pas d'algorithme efficace pour répondre aux requêtes.

Nous allons prouver que la composition de requêtes paramétrées restreinte aux sous-arbres capture toutes les requêtes régulières, et donc, comme c'est un cas particulier de la composition de requêtes paramétrées, cette dernière capture aussi toutes les requêtes régulières.

3.2 Langage de composition

Nous proposons ici un langage de composition de requêtes monadiques, avec restriction de l'évaluation dans les sous-arbres. Intuitivement, une première requête sélectionne un ensemble de noeuds, et des sous-requêtes sont évaluées dans les sous-arbres induits par les noeuds trouvés. A cela s'ajoute la conjonction, la disjonction de requêtes, et la projection des n-uplets de noeuds sélectionnés.

Nous donnons la syntaxe et la sémantique des formules de composition, ainsi que deux algorithmes pour répondre aux requêtes. Nous prouvons dans le chapitre 4 que ce formalisme capture toutes les requêtes n-aires régulières dès lors que les requêtes monadiques sont régulières.

3.2.1 Syntaxe et Sémantique

Dans la suite, l'ensemble `Var` représente un ensemble dénombrables de variables $x, y, \ldots$.

Définition 17 (formules de composition). Étant donné un ensemble (possiblement infini) Q de requêtes monadiques, les *formules de composition* sur Q sont obtenues par la grammaire suivante :

$$
\begin{array}{lll}
\phi \quad ::= & & \\
 & q(x) & q \in Q \text{ et } x \text{ est une variable} \\
 \mid & q(x).\phi & \textit{composition} \\
 \mid & \phi \wedge \phi & \textit{conjonction} \\
 \mid & \phi \vee \phi & \textit{disjonction}
\end{array}
$$

L'ensemble des formules de composition sur Q est noté $\texttt{COMP}[Q]$. Sans ambiguïté sur les requêtes monadiques, on parlera simplement de *formules de composition* voire de *formules*. La *taille* $|.|$ d'une formule est définie inductivement par $|q| = 1$, $|q.\phi| = 1 + |\phi|$, $|\phi_1 \diamond \phi_2| = |\phi_1| + |\phi_2| + 1$ où $\diamond \in \{\vee, \wedge\}$.

Dans la suite Q désigne un ensemble de requêtes monadiques. Avant de définir la sémantique nous définissons une notion de variables libres :

Définition 18 (variables libres). Étant donnée une formule de composition $\phi \in \texttt{COMP}[Q]$, l'ensemble $\mathrm{VL}(\phi)$ des variables libres de ϕ est défini inductivement par :

$$
\begin{array}{lcl}
\mathrm{VL}(q(x)) & = & \{x\} \\
\mathrm{VL}(q(x).\phi) & = & \{x\} \cup \mathrm{VL}(\phi) \\
\mathrm{VL}(\phi_1 \wedge \phi_2) & = & \mathrm{VL}(\phi_1) \cup \mathrm{VL}(\phi_2) \\
\mathrm{VL}(\phi_1 \vee \phi_2) & = & \mathrm{VL}(\phi_1) \cup \mathrm{VL}(\phi_2)
\end{array}
$$

Définition 19 (relativisation d'un chemin). Étant donné un mot $\pi \in \mathbb{N}\neg^*$ et un mot π' tel qu'il existe π'' tel que $\pi' = \pi.\pi''$, la *relativisation* de π' par π est π''. La fonction partielle de *relativisation* par π, $\texttt{relat}_\pi$, est définie pour les mots dont π est préfixe, par :

$$\begin{array}{rcccl} \texttt{relat}_\pi & : & \mathbb{N}^* & \to & \mathbb{N}^* \\ & & \pi' & \mapsto & \begin{cases} \pi'' \text{ si il existe } \pi'' \text{ tel que } \pi' = \pi.\pi'' \\ \text{non défini sinon} \end{cases} \end{array}$$

Par exemple, $\texttt{relat}_{1221}(12213232) = 3232$.

Définition 20 (satisfiabilité). Étant donné un arbre $t \in T_\Sigma^k$, une formule de composition ϕ, un *noeud de relativisation* $\pi \in \texttt{dom}(t)$ et une *valuation* ν des variables libres de ϕ telle que $\forall x \in \mathrm{VL}(\phi), \pi \lhd^* \nu(x)$, la relation de *satisfiabilité* $t, \nu, \pi \models \phi$ est inductivement définie sur la structure de ϕ par :

$$\begin{array}{lll} t, \nu, \pi \models q(x) & \text{ssi} & \pi \lhd^* \nu(x) \text{ et } \texttt{relat}_\pi(\nu(x)) \in q(t|_\pi) \\ t, \nu, \pi \models q(x).\phi & \text{ssi} & t, \nu, \pi \models q(x) \text{ et } t, \nu|_{\mathrm{VL}(\phi)}, \nu(x) \models \phi \text{ et } \forall y \in \mathrm{VL}(\phi)\ \nu(x) \lhd^* \nu(y) \\ t, \nu, \pi \models \phi_1 \wedge \phi_2 & \text{ssi} & t, \nu, \pi|_{\mathrm{VL}(\phi_1)} \models \phi_1 \text{ et } t, \nu, \pi|_{\mathrm{VL}(\phi_2)} \models \phi_2 \\ t, \nu, \pi \models \phi_1 \vee \phi_2 & \text{ssi} & t, \nu, \pi|_{\mathrm{VL}(\phi_1)} \models \phi_1 \text{ ou } t, \nu, \pi|_{\mathrm{VL}(\phi_2)} \models \phi_2 \end{array}$$

Remarque 3. $\texttt{relat}_\pi(\nu(x))$ existe car nous demandons que $\pi \lhd^* \nu(x)$.
De même, dans la deuxième ligne, on a aussi $\pi \lhd^* \nu(x)$, car on demande que $t, \nu, \pi \models q(x)$.

Intuitivement, le noeud π de relativisation π indique que l'évaluation d'une formule ϕ peut se faire dans le sous-arbre en position π.

Avant de donner quelques exemples, définissons la notion de requête.

Définition 21 (requête de composition). Étant donnée une formule de composition $\phi \in \texttt{COMP}[Q]$ et un n-uplet $\overrightarrow{x} = (x_1, \ldots, x_n) \in \mathrm{VL}(\phi)^n$ de variables libres de ϕ, dit *n-uplet de sélection*, la requête de composition $\texttt{query}_{\phi,\overrightarrow{x}}$ représente la requête n-aire définie par :

$$\forall t \in T_\Sigma^k \quad \texttt{query}_{\phi,\overrightarrow{x}}(t) = \{\nu(\overrightarrow{x}) \mid \nu \ :\ \mathrm{VL}(\phi) \to \texttt{nodes}(t),\ t, \nu, \epsilon \models \phi\}$$

où $\nu(\overrightarrow{x}) = (\nu(x_1), \ldots, \nu(x_n))$.

Lorsqu'il n'y aura pas d'ambiguité, on notera indifféremment l'ensemble des requêtes de composition et l'ensemble des formules de composition sur un ensemble de requêtes monadiques Q, $\texttt{COMP}[Q]$.

Voyons tout de suite quelques exemples.

Exemple 4. Les exemples de requêtes sont des requêtes dans les arbres binaires. L'exemple de la figure 3.2 définit une requête de composition q' qui sélectionne toutes les paires de noeuds (π_1, π_2) tels que π_1 et π_2 soient des feuilles, π_1 est frère gauche de π_2, π_1 est étiqueté par a, et π_2 par b. L'idée est d'abord de sélectionner tous les noeuds, et, dans chaque sous-arbre induit par ces noeuds, sélectionner le premier fils de la racine, s'il est étiqueté par a et si c'est une feuille, ainsi que le deuxième fils de la racine, s'il est étiqueté par b et si c'est une feuille. Si le noeud dénoté par la variable x n'est pas l'ancêtre commun d'une paire de noeuds candidate, alors la composition échoue.

Pour illustrer le rôle de la disjonction, nous étendons la requête précédente de telle sorte qu'elle sélectionne aussi les couples de feuilles soeurs étiquetées par (b, a). La requête q' correspondante est donnée figure 3.3. L'idée est la même que précédemment, en commençant par une requête qui sélectionne tous les noeuds.

Soit les formules MSO suivantes :

$$\begin{aligned} \texttt{All}(x) &\equiv \texttt{True} \\ \psi_1(x) &\equiv \exists r\ \texttt{root}(r) \wedge S_1(r,x) \wedge \texttt{label}_a(x) \wedge \texttt{leaf}(x) \\ \psi_2(x) &\equiv \exists r\ \texttt{root}(r) \wedge S_2(r,x) \wedge \texttt{label}_b(x) \wedge \texttt{leaf}(x) \end{aligned}$$

et les requêtes monadiques associées :

$$\begin{aligned} q &= \texttt{query}_{\texttt{All}(x)} \\ q_1 &= \texttt{query}_{\psi_1(x)} \\ q_2 &= \texttt{query}_{\psi_2(x)} \end{aligned}$$

Enfin définissons $q' = \texttt{query}_{q(x).(q_1(y)\wedge q_2(z)),(y,z)}$.

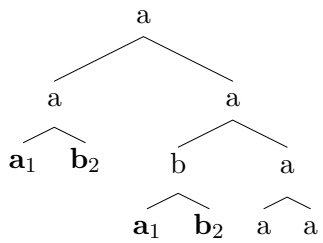


FIG. 3.2 – requête de composition binaire et les couples extraits dans l'arbre a(a(a,b),a(a(b,a),a(a,a))) (en gras, indexés par leur numéro de composante)

Donnons maintenant des algorithmes pour répondre aux requêtes :

3.2.2 Algorithmes de réponse aux requêtes

Nous donnons tout d'abord un algorithme en temps linéaire pour le problème du model-checking, qui sera utilisé pour définir un premier algorithme de réponse aux requêtes de composition.

Model-Checking

Le problème du model checking d'une formule de composition $\phi \in \texttt{COMP}[Q]$ par un arbre t, une valuation ν des variables de ϕ, et un noeud de relativisation $\pi \in \texttt{dom}(t)$ est défini par :

- **ENTRÉE :** une formule ϕ, un arbre $t \in T^k_\Sigma$, une valuation $\nu : \mathrm{VL}(\phi) \to \texttt{dom}(t)$ et un noeud $\pi \in \texttt{dom}(t)$
- **SORTIE :** 1 si $t, \nu, \pi \models \phi$, 0 sinon

Soit `monadique-checking` un algorithme de réponse aux requête monadiques de Q, l'algorithme naturel `model-checking` répond au problème du model-checking. Il est écrit en pseudo-ML.

Algorithm 1 Model Checking Problem

1: **Let** $\texttt{model-checking}(\phi, t, \nu, \pi) =$
2: **match** ϕ **with**
3: $\quad q(x) \rightarrow \texttt{monadique-checking}(q, t|_\pi, \texttt{relat}_\pi(\nu(x)))$
4: $\quad q(x).\phi \rightarrow \texttt{monadique-checking}(q, t|_\pi, \texttt{relat}_\pi(\nu(x))) \wedge \texttt{model-checking}(\phi, t, \nu, \nu(x)) \wedge \forall y \in \mathrm{VL}(\phi)\ \nu(x) \lhd^* \nu(y)$
5: $\quad \phi_1 \vee \phi_2 \rightarrow \texttt{model-checking}(\phi_1, t, \nu, \pi) \vee \texttt{model-checking}(\phi_2, t, \nu, \pi)$
6: $\quad \phi_1 \wedge \phi_2 \rightarrow \texttt{model-checking}(\phi_1, t, \nu, \pi) \wedge \texttt{model-checking}(\phi_2, t, \nu, \pi)$

Soit les formules MSO suivantes :

$$\begin{array}{lcl}
\texttt{All}(x) & \equiv & \texttt{True} \\
\psi_1(x) & \equiv & \exists r\ \texttt{root}(r) \wedge S_1(r,x) \wedge \texttt{label}_a(x) \wedge \texttt{leaf}(x) \\
\psi_2(x) & \equiv & \exists r\ \texttt{root}(r) \wedge S_2(r,x) \wedge \texttt{label}_b(x) \wedge \texttt{leaf}(x) \\
\psi_1^r(x) & \equiv & \exists r\ \texttt{root}(r) \wedge S_1(r,x) \wedge \texttt{label}_b(x) \wedge \texttt{leaf}(x) \\
\psi_2^r(x) & \equiv & \exists r\ \texttt{root}(r) \wedge S_2(r,x) \wedge \texttt{label}_a(x) \wedge \texttt{leaf}(x)
\end{array}$$

et les requêtes monadiques associées :

$$\begin{array}{l}
q = \texttt{query}_{\texttt{All}(x)} \\
q_1 = \texttt{query}_{\psi_1(x)} \\
q_2 = \texttt{query}_{\psi_2(x)} \\
q_1^r = \texttt{query}_{\psi_1^r(x)} \\
q_2^r = \texttt{query}_{\psi_2^r(x)}
\end{array}$$

$$\phi \equiv q(x).((q_1(y) \wedge q_2(z)) \vee (q_1^r(y) \wedge q_2^r(z)))$$

Enfin définissons $q' = \texttt{query}_{\phi,(y,z)}$.

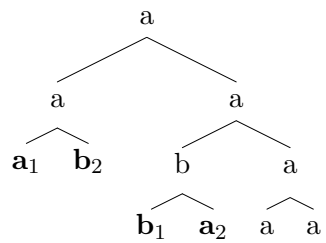


FIG. 3.3 – requête de composition avec disjonction et les couples extraits dans l'arbre a(a(a,b),a(a(b,a),a(a,a))) (en gras, indexés par leur numéro de composante)

Les opérations $\wedge$ et $\vee$ sont ici des opérations booléennes classiques sur $\{0, 1\}$.

Complexité Soit $C(q, t)$ la complexité en temps de l'algorithme du model-checking monadique. . La complexité en temps de `model-checking` est $\mathcal{O}(|\phi| max_{q \in \phi} C(q, t))$, où $q \in \phi$ signifie «q apparait dans ϕ ».

Model-checking dans le cas de requêtes monadiques par automates Quand les requêtes monadiques sont des requêtes par automates de la forme $\texttt{query}_{A,s}$, alors $C(\texttt{query}A, S, t) = |A||t|$ et la complexité du model-checking est alors $\mathcal{O}(|\phi||t| max|A|)$, où $max|A|$ est la taille maximale sur tous les automates apparaissant dans ϕ, i.e. tels que $\texttt{query}_{A,S}$ apparait dans ϕ.

Algorithme naïf pour répondre aux requêtes Un premier algorithme pour répondre aux requêtes de composition découle du problème du model-checking : étant donnés une formule de composition ϕ et un arbre de t, il suffit de générer toutes les valuations possibles des variables de ϕ dans $\texttt{dom}(t)$, il y en a $|t|^{|\mathrm{VL}(\phi)|}$, et d'appliquer l'algorithme du model-checking sur chaque valuation. Enfin, pour chaque valuation, il suffit de projeter suivant le n-uplet de sélection de la requête. Cet algorithme n'a néanmoins pas une bonne complexité en temps, car dans tous les cas, elle est de $\mathcal{O}((|\phi| max_{q \in \phi} C(q, t))|t|^{\mathrm{VL}(\phi)})$.

Deuxième algorithme

Nous définissons un deuxième algorithme, dont nous espérons une meilleure complexité en pratique, que l'algorithme naïf précédent, puisqu'il ne calcule pas toutes les valuations. Il est assez naturel, puisque qu'il calcule toutes les valuations satisfaisant la formule inductivement sur la structure de la formule. Par exemple, pour une formule de la forme $\phi = q(x).q'(y)$, dans un arbre t, l'algorithme évalue la première requête monadique, et, dans chaque sous-arbre de t induit par les noeuds trouvés, évalue la deuxième requête monadique. Le résultat est donc un ensemble de valuations tel que chaque valuation ν vérifie $t, \nu, \epsilon \models \phi$. L'algorithme est donc naturellement récursif, et doit gérer des ensembles de valuations. Par exemple, dans le cas de la conjonction $\phi = \phi_1 \wedge \phi_2$, l'algorithme est appliqué récursivement sur ϕ_1 et sur ϕ_2, retournant deux ensembles de valuations V_1 et V_2, ensembles qu'il faut alors combiner pour créer des valuations satisfaisant la conjonction. Plus formellement, étant donnée une valuation $\nu_1 \in V_1$, et une valuation $\nu_2 \in V_2$, si les deux valuations coïncident sur leur domaine commun, alors on peut les combiner en une valuation ν qui satisfera ϕ, telle que $\nu|_{\mathrm{VL}(\phi)-\mathrm{VL}(\phi_1)} = \nu_2|_{\mathrm{VL}(\phi)-\mathrm{VL}(\phi_1)}$ et $\nu|_{\mathrm{VL}(\phi)-\mathrm{VL}(\phi_2)} = \nu_1|_{\mathrm{VL}(\phi)-\mathrm{VL}(\phi_2)}$. Dans le cas d'une disjonction $\phi = \phi_1 \vee \phi_2$, on applique l'algorithme récursivement sur ϕ_1, ce qui donne un ensemble de valuations V_1, et sur ϕ_2, ce qui donne un ensemble de valuations ϕ_2. Il ne suffit pas ici de retourner $V_1 \cup V_2$, car le domaine de définition de chaque valuation retournée doit être égale à $\mathrm{VL}(\phi)$, il faut donc étendre le domaine de définition de chaque valuation de V_1 et de chaque valuation de V_2, à $\mathrm{VL}(\phi)$. Dans le cas d'une extension d'une valuation ν de V_1, il faudra affecter des valeurs quelconques du domaine de l'arbre aux variables de $\mathrm{VL}(\phi) - \mathrm{VL}(\phi_1)$.

