UNIVERSITÉ DES SCIENCES ET TECHONOLOGIES DE LILLE
LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

THÈSE

présentée en première version en vue d'obtenir le grade de Docteur,
spécialité Informatique

par

Emmanuel Filiot

LOGIQUES POUR REQUÊTES n-AIRES DANS LES ARBRES

(LOGICS FOR n-ARY QUERIES IN TREES)

Thèse soutenue le 13 Octobre 2008 devant le jury composé de :

Mr	GIORGIO GHELLI	Università di Pisa	Rapporteur
Mr	CHRISTOF LÖDING	RWTH Aachen	Rapporteur
Mr	LUC SEGOUFIN	INRIA, ENS Cachan	Rapporteur
Mr	JEAN-FRANÇOIS RASKIN	Université Libre de Bruxelles	Examinateur
Mme	SOPHIE TISON	Université de Lille 1	Directrice
Mr	JEAN-MARC TALBOT	Université de Provence	Co-Directeur
Mme	LAURENCE DUCHIEN	Université de Lille 1	Présidente

REMERCIEMENTS

L'accomplissement d'une thèse ne se fait pas seul et j'aimerais remercier ici toutes les personnes qui ont rendu possible cette entreprise.
Mes remerciements vont d'abord à ma directrice de thèse Sophie Tison. Il m'apparaît difficile de lui témoigner toute ma gratitude tant les raisons en sont nombreuses. Je me contenterais donc simplement de la remercier pour m'avoir donné le goût de la recherche, pour sa disponibilité et son humanité. Pour tout cela et tout ce que je ne dis pas, merci Sophie !
J'aimerais remercier mon co-directeur Jean-Marc Talbot, pour m'avoir guidé et précieusement conseillé, pour sa pédagogie et, pour m'avoir permis occasionnellement de goûter aux douceurs du soleil marseillais.
Durant ma thèse, j'ai travaillé avec Joachim Niehren. Je le remercie chaleureusement pour ses nombreux conseils, très précieux, lorsque l'on découvre le monde de la recherche. J'ai aussi fait la rencontre d'Arnaud Durand, à qui je voudrais témoigner mon amitié et ma joie de collaborer avec lui.
Je remercie Giorgio Ghelli, Christof Löding, et Luc Ségoufin, pour m'avoir fait l'honneur d'être rapporteurs de ma thèse. Je vous suis très reconnaissant d'avoir ajouté cette lourde tâche à vos emplois du temps déjà chargés. Je remercie aussi Jean-François Raskin pour avoir accepté le rôle d'examinateur, et pour m'avoir offert la possibilité de poursuivre ma vocation dans la recherche au sein de son équipe.
Je tiens profondément à remercier tous les membres de l'équipe Mostrare qui ont rendu mon quotidien si agréable pendant ma thèse. J'ai une pensée particulière pour Yves Roos, dont j'ai partagé le bureau avec joie, ainsi que pour Rémi Gilleron, qui m'a accepté dans son équipe et m'a soutenu depuis le début. Enfin, j'aimerais remercier l'ensemble des doctorants et post-doctorants qui ont fait de cette thèse une belle expérience humaine et en particulier, mes collègues et amis Olivier Gauwin et Sławek Staworko.
Je remercie tous mes amis, ma famille, et ma belle-famille pour m'avoir soutenu durant la thèse. J'ai une pensée pour Marion, Gauthier et leur nouvelle venue Célia. Enfin, je témoigne toute mon amitié à Julien Tierny, une des plus belles rencontres que m'ait offert cette thèse.
Je remercie aussi tous mes amis MIM, avec qui a germé l'idée d'accomplir une thèse, avec une pensée particulière pour Christoph et mon partenaire au club zik-saucisson-vodka Samuel.
J'aimerais remercier et témoigner toute mon affection à mes parents. Merci infiniment de votre écoute, de vos conseils et de votre inconditionnel soutien. Je remercie également mes frères, Alexandre, Pierre-Louis et David, pour toutes ces heures joyeuses passées avec vous.
Enfin mes plus profonds remerciements vont à Camille, qui m'accompagne depuis toutes ces années. Cette thèse c'était aussi la tienne, tu l'as acceptée dans ton quotidien tout en me supportant, avec ta gentillesse et ta générosité. C'est un immense bonheur de t'avoir rencontrée.

Résumé en Français Beaucoup de données informatiques sont structurées de manière arborescente. Dans le contexte du Web, c'est le cas en particulier des données au format XML. De par sa généricité, ce format est rapidement devenu un standard pour l'échange et la sauvegarde d'informations.
A l'instar des langages de requêtes pour les bases de données relationnelles, le besoin d'avoir des langages de requêtes pour les documents XML est devenu crucial. On distingue les requêtes unaires (sélection d'un ensemble de sous-parties d'un document) des requêtes n-aires (sélection d'un ensemble de n-uplets de sous-parties d'un document). Beaucoup de formalismes logiques pour les requêtes unaires ont été étudiés, en revanche, peu d'approches logiques existent pour les requêtes n-aires.
Cette thèse étudie de manière fondamentale les requêtes n-aires, en proposant et en étudiant principalement deux formalismes logiques pour requêtes n-aires: une extension du paradigme navigationnel du standard W3C XPath au cas n-aire, appelée langage de composition, et une adaptation de la logique spatiale d'arbres TQL, introduite par Cardelli et Ghelli.
Les questions de pouvoir d'expressivité, de complexité d'évaluation des requêtes ainsi que leur satisfiabilité sont abordées. L'étude du problème de satisfiabilité pour la logique TQL a nécessité l'introduction de nouveaux automates d'arbres avec tests globaux, dont l'étude est réalisée de manière indépendante.

Titre en Anglais Logics for n-Ary Queries in Trees.

Résumé en Anglais In computer science many data are shaped as trees. In the context of the Web, it is the case for XML formatted data in particular. XML is a markup language that has rapidly become a standard for information storage and data exchange.
As query languages for relational databases are not well-suited to XML data, the need to have query languages specific to XML documents has increased. We distinguish unary queries which select a set of subparts of a document from n-ary queries which select a set of n-tuples of subparts of a document. Many logical formalisms for unary queries have been proposed, but less work has been done on logical formalisms for n-ary queries.
This thesis is a fundamental study of n-ary queries that proposes two logical formalisms for n-ary queries: an extension of the navigational paradigm of the W3C standard XPath to n-ary queries, called the *composition language*, and an adapation of the spatial logic *TQL* introduced by Cardelli and Ghelli.
The question of expressive power, the complexity of the query evaluation problem as well as the satisfiability problem are considered. In particular, the satisfiability problem for a TQL fragment is proved to be decidable by reduction to the emptiness test of a new class of tree automata with global constraints that is studied independently.

Mots clés en Français XML, requêtes n-aires, logique, arbres, TQL, XPath.

Mots clés en Anglais XML, n-ary queries, logic, trees, TQL, XPath.

Laboratoire de préparation de la thèse LIFL - UMR USTL/CNRS 8022
Bâtiment M3
59655 Villeneuve d'Ascq Cédex - FRANCE

CONTENTS

FOREWORD

This dissertation presents my research work done during the last 3 years for the obtention of a Ph.D. in Computer Science, under the supervision of Pr. Jean-Marc Talbot and Pr. Sophie Tison. This research has been partially funded by *Conseil Régional du Nord-Pas de Calais* and by *Institut National de Recherche en Informatique et en Automatique* (INRIA). This work was done in the *INRIA Lille-Nord Europe* Institute and in the *Laboratoire d'Informatique Fondamentale de Lille (LIFL)*, within the INRIA project-team Mostrare. It was also supported by the national ANR project *XML Transformation Languages: logic and applications (TraLaLA)* and the national ANR project *Algorithms and complexity for answer enumeration (ENUM)*.

The main results of this thesis concern logical formalisms to define n-ary queries. Two query languages are presented in particular: a query composition language and its relationship to XPath 2.0, and the TQL spatial logic and its correspondence with a novel class of tree automata with global constraints, called *TAGED*. The work on the composition language and XPath was achieved with the precious collaboration of Joachim Niehren (INRIA, project-team Mostrare). It has been presented in (1, 2). The work on TQL and TAGED has been presented in (3). TAGED are also more specifically studied in (4) where an application to unification problems with membership constraints is proposed, but not presented in this dissertation.

I started several other works during the thesis period that are not presented in this manuscript. In (5), we study the variable independence problem (previously introduced for databases) for n-ary queries defined in monadic second-order logic in trees. I also collaborated with Slawomir Staworko (INRIA project-team Mostrare, University at Buffalo) and Jan Chomicki (University at Buffalo) on the consistent query answering problem for n-ary queries in trees (6). Finally, I started a collaboration with Olivier Gauwin (INRIA project-team Mostrare) and Pr. Arnaud Durand (Paris 7 University) on efficient enumeration algorithms for conjunctive queries over trees.

AUTHOR'S PUBLICATIONS

[1] EMMANUEL FILIOT, JOACHIM NIEHREN, JEAN-MARC TALBOT AND SOPHIE TISON: Composing monadic queries in trees. In *PLAN-X International Workshop*. 2006. BRICS, pp. 61–70. (Cited page ix.)

[2] EMMANUEL FILIOT, JOACHIM NIEHREN, JEAN-MARC TALBOT AND SOPHIE TISON. Polynomial time fragments of XPath with variables. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles*

of Database Systems (PODS). 2007. ACM press, pp. 205–214. Available at http://hal.inria.fr/inria-00135678/en. (Cited page ix.)

[3] EMMANUEL FILIOT, JEAN-MARC TALBOT AND SOPHIE TISON: Satisfiability of a spatial logic with tree variables. In *Proceedings of the EACSL Annual Conference on Computer Science Logic (CSL)*. 2007. LNCS, Springer Verlag, pp. 130–145. Available at hal.inria.fr/inria-00148462/en. (Cited page ix.)

[4] EMMANUEL FILIOT, JEAN-MARC TALBOT AND SOPHIE TISON: Tree automata with global constraints. In *International Conference on Developments in Language Theory (DLT)*. 2008, LNCS, Springer Verlag. To appear. Available at http://hal.inria.fr/inria-00292027/en. (Cited page ix.)

[5] EMMANUEL FILIOT AND SOPHIE TISON: Regular n-ary queries in trees and variable independence. In *IFIP International Conference on Theoretical Computer Science (IFIP TCS)*. 2008. To appear. Available at http://hal.inria.fr/inria-00274648/en. (Cited page ix.)

[6] SLAWOMIR STAWORKO, EMMANUEL FILIOT AND JAN CHOMICKI: Querying Regular Sets of XML Documents. In *International Workshop on Logic in Databases (LiD)*. 2008. Available at http://hal.inria.fr/inria-00275491/en/. (Cited page ix.)

1 INTRODUCTION

1.1 MOTIVATIONS AND OBJECTIVES

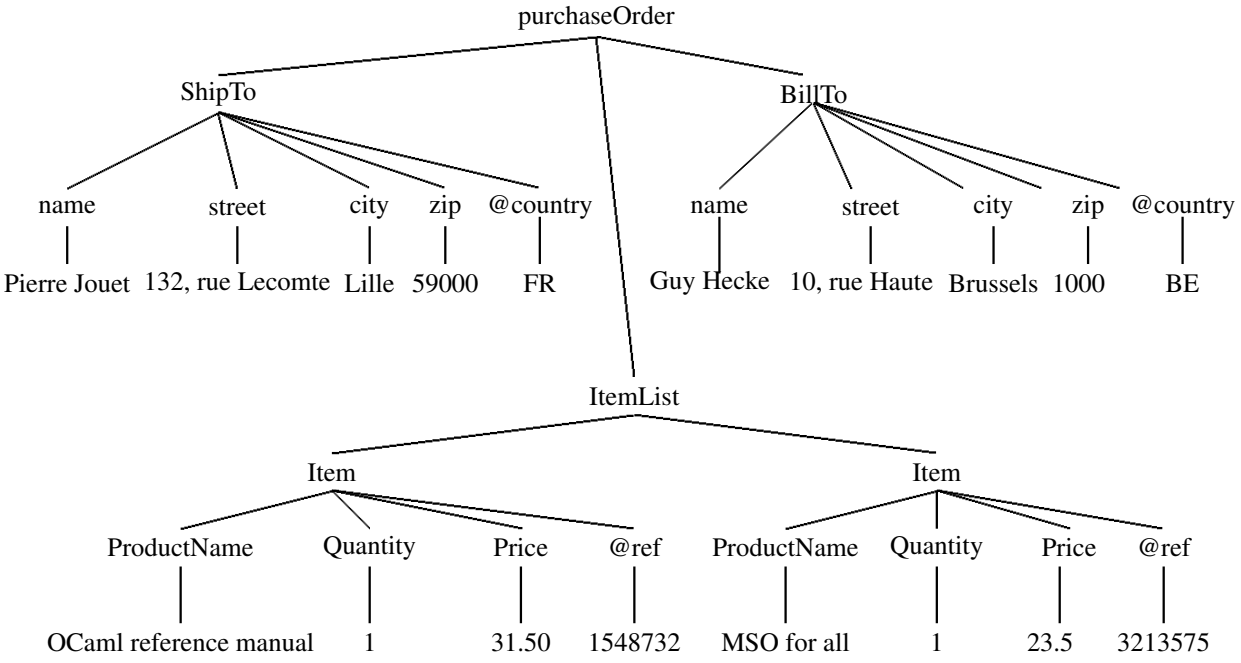
Query languages for relational databases have extensively been studied by the database community. In the context of the Web however, data tend to be shaped as trees, as proved by the growing popularity of markup languages such as HTML and XML (BPSM*06). Relational database systems are not well-suited to handle tree-structured data. Therefore, with the development of web applications, the need to have database tools specific to tree-structured data has increased. In particular, the ability to query XML data has become crucial because XML has emerged as a standard for data exchange and information storage. This has strengthened the necessity to have query languages designed specifically for XML data. Formal approaches to XML query languages aim at better understanding their practical issues. XML documents are naturally modeled as trees. Although trees have been widely studied in computer science research, XML applications have provided new issues that require to go back and forth between theory and practice.