Plus formellement, définissons ces opérations, qui seront naturellement étendues aux ensembles de valuations.

Définition 22 (Opérateurs sur les valuations). Nous appelons $\texttt{dom}(\nu)$ l'ensemble de définition de la valuation ν, et supposons que $\texttt{dom}(\nu) \neq \emptyset$ pour toute valuation ν. Une valuation ν telle que $\texttt{dom}(\nu) = \{x\}$ pour $x \in \texttt{Var}$ est notée $[x \mapsto \nu(x)]$.

Nous définissons la *concaténation* $\pi.\nu$ d'une valuation ν par un noeud π par :

$$\begin{array}{rcccl} \pi.\nu & : & V & \rightarrow & \mathtt{dom}(t) \\ & & x & \mapsto & \pi.\nu(x) \end{array}$$

Ensuite nous définissons la *jointure* de deux valuations ν et ν' telles que $\forall x \in \mathtt{dom}(\nu) \cap \mathtt{dom}(\nu'),\ \nu(x) = \nu'(x)$.

$$\begin{array}{rcccl} \nu \bowtie \nu' & : & V & \rightarrow & \mathtt{dom}(t) \\ & & x & \mapsto & \left\{ \begin{array}{ll} \nu(x) & \text{if} \quad x \in dom(\nu) \\ \nu'(x) & \text{if} \quad x \in dom(\nu') \end{array} \right. \end{array}$$

Enfin nous définissons l'*extension* d'une valuation ν dans par un domaine D et un ensemble de variables V tel que $V \cap \mathtt{dom}(\nu) = \emptyset$ par :

$$\mathrm{extend}_D^V(\nu) = \{\nu' \mid \nu'|_{\mathtt{dom}(\nu)} = \nu \ \wedge\ \nu'|_V \ :\ V \rightarrow D\}$$

Ces opérateurs sont naturellement étendus aux ensembles de valuations.
En particulier, $\nu \bowtie \emptyset = \emptyset$

Étant donnés une formule de composition ϕ et un arbre $t \in T_\Sigma^k$, nous donnons un algorithme récursif auxiliaire `aux` écrit en pseudo-ML, retournant un ensemble de valuations Val telles que pour tout $\nu \in Val$, $t, \nu, \epsilon \models \phi$. L'algorithme `aux` est paramétré par un algorithme M de réponse aux requêtes monadiques.

Algorithm 2 algorithme auxilaire

1: **Let** $\mathtt{aux}(\phi,t) =$
2: **Match** ϕ **with**
3: $\quad \mid q(x) \rightarrow \{[x \mapsto \pi] \mid \pi \in M(q,t)\}$
4: $\quad \mid q(x).\phi' \rightarrow \bigcup_{\pi \in M(q,t)} \{[x \mapsto \pi] \bowtie \pi.\nu \mid \nu \in \mathtt{aux}(\phi', t|_\pi) \ \wedge\ (x \notin \mathtt{dom}(\nu) \vee \nu(x) = \epsilon)\}$
5: $\quad \mid \phi_1 \wedge \phi_2 \rightarrow \{\nu_1 \bowtie \nu_2 \mid \nu_1 \in \mathtt{aux}(\phi_1,t) \wedge \nu_2 \in \mathtt{aux}(\phi_2,t) \wedge \forall x \in \mathtt{dom}(\nu_1) \cap \mathtt{dom}(\nu_2),\ \nu_1(x) = \nu_2(x)\}$
$\qquad\qquad \mathrm{extend}_{\mathtt{dom}(t)}^{\mathtt{Var}(\phi)-\mathtt{Var}(\phi_1)}(\mathtt{aux}(\phi_1,t))$
6: $\quad \mid \phi_1 \vee \phi_2 \rightarrow \qquad\qquad \cup$
$\qquad\qquad \mathrm{extend}_{\mathtt{dom}(t)}^{\mathtt{Var}(\phi)-\mathtt{Var}(\phi_2)}(\mathtt{aux}(\phi_2,t))$

Remarque 4. Dans le cas de la composition, la condition $x \notin \mathtt{dom}(\nu) \vee \nu(x) = \epsilon$ garantit que la jointure des valuations sera bien définie. De même pour la condition $\forall x \in \mathtt{dom}(\nu_1) \cap \mathtt{dom}(\nu_2),\ \nu_1(x) = \nu_2(x)$ dans le cas de la conjonction.

Nous pouvons maintenant définir un algorithme pour répondre aux requêtes :

Algorithm 3 Évaluation d'une requête

1: $\mathtt{query}_{\phi,\overrightarrow{s}}(t) = \{\nu(\overrightarrow{s}) \mid \nu \in \mathtt{aux}(\phi,t)\}$

Correction de l'algorithme Avant de prouver la correction de l'algorithme, prouvons d'abord une propriété liée au noeud de relativisation, qui intuitivement dit que l'évaluation d'une requête peut se faire dans tout sous-arbre induit par un noeud au dessus du noeud de relativisation dans l'arbre.

Lemme 4. *Pour toute formule de composition $\phi \in \texttt{COMP}[Q]$, pour tout arbre $t \in T_\Sigma^k$, pour tout noeud de relativisation $\pi \in \texttt{dom}(t)$, pour toute valuation ν des variables libres de ϕ dans $\texttt{dom}(t)$ telle que $\forall y \in VL(\phi),\ \pi \lhd^* \nu(y)$, et pour tout noeud $\pi' \in \texttt{dom}(t)$ tel que $\pi' \lhd^* \pi$, on a la propriété suivante :*

$$t, \nu, \pi \models \phi \quad \Leftrightarrow \quad t|_{\pi'}, \texttt{relat}_{\pi'} \circ \nu, \texttt{relat}_{\pi'}(\pi) \models \phi$$

En particulier : $\quad t, \nu, \pi \models \phi \quad \Leftrightarrow \quad t|_{\pi}, \texttt{relat}_{\pi} \circ \nu, \epsilon \models \phi$

Démonstration. Nous prouvons ce lemme par induction sur la structure de la formule. Soit un noeud $\pi' \in \texttt{dom}(t)$ tel que $\pi' \lhd^* \pi$. Alors il existe un noeud π'' tel que $\pi = \pi'\pi''$, i.e. $\texttt{relat}_{\pi'}(\pi) = \pi''$.

$t, \nu, \pi \models q(x)$ ssi $\pi \lhd^* \nu(x)$ et $(\texttt{relat}_\pi \circ \nu)(x) \in q(t|_\pi)$
ssi $\pi'' \lhd^* (\texttt{relat}_{\pi'} \circ \nu)(x)$ et $(\texttt{relat}_\pi \circ \nu(x) \in q((t|_{\pi'})|_{\pi''})$
ssi $\pi'' \lhd^* (\texttt{relat}_{\pi'} \circ \nu)(x)$ et $\texttt{relat}_{\pi''}(\texttt{relat}_{\pi'} \circ \nu(x)) \in q((t|_{\pi'})|_{\pi''})$
ssi $\pi'' \lhd^* (\texttt{relat}_{\pi'} \circ \nu)(x)$ et $\texttt{relat}_{\texttt{relat}_{\pi'}(\pi)}(\texttt{relat}_{\pi'} \circ \nu(x)) \in q((t|_{\pi'})|_{\pi''})$
ssi $t|_{\pi'}, \texttt{relat}_{\pi'} \circ \nu, \texttt{relat}_{\pi'}(\pi) \models q(x)$

$t, \nu, \pi \models q(x).\phi$ ssi $t, \nu, \pi \models q(x)$ et $t, \nu, \nu(x) \models \phi$
ssi (par hypothèse d'induction)
$t|_{\pi'}, \texttt{relat}_{\pi'} \circ \nu, \pi'' \models q(x)$ et $t, \nu|_{\mathrm{VL}(\phi)}, \nu(x) \models \phi$ et $\forall y \in \mathrm{VL}(\phi)\ \nu(x) \lhd^* \nu(y)$
ssi (par hypothèse d'induction, applicable car $\pi' \lhd^* \nu(x)$
$t|_{\pi'}, \texttt{relat}_{\pi'} \circ \nu, \pi'' \models q(x)$ et $t|_{\pi'}, \texttt{relat}_{\pi'} \circ \nu|_{\mathrm{VL}(\phi)}, \texttt{relat}_{\pi'} \circ \nu(x) \models \phi$
ssi $t|_{\pi}, \texttt{relat}_{\pi} \circ \nu, \epsilon \models q(x)$

Les deux derniers cas se prouvent similairement. □

Comme la correction de l'algorithme d'évaluation d'une requête dépend immédiatement de la correction de l'algorithme auxiliaire, nous prouvons la correction de ce dernier :

Lemme 5 (Correction). *Pour toute formule $\phi \in \texttt{COMP}[Q]$, pour tout arbre $t \in T_\Sigma^k$ et pour toute valuation ν de $VL(\phi)$ dans $\texttt{dom}(t)$:*

$$t, \nu, \epsilon \models \phi \quad \text{ssi} \quad \nu \in \texttt{aux}(\phi, t)$$

Démonstration. Par induction sur la structure de la formule.
- $q(x)$: c'est immédiat
- $q(x).\phi$:

$t, \nu, \epsilon \models q(x).\phi$
ssi (par définition de la sémantique)
$\nu(x) \in q(t)$ et $t, \nu|_{\mathrm{VL}(\phi)}, \nu(x) \models \phi$ et $\forall y \in \mathrm{VL}(\phi),\ \nu(x) \lhd^* \nu(y)$
ssi (par définition de l'algorithme)
$\nu(x) \in M(q,t)$ et $t, \nu|_{\mathrm{VL}(\phi)}, \nu(x) \models \phi$ et $\forall y \in \mathrm{VL}(\phi),\ \nu(x) \lhd^* \nu(y)$
ssi (par le lemme 4)
$\nu(x) \in M(q,t)$ et $t|_{\nu(x)}, \mathtt{relat}_{\nu(x)} \circ \nu|_{\mathrm{VL}(\phi)}, \epsilon \models \phi$
ssi (par hypothèse d'induction)
$\nu(x) \in M(q,t)$ et $\mathtt{relat}_{\nu(x)} \circ \nu \in \mathtt{aux}(\phi, t|_{\nu(x)})$
ssi (par définition de l'algorithme)
$[x \to \nu(x)] \bowtie \nu(x).(\mathtt{relat}_{\nu(x)} \circ \nu) \in \mathtt{aux}(q(x).\phi, t)$
ssi
$[x \to \nu(x)] \bowtie \nu \in \mathtt{aux}(q(x).\phi, t)$
ssi (par définition de la jointure)
$\nu \in \mathtt{aux}(q(x).\phi, t)$

– $\phi = \phi_1 \vee \phi_2$
$t, \nu, \epsilon \models \phi_1 \vee \phi_2$
ssi (par définition de la sémantique)
$t, \nu, \epsilon \models \phi_1$ ou $t, \nu, \epsilon \models \phi_2$
ssi
$t, \nu|_{\mathrm{VL}(\phi_1)}, \epsilon \models \phi_1$ ou $t, \nu|_{\mathrm{VL}(\phi_2)}, \epsilon \models \phi_2$
ssi (par hypothèse d'induction)
$\nu|_{\mathrm{VL}(\phi_1)} \in \mathtt{aux}(\phi_1, t)$ ou $\nu|_{\mathrm{VL}(\phi_2)} \in \mathtt{aux}(\phi_2, t)$
ssi (par définition de extend)
$\nu \in \mathrm{extend}^{\mathrm{VL}(\phi)-\mathrm{VL}(\phi_1)}_{\mathtt{dom}(t)}(\mathtt{aux}(\phi_1, t))$ ou $\nu \in \mathrm{extend}^{\mathrm{VL}(\phi)-\mathrm{VL}(\phi_2)}_{\mathtt{dom}(t)}(\mathtt{aux}(\phi_2, t))$
ssi (par définition de l'algorithme)
$\nu \in \mathtt{aux}(\phi, t)$

– $\phi = \phi_1 \wedge \phi_2$
$t, \nu, \epsilon \models \phi_1 \wedge \phi_2$
ssi (par définition de la sémantique)
$t, \nu, \epsilon \models \phi_1$ et $t, \nu, \epsilon \models \phi_2$
ssi
$t, \nu|_{\mathrm{VL}(\phi_1)}, \epsilon \models \phi_1$ et $t, \nu|_{\mathrm{VL}(\phi_2)}, \epsilon \models \phi_2$
ssi (par hypothèse d'induction)
$\nu|_{\mathrm{VL}(\phi_1)} \in \mathtt{aux}(\phi_1, t)$ et $\nu|_{\mathrm{VL}(\phi_2)} \in \mathtt{aux}(\phi_2, t)$
ssi (par définition de la jointure et de l'algorithme)
$t, \nu, \epsilon \in \mathtt{aux}(\phi, t)$

□

Complexité de l'algorithme Les opérations de base sont les jointures, qu'on suppose réalisées en temps constant.

Lemme 6 (Complexité). *Soit $C_q(t)$ la complexité pour répondre à une requête monadique q sur l'entrée t. Alors sur l'entrée ($\mathtt{query}_{\phi,\vec{s}}$, t), l'algorithme calcule toutes les solutions de la requête en temps $\mathcal{O}(|\phi|(M(\phi,t) + |t|^{2|VL(\phi)|}))$ où $M(\phi,t) = \mathtt{argmax}_{q\in\phi}(C_q(t))$.*

Démonstration. Par induction sur les formules nous prouvons que pour toute formule $\phi \in$

$\texttt{COMP}[Q]$ le temps de calcul $\text{Time}(\phi, t)$ pour calculer toutes les valuations solutions sur l'entrée t est borné par $|\phi|(M(\phi,t) + 2|t|^{2|\text{VL}(\phi)|})$.

- si $\phi = q(x)$ alors $\text{Time}(\phi, t) = C_q(t)$.
- si $\phi \equiv q(x).\phi'$ alors ϕ' est d'abord évaluée, et l'évaluation retourne au plus $|t|^{\text{VL}(\phi')}$ valuations, devant être jointes avec au plus $|t|$ valuations de la forme $[x \to \pi]$ pour tout π retourné par $M(q,t)$, ce qui prend au plus $|t||t|^{|\text{VL}(\phi')|} = |t|^{|\text{VL}(\phi')+1|} \leq |t|^{2|\text{VL}(\phi)|}$ opérations. Par conséquent :

$$\begin{array}{rcllcl}
\text{Time}(\phi,t) & = & \text{Time}(\phi',t) & + & C_q(t) & + & |t|^{|\text{VL}(\phi')|+1} \\
& \leq & |\phi'|(M(\phi',t) + 2|t|^{2|\text{VL}(\phi')|}) & + & C_q(t) & + & |t|^{2\text{VL}(\phi)|} \\
& \leq & |\phi'|(M(\phi',t) + 2|t|^{2|\text{VL}(\phi)|}) & + & C_q(t) & + & 2|t|^{|\text{VL}(\phi)|} \\
& \leq & (|\phi'|+1)(M(\phi,t) + 2|t|^{2|\text{VL}(\phi)|}) & & & & \\
& \leq & |\phi|(M(\phi,t) + 2|t|^{2|\text{VL}(\phi)|}) & & & &
\end{array}$$

- si $\phi \equiv \phi_1 \vee \phi_2$ alors ϕ_1 et ϕ_2 sont d'abord évaluées, et retournent, respectivement, au plus $|t|^{|\text{VL}(\phi_1)|}$ et $|t|^{|\text{VL}(\phi_1)|}$ valuations. L'opération d'extension pour chaque ensemble de valuations prend au plus $|t|^{|\text{VL}(\phi)|}$ opérations. Par conséquent :

$$\begin{array}{rcllcllcl}
\text{Time}(\phi,t) & = & \text{Time}(\phi_1,t) & + & \text{Time}(\phi_2,t) & + & 2|t|^{|\text{VL}(\phi)|} \\
& \leq & |\phi_1|(M(\phi_1,t) + 2|t|^{2|\text{VL}(\phi_1)|}) & + & |\phi_2|(M(\phi_2,t) + 2|t|^{2|\text{VL}(\phi_2)|}) & + & 2|t|^{2|\text{VL}(\phi)|} \\
& \leq & (|\phi_1| + |\phi_2|)M(\phi,t) & + & (|\phi_1| + |\phi_2| + 1)2|t|^{2|\text{VL}(\phi)|} & & \\
& \leq & |\phi|(M(\phi,t) + 2|t|^{2|\text{VL}(\phi)|}) & & & &
\end{array}$$

- si $\phi \equiv \phi_1 \wedge \phi_2$, alors ϕ_1 et ϕ_2 sont évaluées et leurs évaluations retournent respectivement deux ensembles de valuations V_1 et V_2 de tailles au plus $|t|^{|\text{VL}(\phi_1)|}$ et $|t|^{|\text{VL}(\phi_1)|}$ respectivement. Chaque paire de valuations de $V_1 \times V_2$ subit une opération de jointure, et le nombre total de jointure est alors $|t|^{|\text{VL}(\phi_1)|+|\text{VL}(\phi_2)|} \leq |t|^{2|\text{VL}(\phi)}$.

Enfin, comme $|\phi|(M(\phi,t) + 2|t|^{2|\text{VL}(\phi)|}) = \mathcal{O}(|\phi|(M(\phi,t) + |t|^{2|\text{VL}(\phi)|}))$ on peut conclure.

Remarque 5. Le pire des cas est obtenu avec la formule suivante :

$$\bigwedge_{j=1}^{p} \left(\bigvee_{i=1}^{n} \text{All}(x_i) \right).$$

où $t, \pi \models \text{All}(x_i)$ ssi $\pi \in \texttt{dom}(t)$.