XML documents are *semi-structured*: the structure and the data are not separated. This makes XML a very flexible format that allows to easily exchange data coming from different applications or database systems. Fig. 8.1 gives two examples of XML documents. The first document represents a purchase order with the shipment and billing addresses, as well as the purchased items. The second document represents a shipment order related to the purchase order. XML documents are structured by opening and closing *tags* which have to be well-matched, such as `<city>` and `</city>`. A pair of respective opening and closing tags may contain well-matched sequences of tags or unstructered data by means of raw text. Opening tags may also contain *attributes*, that is pairs of name and values. The nested structure of XML documents naturally yields a tree representation of them. The respective tree representations of the seller and shipper orders are depicted in Fig. 8.2. Each node is labeled by some symbol and can have an arbitrary number of children. The children are naturally ordered by the sequential order of the respective XML documents. Attributes are represented by additional branches whose node labels are preceded by `@`. The underlying tree model of XML documents is best known as *node labeled unranked trees*, in contrast to node labeled *ranked* trees, for which the number of children of any node is fixed.

```
<purchaseOrder orderDate="2008-06-30">
  <ShipTo country="FR">
    <name>Pierre Jouet</name>
    <street>132, rue Lecomte</street>
    <city>Lille</city>
    <zip>59000</zip>
  </ShipTo>
  <ItemList>
    <Item ref="1548732">
      <ProductName>OCaml reference manual</ProductName>
      <Quantity>1</Quantity>
      <Price>31.50</Price>
    </Item>
    <Item ref="3213575">
      <ProductName>MSO for all</ProductName>
      <Quantity>1</Quantity>
      <Price>23.5</Price>
    </Item>
  </ItemList>
  <BillTo country="BE">
    <name>Guy Hecke</name>
    <street>10, rue Haute</street>
    <city>Brussels</city>
    <zip>1000</zip>
  </BillTo>
</purchaseOrder>

<shipItem>
    <address>
      <name>Pierre Jouet</name>
      <street>132, rue Lecomte</street>
      <city>Lille</city>
      <zip>59000</zip>
      <country>FR</country>
    </address>
    <productRefs>
       <ref>1548732</ref>
       <ref>3213575</ref>
    </productRefs>
</shipItem>
```

Figure 1.1: Seller and shipper XML orders.



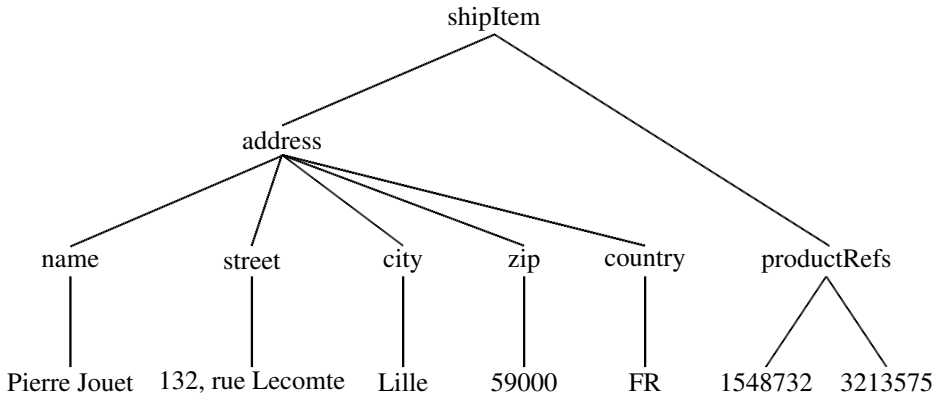


Figure 1.2: Tree representations of seller and shipper XML orders.

The structure and the syntax of a collection of XML documents can be constrained by *schema languages*, a well-known instance of them being *Document Type Definition* (DTD). For example, a DTD may constrain an XML shipment order to be enclosed by `shipItem` tags, in which the address always occurs before the product references, such as in Fig. 8.1. Now, suppose that the seller has to order the shipment of the products listed in the purchase order of Fig. 8.1. Since the shipper application only reads XML shipment order of a specific format, the seller application has to transform the purchase order into a compatible shipment order. This can be done for instance thanks to XML document transformations based on XML queries.

Querying a tree consists in selecting relevant parts of it. The parts of the tree that are selected are usually subtrees of it, often represented by their root nodes. Those queries are consequently called *node-selecting queries*. They are the core of many XML applications, such as document transformation (MBPS05), web information extraction (GK04), XML programming (HP03b, BCF03a), and data exchange (AL08). *Unary* queries select sets of nodes in a tree, while n-ary queries select sets of n-tuples of nodes. Consider again the purchase and shipment orders of Fig. 8.2. As a first step of the transformation of the purchase order into a shipment order, one needs to select the address as well as the product references. Accessing the address can be done thanks to the 5-ary query which select the name, the street, the city, the zip code and the country. Selecting the product references is done via a unary query. Those results can be recombined to construct an XML shipment order that conforms to the shipper schema.

The study of tree formalisms benefits from a long history of research by the formal languages and finite model theory communities. Most of them were designed for ranked trees. However since the growing popularity of XML, several tree formalisms have been revisited and proposed under the perspective of query languages for unranked trees. Those formalisms can be divided into declarative formalisms (mainly tree logics) and procedural formalisms (mainly tree automata). Those two categories have many connections, and procedural formalisms are often used as a computational model for declarative formalisms. One of the best known connection is the correspondence between the monadic second-order logic and finite tree automata (TW68).

Most of the query formalisms which have been proposed are for unary or binary queries (Lib06). Except procedural n-ary query formalisms (NV02, NPTT05), logical n-ary query formalisms have received less attention, although they constitute an important issue of XML research, as mentioned in (GKS04, Lib06, ABL07, Sch07). The main objective of this dissertation is to propose and study logical formalisms for n-ary queries. Two languages are proposed in this dissertation. The first language is a query composition language that allows one to turn any unary or binary query language into a full n-ary query language. One sub-objective was to obtain a composition language with a reasonable expressive power, that admits a polynomial-time algorithm to answer queries. The second language proposition is an adaptation of the *Tree Query Logic* (TQL) to ordered unranked trees as TQL was defined for unordered unranked trees (CG04). The main sub-objective was to define a decidable fragment of TQL as full TQL is undecidable (with respect to the satisfiability problem). Both languages have variables by which to select components of the answer tuples.

1.2 Overview of the Dissertation

As two n-ary query languages are proposed, the dissertation is divided into two independent parts.

1.2.1 Composition Language

The W3C standard XPath (BBC*07) is an XML navigational language by which to select sets of nodes by navigating in the document with path expressions. For instance, the path expression

$$/\text{child} :: \text{BillTo}/\text{child} :: \text{name}/\text{child} :: *$$

selects in the first tree of Fig. 8.2 the name of the person the bill has to be sent to. The '/' is a composition operator. The query is interpreted as follows: starting from the root, go down to a child labeled BillTo, then go down to a child labeled name, and finally access the child labeled by the name of the person (the star means that any label can be matched). XPath is the node selecting core of many other XML languages, such as for instance the query language XQuery (BCF*07), the transformation language XSLT (Cla99), the schema language XML Schema (FW04), or the addressing language XPointer (DMJ01). The essential navigational core of XPath (without syntactic sugar) is formally defined in Chapter 2. Path expressions are usually viewed as unary queries, as they are evaluated relatively to the root, but can also be viewed as binary queries, as they can be evaluated relatively to any node of the tree. They relate starting nodes to ending nodes of the navigation. In particular, a pair of nodes (u, v) is selected by a path expression p if v can be reached starting from u and following the path p.
The composition language extends the XPath navigational paradigm to full n-ary queries. In general, binary queries q can also be used to navigate in trees t: starting from a node u, one can go to a node v with q if (u, v) is a solution of q in t. The idea of the composition language is to use binary queries to navigate in the document, and to capture the components of the output tuples with node variables. The language is then closed by intersection and disjunction to add more expressive power. The main operator of the composition language is the composition operator $\circ$. For instance, selecting a triple (ProductName, Price, Reference) in the first tree of Fig. 8.2 can be done as follows: go from the root to a product name, capture it by some variable x, then go from the product name to the price, capture it by some variable y, and finally go from the price to the reference, and capture it by some variable z. The navigational parts of this query can be expressed in any binary query formalism, XPath for instance. Formally, this composition query is written as follows:

$$p_{prod} \circ x \circ p_{price} \circ y \circ p_{ref} \circ z$$

where p_{prod}, p_{price} and p_{ref} are binary queries defined for instance by the following XPath path expressions[1]:

$$\begin{array}{lcl} p_{prod} & = & /\text{child} :: \text{ItemList}/\text{child} :: \text{Item}/\text{child} :: \text{ProductName}/\text{child} :: * \\ p_{price} & = & /\text{parent} :: */\text{next-sibling} :: \text{Quantity}/\text{next-sibling} :: \text{Price}/\text{child} :: * \\ p_{ref} & = & /\text{parent} :: */\text{next-sibling} :: @\text{ref}/\text{child} :: * \end{array}$$

[1] parent (resp. next-sibling) relate a node to its parent (resp. right nearest sibling)

Of course there are many ways to define this query, and in general, the definition of a query strongly depends on what it is known *a priori* about the form of the documents on which it is applied. Composition queries over a binary query language L are denoted by $\mathcal{C}(L)$. We now review the main results obtained for $\mathcal{C}(L)$, in terms of expressiveness and complexity of query evaluation. These results are proved in Chapter 3.

Query evaluation takes a query and a tree as input, and outputs the query answers. Its associated decision problem is called the *model-checking problem*. It takes a tree, a node tuple and a query as input, and consists in checking whether the tuple is an answer to the query in the given tree. Any model-checking algorithm can be transformed into a query evaluation algorithm as follows: generate all node tuples and filter them by the model-checking algorithm. Although this approach is still reasonable for small trees and queries of small arities, it quickly becomes intractable in other cases. Indeed, trees can be very large and arities quite high. For instance, the XML purchase order of Fig. 8.1 may be part of a long sequence of purchase orders, and may contain other informations that we want to query altogether in a tuple, such as some comments of the buyer, the required shipment mode, the buyer identifier, etc. In this thesis, we define a simple and expressive fragment of $\mathcal{C}(L)$, denoted $\mathcal{C}^{\text{nvs}}(L)$. The idea is simply to disallow the possibility to use the same variable in both parts of a composition, which would create cycles. This notion is related to the well-known notion of *acyclicity* of conjunctive queries over binary predicates. However, it is much simplier to decide and composition queries are closed by disjunction, which is not the case for conjunctive queries. The fragment $\mathcal{C}^{\text{nvs}}(L)$ admits a query evaluation algorithm which is polynomial in the size of the query, in the size of the tree, and also in the size of the output, as long as the query evaluation for L is polynomial.

First-order logic (FO) and *monadic-second order logic* (MSO) are two logics that are widely accepted as standard logics to which the other query formalisms are often compared (Lib06). They can express n-ary queries, thanks to their variables, but their evaluation is intractable, when considering both the query and the tree as input (*combined complexity*). However, when the query is not part of the input, the evaluation becomes tractable, and even linear in the size of input plus the size of the output (Bag06, Cou07), but involves large constants that we want to avoid. We prove that $\mathcal{C}^{\text{nvs}}(L)$ matches the expressiveness of FO and MSO with respect to n-ary queries, provided L is also as expressive as FO and MSO respectively, with respect to binary queries. The proof uses folklore results based on the Shelah's composition method (She, Tho84, Sch00, Mar05a). Based on this proof, we propose a way to compose unary queries from a language L, which still matches the expressiveness of FO and MSO, provided L does. It relies on a domain restriction called *subhedge restriction*.

In contrast to the first version of XPath, XPath 1.0 (CD99), its second version XPath 2.0 (BBC*07) has node variables. While XPath 2.0 is still considered as a unary query language, we show that it can also be used as an n-ary query language. In Chapter 4, we present two n-ary query languages based on the composition language and XPath. The first language extends with variables the FO-expressive Marx's extension of XPath called *Conditional XPath* (Mar05a). The second language is an FO-expressive fragment of XPath 2.0. Both languages admit polynomial-time query evaluation.

1.2.2 Tree Query Logic

The *Tree Query Logic* (TQL) is a spatial logic introduced in (CG04) to query XML documents represented as unordered trees. It can express n-ary queries thanks to variables. The query evaluation problem of TQL queries has already been investigated in (CFG02). Satisfiability and expressiveness of variable-free fragments of TQL have been studied in (BTT05, Bon06). However, nothing is known about the satisfiability of TQL fragments which can define n-ary queries, *ie* fragments with variables. In Chapter 6, we study the satisfiability problem of TQL fragment with variables, on **ordered** trees. The study of TQL in the context of unranked ordered trees was mentioned as an open issue in (CG04, Gen06, Bon06). TQL is a query language in the spirit of the pattern-matching languages of XDuce and CDuce (HP03b, BCF03a). The TQL formula

$$\text{purchaseOrder}[\text{ShipTo}[\top] \wedge X \mid \top]$$

matches the first tree of Fig. 8.2, but not the second one. The result of this matching is the binding which associates with X the subtree whose root is labeled by "ShipTo". Every node label is viewed as tree constructor. The symbol $|$ is also a constructor which takes two sequences of trees and concatenate them. The formula $\top$ is matched by any sequence of trees. As shown by the previous example, variables denote trees, in contrast to the variables of the composition language. By repeating the same variable, equality tests can be performed. The formula

$$\text{purchaseOrder}[\top \mid \text{ItemList}[\top|X|\top|X|\top] \mid \top]$$

is successfully matched against a tree representing a purchase order if the item list contains two identical items. By using negation, disequality tests can also be done:

$$\text{purchaseOrder}[\top \mid \text{ItemList}[\top|X|\top|\neg X|\top] \mid \top]$$

is successfully matched if the item list contains at least two different items. TQL formulas may also contain fixpoint operators to allow recursion. For instance, the formula

$$\text{purchaseOrder}[\top \mid \text{ItemList}[\mu\xi.(X|\xi \vee X)] \mid \top]$$

is successfully matched if the item list contains only identical items.
The main difficulty to decide the satisfiability of TQL formulas comes from their ability to test equalities or inequalities of whole subtrees. We define the *bounded fragment*, where disequality tests can be done only a bounded number of times along any root-to-leaf path.
The satisfiability problem for the bounded fragment is solved by reduction to the emptiness test of a new class of tree automata with global equality and disequality constraints (TAGEDs). Tree equality and disequality tests can be done between subtrees which are arbitrarily faraway, in contrast to usual automata with constraints, where tests are done only between children and cousins (CDG*07). TAGEDs are studied in Chapter 5, which can be read independently of the other chapters. It is proved that TAGED definable languages are effectively closed by intersection and union. However, they are not closed by complement. Moreover, TAGEDs are not determinizable and universality is undecidable. The emptiness problem is proved to be decidable for several subclasses: *positive TAGEDs*, which performs only equality tests, and *negative TAGEDs*, which performs only disequality tests. We also prove decidability of the emptiness problem for a subclass ,

called *vertically bounded TAGEDs*, that allows both kind of tests, but in a bounded manner. In particular, only a bounded number of disequality tests can be performed along any root-to-leaf path, similarly to the bounded TQL fragment. A natural correspondence between vertically bounded TAGEDs and an extension of MSO with tree equality tests is given. The satisfiability of this MSO extension is therefore decidable.
In Chapter 6, we formally define our adaptation of TQL on unranked ordered trees, illustrated by many query examples. We prove that the model-checking problem is polynomial, and finally we give a reduction of TQL formulas into TAGEDs. This reduction introduces a novel type of construction that deals with the variables of the logic.