□

3.2.3 Expressivité

Comme nous verrons dans le prochain chapitre, les requêtes de composition sur des requêtes monadiques régulières (donc MSO-définissable par le théorème **??**) sont régulières, et réciproquement. Plus formellement, notons $\texttt{COMP}[MSO]$ les formules de compositions sur des requêtes monadiques MSO-définissables. Dans le chapitre 4, nous prouvons que pour toute formule MSO $\psi(x_1, \ldots, x_n)$ à n variables libres, il existe une formule de composition $\phi \in \texttt{COMP}[MSO]$ avec $(x_1, \ldots, x_n)\text{VL}(\phi)$, telle que :

$$\texttt{query}_{\psi(x_1,\ldots,x_n)} = \texttt{query}_{\phi,(x_1,\ldots,x_n)}$$

et réciproquement.

Chapitre 4

Expressivité du langage de composition

Dans ce chapitre nous prouvons que la composition de requêtes monadiques MSO-définissables COMP[MSO] capture exactement la classe des requêtes MSO-définissables, i.e. la classe des requêtes n-aires régulières. Plus formellement, le théorème suivant énonce ce résultat :

Théorème 5 (MSO-Complétude). *Toute requête de composition* $\texttt{query}_{\phi,\overrightarrow{x}} \in$COMP[MSO] *est égale à une requête* MSO $\texttt{query}_{\psi(\overrightarrow{x})}$*, et réciproquement.*

Il suffit pour cela de montrer que toute formule de composition ϕ à n variables libres est équivalente à une formule MSO ψ à n variables libres, et réciproquement. «Équivalente» signifie ici que pour tout arbre $t \in T_\Sigma^k$ et pour tout valuation ν des variables libres de ϕ dans $\texttt{dom}(t)$, $t, \nu, \epsilon \models_{\texttt{COMP}[MSO]} \phi$ si et seulement si $t, \nu \models_{MSO} \psi$.

Le chapitre se décompose en deux sections, chacune contenant la preuve d'une direction du théorème 5.

4.1 COMP[MSO] $\Rightarrow$ MSO

Dans cette section nous établissons la preuve du lemme suivant :

Lemme 7. *Toute requête de composition sur des requêtes monadiques MSO-définissables est MSO-définissable.*

Avant de prouver ce lemme, nous établissons un lemme plus général sur la composition de requêtes monadiques quelconques. Pour un ensemble fini Q de requêtes monadiques, on note $(B_q)_{q\in Q}$ la famille finie de prédicats B_q, pour une requête monadique $q \in Q$, où B_q est interprété par :

$$\forall t \in T_\Sigma \quad B_q^t = \{(\pi_1, \pi_1\pi_2) \in \texttt{dom}(t)^2 \mid \pi_2 \in q(t|_{\pi_1})\}$$

De plus, rappelons que pour un ensemble Q de requêtes monadiques, FO$[(B_q)_{q\in Q} \cup \mathbb{T}]$ désigne l'ensemble des formules de la logique du premier ordre sur le vocabulaire $(B_q)_{q\in Q} \cup \mathbb{T}$.

Lemme 8. *Pour toute formule de composition* $\phi \in \texttt{COMP}[P]$ *sur un ensemble* P *de requêtes monadiques quelconques telle que* $VL(\phi) = \{x_1, \ldots, x_n\}$*, il existe un sous-ensemble fini* $Q \subseteq P$ *de requêtes monadiques et une formule* $\gamma(x_1, \ldots, x_n)$ *de* FO$[(B_q)_{q\in Q} \cup \mathbb{T}]$ *telle que :*

$$t, \nu, \epsilon \models \phi \quad \text{ssi} \quad t \models_{FO} \gamma(\nu(x_1), \ldots, \nu(x_n))$$

Démonstration. Prenons Q l'ensemble des requêtes monadiques apparaissant dans ϕ. Alors on définit la fonction de codage $[\![.]\!]$ d'une formule de composition vers une FO$[(B_q)_{q\in Q} \cup \mathbb{T}]$-formule inductivement par :

$$\begin{array}{lcl} [\![\phi]\!] & = & \exists y\ \mathtt{root}(y) \wedge [\![\phi]\!]_y \quad \text{avec } y \notin \mathrm{VL}(\phi) \\ \\ [\![q(y)]\!]_x & = & B_q(x,y) \\ [\![q(y).\phi]\!]_x & = & B_q(x,y) \wedge [\![\phi]\!]_y \\ [\![\phi_1 \wedge \phi_2]\!]_x & = & [\![\phi_1]\!]_x \wedge [\![\phi_2]\!]_x \\ [\![\phi_1 \vee \phi_2]\!]_x & = & [\![\phi_1]\!]_x \vee [\![\phi_2]\!]_x \end{array}$$

Intuitivement, la fonction $[\![.]\!]_x$ indexée par la variable x définit le codage d'une formule sous une variable x qui dénote le noeud de relativisation. Plus formellement, il faut montrer tout d'abord que $t, \nu \models [\![\phi]\!]_x$ ssi $t, \nu, \nu(x) \models \phi$ (*). Nous donnons la preuve pour le cas $\phi = q(y)$, les autres cas étant de simples applications de l'hypothèse d'induction et/ou du premier cas.

$$\begin{array}{rll} t, \nu, \nu(x) \models q(y) & \text{ssi} & \mathtt{relat}_{\nu(x)}(\nu(y)) \in q(t|_{\nu(x)}) \\ & \text{ssi} & (\nu(x), \nu(x).(\mathtt{relat}_{\nu(x)}(\nu(y)))) \in B^t_q \\ & \text{ssi} & t, \nu \models B_q(x,y) \\ & \text{ssi} & t, \nu \models [\![q(y)]\!]_x \end{array}$$

On peut maintenant démontrer que le codage est correct :

$$\begin{array}{rll} t, \nu, \epsilon \models \phi & \text{ssi} & t, \nu[y \leftarrow \epsilon], \nu(y) \models \phi \text{ et } y \notin \mathrm{VL}(\phi) \\ & \text{ssi (par (*))} & t, \nu[y \leftarrow \epsilon] \models [\![\phi]\!]_y \\ & \text{ssi} & t, \nu \models \exists y\ \mathtt{root}(y) \wedge [\![\phi]\!]_y \\ & \text{ssi} & t, \nu \models [\![\phi]\!] \end{array}$$

De plus, $\mathtt{root}(y)$ est définissable en FO, donc en MSO, ce qui termine la preuve. □

On peut alors énoncer le premier lemme d'équivalence des requêtes :

Lemme 9. *Pour toute requête de composition* $\mathtt{query}_{\phi,\overrightarrow{x}}$ *sur un ensemble de requêtes monadiques, il existe un ensemble fini* Q *de requêtes monadiques et une formule* $\gamma(\overrightarrow{x})$ *de* FO$[(B_q)_{q\in Q} \cup \mathbb{T}]$ *telle que :*

$$t \models \gamma(\overrightarrow{\pi}) \text{ ssi } \overrightarrow{\pi} \in \mathtt{query}_{\phi,\overrightarrow{x}}(t)$$

Démonstration. En utilisant le lemme 8 et le codage défini dans sa preuve, il suffit de prendre la formule :

$$\gamma(\overrightarrow{x}) \equiv \exists \mathrm{VL}(\phi) - \overrightarrow{x} \quad [\![\phi]\!]$$

□

Nous pouvons maintenant prouver le lemme 7.

preuve du lemme 7. En utilisant le lemme 8 et le lemme 9, il suffit alors de montrer que pour toute requête q monadique MSO-définissable , le prédicat B_q est MSO-définissable. Nous définissons le codage inductivement sur la structure de la formule MSO définissant la requête monadique ($\psi(z)[z \leftarrow y]$ est la formule ψ dans laquelle toute les occurences libres de z ont été remplacées par y) :

$$
\begin{array}{lcll}
B_{\mathtt{query}_{\psi(z)}}(x,y) & = & x \lhd^* y \wedge \wr\psi(z)[z \leftarrow y]\int_x & \\
\wr S_i(u,v)\int_x & = & S_i(u,v) & \forall i \in \{1,\ldots,k\} \\
\wr z \in X\int_x & = & z \in X & \\
\wr \exists z \phi\int(x) & = & \exists z\ x \lhd^* z \wedge \wr\phi\int_x & \\
\wr \exists Z \phi\int_x & = & \exists Z\ x \lhd^* Z\ \wedge\ \wr\phi\int_x & \\
\wr \phi_1 \wedge \phi_2\int_x & = & \wr\phi_1\int_x \wedge \wr\phi_2\int_x & \\
\wr \phi_1 \vee \phi_2\int_x & = & \wr\phi_1\int_x \vee \wr\phi_2\int_x & \\
\wr \neg\phi\int_x & = & \neg\wr\phi\int_x &
\end{array}
$$

où $(x \lhd^* Z)$ est défini par $(\forall z\ \ z \in Z \rightarrow x \lhd^* z)$, et $z, u, v \in \mathtt{Var}$.

La première ligne permet de remplacer toutes les occurences libres de z de la formule ψ par la variable y, et le codage défini par $\wr\psi(y)\int_x$ transforme la formule ψ afin que toute quantification dans ψ soit relativisée par rapport à x, ceci permet de relativiser le domaine d'interprétation des variables quantifiées dans ψ à tout noeud en dessous de x. Pour prouver la correction du codage, il faut prouver la proposition (*) suivante :

$$
\forall t \in T^k_\Sigma,\ \forall \pi_1, \pi_2 \in \mathtt{dom}(t) \qquad \begin{array}{c} t \models B_{\mathtt{query}_{\psi(z)}}(\pi_1, \pi_2) \\ \text{ssi} \\ (\pi_1, \pi_2) \in \{(\pi, \pi\pi') \mid t|_\pi \models \psi(\pi')\} \end{array} \qquad (*)
$$

Soit $t \in T^k_\Sigma$, et ν une valuation telle que $\nu(x) = \pi$, alors pour toute formule $\psi \in$ MSO, $t, \nu \models \wr\psi\int_x$ implique par définition que pour toute variable libre z et toute variable libre du second ordre Z apparaissant dans ψ, on a $\nu(x) \lhd^* \nu(z)$ et $\nu(x) \lhd^* \nu(Z)$. Pour prouver (*), prouvons d'abord par induction sur les formules que : $t, \nu \models \wr\psi\int_x$ si et seulement si $t|_{\nu(x)}, \mathtt{relat}_{\nu(x)}(\nu) \models \psi$ (**).

$$
\begin{array}{lll}
t, \nu \models \wr S_i(u,v)\int_x & \text{ssi} & t, \nu \models S_i(u,v) \text{ et } \nu(x) \lhd^* \nu(u) \text{ et } \nu(x) \lhd^* \nu(v) \\
& \text{ssi} & t|_{\nu(x)} \models S_i(\mathtt{relat}_{\nu(x)}(\nu(u)), \mathtt{relat}_{\nu(x)}(\nu(v))) \\
\\
t, \nu \models \wr z \in X\int_x & \text{ssi} & t, \nu \models z \in X \text{ et } \nu(x) \lhd^* \nu(z) \text{ et } \nu(x) \lhd^* \nu(X) \\
& \text{ssi} & t|_{\nu(x)}, \mathtt{relat}_{\nu(x)}(\nu(z)), \mathtt{relat}_{\nu(x)}(\nu(X)) \models (z \in X) \\
\\
t, \nu \models \wr \exists z \phi\int_x & \text{ssi} & t, \nu \models \exists z\ x \lhd^* z \wedge \wr\phi\int_x \\
& \text{ssi} & \exists \pi \in \mathtt{dom}(t) \text{ tq } t, \nu[z \leftarrow \pi] \models (\wr\phi\int_x) \text{ et } \nu(x) \lhd^* \pi \\
& \text{ssi} & \exists \pi' \in \mathtt{dom}(t|_{\nu(x)}) \text{ tq } t|_{\nu(x)}, \mathtt{relat}_{\nu(x)}(\nu[z \leftarrow \pi]) \models \psi \\
& \text{ssi} & t|_{\nu(x)}, \mathtt{relat}_{\nu(x)}(\nu) \models \exists z \psi
\end{array}
$$

Les autres cas sont similaires. Prouvons maintenant (*), soit $t \in T^k_\Sigma$ et $\pi_1, \pi_2 \in \mathtt{dom}(t)$, alors :

$$
\begin{array}{lll}
t, [x \leftarrow \pi_1, y \leftarrow \pi_2] \models B_{\mathtt{query}_{\psi(z)}}(x,y) & \text{ssi} & \pi_1 \lhd^* \pi_2 \text{ et } t, [x \leftarrow \pi_1, y \leftarrow \pi_2] \models \wr\psi(z)[z \leftarrow y]\int_x(x,y) \\
& \text{ssi} & t|_{\pi_1}, [y \leftarrow \mathtt{relat}_{\pi_1}(\pi_2)] \models \psi(y) \qquad (\text{par } (**)) \\
& \text{ssi} & (\pi_1, \pi_2) \in \{(\pi, \pi\pi') \mid t|_\pi \models \psi(\pi')\}
\end{array}
$$

□

4.2 MSO ⇒ COMP[MSO]

Dans cette section nous établissons la preuve du lemme suivant :

Lemme 10. *Toute requête MSO-définissable est égale à une requête de composition sur des requêtes monadiques MSO-définissables, i.e. de* $\texttt{COMP}[MSO]$.

Il suffit (et c'est l'objet de cette section) de prouver le lemme suivant :

Lemme 11. *Pour toute formule MSO* $\psi(x_1, \ldots, x_n)$ *à* n *variables libres il existe un ensemble fini* Q *de requêtes monadiques et une formule de composition* $\phi \in \texttt{COMP}[Q]$ *telle que* $\{x_1, \ldots, x_n\} \subseteq VL(\phi)$ *et* $\texttt{query}_{\phi,(x_1,\ldots,x_n)} = \texttt{query}_{\psi(x_1,\ldots,x_n)}$.

La preuve est moins directe que la preuve du lemme 7, et se décompose en plusieurs étapes :

1. par le théorème 3, il existe un automate d'arbre $\mathcal{A}$ et un ensemble de sélection $S \subseteq \texttt{states}(\mathcal{A})^n$ tel que $\texttt{query}_\psi = \texttt{query}_{\mathcal{A},S}$
2. il existe un nombre fini de *configurations* dans lesquelles se trouvent les éléments sélectionnés (par exemple, dans le cas binaire, si (π_1, π_2) est solution de la requête, alors $\pi_1 \lhd^* \pi_2$, ou $\pi_2 \lhd^* \pi_1$, ou π_1 et π_2 sont incomparables et π_1 est à droite de π_2 dans l'arbre, ou π_1 et π_2 sont incomparables et π_2 est à droite de π_1 dans l'arbre)
3. pour toute configuration, on décompose en *runs partiels* un run acceptant de l'automate $\mathcal{A}$ extrayant des n-uplets qui respectent cette configuration. Un run partiel sélectionne un élément du n-uplet, et en recomposant les runs partiels en respectant la configuration donnée, on arrive à reconstruire un run acceptant global qui sélectionne le n-uplet entier.
4. pour décrire une configuration, on décrit la position relative des ancêtres communs des composantes d'un n-uplet sélectionné
5. la requête de composition finale est une disjonction suivant toutes les configurations des différentes descriptions de ces configurations et des runs acceptants sur celles-ci.

Afin de donner l'intuition de la preuve, nous étudions informellement le cas des requêtes ternaires dans les arbres binaires.

4.2.1 Cas ternaire

Nous étudions informellement le cas des requêtes ternaires, afin de donner l'intuition de la notion de configuration, et de la décomposition d'un run en runs partiels. Partant d'une formule MSO à trois variables libres $\psi(x, y, z)$, il s'agit donc de construire une requête de composition $\texttt{query}_{\phi,(x,y,z)}$ équivalente. D'après le théorème 3 il existe un automate $\mathcal{A}_S$ et un ensemble de sélection S tel que $\texttt{query}_\psi = \texttt{query}_{\mathcal{A}_S,S}$. Les requêtes monadiques utilisées serviront à décrire des runs partiels de $\mathcal{A}_S$, et les configurations. Elles seront de quatre types :

$\pi \in \texttt{Run}^p_{p'}(t)$	ssi	il existe un run de $\mathcal{A}_S$ sur t qui étiquette la racine par p et π par p'
$\pi \in \texttt{Run}^{\text{accept}}_p(t)$	ssi	il existe un run acceptant de $\mathcal{A}_S$ sur t qui étiquette π par p
$\pi \in \texttt{Get}_i(t)$	ssi	π est le i-ème fils de la racine de t
$\pi \in \texttt{Lab}_f(t)$	ssi	π est la racine de t et est étiquetée par $f \in \Sigma$

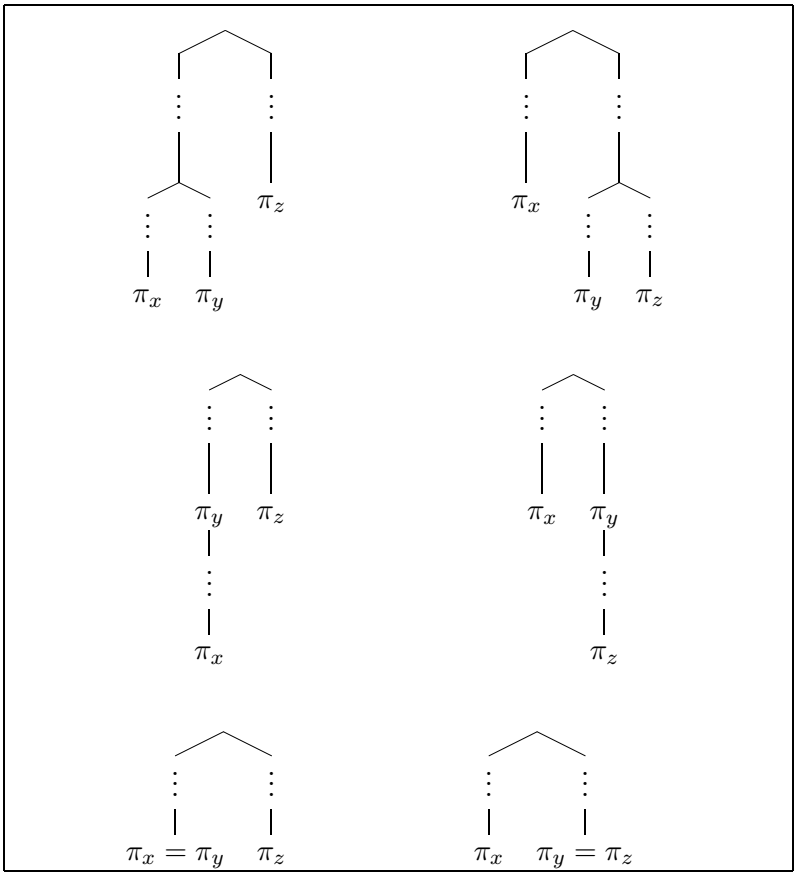


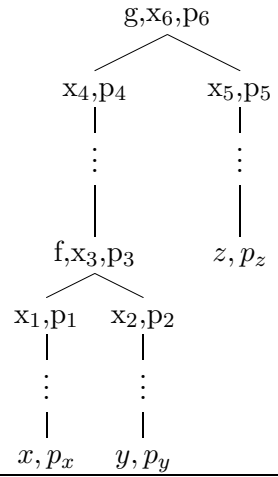
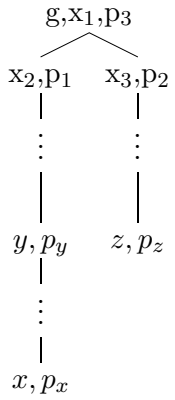
FIG. 4.1 – exemples de positionnements relatifs

pour tout $p, p' \in \texttt{states}(\mathcal{A}_S)$, et pour tout $i \in \{1, 2\}$. Ces requêtes monadiques sont MSO-définissables (voir proposition 3).

Étant donné un arbre $t \in T^2_\Sigma$ tel que $t, (\pi_x, \pi_y, \pi_z) \models \psi(x, y, z)$, il existe plusieurs positionnements relatifs possibles (mais en nombre fini) pour les positions relatives des éléments du triplet (π_x, π_y, π_z). La figure 4.1 donne une liste d'exemples de positionnements relatifs, liste non exhaustive ; il manque par exemple tous les cas où les composantes sont permutées. On peut remarquer sur ces exemples que la description des positionnements relatifs fait intervenir les ancêtres communs, et donnent les positions relatives de ces ancêtres communs et des éléments du triplet. Étant donnée une description C des positionnements relatifs du triplet (π_x, π_y, π_z), on cherche alors une formule de composition ϕ telle que tout triplet (π_x, π_y, π_z) est solution de $\texttt{query}_{\phi,(x,y,z)}(t)$ si et seulement si il est solution de $\texttt{query}_{\psi(x,y,z)}(t)$ et respecte les positionnements relatifs définis par C. La formule de composition recherchée doit donc traduire l'existence d'un run global qui sélectionne le triplet (π_x, π_y, π_z) qui respecte C. Se placer dans une description des positionnements relatifs C permet de traduire l'existence d'un run par l'existence de runs partiels entre les ancêtres communs des éléments des n-uplets résultats de la requête $\texttt{query}_{\psi(x,y,z)}$, les positions relatives des ancêtres communs devant être cohérentes avec C. Afin de décomposer un run de $\mathcal{A}_S$ sélectionnant des triplets dans une description donnée des positions relatives, il est nécessaire d'enrichir cette description avec les étiquettes des noeuds des ancêtres communs, des états de l'automate, et des noms donnés aux

variables dénotant les ancêtres communs, ainsi que les éléments du triplet, en respectant les règles de l'automate. De plus, afin de reconstruire un run global de l'automate, nous devons utiliser les fils des ancêtres communs, auxquels nous devons aussi donner un nom de variable. Cette notion de description des positions relatives enrichie est appelée *configuration*. Nous donnons maintenant deux cas particuliers de configurations, et la description de la configuration et d'un run de $\mathcal{A}_S$ sélectionnant (x, y, z) sous forme d'une requête de composition.

En supposant $p_1, \ldots, p_6 \in \texttt{states}(\mathcal{A}_S)$, $f(p_1, p_2) \to p_3 \in \texttt{rules}(\mathcal{A}_S)$, $g(p_4, p_5) \to p_6 \in \texttt{rules}(\mathcal{A}_S)$, $g(p_1, p_2) \to p_3 \in \texttt{rules}(\mathcal{A}_S)$, et $(p_x, p_y, p_z) \in S$, voici deux configurations et la description des runs acceptant associés à ses configurations :

Configurations	Runs
C_1 	$\texttt{Run}^{\text{accept}}_{p_6}(x_6) \wedge$ $\texttt{Lab}_g(x_6).$ $\quad(\texttt{Get}_1(x_4).$ $\quad\quad\texttt{Run}^{p_4}_{p_3}(x_3)) \wedge$ $\quad\quad\texttt{Lab}_f(x_3).$ $\quad\quad\quad\texttt{Get}_1(x_1).\texttt{Run}^{p_1}_{p_x}(x)$ $\quad\quad\quad\wedge$ $\quad\quad\quad\texttt{Get}_2(x_2).\texttt{Run}^{p_2}_{p_y}(y)$ $\quad\wedge$ $\quad(\texttt{Get}_2(x_5).\texttt{Run}^{p_5}_{p_z}(z)))$
C_2 	$\texttt{Run}^{\text{accept}}_{p_3}(x_1) \wedge$ $\texttt{Lab}_g(x_1).$ $\quad(\texttt{Get}_1(x_2).\texttt{Run}^{p_1}_{p_y}(y).\texttt{Run}^{p_y}_{p_x}(x)$ $\quad\wedge$ $\quad\texttt{Get}_2(x_3).\texttt{Run}^{p_2}_{p_z}(z))$

Avant de donner une intuition sur la preuve d'équivalence des requêtes, définissons formellement la fonction `var`(resp. `sta`) qui à une étiquette d'un noeud d'une configuration associe la variable(resp. l'état) apparaissant dans cette étiquette :

$$\begin{aligned}
\texttt{var} \;:\; & (\texttt{Var} \times \texttt{states}(\mathcal{A}_S)) \cup (\Sigma \times \texttt{Var} \times \texttt{states}(\mathcal{A}_S)) \longrightarrow \texttt{Var} \\
& \begin{cases} (x, p) & \mapsto x \\ (f, x, p) & \mapsto x \end{cases} \\
\texttt{sta} \;:\; & (\texttt{Var} \times \texttt{states}(\mathcal{A}_S)) \cup (\Sigma \times \texttt{Var} \times \texttt{states}(\mathcal{A}_S)) \longrightarrow \texttt{states}(\mathcal{A}_S) \\
& \begin{cases} (x, p) & \mapsto p \\ (f, x, p) & \mapsto p \end{cases}
\end{aligned}$$

Si on appelle ϕ_1 la formule correspondant au run sur la configuration C_1, alors $t, \nu, \epsilon \models \phi_1$ si et seulement si $(\nu(x), \nu(y), \nu(z))$ est dans la configuration C_1 et il existe un run acceptant r de l'automate $\mathcal{A}_S$ qui étiquette $\nu(x)$ par p_x, $\nu(y)$ par p_y, et $\nu(z)$ par p_z, $\nu(x_2)$ par p_2, $\nu(x_3)$ par p_3, etc... Intuivement, on dira qu'un run r sur t et une valuation ν dans $\texttt{dom}(t)$ des variables de C_1 respectent C_1 si et seulement si, si on voit C_1 comme un arbre, alors l'arbre des positions relatives des noeuds valués par ν est «similaire »à C_1, et que le run r est cohérent avec les états associés aux variables dans la configuration, i.e. $r(\nu(\texttt{var}(C_1(\pi)))) = \texttt{sta}(C_1(\pi))$.

respecte la configuration C_1 si et seulement si ν

Alors, si $t, \nu, \epsilon \models \phi_1$, alors il existe un run r acceptant tel que (ν, r) respecte la configuration C_1, et par conséquent, le run r sélectionne dans t le triplet $(\nu(x), \nu(y), \nu(z))$, qui est donc solution de la requête de départ.

En supposant que $t, \nu, \epsilon \models \phi_1$, voyons comment construire un run acceptant r tel que (ν, r) respecte la configuration C_1. Puisque $\texttt{Run}_{p_6}^{\text{accept}}(x_6)$ est satisfaite par $\nu(x_6)$, il existe un run r_{x_6} acceptant qui étiquette $\nu(x_6)$ par p_6, par définition. De même, comme $\texttt{Run}_{p_3}^{p_4}(x_3)$ est satisfaite par $\nu(x_3)$, il existe un run r_{x_4,x_3} acceptant qui étiquette $\nu(x_3)$ par p_3 et $\nu(x_4)$ par p_4, etc... Le run r est alors défini par :

$$r : \pi \mapsto \begin{cases} r_{x_6}(\pi) & \text{si } \pi \lhd^* \nu(x_6) \\ r_{x_4,x_3}(\pi) & \text{si } \nu(x_4) \lhd^* \pi \text{ et non } \pi \lhd^* \nu(x_3) \\ r_{x_5,z}(\pi) & \text{si } \nu(x_5) \lhd^* \pi \\ r_{x_1,x}(\pi) & \text{si } \nu(x_1) \lhd^* \pi \\ r_{x_2,y}(\pi) & \text{si } \nu(x_2) \lhd^* \pi \end{cases}$$

r est une application des noeuds de l'arbre vers les états, car définie en correspondance avec la configuration, et c'est bien un run, car les règles $f(p_1, p_2) \to p_3$, et $g(p_4, p_5) \to p_6$ appartiennent à l'automate, et il est acceptant car $r_{x_6}(\epsilon) \in \texttt{finals}(\mathcal{A}_S)$. Par définition de r, et par l'affirmation (*), (r, ν) respecte la configuration C_1.

Réciproquement, s'il existe un run acceptant r et une valuation ν qui respectent la configuration C_1, alors on peut montrer par induction sur la formule que $t, \nu, \epsilon \models \phi_1$.

Si on note γ_C la formule de composition décrivant un run associé à une configuration C, et ϕ la formule $\bigvee_{C \text{ configuration}} \gamma_C$, alors $\texttt{query}_{\psi(x,y,z)} = \texttt{query}_{\phi,(x,y,z)}$.

Remarque 6. Remarquons que la combinaison de tous les runs partiels r_{x_6}, r_{x_4,x_3}, etc... en un run acceptant r est rendue possible par l'intégration dans la configuration de règles de transition de l'automate au niveau des «points de jonctions » (ici le noeud étiqueté par (g, x_6, p_6) et (f, x_3, p_3)). Intégrer des règles de transition dans les configurations suppose qu'on ait un contrôle de tous les fils du noeud au niveau duquel on applique la règle de transition. Ce contrôle se fait un donnant un nom de variable à chacun de ses fils. Par exemple, dans le cas des arbres ternaires, la figure 4.2 donne l'exemple d'une configuration qui explicite l'importance d'avoir ce contrôle.

Avant de donner la preuve dans le cas général, prouvons la proposition 3 qui établit la MSO-définissabilité des requêtes monadiques :

Proposition 3 (MSO-définissabilité des requêtes monadiques). *Pour tout* $p, p' \in \texttt{states}(\mathcal{A}_S)$*, et pour tout* $i \in \{1, \ldots, k\}$, les requêtes monadiques $\texttt{Run}_{p'}^{p}$, $\texttt{Run}_{p}^{\text{accept}}$, $\texttt{Lab}_f$ *et* $\texttt{Get}_i$ *sont MSO-définissables.*

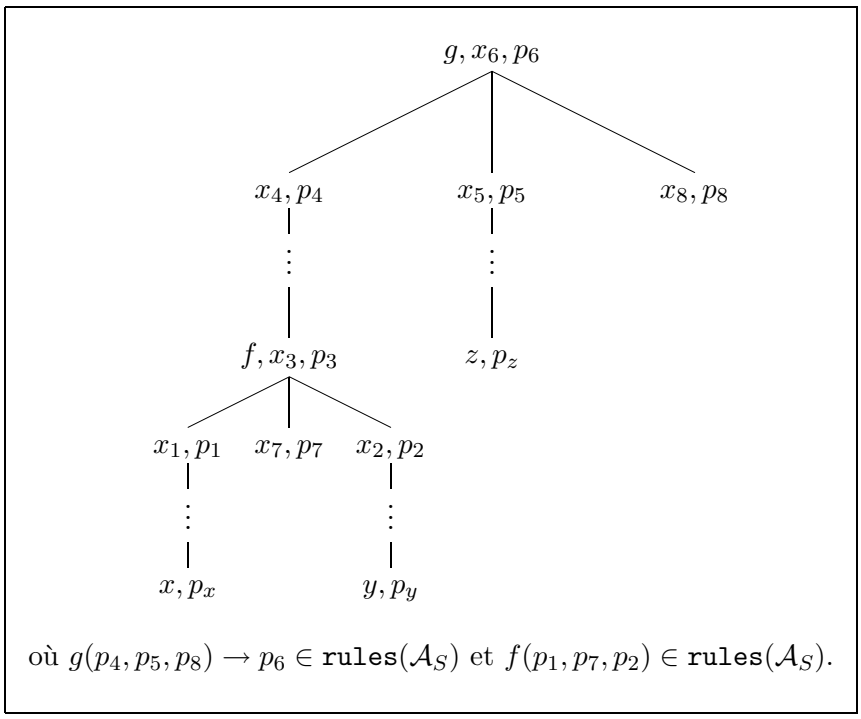


FIG. 4.2 – Exemple de configuration dans le cas des arbres ternaires

Démonstration. Supposons que $\texttt{states}(\mathcal{A}_S) = \{p_1, \ldots, p_n\}$, et soit $\phi(X_{p_1}, \ldots, X_{p_n})$ la formule à n variables libres du second ordre définie par :

$$\begin{aligned}\phi(X_{p_1}, \ldots, X_{p_n}) \equiv \quad & \forall x \ \textstyle\bigvee_i (x \in X_{p_i}) \wedge \bigwedge_i \forall x (x \in X_{p_i} \to \bigwedge_{j \neq i} x \notin X_{p_j}) \\ \wedge \forall x \quad & \bigwedge_{\substack{f \in \Sigma \\ p \in \texttt{states}(\mathcal{A}_S)}} ((\texttt{label}_f(x) \wedge x \in X_p) \to \\ & \Big(\bigvee_{\substack{f(p_1, \ldots, p_n) \to p \\ \in \texttt{rules}(\mathcal{A}_S)}} \bigwedge_{i=1}^{n} \exists y \ S_i(x, y) \wedge y \in Y_{p_i} \Big)\end{aligned}$$

(pour plus d'explications sur la formule voir section 2.4.1)

Soient maintenant les formules suivantes :

$\phi^p_{p'}(x) \equiv \exists X_{p_1} \ldots \exists X_{p_n} \ \phi(X_{p_1}, \ldots, X_{p_n}) \wedge x \in X_{p'} \wedge \exists y \ \texttt{root}(y) \wedge y \in X_p$
$\phi^{\text{accept}}_p(x) \equiv \exists X_{p_1} \ldots \exists X_{p_n} \ \phi(X_{p_1}, \ldots, X_{p_n}) \wedge x \in X_p \wedge \exists y \ \texttt{root}(y) \wedge \bigvee_{p \in \texttt{finals}(\mathcal{A}_S)} y \in X_p$
$\psi_i(x) \equiv \exists y \ \texttt{root}(y) \wedge S_i(y, x)$

Alors $\texttt{Run}^p_{p'} = \texttt{query}_{\phi^p_{p'}(x)}$, $\texttt{Run}^{\text{accept}}_p = \texttt{query}_{\phi^{\text{accept}}_p(x)}$ et $\texttt{Get}_i = \texttt{query}_{\psi_i(x)}$.

□

4.2.2 Cas général

Ici nous prouvons le lemme 11 en généralisant la construction vue dans la section précédente au cas des requêtes n-aire dans les arbres d'arité bornée k sur un alphabet Σ. Nous formalisons la notion de configurations et la construction des runs acceptants associés.

Fixons d'abord quelques notations :

k	:	la borne sur l'arité des arbres
$\psi(\overrightarrow{x})$	:	une formule MSO à n variables libres du premier ordre
$\phi(\overrightarrow{x})$	:	une formule de composition telle que $\texttt{query}_{\psi(\overrightarrow{x})} = \texttt{query}_{\phi,\overrightarrow{x}}$ (et qu'on va construire)
$x_1, \ldots, x_n$	:	les variables du premier ordre telles que $(x_1, \ldots, x_n) = \overrightarrow{x}$
$\texttt{Var}^-$	:	$\texttt{Var} - \{x_1, \ldots, x_n\}$
$\mathcal{A}_S$	:	Un automate d'arbre **complet** $\mathcal{A}_S$ tel que $S \subseteq \texttt{states}(\mathcal{A}_S)$ et $\texttt{query}_{\mathcal{A}_S,S} = \texttt{query}_{\psi(\overrightarrow{x})}$ (le théorème **??** montre qu'il existe)

ainsi que $\texttt{root}^{\texttt{var}}$ qui retourne la variable contenue dans l'étiquette de la racine d'une configuration, et $\texttt{root}^{\texttt{sta}}$ qui retourne l'état contenue dans l'étiquette de la racine d'une configuration, i.e. $\texttt{root}^{\texttt{var}} = \texttt{var} \circ \texttt{label} \circ \texttt{root}$ et $\texttt{root}^{\texttt{sta}} = \texttt{sta} \circ \texttt{label} \circ \texttt{root}$.