1.2.3 Organization of the Dissertation

Chapter 2 introduces the notions of trees, queries, the classical logics FO and MSO, and tree automata. It also gives an overview of the main existing query languages and logics for trees.
Chapter 3 introduces the composition language, gives a query evaluation algorithm and proves the FO and MSO expressiveness results. A complete comparison with conjunctive queries is also given.
Chapter 4 presents two applications of the composition language by means of two FO-expressive n-ary query languages: Conditional XPath with variables and a fragment of XPath 2.0.
Chapter 5 introduces TAGEDs and their properties. The emptiness problem for several subclasses is also investigated, and the extension of MSO with tree equality tests is introduced.
Chapter 6 defines the TQL logic, illustrated by several examples, gives a model-checking algorithm, and finally proves a correspondence between TQL and TAGEDs.

TREES AND QUERIES

2

CONTENTS

THIS chapter introduces important preliminary notions used in the next chapters, such as tree models, n-ary queries, and several related decision problems. Existing query languages are then presented, for Boolean, unary, and n-ary queries. In particular, they are surveyed through the questions of expressiveness and complexity of query evaluation, model-checking, inclusion, satisfiability.

2.1 TREE MODELS

We consider finite unranked ordered trees whose nodes are labeled over a finite or infinite alphabet. In particular, a node can have a finite but unbounded number of ordered children. Those trees are commonly admitted as a model of XML documents (Nev02). In the rest of this thesis, they are just called *unranked trees*. Hedges are another model which underlies unranked trees: a hedge is a finite sequence of unranked trees while an unranked tree is a hedge rooted by a single node.
We also define ranked trees as some results of this thesis are proved for ranked trees and then lifted to unranked trees via binary encodings.

2.1.1 Alphabets

An *(unranked) alphabet* Σ is a finite set of symbols often denoted by f, g, a.
A *ranked alphabet* is a pair (Σ_r, ar) where Σ_r is a finite set of symbols, and $\mathsf{ar} : \Sigma_r \to \mathbb{N}$ is total *arity* function. A symbol $f \in \Sigma_r$ is said to be $\mathsf{ar}(f)$-ary. The 0-ary symbols are called *constants*. The function ar is often omitted and considered as implicit. For all $k \in \mathbb{N}$, Σ_r is called k-ary if every symbol $f \in \Sigma_r$ has arity 0 or k.
A *weakly ranked alphabet* is a pair $(\Sigma_r, \mathsf{ar}(\Sigma_r))$ where $\mathsf{ar}(\Sigma_r) \in \mathbb{N}$. Symbols from Σ_r may have multiple arities, but these arities are bounded by $\mathsf{ar}(\Sigma_r)$.

2.1.2 Logical Structures

A *signature* (or *vocabulary*) σ is a finite set of relational symbols[1] $R_1, R_2, \ldots, R_n$. Each relational symbol R is associated with a natural number $\mathsf{ar}(R)$ called its *arity*. A finite *σ-structure* M is a tuple $M = (\mathrm{Dom}(M), R_1^M, R_2^M, \ldots, R_n^M)$ where $\mathrm{Dom}(M)$ is a finite set called the *domain*, and for all relational symbols, R_i^M is an $\mathsf{ar}(R_i)$-ary relation on $\mathrm{Dom}(M)$. We always assume that a signature contains an equality predicate $=$ interpreted on M by the identity on $\mathrm{Dom}(M)$.
Finally, we will sometimes consider constant symbols c in the signature, which are nothing else than unary relation symbols interpreted by singleton sets.

2.1.3 Unranked Trees

Let Σ be an unranked alphabet. We let $\sigma_{unr}(\Sigma) = \{\prec_{ns}, \prec_{fc}, (\mathsf{lab}_a)_{a\in\Sigma}\}$ be the signature[2] of unranked trees over Σ, where the predicates $\prec_{ns}$ and $\prec_{fc}$ are binary, and the predicates lab_a are unary. A *labeled ordered unranked tree* t over Σ is a finite $\sigma_{unr}(\Sigma)$-structure $t = (\mathrm{Dom}(t), \prec_{ns}^t, \prec_{fc}^t, (\mathsf{lab}_a^t)_{a\in\Sigma})$ such that:

- elements of $\mathrm{Dom}(t)$ are called *nodes*;
- $\prec_{fc}^t$ relates a node to its *first-child*. Hence it must satisfy: for all $u \in \mathrm{Dom}(t)$ there exists at most one node $v \in \mathrm{Dom}(t)$ such that $u \prec_{fc}^t v$;
- $\prec_{ns}^t$ relates a node to its *next-sibling*. Hence it must satisfy: for all $u \in \mathrm{Dom}(t)$ there is at most one node $v \in \mathrm{Dom}(t)$ such that $u \prec_{ns}^t v$;
- for all $a \in \Sigma$, lab_a^t is a set of nodes labeled a, and for all nodes $u \in \mathrm{Dom}(t)$ there is a unique $a \in \Sigma$ such that $u \in \mathsf{lab}_a^t$ (also denoted $\mathsf{lab}_a^t(u)$);

[1] In general, a signature also contains function symbols but they are not needed in this thesis
[2] When it is clear from the context, we write σ_{unr} instead of $\sigma_{unr}(\Sigma)$

- $(\mathrm{Dom}(t), \prec^t_{fc} \cup \prec^t_{ns})$ is an acyclic directed and connected graph, and every node has a unique incoming edge, except for exactly one node that we call the *root* of the tree;
- the root has no next-sibling;

The set of unranked trees over Σ is denoted by $\mathcal{T}_{unr}(\Sigma)$. Nodes are often denoted by the letters u, v, w, and tuples of nodes by $\overline{u}$. The size of a tree $\|t\|$ is the number of its nodes, *ie* $\|t\| = |\mathrm{Dom}(t)|$.
The following predicates (directly given with their interpretation) are often used:

- $\prec^t_{ns^*}$: the transitive and reflexive closure of $\prec^t_{ns}$;
- $\prec^t_{ch}$: the *child* relation, defined by the composition $\prec^t_{fc} \circ \prec^t_{ns^*}$;
- $\prec^t_{ch_k},\ k \in \mathbb{N}$: the *k-th child successor* relation, which relates a node to its k-th child. Formally, $\prec^t_{ch_k} = \prec^t_{fc} \circ (\prec^t_{ns})^{k-1}$;
- $\prec^t_{ch^*}$: the *descendant* relation, *ie* the transitive and reflexive closure of $\prec^t_{ch}$;
- $\prec^t_{ch^+}$: the transitive closure of $\prec^t_{ch}$;
- root^t: the *root node* of t (root is a constant symbol).

For all nodes $u \in \mathrm{Dom}(t)$, the *children* of u are given by the least set containing the first-child of u (if it exists) and closed by $\prec^t_{ns}$). Note that the children of u are totally ordered by the transitive closure of $\prec^t_{ns}$, that is why unranked trees are said to be *ordered*. For all $k \in \mathbb{N}$, the k-th child of u (if it exists) is denoted by $u.k \in \mathrm{Dom}(t)$. More generally, if $\overline{k} = k_1 k_2 \dots k_n$ is a word over $\mathbb{N}-0$, we denote by $u.\overline{k}$ the node $(\dots((u.k_1).k_2).\dots).k_n$ (if it exists). The node that can be reached from root^t by following the path $\overline{k}$ is denoted $\mathsf{node}^t(\overline{k})$, *ie* $\mathsf{node}^t(\overline{k}) = \mathsf{root}^t.\overline{k}$. A node without children is called a *leaf*. All other nodes are called *inner-nodes*. Finally, for all nodes $u \in \mathrm{Dom}(t)$, $\mathsf{lab}^t(u)$ denotes the label of u in t, *ie* if $a = \mathsf{lab}^t(u)$, then $\mathsf{lab}^t_a(u)$ holds. Fig. 2.1 partially represents the main relations of an unranked tree structure.