Nous avons explicité dans le cas ternaire le rôle de certains noeuds de la configuration, fils des noeuds au niveau desquels on veut appliquer une transition de l'automate afin de combiner les runs partiels. Si ces noeuds ne représentent pas des noeuds à sélectionner, on parle alors de *noeuds intermédiaires*. Nous allons définir, par une grammaire, les configurations comme des arbres dont la racine n'est pas un noeud intermédiaire. Pour cette raison, nous typons les arbres générés (de manière implicite dans la grammaire). Certains seront des arbres générés à partir d'un axiome i_p («i »pour intermédiaire), et d'autres, à partir d'un axiome c_p («c »pour configuration), pour $p \in \texttt{states}(\mathcal{A}_S)$. Les noeuds des arbres générés seront étiquetés dans $\texttt{Var} \times \texttt{states}(\mathcal{A}_S)$ ou $\Sigma \times \texttt{Var} \times \texttt{states}(\mathcal{A}_S)$ (si le noeud est un noeud au niveau duquel on veut appliquer une règle de transition de l'automate). Plus formellement, définissons l'ensemble $\mathbb{C}^{\overrightarrow{s}}_{\overrightarrow{x}}(\mathcal{A}_S)$ des configurations d'un automate de requête $\mathcal{A}_S$ sur des arbres d'arité bornée k et d'un n-uplet de variables $\overrightarrow{x}$, pour un n-uplet de sélection $\overrightarrow{s} \in S$ donné.

Définition 23 (Configuration). Une *configuration* C d'un n-uplet de variables $\overrightarrow{x} = (x_1, \ldots, x_n)$ et d'un automate de requête $\mathcal{A}_S$ sur des arbres d'arité borné k est, pour un n-uplet de sélection $\overrightarrow{s} \in S$ fixé, un arbre d'arité bornée k sur l'alphabet $(\texttt{Var} \times \texttt{states}(\mathcal{A}_S)) \cup (\Sigma \times \texttt{Var} \times \texttt{states}(\mathcal{A}_S))$ engendré par la grammaire G suivante à partir de l'axiome c :

$$
\begin{array}{lcll}
o_p & ::= & c_p \mid i_p & p \in \mathtt{states}(\mathcal{A}_S) \\
\\
i_p & ::= & (x,p) & x \in \mathtt{Var}^-,\ p \in \mathtt{states}(\mathcal{A}_S) \\
 & & \mid (x,p)(c_{p'}) & x \in \mathtt{Var}^-,\ p \in \mathtt{states}(\mathcal{A}_S) \\
\\
c_p & ::= & (x_i,p) & p = \mathtt{proj}_i(\overrightarrow{s}) \\
 & & \mid (x,p)(c_{p'}) & \begin{cases} x \in \mathtt{Var},\ p' \in \mathtt{states}(\mathcal{A}_S) \\ \text{si } x = x_i \ \text{ alors } p = \mathtt{proj}_i(\overrightarrow{s}) \end{cases} \\
 & & \mid (f,x,p)(o_{p_1},\ldots,o_{p_n}) & \begin{cases} f \in \Sigma,\ x \in \mathtt{Var},\ p,p_1,\ldots,p_n \in \mathtt{states}(\mathcal{A}_S),\ n \leq k & (1) \\ |\{i \mid o_{p_i} = c_{p_i}\}| \geq 2 & (2) \\ \text{si } x = x_i \ \text{ alors } p = \mathtt{proj}_i(\overrightarrow{s}) & (3) \\ f(p_1,\ldots,p_n) \to p \ \in \mathtt{rules}(\mathcal{A}_S) & (4) \end{cases} \\
\\
c & ::= & c_{p_1} \mid c_{p_2} \mid \ \ldots \ \mid c_{p_m} & \{p_1,\ldots,p_m\} = \mathtt{states}(\mathcal{A}_S)
\end{array}
$$

tel que pour tout $1 \leq i \leq n$, il existe un noeud π et **un seul** tel que x_i apparait dans $\mathtt{label}^C(\pi)$, et que toute variable de $\mathtt{Var}$ apparait au plus une fois dans la configuration.

Remarque 7. Détaillons les différentes règles de la grammaire une à une.

Règles dont l'axiome est de type $\mathbf{i_p}$. Intuitivement, la racine d'un arbre I est un noeud intermédiaire (I est alors généré à partir d'un axiome i_p) si et seulement si :

- c'est une feuille qui ne représente pas une composante à sélectionner (c'est le cas par exemple de la feuille étiquetée par (x_7,p_7) dans l'arbre de la figure 4.2)
- ou c'est un noeud qui ne représente pas une composante à sélectionner, et qui a un seul fils, qui lui-même n'est pas un noeud intermédiaire (c'est le cas par exemple du noeud étiqueté par (x_4,p_4) dans l'arbre de la figure 4.2).

Règles dont l'axiome est de type $\mathbf{c_p}$. La racine d'un arbre C **n'est pas** un noeud intermédiaire (C est alors généré à partir d'un axiome c_p) si et seulement si :

- c'est une feuille qui représente une composante à sélectionner (noeud étiqueté par (x,p_x) dans l'arbre de la figure 4.2). L'état associé doit alors être sélectionnant pour cette composante.
- ou c'est un noeud qui a un fils qui n'est pas un noeud intermédiaire. S'il représente une composante à sélectionner, alors l'état associé doit être sélectionnant pour cette composante.
- ou c'est un noeud qui est un noeud de jonction, c'est à dire un noeud qui permettra de recombiner les différents runs, par application locale d'une règle de transition de l'automate. On a besoin de recombiner plusieurs runs d'automates si et seulement si le noeud de jonction représente l'ancêtre commun d'au moins deux composantes (différentes de lui-même) à sélectionner, c'est ce que traduit la condition (2) de la règle de production. La condition (3) impose comme d'habitude que si le noeud représente une composante à sélectionner, alors l'état associé doit être sélectionnant pour cette composante. Enfin la condition (4) impose, pour pouvoir faire la jonction des runs en appliquant une règle de transition de l'automate, que la règle formée par les états associées au noeud soit une règle de l'automate.

Enfin, nous imposons que toutes les variables utilisées soient différentes, et que chaque variable sélectionnante soit utilisée exactement une fois.

Comme pour les formules, nous définissons une notion de variables libres. Intuitivement, les variables libres d'une configuration sont les variables qui sont utilisées dans cette configuration.

Définition 24 (variables libres d'une configuration). On définit l'ensemble $\mathrm{VL}(C)$ des variables libres d'une configuration $C \in \mathbb{C}^{\overrightarrow{s}}_{(x_1,\ldots,x_n)}(\mathcal{A}_S)$ inductivement par :
- $\mathrm{VL}((x,p)) = \{x\}$
- $\mathrm{VL}((x,p)(c)) = \{x\} \cup \mathrm{VL}(c)$
- $\mathrm{VL}((f,x,p)(c_1,\ldots,c_n)) = \{x\} \cup \bigcup_i \mathrm{VL}(c_i)$

Remarque 8. Pour tout configuration $C \in \mathbb{C}^{\overrightarrow{s}}_{(x_1,\ldots,x_n)}(\mathcal{A}_S)$, on a $\{x_1,\ldots,x_n\} \subseteq \mathrm{VL}(C)$, où $\overrightarrow{x} = (x_1,\ldots,x_n)$.

Comme les noms des variables (différentes des x_i) qui apparaissent dans les étiquettes d'une configuration ne sont pas importants, et que la seule chose importante est qu'ils soient tous différents, nous définissons une relation d'équivalence (α-conversion $\equiv_\alpha$) entre deux configurations, et diront que deux configurations sont égales si et seulement si elles appartiennent à la même classe d'équivalence.

Définition 25 (α-conversion). Deux configurations C_1 et C_2 de $\mathbb{C}^{\overrightarrow{s}}_{(x_1,\ldots,x_n)}(\mathcal{A}_S)$ sont équivalentes (noté $C_1 \equiv_\alpha C_2$) si et seulement si il existe une bijection χ de $\mathrm{VL}(C_1)$ vers $\mathrm{VL}(C_2)$ telle que pour tout $i \in \{1,\ldots,n\}$, $\chi(x_i) = x_i$ et les conditions suivantes sont satisfaites :
- $\mathtt{dom}(C_1) = \mathtt{dom}(C_2)$
- $\forall \pi \in \mathtt{dom}(C_1)$, $C_1(\pi) = (x,q)$ ssi $C_2(\pi) = (\chi(x),q)$
- $\forall \pi \in \mathtt{dom}(C_1)$, $C_1(\pi) = (f,x,q)$ ssi $C_2(\pi) = (f,\chi(x),q)$

Proposition 4. *La relation $\equiv_\alpha$ est une relation d'équivalence.*

Étant donnée une configuration, on cherchera ensuite, comme dans le cas ternaire, à décrire un run respectant cette configuration, et on fera une disjonction sur toutes les configurations. Il faut donc que l'ensemble des configurations soit fini, c'est ce que prouve le lemme suivant :

Lemme 12. *L'ensemble $\mathbb{C}^{\overrightarrow{s}}_{(x_1,\ldots,x_n)}(\mathcal{A}_S)$ est fini modulo l'alpha-conversion $\equiv_\alpha$.*

Démonstration. Il suffit de montrer que l'ensemble des arbres générés par la grammaire G de la définition 23 est fini modulo α-conversion.

Soit C un arbre généré à partir d'un axiome c_p par la grammaire G (voir définition 23). Montrons qu'il existe au moins une de ses étiquettes qui contient une variable de $\overrightarrow{x}$ (propriété (*)). Montrons le par induction sur la structure de C. Soit C est une feuille, alors par définition de la grammaire, cette feuille contient nécessairement une variable de $\overrightarrow{x}$. Soit la racine de C n'a qu'un seul fils, alors il nécessairement généré à partir d'un axiome $c_{p'}$, d'après les règles de production, et on peut appliquer l'hypothèse d'induction. Soit la racine possède au moins deux fils générés par des axiomes $c_{p'}$ et $c_{p''}$, alors on applique l'hypothèse d'induction.

Montrons maintenant par induction que la hauteur d'un arbre C généré à partir d'un axiome α est bornée par n_C le nombre de noeuds de C contenant des variables de $\overrightarrow{x}$ si $\alpha = c_p$, et $n_C + 1$ si $\alpha = i_p$. Pour simplifier on dira qu'un arbre est de type c_p(resp. i_p) s'il est généré dans G à partir d'un axiome c_p(resp. i_p).

- $C = (x, p)$: c'est immédiat.
- $C = (x, p)(C')$: alors C' est forcément de type $c_{p'}$, donc $h(c) \leq n_{C'}$. Si C est de type c_p, alors x est sélectionnante, et $h(C) = h(C') + 1 \leq n_{C'} + 1 = n_C$. Si C est de type i_p, alors $h(C) = h(C') + 1 \leq n_{C'} + 1 = n_C + 1$.
- $C = (f, x, p)(C_1, \ldots, C_n)$. Soit I tel que pour tout $i \in I$, C_i est de type c_{p_i}, et soit J tel que pour tout $i \in J$, C_i est de type i_{p_i}. Alors $h(C) = max\ (max_I\ h(C_i), max_J\ h(C_i)) + 1$. Notons $m = max\ (max_I\ n_{C_i}, max_J\ n_{C_i} + 1) + 1$, alors par hypothèse d'induction, $h(C) \leq m$. De plus, on sait que $\Sigma_I n_{C_i} + \Sigma_J n_{C_i} \leq n_C$, et d'après la condition (2) de la grammaire G et la propriété (*), $\Sigma_I n_{C_i} \geq 2$. Il y a alors deux cas :
  - Si il existe $i_0 \in J$ tel que $m = n_{C_{i_0}} + 2$, alors $h(C) \leq n_{C_{i_0}} + 2 \leq \Sigma_I n_{C_i} + \Sigma_J n_{C_i} \leq n_C$.
  - Si il existe $i_0 \in I$ tel que $m = n_{C_{i_0}} + 1$, alors d'après la condition (2) de la grammaire G il existe $j \neq i_0$ tel que $n_{C_j} \leq 1$, donc $h(C) \leq n_{C_{i_0}} + 1 \leq n_{C_{i_0}} + n_{C_j} \leq \Sigma_I n_{C_i} \leq \Sigma_I n_{C_i} + \Sigma_J n_{C_i} \leq n_C$.

Soit maintenant C une configuration, alors comme chaque variable de $\overrightarrow{x} = (x_1, \ldots, x_n)$ apparaît exactement une fois dans C, on a $n_C \leq n$, et par conséquent $h(C) \leq n$. Comme l'arité est bornée par k, et que le nombre d'arbres d'arité bornée par k et de hauteur bornée par n sur un alphabet fini est fini, l'ensemble des configurations est fini (modulo α-conversion). □

Nous allons maintenant définir formellement les notions évoquées dans l'étude du cas ternaire, à savoir la notion de respect d'une configuration C par une valuation ν des variables libres de la configuration vers le domaine d'un arbre $t \in T_\Sigma^k$ et par un run sur l'arbre t. Informellement ν respecte la configuration C si il y a un isomorphisme d'ordre préfixe entre les noeuds de la configurations et les valuations dans $\mathtt{dom}(t)$ des variables libres de C. D'autre part, un run r respecte une configuration s'il étiquette les noeuds de l'arbre par des états, en cohérence avec les états de la configuration, selon l'ordre préfixe. On prouvera ensuite que si un run et une valuation respecte une configuration, alors le run est acceptant, et la requête de composition est équivalente à la requête par automate. Nous prouverons aussi la réciproque, en montrant comment, à partir d'une valuation satisfaisant une formule de composition décrivant une configuration, on peut reconstruire un run acceptant respectant la configuration.

Définition 26 (valuation d'une configuration dans un domaine). Étant donnés une configuration $C \in \mathbb{C}^{\overrightarrow{s}}_{(x_1,\ldots,x_n)}(\mathcal{A}_S)$ et un arbre $t \in T_\Sigma^k$, une valuation ν des variables libres de C est une application de $\mathrm{VL}(C)$ dans $\mathtt{dom}(t)$.

Étant donné une configuration C, notons $\lhd_C^*$ l'ordre préfixe sur les noeuds de C. De même pour un arbre $t \in T_\Sigma^k$, notons $\lhd_t^*$ l'ordre préfixe sur les noeuds de t.

Étant donnés une configuration C, un arbre $t \in T_\Sigma^k$, et une valuation ν des variables libres de C dans $\mathtt{dom}(t)$, définissons $\mathtt{corr}_\nu$ l'application qui fait correspondre les noeuds de C avec certains noeuds de t :

$$\begin{array}{rccl} \mathtt{corr}_\nu & : & \mathtt{dom}(C) & \rightarrow & \mathtt{dom}(t) \\ & & \pi & \mapsto & \nu(\mathtt{var}(C(\pi))) \end{array}$$

Définition 27 (ordre induit par une configuration). Étant donnés une configuration C, un arbre $t \in T_\Sigma^k$, et une valuation ν des variables libres de C dans $\mathtt{dom}(t)$, on dit que *ν respecte l'ordre induit par la configuration C* si et seulement si l'application $\mathtt{corr}_\nu$ est un isomorphisme de $(\mathtt{dom}(C), \lhd_C^*)$ vers $(\nu(\mathrm{VL}(C)), \lhd_t^*)$, i.e. $\pi \lhd_C^* \pi'$ ssi $\mathtt{corr}_\nu(\pi) \lhd_t^* \mathtt{corr}_\nu(\pi')$.

Définition 28 (respect d'une configuration). Étant donnés une configuration $C \in \mathbb{C}^{\overrightarrow{s}}_{(x_1,\dots,x_n)}(\mathcal{A}_S)$, un arbre $t \in T^k_\Sigma$, une valuation ν des variables libres de C dans $\mathtt{dom}(t)$, et un run r de l'automate $\mathcal{A}_S$ sur t. On dit que (t, r, ν) *respecte* la configuration C si et seulement si :
- ν respecte l'ordre induit par la configuration C
- pour tout noeud π de C tel que $\mathtt{label}^C(\pi) = (f, x, p)$, $t(\mathtt{corr}_\nu(\pi)) = f$
- pour tout noeud π de C tel que p soit un état de l'automate apparaissant dans $\mathtt{label}^C(\pi)$ (ie $\mathtt{sta}(\mathtt{label}^C(\pi)) = p$), $r(\mathtt{corr}_\nu(\pi)) = p$

Le lemme suivant établit la correspondance entre requêtes et configurations :

Lemme 13. *Étant donnés un arbre $t \in T^k_\Sigma$ et $(\pi_1, \dots, \pi_n) \in \mathtt{dom}(t)$, $(\pi_1, \dots, \pi_n) \in \mathtt{query}_{\mathcal{A}_S,S}(t)$ si et seulement si il existe une configuration $C \in \mathbb{C}^{\overrightarrow{s}}_{(x_1,\dots,x_n)}(\mathcal{A}_S)$ et une valuation ν des variables libres de C avec pour tout $i \in \{1, \dots, n\}$, $\nu(x_i) = \pi_i$, telles qu'il existe un run acceptant r de l'automate $\mathcal{A}_S$ sur t tel que (t, r, ν) respecte C.*

Démonstration. Nous montrons les deux directions de l'équivalence séparément.
$\Rightarrow$: par hypothèse, $(\pi_1, \dots, \pi_n) \in \mathtt{query}_{\mathcal{A}_S,S}(t)$, donc il existe un run r sur t acceptant, et $(s_1, \dots, s_n) \in S$ tel que $r(\pi_i) = s_i$ pour tout i. Nous définissons maintenant informellement la construction de la configuration C. Premièrement nous donnons une transformation δ du domaine de t vers le domaine de C. Intuitivement, la transformation δ ne conserve que les noeuds de t qui vérifient les propriétés (*) : ils sont des composantes sélectionnées, ou des plus petits ancêtres communs de composantes sélectionnées, ou des fils de plus petits ancêtres communs de composantes sélectionnées si cela est nécessaire (noeuds intermédiaires) ; et crée l'arbre de la relation préfixe de ces noeuds. Plus formellement, décomposons la fonction δ en une fonction `extract` qui extrait d'un arbre t, et d'un n-uplet de noeuds donnés, les noeuds satisfiant (*), et en une fonction `pref` qui à un ensemble de noeuds, associe l'arbre de ses relations préfixes.

Définissons formellement l'ensemble $\mathtt{extract}(t) \subseteq \mathtt{dom}(t)$, pour un arbre t donné, par un calcul de plus petit point fixe :

$$\frac{\pi \in (\pi_1, \dots, \pi_n)}{\pi \in \mathtt{extract}(t)} \qquad 1 \text{ (composantes sélectionnées)}$$

$$\frac{\begin{array}{c}\pi_1, \dots, \pi_n \in \mathtt{extract}(t) \\ \pi = \mathtt{ppac}(\pi_1, \dots, \pi_n)\end{array}}{\pi \in \mathtt{extract}(t)} \qquad 2 \text{ (plus petit ancêtre commun)}$$

$$\frac{\begin{array}{c}\pi \in \mathtt{extract}(t) \\ |\{i \mid \pi i \in \mathtt{dom}(t) \ \wedge \ \exists \pi' \in \mathtt{extract}(t),\ \pi i \lhd^* \pi'\}| \geq 2\end{array}}{\{\pi i \mid \pi i \in \mathtt{dom}(t)\} \subseteq \mathtt{extract}(t)} \qquad 3 \text{ (noeuds intermédiaires)}$$

La fonction `extract` est illustrée figure 4.3.

Définissons maintenant la fonction `pref`, qui à tout ensemble de noeud $D \subseteq \mathtt{dom}(t)$ tel que le plus petit élément de D par l'ordre préfixe existe, associe l'arbre $\mathtt{pref}(D)$ des relations préfixes des éléments de D. À cette fin définissons la relation d'équivalence préfixe $\sim_D$ sur un ensemble D de noeuds par :

$$\forall \pi, \pi' \in D \quad \pi \sim \pi' \quad \text{ssi} \quad \begin{array}{l}\text{il existe un élément minimal } \pi_{min} \text{ par } \lhd^* \text{ dans } D \text{ tel que} \\ \pi_{min} \text{ est le plus petit ancêtre commun de } \pi \text{ et } \pi'\end{array}$$

Remarquons que $\sim_D$ est une relation d'équivalence. Notons maintenant $\texttt{ppe}(D')$ le plus petit élément, s'il existe, d'un ensemble de noeud D'. Par définition de $\sim_D$, pour tout $D' \in D/_{\sim_D}$, $\texttt{ppe}(D')$ existe. De plus, notons $\lhd^*_{\texttt{lex}}$ l'ordre lexicographique sur les noeuds. Alors, $\texttt{pref}(D)$ est définie par :

$$\begin{array}{lcll}\texttt{pref}(\{\pi\}) & = & \pi & \text{si } D = \{\pi\} \\ \texttt{pref}(D) & = & (\texttt{ppe}(D))(\texttt{pref}(D_1), \ldots, \texttt{pref}(D_n)) & \text{si } |D| \geq 2, \text{ avec} \\ & & & \{D_1, \ldots, D_n\} = (D - \texttt{ppe}(D))/_{\sim_{D-\texttt{ppe}(D)}} \\ & & & \text{et } \texttt{ppe}(D_1) \lhd^*_{\texttt{lex}} \ldots \lhd^*_{\texttt{lex}} \texttt{ppe}(D_n)\end{array}$$

La fonction `pref` est illustrée figure 4.3.

Alors le domaine de la configuration C est défini par $\texttt{dom}(\texttt{pref}(\texttt{extract}(t)))$, et la fonction δ, si on voit $\texttt{pref}(\texttt{extract}(t))$ comme une application d'un domaine d'arbre vers un ensemble d'étiquettes (ici les noeuds de $\texttt{extract}(t)$), δ est définie par :

$$\forall t \in T^k_\Sigma, \quad \begin{array}{lccl}\delta(t) & : & \texttt{dom}(C) & \rightarrow & \texttt{extract}(t) \\ & & \pi & \mapsto & \texttt{pref}(\texttt{extract}(t))(\pi)\end{array}$$

Remarquons que $\texttt{dom}(C)$ est bien un domaine d'arbre d'arité bornée k, car si un noeud de $\texttt{extract}(t)$ à plus de deux fils dans $\texttt{extract}(t)$, alors, par définition de $\texttt{extract}(t)$, tous ses fils (dans t) sont dans $\texttt{extract}(t)$, donc il n'existe qu'au plus k classes d'équivalences à chaque étape.

La transformation δ définit donc une bijection entre $\texttt{extract}(t)$ et les noeuds de $\texttt{dom}(C)$, et par définition, δ est un isomorphisme de $(\texttt{extract}(t), \lhd^*_t)$ dans $\texttt{dom}(C), \lhd^*_C$ (*).

Ensuite, il faut construire l'étiquettage de $\texttt{dom}(C)$, se qui se fait assez naturellement. Soit $\pi \in \texttt{dom}(C)$, alors :

1. Si $\delta^{-1}(\pi)$ est une composante sélectionnée π_i et si π a au plus un fils, alors $C(\pi) = (x_i,\ r(\delta^{-1}(\pi)))$
2. Si $\delta^{-1}(\pi)$ est une composante sélectionnée π_i et si π a strictement plus d'un fils, π est un noeud de jonction, alors $C(\pi) = (t(\delta^{-1}(\pi)),\ x_i,\ r(\delta^{-1}(\pi)))$
3. ce cas reprend les deux premiers, à la différence que $\delta^{-1}(\pi)$ n'est pas une composante sélectionnée, dans ce cas la variable choisie doit être différente des variables de $\overrightarrow{x}$

On s'arrange de plus pour que toutes les variables soient différentes. La transformation complète de t, r, et $(\pi_1, \ldots, \pi_n)$ en configuration est illustrée figure 4.4. Le run et l'arbre sont représentés sur le même arbre et les composantes sélectionnées sont en caractère gras. Enfin, on définit la valuation ν des variables de C dans $\texttt{dom}(t)$ par $(\texttt{var} \circ \delta)^{-1}$ (qui est bien définie puisque `var` et δ sont bijectives, puisque qu'on impose que toutes les variables soient différentes). Par construction et par la remarque (*), (t, r, ν) respecte la configuration C, et on a fini de prouver la première direction.

$\Leftarrow$ Ce cas est plus immédiat. En effet, s'il existe une configuration C et une valuation ν telle que $\nu(x_i) = \pi_i$, et un run acceptant r tel que (t, r, ν) respecte C, alors comme les états

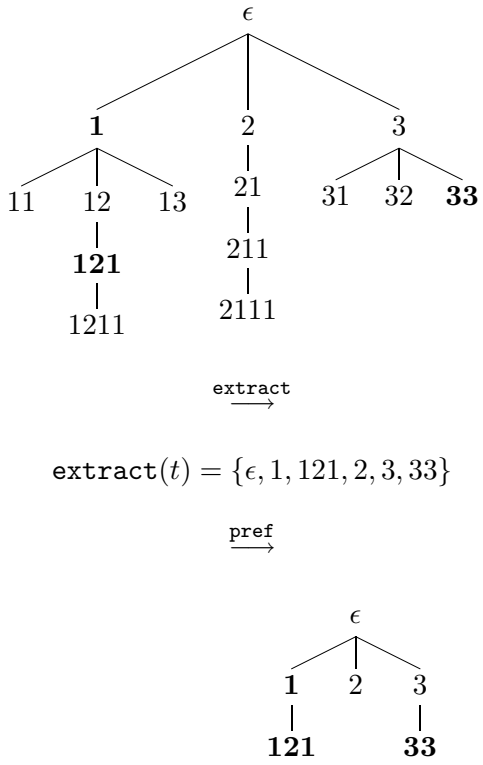


FIG. 4.3 – Fonctions `extract` et `pref` avec un triplet de sélection (**1,121,33**)

p_i associés aux variables x_i dans la configuration C sont tels que $(p_1, \ldots, p_n) \in S$, et que par définition du respect d'une configuration, $r(\nu(x_i)) = r(\pi_i) = p_i$, on peut conclure, par définition d'une requête par automate. □

Nous donnons maintenant le lemme central de la preuve :

Lemme 14. *Soient une configuration $C \in \mathbb{C}^{\vec{s}}_{(x_1,\ldots,x_n)}(\mathcal{A}_S)$, un arbre $t \in T^k_\Sigma$, un n-uplet de noeuds $(\pi_1, \ldots, \pi_n)$, et une valuation ν des variables libres de C telle que $\nu(x_i) = \pi_i$ pour tout $i \in \{1, \ldots, n\}$. Alors il existe une formule de composition $[\![C]\!]$ sur des requêtes monadiques MSO définissables avec $VL([\![C]\!]) = VL(C)$, telle que les deux propositions suivantes soient équivalentes :*

- *il existe un run acceptant r de $\mathcal{A}_S$ sur t tel que (t, r, ν) respecte C*
- $t, \nu, \epsilon \models [\![C]\!]$

Pour construire la formule $[\![C]\!]$ pour une configuration C, comme dans le cas ternaire, il est nécessaire de définir quelques requêtes monadiques :

$\pi \in \mathtt{Run}^p_{p'}(t)$	ssi	il existe un run de $\mathcal{A}_S$ sur t qui étiquette la racine par p et π par p'
$\pi \in \mathtt{Run}^{\text{accept}}_p(t)$	ssi	il existe un run acceptant de $\mathcal{A}_S$ sur t qui étiquette π par p
$\pi \in \mathtt{Get}_i(t)$	ssi	π est le i-ème fils de la racine de t
$\pi \in \mathtt{Lab}_f(t)$	ssi	π est la racine de t et est étiquetée par $f \in \Sigma$
$\pi \in \mathtt{True}(t)$	ssi	$\pi \in \mathtt{dom}(t)$

Preuve du lemme 14. La formule $[\![C]\!]$ est définie par :

$$[\![C]\!] = \begin{cases} \mathtt{Run}_p^{\text{accept}}(x) \wedge \mathtt{Lab}_g(x).\wr C \wr & \text{si } C(\epsilon) = (g, x, p) \\ \mathtt{Run}_p^{\text{accept}}(x).\wr C \wr & \text{si } C(\epsilon) = (x, p) \end{cases}$$

où $\wr C \wr$ est inductivement définie par :

$$\begin{array}{lcll} \wr (x,p) \wr & = & \mathtt{True} & \\ \wr (x,p)(C) \wr & = & (\mathtt{Run}_{p'}^{p}(y).\wr C \wr & \text{si } C(\epsilon) = (y,p') \text{ ou } C(\epsilon) = (g,y,p') \\ \wr (g,x,p)(C_1,\ldots,C_n) \wr & = & \begin{cases} \bigwedge_i \ \mathtt{Get}_i(y_i).\wr C_i \wr & \text{si } \forall i,\ C_i(\epsilon) = (y_i, p) \\ \bigwedge_i \ \mathtt{Get}_i(y_i) \wedge \mathtt{Lab}_g(y_i).\wr C_i \wr & \text{si } \forall i,\ C_i(\epsilon) = (g, y_i, p) \end{cases} & \end{array}$$

$\forall x, y_1, \ldots, y_n \in \mathtt{Var}\ \forall p, p' \in \mathtt{states}(\mathcal{A}_S)\ \forall g \in \Sigma$

Nous montrons d'abord un résultat (**) liée à la transformation $\wr . \wr$:

Étant donné un arbre t, une configuration C et une valuation ν dans $\mathtt{dom}(t)$ de ses variables libres, qui vérifient les conditions du lemme, alors il y a équivalence entre les deux propositions :

- il existe un run r de $\mathcal{A}_S$ sur t tel que (t, r, ν) respecte C
- $t, \nu, \nu(\mathtt{root}^{\mathtt{var}}(C)) \models \wr C \wr$

Nous montrons la première direction par induction :

- $C = (x, p)$: c'est immédiat
- $C = (x, p)(C')$: supposons que $\mathtt{root}^{\mathtt{var}}(C') = y$ et $\mathtt{root}^{\mathtt{sta}}(C') = p'$, alors comme (t, r, ν) respecte la configuration C, $r(\nu(x)) = p$ et $r(\nu(y)) = p'$. Si on note $r|_{\nu(x)}$ le run sur $t|_{\nu(x)}$ induit par r en position $\nu(x)$, alors comme $r(\nu(x)) = r|_{\nu(x)}(\epsilon) = p$ et $r(\nu(y)) = r|_{\nu(x)}(\mathtt{relat}_{\nu(x)}(\nu(y))) = p'$, on a $\mathtt{relat}_{\nu(x)}(\nu(y)) \in \mathtt{Run}_{p'}^{p}(t|_{\nu(x)})$ (*). Comme (t, ν, r) respecte la configuration C, alors il respecte aussi la configuration C', donc par hypothèse d'induction, $t, \nu, \nu(\mathtt{root}^{\mathtt{var}}(C')) \models \wr C' \wr$, i.e. $t, \nu, \nu(y) \models \wr C' \wr$. En combinant avec (*) et par définition de la sémantique de la composition, $t, \nu, \nu(\mathtt{root}^{\mathtt{var}}(C)) \models \wr C \wr$.
- $C = (g, x, p)(C')$: ce cas se traite similairement.

Montrons l'autre direction :

- $C = (x, p)$: supposons que $\mathtt{root}^{\mathtt{var}}(C) = x$, alors si $t, \nu, \nu(x) \models \wr C \wr$, c'est donc que $t, \nu, \epsilon \models [\![C]\!]$, et donc que $t, \nu, \epsilon \models \mathtt{Run}_p^{\text{accept}}(x)$, donc il existe un run sur $t|_{\nu(x)}$ qui étiquette $\mathtt{relat}_{\nu(x)}(\nu(x)) = \epsilon$ par p, ie $t|_{\nu(x)}(\epsilon) = p$, donc, l'automate étant complet, il existe un run sur t qui étiquette $\nu(x)$ par p, et tel que (t, r, ν) respecte C.
- $C = (x, p)(C')$: supposons que $\mathtt{root}^{\mathtt{var}}(C') = y$ et $\mathtt{root}^{\mathtt{sta}}(C') = p'$, alors $\mathtt{relat}_{\nu(x)}(\nu(y)) \in \mathtt{Run}_{p'}^{p}(t|_{\nu(x)})$. Donc il existe un run sur $t|_{\nu(x)}$ qui étiquette la racine de $t|_{\nu(x)}$ par p et $\mathtt{relat}_{\nu(x)}(\nu(y))$ par p', et, comme l'automate est complet, il existe donc un run r' sur t qui étiquette $\nu(x)$ par p et $\nu(y)$ par p'.
  De plus, comme par hypothèse on a $t, \nu, \nu(y) \models \wr C' \wr$, par hypothèse d'induction, il existe un run r'' sur t tel que (t, r'', ν) respecte la configuration C'. Définissons maintenant le run r sur t de la manière suivante :

$$r : \pi \mapsto \begin{cases} r'(\pi) & \text{si } \pi \lhd^* \nu(x) \\ r''(\pi) & \text{sinon} \end{cases}$$

  Alors par construction, r est run sur t tel que (t, r, ν) respecte C.