Let $u \in \mathrm{Dom}(t)$. The *subtree of t at node u*, denoted $t|_u$, is the unranked tree $t|_u = (\mathrm{Dom}(t|_u), \prec^{t|_u}_{fc}, \prec^{t|_u}_{ns}, (\mathsf{lab}^{t|_u}_a)_{a\in\Sigma})$ where:

- $\mathrm{Dom}(t|_u) = \{v \mid u \prec^t_{ch^*} v\}$
- $\prec^{t|_u}_{fc} = \prec^{t|_u}_{fc} \cap (\mathrm{Dom}(t|_u) \times \mathrm{Dom}(t|_u))$
- $\prec^{t|_u}_{ns} = \prec^{t|_u}_{ns} \cap (\mathrm{Dom}(t|_u) \times \mathrm{Dom}(t|_u))$
- $\forall a \in \Sigma,\ \mathsf{lab}^{t|_u}_a = \mathsf{lab}^t_a \cap \mathrm{Dom}(t|_u)$

Two trees $t, t' \in \mathcal{T}_{unr}(\Sigma)$ are *equal* (or *isomorphic*), if they are structurally equal. More formally, $t = t'$ if their is a one-to-one mapping Ψ from $\mathrm{Dom}(t)$ into $\mathrm{Dom}(t')$ such that for all nodes $u, u' \in \mathrm{Dom}(t)$ and all $a \in \Sigma$:

$$\begin{array}{lll} u \prec^t_{fc} u' & \text{iff} & \Psi(u) \prec^{t'}_{fc} \Psi(u') \\ u \prec^t_{ns} u' & \text{iff} & \Psi(u) \prec^{t'}_{ns} \Psi(u') \\ \mathsf{lab}^t_a(u) & \text{iff} & \mathsf{lab}^{t'}_a(\Psi(u)) \end{array}$$

Ψ is called an *isomorphism*. We say that t and t' are *edge-isomorphic* (or have the *same shape*) if Ψ respects the edge relations $\prec_{fc}$ and $\prec_{ns}$ but not necessarily the label relation. If $t = t'$ or if t and t' have the same shape, we identify their sets of nodes and assume that $\Psi(u) = u$, for all $u \in \mathrm{Dom}(t)$ (hence $\mathrm{Dom}(t) = \mathrm{Dom}(t')$).

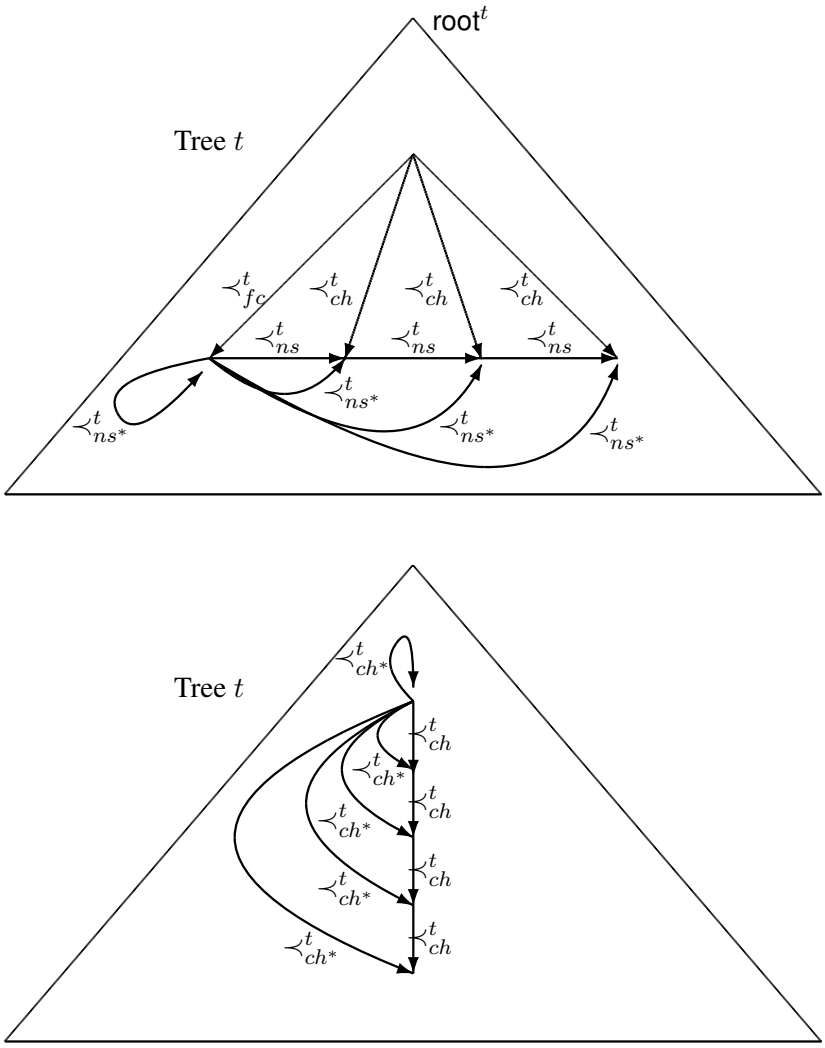


Figure 2.1: Main relations of an unranked tree structure t

2.1.4 Hedges and Tree Operations

A *hedge* h is an ordered finite sequence of unranked trees $t_1, \ldots, t_n$ over Σ. The empty hedge is denoted by $\mathsf{0}$. Hedges can naturally be viewed as $\sigma_{unr}(\Sigma)$-structures, and the set of hedges over Σ is denoted by $\mathcal{H}(\Sigma)$. A hedge have several roots linked by the next-sibling relation, which are the roots of the trees it is composed of. However we distinguish one element root^h which is the root of the left-most tree of h, *ie* the root of t_1.
Let σ be the signature $\{\mathsf{0}, |\} \cup \{a \mid a \in \Sigma\}$, where $\mathsf{0}$ is a constant, $|$ a binary symbol and as are viewed as unary symbols. The *free σ-algebra* is the set of terms h generated by the following grammar:

$$h ::= \mathsf{0} \quad | \quad h|h \quad | \quad a(h)$$

We call *hedge term* an element of the σ-algebra Hedge obtained by quotienting the free σ-algebra by the following three axioms:

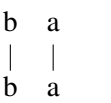
$$\mathsf{0}|h = h \qquad h|\mathsf{0} = h \qquad (h_1|h_2)|h_3 = h_1|(h_2|h_3)$$

Every hedge term h represents a hedge $h^I \in \mathcal{H}(\Sigma)$ via the interpretation $.^I$ defined as follows: $\mathsf{0}^I$ is the $\sigma_{unr}(\Sigma)$-structure with empty domain, $|^I$ takes two hedge structures and concatenate them, and a^I take a hedge structure and enroots it by a node labeled a. This interpretation is extended to terms as follows:

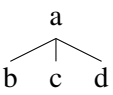
$$(h_1|h_2)^I \;=\; h_1^I \,|^I\, h_2^I \qquad (a(h))^I \;=\; a^I(h^I)$$

To ease the notation, we often identify the set of hedge term and the set of hedge structures. In particular, we omit the interpretation $.^I$ in the notations and freely write, for instance, $a(b(\mathsf{0})|c(\mathsf{0}))$ to denote the hedge structure with three nodes labeled a, b, c respectively. Other examples of the correspondence between hedge term and hedge structures are:

$b(b(\mathsf{0})) \mid a(a(\mathsf{0}))$ $\rightarrow$



$a(b(\mathsf{0})|c(\mathsf{0})|d(\mathsf{0}))$ $\rightarrow$



Finally sometimes we may omit $|$ and write $a(h), b(h'), c(h'')$ or just $a(h)b(h')c(h'')$ instead of $a(h)|b(h')|c(h'')$. Similarly, we sometimes omit $\mathsf{0}$ and write $a(b)$ and a instead of $a(b(\mathsf{0}))$ and $a(\mathsf{0})$ respectively.

2.1.5 Ranked Trees and Binary Encoding

Ranked trees are trees over a ranked alphabet. In particular, the number of children of a node is determined by the arity of its label.
Let Σ_r be a ranked alphabet. The set of ranked trees over Σ_r, denoted $\mathcal{T}_{ran}(\Sigma_r)$, is the set of trees $t \in \mathcal{T}_{unr}(\Sigma_r)$ such that for all nodes $u \in \mathrm{Dom}(t)$, and all $f \in \Sigma_r$, if u is labeled f, then u has exactly $\mathsf{ar}(f)$ children.
If Σ_r is k-ary, the set of ranked trees over Σ_r is also called k-ary and the set of k-ary trees over Σ_r is sometimes denoted $\mathcal{T}_k(\Sigma_r)$, to emphasize that Σ_r is k-ary.
A well-known encoding of unranked trees (or hedges) into binary trees is the *first-child next-sibling* encoding. Many results of the literature are proved for binary trees, and then lifted to unranked trees (or hedges) via this encoding. In particular, every hedge h over an unranked alphabet Σ can be viewed as a binary tree over $\Sigma_r = \Sigma \cup \{\bot\}$ for a fresh symbol $\bot \notin \Sigma$, where $\mathsf{ar}(f) = 2$ for all $f \in \Sigma$, and $\mathsf{ar}(\bot) = 0$. This is done via the binary encoding $\mathsf{enc}^{\mathrm{bin}}$ defined as follows over hedges:

$$\mathsf{enc}^{\mathrm{bin}}(\mathsf{0}) = \bot \qquad \mathsf{enc}^{\mathrm{bin}}(a(h)|h') = a(\mathsf{enc}^{\mathrm{bin}}(h), \mathsf{enc}^{\mathrm{bin}}(h'))$$

Conversely, any binary tree over Σ_r represents a hedge over Σ via the following decoding $\mathsf{dec}^{\mathrm{bin}}$:

$$\mathsf{dec}^{\mathrm{bin}}(\bot) = \mathsf{0} \qquad \mathsf{dec}^{\mathrm{bin}}(f(t_1, t_2)) = f(\mathsf{dec}^{\mathrm{bin}}(t_1))|\mathsf{dec}^{\mathrm{bin}}(t_2)$$

This encoding is illustrated in Fig. 2.2.

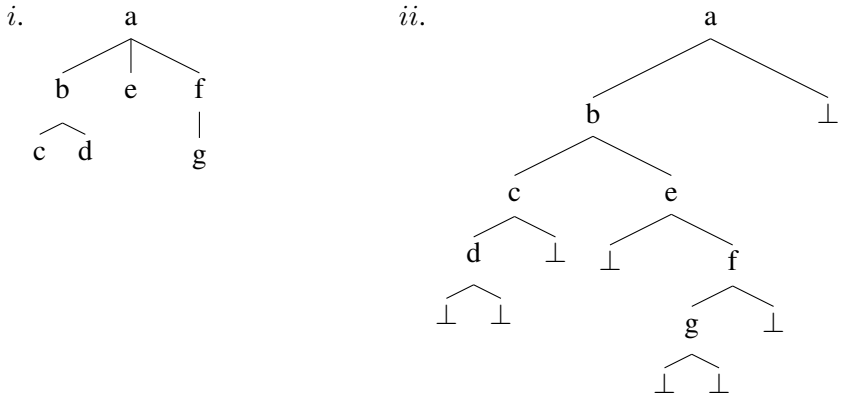


Figure 2.2: An unranked tree (i) and its *first-child next-sibling* encoding (ii).

Weakly Ranked Trees Let Σ_r be a weakly ranked alphabet. A weakly ranked tree t over Σ_r is a tree such that every node has less than $\mathsf{ar}(\Sigma_r)$ children. The set of weakly ranked trees over Σ is denoted by $\mathcal{T}_{wran}(\Sigma_r)$.

2.1.6 Contexts

Let $n \in \mathbb{N}$, and Σ be an (unranked) alphabet. An n-ary context C over Σ is a tree over $\Sigma \cup \{\circ_1, \ldots, \circ_n\}$, where $\circ_1, \ldots, \circ_n$ are linearly ordered constant symbols not in Σ and occurring uniquely at the leaves of C. These symbols are often called *holes*. The substitution of the holes $\circ_1, \ldots, \circ_n$ by trees $t_1, \ldots, t_n$ is denoted $C[t_1, \ldots, t_n]$. Note that $C[t_1, \ldots, t_n]$ is a tree over Σ. Figure 2.3 illustrates the substitution of a context by trees.

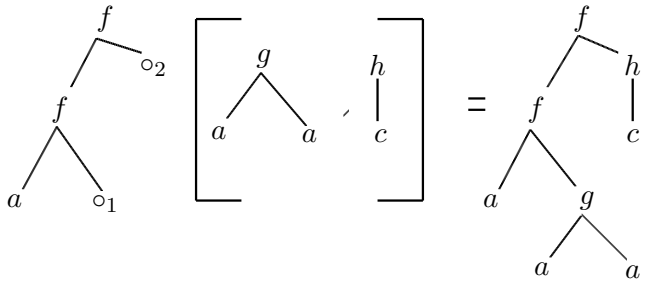


Figure 2.3: Substitution of a binary context by two trees

2.1.7 Trees over an Infinite Alphabet

In the second part of this thesis, we will consider an infinite countable alphabet, which in contrast to finite alphabets will be denoted Λ. All the notions defined before carry over to an infinite alphabet. In particular, in the signature of unranked trees, we assume an infinite set of relational symbols lab_a, $a \in \Lambda$. Only finitely many of them have a non-empty interpretation in a tree.

A set of labels $L \subseteq \Lambda$ is *cofinite* if it is finite or its complement in Λ is finite. The set of cofinite subsets of Λ is denoted $\mathcal{P}_{cf}(\Lambda)$. For instance, $\varnothing$ and Λ are cofinite. Cofinite sets of labels are often denoted α, and their complement $\overline{\alpha}$. Note that $\mathcal{P}_{cf}(\Lambda)$ is stable by complement, intersection, and union. Cofinite sets, although potentially infinite, admit a simple finite representation.
We extend the relational symbols lab_a to cofinite sets α with the following interpretation:

$$\mathsf{lab}^t_\alpha \;=\; \{u \in \mathrm{Dom}(t) \mid \exists a \in \alpha,\ u \in \mathsf{lab}^t_a\}$$

2.2 QUERIES

In this section, we define the notion of n-ary queries and the problems considered in this thesis.