– $C = (g, x, p)(C')$: ce dernier cas se traite de manière similaire aux deux premiers

Enfin, prouvons l'équivalence du lemme, dans le cas d'une configuration C telle que $C(\epsilon) = (x, p)$:

$t, \nu, \epsilon \models [\![C]\!]$ ssi $\mathtt{Run}_p^{\text{accept}}(x).\wr C \wr$
ssi (par définition de la sémantique)
$t, \nu, \epsilon \models \mathtt{Run}_p^{\text{accept}}(x)$ et $t, \nu|_{\text{VL}(\wr C \wr)}, \nu(x) \models \wr C \wr$ et $\forall y \in \text{VL}(\wr C \wr),\ \nu(x) \lhd^* \nu(y)$
ssi (par définition de la requête monadique)
il existe un run r' acceptant sur t tel que
la racine de t soit étiquetée par un état p et $t, \nu|_{\text{VL}(\wr C \wr)}, \nu(x) \models \wr C \wr$
ssi (par (**))
il existe un run r' acceptant sur t tel que
la racine de t soit étiquetée par un état p
et il existe un run r'' sur t tel que $(t, \nu|_{\text{VL}(\wr C \wr)}, r'')$ respecte C
ssi (en prenant r tel que définie plus haut, à partir de r' et r'')
il existe un run r acceptant sur t tel que (t, r, ν) respecte C

□

Nous pouvons maintenant prouver le lemme 11 :

Preuve du lemme 11. Soient $\psi(x_1, \ldots, x_n)$ une formule MSO à n variables libres. Alors d'après le théorème 3, il existe un automate d'arbre $\mathcal{A}_S$ et un ensemble de sélection $S \subseteq \mathtt{states}(\mathcal{A}_S)^n$ tel que $\mathtt{query}_\psi = \mathtt{query}_{\mathcal{A}_S, S}$. Soit Q l'ensemble fini de requêtes monadiques suivant :

$$Q = \{(\mathtt{Run}_p^{p'})_{p,p' \in \mathtt{states}(\mathcal{A}_S)}, (\mathtt{Run}_p^{\text{accept}})_{p \in \mathtt{states}(\mathcal{A}_S)}, (\mathtt{Get}_i)_{1 \leq i \leq k}, (\mathtt{Lab}_f)_{f \in \Sigma}\}$$

Soit ϕ la formule de composition sur Q définie par :

$$\phi = \bigvee_{\vec{s} \in S} \bigvee_{C \in \mathbb{C}^{\vec{s}}_{(x_1, \ldots, x_n)}(\mathcal{A}_S)} [\![C]\!]$$

Soit $t \in T_\Sigma^k$ et $(\pi_1, \ldots, \pi_n) \in \mathtt{dom}(t)^n$. On a alors :

$(\pi_1, \ldots, \pi_n) \in \texttt{query}_{\mathcal{A}_S,S}(t)$

ssi (par les lemmes 13 et 14)

il existe $\overrightarrow{s} \in S$, $C \in \mathbb{C}^{\overrightarrow{s}}_{(x_1,\ldots,x_n)}(\mathcal{A}_S)$ et $\nu : \mathrm{VL}(C) \to \texttt{dom}(t)$
telle que pour tout $i \in \{1, \ldots, n\}$, $\nu(x_i) = \pi_i$
tel que $t, \nu, \epsilon \models [\![C]\!]$

ssi (par égalité des ensembles de variables libres)

il existe $\overrightarrow{s} \in S$, $C \in \mathbb{C}^{\overrightarrow{s}}_{(x_1,\ldots,x_n)}(\mathcal{A}_S)$ et $\nu : \mathrm{VL}([\![C]\!]) \to \texttt{dom}(t)$
telle que pour tout $i \in \{1, \ldots, n\}$, $\nu(x_i) = \pi_i$
tel que $t, \nu, \epsilon \models [\![C]\!]$

ssi (par définition de ϕ)

il existe $\nu : \mathrm{VL}(\phi) \to \texttt{dom}(t)$ telle que pour tout $i \in \{1, \ldots, n\}$, $\nu(x_i) = \pi_i$
et telle que $t, \nu, \epsilon \models \phi$

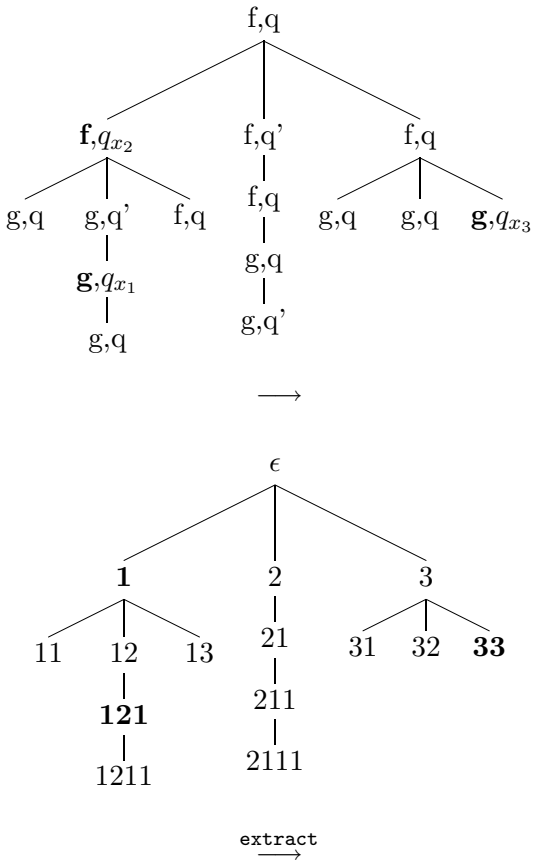
Enfin, la conclusion vient de $\texttt{query}_{\mathcal{A}_S,S} = \texttt{query}_{\phi,(x_1,\ldots,x_n)}$.

□

4.3 Corollaire

Corollaire 1. *Pour toute formule $\phi(x_1, \ldots, x_n)$ de* MSO$[\mathbb{T}]$ *à n-variables libres, il existe un ensemble fini B de prédicats binaires MSO-définissables tel que ϕ est équivalente à une formule de* FO$[\mathbb{T} \cup B]$

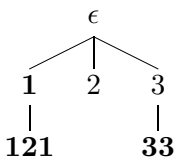
Démonstration. D'après le théorème 5, il existe $\psi \in \texttt{COMP}[MSO]$ telle que $\texttt{query}_{\phi(x_1,\ldots,x_n)} = \texttt{query}_{\psi,(x_1,\ldots,x_n)}$. La formule de composition ψ est définie sur un ensemble fini M de requêtes monadiques MSO-définissables. Or, d'après le lemme 9, il existe un ensemble fini B de prédicats binaires MSO-définissables et une formule γ de FO$[\mathbb{T} \cup B]$ tels que $\texttt{query}_{\phi,(x_1,\ldots,x_n)} = \texttt{query}_{\gamma(x_1,\ldots,x_n)} = \texttt{query}_{\phi(x_1,\ldots,x_n)}$, ce qui permet de conclure. □



$\texttt{extract}(t) = \{\epsilon, 1, 121, 2, 3, 33\}$

$\xrightarrow{\texttt{pref}}$

$\texttt{pref}(\texttt{extract}(t)) =$



$\longrightarrow$

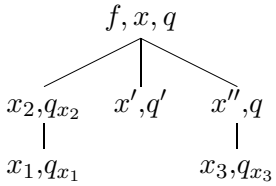


FIG. 4.4 – transformation d'un arbre, son run et un triplet de composantes sélectionnées, en configuration

Chapitre 5

Composition dans les arbres ordonnés d'arité non bornée

Comme nous l'avons souligné dans l'Introduction, les documents XML/HTML sont naturellement représentés par des *arbres d'arité non bornée finie* ou *unranked trees*, ie pour lesquels le nombre de fils d'un noeud n'est pas borné. Par exemple, dans un arbre HTML correspondant à une liste, le nombre de fils du noeud étiqueté par LU n'est pas borné (voir figure 5.1).

Nous allons montrer comment étendre la composition dans les arbres d'arité non bornée, en passant par un codage des arbres d'arité non bornée dans les arbres d'arité bornée.

5.1 Requêtes dans les arbres d'arité non bornée

5.1.1 Arbres d'arité non bornée

Définissons les arbres d'arité non bornée et voyons quel vocabulaire leur est associé, pour les voir comme des structures.

Les arbres d'arité non bornée sont définis comme les arbres d'arité bornée, mais sans restriction sur le nombre de fils des noeuds.

Définition 29 (Domaine d'arbre d'arité non bornée). Un *domaine d'arbre d'arité non bornée* est un sous-ensemble fini D de $\mathbb{N}^*$ (monoïde libre sur $\mathbb{N}$) tel que :

1. si $\pi \in D$ et π' est un préfixe de π, alors $\pi' \in D$
2. si $\pi i \in D$ pour $i \in \mathbb{N}$, alors pour tout $0 < j < i,\ \pi j \in D$

Définition 30 (Arbres ordonnés d'arité non bornée). Un *arbre ordonné t d'arité non bornée* sur un alphabet Σ est une application d'un domaine d'arbre D d'arité non bornée vers Σ. L'ensemble D des noeuds de l'arbre sera aussi noté $\texttt{dom}(t)$.

Enfin, l'ensemble des arbres ordonnés d'arité non bornée sera noté T_Σ.

Remarquons que $T_\Sigma = \bigcup_{k\in\mathbb{N}} T_\Sigma^k$.

Les notions de sous-arbre, d'ordre préfixe, de taille et de hauteur sont naturellement étendues sans difficulté aux arbres d'arité non bornée.

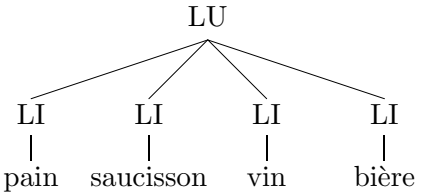


FIG. 5.1 – Une liste en HTML représenté par un arbre d'arité non bornée

5.1.2 Arbres d'arité non bornée comme structures [15]

Un arbre d'arité non bornée $t \in T_\Sigma$ peut être vu comme une structure sur le vocabulaire $\mathbb{T}^u = \{\texttt{label}_a(.), \texttt{first\_child}(.,.), \texttt{next\_sibling}(.,.)\}_{a\in\Sigma}$ de domaine $\texttt{dom}(t)$, où les prédicats $\texttt{label}_a$ sont interprétés comme d'habitude, et où :

$$\begin{array}{lcl} \texttt{first\_child}^t & = & \{(\pi, \pi 1) \mid \pi 1 \in \texttt{dom}(t)\} \\ \texttt{next\_sibling}^t & = & \{(\pi i, \pi(i+1)) \mid \pi(i+1) \in \texttt{dom}(t)\} \end{array}$$

5.1.3 Requêtes

La définition d'une requête n-aire est aussi naturellement étendue aux arbres d'arité non bornée par :

Définition 31 (requête sur arbres d'arité non bornée). Une *requête n-aire* q sur les arbres d'arité k est une application qui à tout arbre $t \in T_\Sigma$ associe un sous-ensemble de $\texttt{dom}(t)^n$:

$$\forall t \in T_\Sigma \; q(t) \subseteq \texttt{dom}(t)^n$$

Les requêtes par formules MSO définies de la même manière que dans les arbres d'arité non bornée, mais sur le vocabulaire $\mathbb{T}^u$. La figure 5.2 donne l'exemple d'une requête $query_{\psi(x,y)}$ qui sélectionne toutes les paires de noeuds (π, π') tel que π est étiqueté par a, et π' est un frère droit de π. Dans cet exemple, π est un frère de π' si et seulement si $t \models \texttt{next\_sibling}^*(\pi, \pi')$.

$\texttt{next\_sibling}^*(x, y) \equiv$

$$\begin{array}{ll} \forall X & [\, x \in X \wedge \\ & \forall z_1 \forall z_2 \; (\texttt{next\_sibling}(z_1, z_2) \wedge z_1 \in X \rightarrow z_2 \in X)] \\ & \quad \rightarrow \; y \in X \end{array}$$

$$\psi(x, y) = \texttt{label}_a(x) \wedge \texttt{next\_sibling}^*(x, y)$$

FIG. 5.2 – Requête MSO sur les arbres d'arité non bornée

5.1.4 Automates d'arbres d'arité non bornée et requêtes régulières

Similairement au cas des arbres d'arité bornée, il existe des notions d'automates sur de tels arbres, tel que si on étend la notion de régularité aux langages d'arbres d'arité non bornée, le théorème de Thatcher et Wright (2) est toujours valide [19, 15]. Nous ne présentons pas

ici en détail les automates, mais nous citons par exemple les *automates à pas* ([8]), qui utilise directement une construction algébrique des arbres d'arité non bornée, et les automates à *haies* [19], dont les règles de transitions sont de la forme $f(L) \to q$, où L est un langage rationnel de mots d'états de l'automate. Ces deux notions d'automates sont équivalentes [8], en terme de reconnaissancance de langage, et en terme de runs. On parlera donc d'automates d'arbres d'arité non bornée. La notion de *run* est de même étendue au cas des automates d'arité non bornée.

On définit de manière similaire au cas bornée les requêtes par automates :

Définition 32 (Requêtes par automates [17]). Étant donné un automate d'arbres $\mathcal{A}$ d'arité non bornée, un entier n, et un ensemble $S \subseteq \texttt{states}(\mathcal{A})^n$ appelé *ensemble de sélection*, la *requête* $\texttt{query}_{\mathcal{A},S}$ *par l'automate* $\mathcal{A}$ *et l'ensemble de sélection* S est définie par :

$$\forall t \in T_\Sigma \quad \texttt{query}_{\mathcal{A},S}(t) = \{(\pi_1, \ldots, \pi_n) \mid \begin{array}{l}\text{il existe un run acceptant } r \text{ sur } t \\ \text{tel que } (r(\pi_1), \ldots, r(\pi_n)) \in S\}\end{array}$$

Une requête est dite *régulière* si elle est égale à une requête par automate.

Remarque 9. On peut indifféremment parler de requêtes par automates à pas et de requêtes par automates à haies, puisque les deux formalismes de requêtes sont équivalents [8].

Le théorème de Thatcher et Wright est toujours vrai :

Théorème 6. *Une requête est régulière si et seulement si elle est* MSO$[\mathbb{T}^u]$*-définissable.*

5.2 Composition dans les arbres d'arité non bornée

On définit la composition dans les arbres d'arité non bornée via un codage vers les arbres binaires. La requête de composition sera donc définie sur le codage binaire des arbres. Notons $\texttt{bin}(t)$ le codage d'un arbre d'arité non bornée $t \in T_\Sigma$ dans l'ensemble des arbres $T^2_{\Sigma'}$ sur un alphabet Σ'. Ce codage définit donc une bijection $\texttt{corr}_t$ de l'ensemble des noeuds de $\texttt{dom}(t)$ dans l'ensemble des noeuds de $\texttt{dom}(\texttt{bin}(t))$. Soit Q un ensemble de requêtes monadiques sur les arbres binaires. Soit $\phi \in \texttt{COMP}[Q]$ une formule de composition dans les arbres binaires sur $T^2_{\Sigma'}$, alors la requête $\texttt{query}^u_{\phi,\overrightarrow{x}}$ dans les arbres d'arité non bornée est définie par :

$$\forall t \in T_\Sigma, \qquad \texttt{query}^u_{\phi,\overrightarrow{x}} = \texttt{corr}_t^{-1}(\texttt{query}_{\phi,\overrightarrow{x}}(\texttt{bin}(t)))$$

Définissons maintenant le codage binaire $\texttt{bin}(t)$ d'un arbre d'arité non bornée t.

5.2.1 Codage binaire

Nous utilisons un codage «$\texttt{first\_child} - \texttt{next\_sibling}$»[15]. Intuitivement, si un noeud $\pi \in \texttt{dom}(\texttt{bin}(t))$ est deuxième fils d'un noeud $\pi' \in \texttt{dom}(\texttt{bin}(t))$, alors, c'est que $\texttt{corr}_t^{-1}(\pi)$ est le frère droit immédiat de $\texttt{corr}_t^{-1}(\pi')$; si c'est le premier fils d'un noeud π' qui a deux fils, alors, $\texttt{corr}_t^{-1}(\pi)$ est le premier fils de $\texttt{corr}_t^{-1}(\pi')$; par contre, si c'est le premier fils d'un noeud π' qui n'a qu'un seul fils, alors il faut pouvoir déterminer si $\texttt{corr}_t^{-1}(\pi)$ et $\texttt{corr}_t^{-1}(\pi')$ sont liés par une relation de frères, ou s'ils sont liés par une relation de fils. Pour cela, nous enrichissons l'alphabet avec des nouveaux symboles qui gardent cette information.

Soit $\Sigma' = \Sigma \cup (\Sigma \times \{\texttt{se}, \texttt{sf}, \texttt{sef}\})$. Intuitivement un noeud $\pi \in \texttt{dom}(\texttt{bin}(t))$ est étiqueté par $(a, \texttt{se})$ si et seulement si $\texttt{corr}_t^{-1}(\pi)$ est étiqueté par a dans t, et est sans enfants (se).

Un noeud $\pi \in \mathtt{dom}(\mathtt{bin}(t))$ est étiqueté par $(a, \mathtt{sf})$ si et seulement si $\mathtt{corr}_t^{-1}(\pi)$ est étiqueté par a dans t, et est sans frères (sf). Un noeud $\pi \in \mathtt{dom}(\mathtt{bin}(t))$ est étiqueté par $(a, \mathtt{sef})$ si et seulement si $\mathtt{corr}_t^{-1}(\pi)$ est étiqueté par a dans t, et est sans frères et sans enfants (sef). Enfin, un noeud $\pi \in \mathtt{dom}(\mathtt{bin}(t))$ est étiqueté par $a \in \Sigma$ si et seulement si $\mathtt{corr}_t^{-1}(\pi)$ a un frère et un enfant. La figure 5.3 donne un exemple de codage. Plus formellement, si on note $\mathtt{addr}(t)$ la fonction qui à un arbre t associe 1 si t est une feuille, et 2 sinon, la correspondance de domaines $\mathtt{corr}_t : \mathtt{dom}(t) \to \mathtt{dom}(\mathtt{bin}(t))$ est définie récursivement par :

$$\begin{array}{lcl} \mathtt{corr}_t(\epsilon) & = & \epsilon \\ \mathtt{corr}_{f(t_1,\ldots,t_n)}(i\pi) & = & 1.\mathtt{addr}(t_1)\ldots\mathtt{addr}(t_{i-1}).\mathtt{corr}_{t_i}(\pi) \end{array}$$

Enfin, l'arbre $\mathtt{bin}(t)$ est définie par l'application :

$$\begin{array}{lcccl} \mathtt{bin}(t) & : & \mathtt{corr}_t(\mathtt{dom}(t)) & \to & \mathtt{dom}(t) \\ & & \pi & \mapsto & \begin{cases} \mathtt{label}(\mathtt{corr}_t^{-1}(\pi)) & \text{si } \mathtt{corr}_t^{-1}(\pi) \text{ a un fils et un frère} \\ (\mathtt{label}(\mathtt{corr}_t^{-1}(\pi)), \mathtt{se}) & \text{si } \mathtt{corr}_t^{-1}(\pi) \text{ a un frère mais pas de fils} \\ (\mathtt{label}(\mathtt{corr}_t^{-1}(\pi)), \mathtt{sf}) & \text{si } \mathtt{corr}_t^{-1}(\pi) \text{ a un fils mais pas de frères} \\ (\mathtt{label}(\mathtt{corr}_t^{-1}(\pi)), \mathtt{sef}) & \text{si } \mathtt{corr}_t^{-1}(\pi) \text{ n'a ni fils ni frères} \end{cases} \end{array}$$

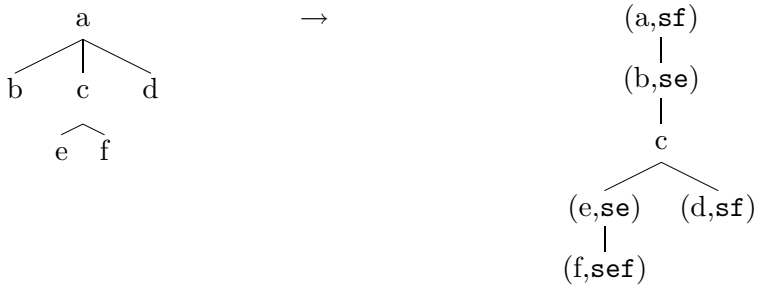


FIG. 5.3 – Codage binaire

5.2.2 MSO-complétude

Similairement au cas borné, nous montrons la MSO-complétude de la composition dans le cas des arbres d'arité non bornée :

Théorème 7 (MSO-complétude). *Pour toute requête* $\mathtt{query}_{\psi(x_1,\ldots,x_n)}$, $\mathrm{MSO}[\mathbb{T}^u]$*-définissable, il existe un ensemble fini* Q *de requêtes monadiques régulières sur les arbres de* $T^2_{\Sigma'}$*, et une formule de composition* $\phi \in \mathtt{COMP}[Q]$*, telle que* $(x_1, \ldots, x_n) \in \mathrm{VL}(\phi)$ *et telle que :*

$$\forall t \in T_\Sigma, \quad \mathtt{query}_{\psi(x_1,\ldots,x_n)}(t) = \mathtt{corr}_t^{-1}(\mathtt{query}_{\phi,(x_1,\ldots,x_n)}(\mathtt{bin}(t)))$$

Réciproquement, pour toute formule de composition $\phi \in \mathtt{COMP}[Q]$ *pour un ensemble fini de requêtes monadiques régulières* Q*, telle que* $x_1, \ldots, x_n \in VL(\phi)$*, il existe une formule* $\psi(x_1, \ldots, x_n)$ $\mathrm{MSO}[\mathbb{T}^u]$*-définissable telle que :*

$$\forall t \in T_\Sigma, \quad \mathtt{query}_{\psi(x_1,\ldots,x_n)}(t) = \mathtt{corr}_t^{-1}(\mathtt{query}_{\phi,(x_1,\ldots,x_n)}(\mathtt{bin}(t)))$$

Démonstration. Pour prouver ce théorème, nous définissons un codage $[\![.]\!]_r$ des formules de MSO$[\mathbb{T}^u]$ vers les formules de MSO$[\mathbb{T}]$, tel que leurs modèles soient égaux modulo le codage binaire (ie que l'équation (1) définie plus loin est satisfaite).

Réciproquement, nous définissons un codage $[\![.]\!]_u$ des formules de MSO$[\mathbb{T}]$ vers les formules de MSO$[\mathbb{T}^u]$ tel que leurs modèles soient équivalents modulo le codage binaire.

Ainsi, il suffira d'utiliser le théorème 5 dans le cas borné pour conclure.

Les codages sont définis figure 5.4.

Il faut maintenant prouver que pour tout arbre $t \in T_\Sigma$, et pour tout $t' \in T^2_{\Sigma'}$:

– $t, \nu \models_{MSO[\mathbb{T}^u]} \phi$ ssi $\mathtt{bin}(t), \mathtt{corr}_t \circ \nu \models_{MSO[\mathbb{T}]} [\![\phi]\!]_r$ (1)

– $t', \nu \models_{MSO[\mathbb{T}]} \phi$ ssi $\mathtt{bin}^{-1}(t'), \mathtt{corr}_{t'}^{-1} \circ \nu \models_{MSO[\mathbb{T}^u]} [\![\phi]\!]_u$ (2)

La preuve se fait par induction sur la structure des formules, et la difficulté se situe au niveau des cas de bases. Nous donnons simplement une idée de la preuve de (1) dans le cas de la formule $\mathtt{next\_sibling}(x, y)$: si $t \models \mathtt{next\_sibling}(\pi, \pi')$, alors π est le frère de π, donc si π n'a pas d'enfants, alors, $\mathtt{corr}_t(\pi')$ est le premier fils de $\mathtt{corr}_t(\pi)$ dans le codage binaire, et s'il a des enfants, alors c'est le deuxième fils, donc, $\mathtt{bin}(t), \mathtt{corr}_t(\pi), \mathtt{corr}_t(\pi') \models [\![\mathtt{next\_sibling}(x, y)]\!]_r$.

Réciproquement, si $\mathtt{bin}(t), \mathtt{corr}_t(\pi), \mathtt{corr}_t(\pi') \models [\![\mathtt{next\_sibling}(x, y)]\!]_r$, alors comme $t, \mathtt{corr}_t(\pi) \models \mathtt{se}(x) \vee \neg\mathtt{se}(x)$, alors si $t, \mathtt{corr}_t(\pi) \models \mathtt{se}(x)$, $\mathtt{corr}_t(\pi')$ est le premier enfant de $\mathtt{corr}_t(\pi)$, et par le codage, π' est bien un frère de π. L'autre cas est similaire.

Soit maintenant une formule $\psi(x_1, \ldots, x_n)$ de $MSO[\mathbb{T}^u]$ et $t \in T_\Sigma$, alors par l'équation (1), on a : $\mathtt{query}_{\psi(\overrightarrow{x})}(t) = \mathtt{corr}_t^{-1}(\mathtt{query}_{[\![\psi(\overrightarrow{x})]\!]_r}(\mathtt{bin}(t)))$ et d'après le lemme 10, il existe une famille finie de requêtes monadiques régulières Q sur les arbres de $T^2_{\Sigma'}$, et une formule de composition $\phi \in \mathtt{COMP}[Q]$ telle que $\overrightarrow{x} \in \mathrm{VL}(\phi)$ et $\mathtt{query}_{[\![\psi(\overrightarrow{x})]\!]_r}(\mathtt{bin}(t)) = \mathtt{query}_{\phi,\overrightarrow{x}}(\mathtt{bin}(t))$; d'où $\mathtt{query}_{\psi(\overrightarrow{x})}(t) = \mathtt{corr}_t^{-1}(\mathtt{query}_{\phi,\overrightarrow{x}}(\mathtt{bin}(t)))$.

Réciproquement, soit Q une famille finie de requêtes monadiques régulières, soit ϕ une formule de composition sur Q tel que $\overrightarrow{x} \in \mathrm{VL}(\phi)$, et soit $t \in T_\Sigma$, alors par le lemme 7, il existe une formule $\psi(\overrightarrow{x})$ MSO$[\mathbb{T}]$-définissable telle que $\mathtt{query}_{\phi,\overrightarrow{x}}(\mathtt{bin}(t)) = \mathtt{query}_{\psi(\overrightarrow{x})}(\mathtt{bin}(t))$. Alors par l'équation (2), on a $\mathtt{corr}_t^{-1}(\mathtt{query}_{\phi,\overrightarrow{x}}(\mathtt{bin}(t))) = \mathtt{query}_{[\![\psi(\overrightarrow{x})]\!]_u}(\mathtt{bin}^{-1}(\mathtt{bin}(t))) = \mathtt{query}_{[\![\psi(\overrightarrow{x})]\!]_u}(t)$. Ce qui termine la preuve.

□

Définissons tout d'abord quelques formules sur les arbres d'arité non bornée :
- $\texttt{se}(x) \equiv \forall y\ \neg\texttt{first\_child}(x,y)$
- $\texttt{sf}(x) \equiv \forall y\ \neg\texttt{next\_sibling}(x,y)$
- $\texttt{sef}(x) \equiv \texttt{se}(x) \wedge \texttt{sf}(x)$

Arité bornée vers arité non bornée

$$
\begin{array}{lcl}
[\![\texttt{label}_{(a,\texttt{se})}(x)]\!]_u & \equiv & \texttt{label}_a(x) \wedge \texttt{se}(x) \\
[\![\texttt{label}_{(a,\texttt{sf})}(x)]\!]_u & \equiv & \texttt{label}_a(x) \wedge \texttt{sf}(x) \\
[\![\texttt{label}_{(a,\texttt{sef})}(x)]\!]_u & \equiv & \texttt{label}_a(x) \wedge \texttt{sef}(x) \\
[\![\texttt{label}_a(x)]\!]_u & \equiv & \texttt{label}_a(x) \\
[\![\texttt{child}_1(x,y)]\!]_u & \equiv & \left\{ \begin{array}{l} (\bigvee_{a\in\Sigma} [\![\texttt{label}_{(a,\texttt{se})}(x)]\!]_u) \rightarrow \texttt{next\_sibling}(x,y) \wedge \\ (\bigvee_{a\in\Sigma} [\![\texttt{label}_{(a,\texttt{sf})}(x)]\!]_u) \rightarrow \texttt{first\_child}(x,y) \wedge \\ (\bigvee_{a\in\Sigma} [\![\texttt{label}_a(x)]\!]_u) \rightarrow \texttt{first\_child}(x,y) \end{array} \right. \\
[\![\texttt{child}_2(x,y)]\!]_u & \equiv & \texttt{next\_sibling}(x,y) \wedge \neg\texttt{se}(x) \\
[\![x \in X]\!]_u & \equiv & x \in X \\
[\![\exists x\phi]\!]_u & \equiv & \exists x [\![\phi]\!]_u \\
[\![\exists X\phi]\!]_u & \equiv & \exists X [\![\phi]\!]_u \\
[\![\neg\phi]\!]_u & \equiv & \neg [\![\phi]\!]_u \\
[\![\phi_1 \wedge \phi_2]\!]_u & \equiv & [\![\phi_1]\!]_u \wedge [\![\phi_2]\!]_u
\end{array}
$$

Arité non bornée vers arité bornée

$$
\begin{array}{lcl}
[\![\texttt{label}_a(x)]\!]_r & \equiv & \texttt{label}_a(x) \vee \texttt{label}_{(a,\texttt{se})}(x) \vee \texttt{label}_{(a,\texttt{sf})}(x) \ \vee \texttt{label}_{(a,\texttt{sef})}(x) \\
[\![\texttt{first\_child}(x,y)]\!]_r & \equiv & \texttt{child}_1(x,y) \ \wedge \ \neg \bigvee_{a\in\Sigma} \texttt{label}_{(a,\texttt{se})}(x) \\
[\![\texttt{next\_sibling}(x,y)]\!]_r & \equiv & ([\![\texttt{se}(x)]\!]_r \rightarrow \texttt{child}_1(x,y)) \wedge ([\![\neg\texttt{se}(x)]\!]_r \rightarrow \texttt{child}_2(x,y)) \\
[\![x \in X]\!]_r & \equiv & x \in X \\
[\![\exists x\phi]\!]_r & \equiv & \exists x [\![\phi]\!]_r \\
[\![\exists X\phi]\!]_r & \equiv & \exists X [\![\phi]\!]_r \\
[\![\neg\phi]\!]_r & \equiv & \neg [\![\phi]\!]_r \\
[\![\phi_1 \wedge \phi_2]\!]_r & \equiv & [\![\phi_1]\!]_r \wedge [\![\phi_2]\!]_r
\end{array}
$$

FIG. 5.4 – Codages des formules MSO

Chapitre 6

Conclusion

Nous avons défini un formalisme pour les requêtes n-aires dans les arbres d'arité bornée, basé sur la composition de requêtes monadiques. Nous avons donné un algorithme efficace pour répondre aux requêtes, et avons prouvé que si les requêtes monadiques sont régulières, alors toutes les requêtes que ce formalisme permet d'exprimer sont régulières, et réciproquement, toute requête régulière (et donc MSO-définissable) est exprimable dans ce formalisme, avec des requêtes monadiques régulières. Nous en avons déduit un corollaire important : toute formule MSO à n-variables libres sur les arbres d'arité bornée est équivalente à une formule de FO sur des prédicats **binaires** MSO-définissables.

Nous avons ensuite montré la nécessité de représenter les arbres comme des arbres d'arité non bornée, dans le contexte XML/HTML, et avons étendu le formalisme de composition à de tels arbres, via un codage des arbres d'arité non bornée vers les arbres binaires.

Perspectives Plusieurs pistes sont possibles. Nous aimerions pouvoir définir un langage de composition directement dans les arbres d'arité non bornée, sans passer par le codage binaire, ayant l'expressivité de MSO. Cela doit être possible en se restreignant non pas aux sous-arbres, mais au sous-haies (on garde aussi toute la partie à droite – par la relation de frère – du noeud sélectionné). La preuve de MSO-complétude doit être alors similaire au cas borné, en adaptant la notion de configurations aux automates à haies.

Il serait aussi intéressant d'étendre le langage de composition aux arbres non ordonnés. Il existe plusieurs formalismes logiques sur les arbres non-ordonnés [6] : MSO[`child`(.,.)], avec un seul prédicat `child` qui dénote la relation père-fils, CMSO, extension de MSO[`child`(.,.)] avec la possibilité de compter modulo des entiers ($|X| = i[j]$), et PMSO, extension de CMSO avec des formules de Presburger (comparaison entières $\leq$ et divisibilité $|$) sur les cardinaux des ensembles dénotés par les variables du second-ordre. Á chaque formalisme logique correspond une notion d'automate, on peut alors étendre le théorème de Thatcher et Wright à ces trois formalismes. Nous pensons que toute requête MSO[resp. CMSO, PMSO]-définissable sur les arbres non ordonnés est exprimable comme une composition de requêtes monadiques MSO[resp. CMSO, PMSO]-définissables. Cela impliquerait que toute formule MSO[resp. CMSO, PMSO]-définissable sur les arbres non ordonnés, à n variables libres, serait équivalente à une formule de FO sur les arbres non ordonnés, définies sur des prédicats **binaires** MSO[resp. CMSO, PMSO]-définissables. Il pourrait alors être intéressant de trouver une structure (les graphes par exemple), avec sa logique associée (dans le cas des graphes, MSO[`edge(.,.)`], où `edge` dénote la relation d'arête), pour laquelle ce corollaire ne serait

plus vrai. De manière plus générale, il serait intéressant de bien formaliser la «décomposition »d'une formule n-aire en formules p-aires, $p < n$, et d'établir certaines propriétés générales sur les structures et les logiques pour lesquelles cette décomposition est possible.

En outre, il serait intéressant d'implémenter ce formalisme, en utilisant des bibliothèques de requêtes monadiques, et en offrant des facilités pour exécuter une requête directement sur un fichier html ou xml.

Enfin, nous allons étudier les propriétés d'apprenabilité du formalisme de composition, en relation avec les travaux de Julien Carme et Patrick Marty, de l'équipe Grappa [1]. Il serait alors intéressant de développer une application d'apprentissage intéractif de wrappers, comme dans le logiciel Squirrel [3] de l'équipe Grappa, mais orienté vers l'apprentissage de la composition. Par là même, il faudra étudier certains formalisme de composition, pour lesquels il existerait un algorithme de réponse plus efficace, l'objectif étant d'obtenir une complexité linéaire en la taille du résultat de la requête. De plus, il serait important de trouver une implémentation efficace de l'algorithme de réponse aux requêtes compositionnelles, afin d'obtenir une meilleure complexité que celle qui a été donnée dans le chapitre *Langage de composition*.

Bibliographie

[1] Équipe GRAPPA. http ://www.grappa.univ-lille3.fr.

[2] Extensible Markup Language (XML). http ://www.w3.org/XML/.

[3] Logiciel Squirrel. http ://www.grappa.univ-lille3.fr/ carme/squirrel.

[4] Projet INRIA MOSTRARE. http ://www.grappa.univ-lille3.fr/mostrare.

[5] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *The VLDB Journal*, pages 119–128, 2001.

[6] Iovka Boneva and Jean-Marc Talbot. Automata and Logics for Unranked and Unordered Trees. In *RTA '05*, volume 3467 of *LNCS*, pages 500–515. Springer-Verlag, 2005.

[7] Julien Carme, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Interactive learning of node selection queries in tree structured documents. In *IJCAI Workshop on Grammatical Inference*, 2005.

[8] Julien Carme, Joachim Niehren, and Marc Tommasi. Querying unranked trees with stepwise tree automata. In Vincent van Oostrom, editor, *International Conference on Rewriting Techniques and Applications, Aachen*, volume 3091 of *Lecture Notes in Computer Science*, pages 105–118. Springer-Verlag, June 2004.

[9] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on : http ://www.grappa.univ-lille3.fr/tata, 1997.

[10] H. Ebbinghaus, J. Flum, and W. Thomas. *Finite Model Theory*. Springer-Verlag, Berlin, 1999.

[11] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. In *Proc. LICS '02 : Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pages 215–224, Washington, DC, USA, 2002. IEEE Computer Society.

[12] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The lixto data extraction project — back and forth between theory and practice. In *PODS '04*, 2004.

[13] Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of languages for web information extraction. In *Symposium on Principles of Database Systems (PODS '02)*, pages 17–28, 2002.

[14] Georg Gottlob and Christoph Koch. Monadic queries over tree-structured data. In *Proc. LICS '02*, Copenhagen, 2002.

[15] Leonid Libkin. Logics over unranked trees : an overview. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP'05)*, 2005.

[16] Frank Neven and Thomas Schwentick. Query automata. In *PODS '99 : Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 205–214, New York, NY, USA, 1999. ACM Press.

[17] Joachim Niehren, Laurent Planque, Jean-Marc Talbot, and Sophie Tison. N-ary queries by tree automata. In *19th International Workshop on Unification (VUIF '05)*, May 2005.

[18] Thomas Schwentick. On diving in trees. In *MFCS '00 : Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 660–669, London, UK, 2000. Springer-Verlag.

[19] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1) :57–81, 1968.

[20] Victor Vianu. A web odyssey : from codd to xml. In *PODS '01 : Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–15, New York, NY, USA, 2001. ACM Press.