2.2.1 Definition

Let $n \in \mathbb{N}$. An n-ary query q is a mapping from unranked trees over Σ into tuples of nodes of length n:

$$\forall t \in \mathcal{T}_{unr}(\Sigma), \quad q(t) \subseteq (\mathrm{Dom}(t))^n$$

The *arity* of q, denoted $\mathsf{ar}(q)$ is n. When $n = 0$ (resp. $n = 1, n = 2$), q is said to be *Boolean* (resp. *unary or monadic*, and *binary*). Boolean queries are like tree acceptors: $q(t)$ is either empty or equal to $\{()\}$, where $()$ is the empty tuple. If some tuple of nodes $\overline{u}$ is in $q(t)$, we often say that q *select* $\overline{u}$ on t. Note that if two trees t and t' are isomorphic, since we assume that they have the same domain, we also have $q(t) = q(t')$.
Given two queries q, q' of the same arity, we define $q \cap q'$ and $q \cup q'$ by, for all $t \in \mathcal{T}_{unr}(\Sigma)$:

$$\begin{array}{rcl} q \cap q'(t) & = & q(t) \cap q'(t) \\ q \cup q'(t) & = & q(t) \cup q'(t) \end{array}$$

Given an n-ary query q and a m-ary query q', we let $q \times q'$ be the $n+m$-ary query defined by, for all $t \in \mathcal{T}_{unr}(\Sigma)$:

$$(q \times q')(t) \;=\; q(t) \times q'(t)$$

Let $\overline{u} = (u_1, \ldots, u_n)$ be an n-ary tuple and $I \subseteq \{1, \ldots, n\}$. We denote by $\pi_I(\overline{u})$ the tuple $(u_i)_{i \in I}$. For all $i \in \{1, \ldots, n\}$, π_i stands for $\pi_{\{i\}}$. Note that $\pi_\varnothing(\overline{u}) = ()$.
Let q be an n-ary query, we denote by $\pi_I q$ the $|I|$-ary query defined by, for all trees $t \in \mathcal{T}_{unr}(\Sigma)$:

$$\pi_I q(t) \;=\; \{\pi_I(\overline{u}) \mid \overline{u} \in q(t)\}$$

Two queries q and q' are equal, denoted $q = q'$, if for all trees $t \in \mathcal{T}_{unr}(\Sigma)$, $q(t) = q'(t)$.
The definition of n-ary queries q naturally extends to arbitrary σ-structures, for some signature σ:

$$\text{for all } \sigma\text{-structures } M, \qquad q(M) \subseteq (\mathrm{Dom}(M))^n$$

2.2.2 Query Languages

A *query language* L is a tuple $(\mathbb{Q}_L, \mathsf{ar}_L(.), \|.\|_L, \mathcal{Q}_L(.))$ where:

- $\mathbb{Q}_L$ is a set of objects, called *query expressions*, ranged over by q, q';
- $\mathsf{ar}_L(.) : \mathbb{Q}_L \to \mathbb{N}$ is a total function mapping query expressions to their *arity*;
- $\|.\|_L : \mathbb{Q}_L \to \mathbb{N}$ is a total function mapping query expressions to their *size*;
- $\mathcal{Q}_L(.)$ is a total function mapping query expressions q to the query they represent, such that $\mathcal{Q}_L(\mathrm{q})$ is an $\mathsf{ar}(\mathrm{q})$-ary query.

Intuitively, query expressions may represent logical formulas or query automata, as we will further see, and $\mathcal{Q}_L(.)$ are their interpretation as queries in unranked trees over Σ. This definition implicitly considers that queries are defined in unranked trees over Σ. However, we will sometimes consider other classes of structures such as binary trees over some alphabet Σ. In this case, to avoid any ambiguity, we mention explicitly that L is a query language in binary trees over Σ'. We say that L is Boolean (resp. unary, binary) if for all $\mathrm{q} \in \mathbb{Q}_L$, $\mathcal{Q}_L(\mathrm{q})$ is Boolean (resp. unary, binary). More generally, we say that L is n-ary if L can express queries of arbitrary arities, not necessarily fixed (for all $n \in \mathbb{N}$, there exists $\mathrm{q} \in \mathbb{Q}_L$ such that $\mathsf{ar}(\mathrm{q}) = n$). When it is clear from the context, we omit the subscript L.
L is *closed by projection* if for all $\mathrm{q} \in \mathbb{Q}_L$, if $\mathcal{Q}_L(\mathbb{Q})$ is n-ary, then for all $I \subseteq \{1, \ldots, n\}$, there exists a query expression denoted $\pi_I \mathrm{q} \in \mathbb{Q}_L$ such that $\pi_I \mathcal{Q}_L(\mathrm{q}) = \mathcal{Q}_L(\pi_I \mathrm{q})$.
We consider two important notions to compare two query languages $L = (\mathbb{Q}_L, \|.\|_L, \mathcal{Q}_L(.))$ and $L' = (\mathbb{Q}_{L'}, \|.\|_{L'}, \mathcal{Q}_{L'}(.))$.

EXPRESSIVENESS:

L captures L' (or *L is L'-complete*) if for all $\mathrm{q}' \in \mathbb{Q}_{L'}$, there is $\mathrm{q} \in \mathbb{Q}_L$ such that $\mathcal{Q}_{L'}(\mathrm{q}') = \mathcal{Q}_L(\mathrm{q})$. L and L' are *equally expressive*, denoted $L = L'$ if L captures L' and L' captures L.

2.2.3 Query Evaluation and Decision Problems

We define the main evaluation and decision problems we consider in this thesis. Let $L = (\mathbb{Q}_L, \|.\|_L, \mathcal{Q}_L(.))$ be a query language.

MODEL-CHECKING: $MC(\mathrm{q}, t, \overline{u})$

Input: $\mathrm{q} \in \mathbb{Q}_L$, $t \in \mathcal{T}_{unr}(\Sigma)$, $\overline{u} \in \bigcup_{n \in \mathbb{N}} (\mathrm{Dom}(t))^n$
Output: Yes iff $\overline{u} \in \mathcal{Q}_L(\mathrm{q})(t)$

NON-EMPTINESS: $NE(\mathrm{q}, t)$

Input: $\mathrm{q} \in \mathbb{Q}_L$, $t \in \mathcal{T}_{unr}(\Sigma)$
Output: Yes iff $\mathcal{Q}_L(\mathrm{q})(t) \neq \varnothing$

QUERY EVALUATION:

Input: $\mathrm{q} \in \mathbb{Q}_L$, $t \in \mathcal{T}_{unr}(\Sigma)$
Output: $\mathcal{Q}_L(\mathrm{q})(t)$

Note that the model-checking problem is the decision problem associated with the query evaluation problem. If the language is closed by projection, then the non-emptiness problem reduces to the model-checking problem, since $NE(q, t)$ holds iff $MC(\pi_{\varnothing} q, t, ())$ holds. Since all the query languages in the literature are closed by projection, the non-emptiness problem is in general omitted in favor of model-checking.
The model-checking is often used in the literature to measure the complexity of query evaluation. The complexity of model-checking can provide an argument supporting the non-existence of an efficient query evaluation algorithm. However, the query evaluation algorithm based on enumerating all the tuples and checking their membership to the answer set is not efficient in general. Indeed, as said in Chapter 1, arities of queries can be large, making the set of tuples large as well.
Hence the time complexity of model-checking is in general not precise enough to describe the time complexity of query evaluation. One needs a new notion of complexity:

Definition 2.2.1 (Complexity Measure for Query Evaluation) *Let $L = (\mathbb{Q}_L, \mathsf{ar}_L(.), \|.\|_L, \mathcal{Q}_L(.))$ be a query language. We say that the query evaluation problem is in **polynomial-time** for L if there is a polynomial p, and an algorithm A such that for all inputs $q \in \mathbb{Q}$ and $t \in \mathcal{T}_{unr}(\Sigma)$, it outputs $\mathcal{Q}_L(q)(t)$ in time complexity lesser than $p(\|q\|_L, \|t\|, |\mathcal{Q}_L(q)(t)|)$.*

We will also consider the satisfiability problem, as it is of particular importance for static analysis of queries (Gen06).

SATISFIABILITY: $SAT(q, n)$

Input: $q \in \mathbb{Q}_L, n \in \mathbb{N}$
Output: Yes iff there is $t \in \mathcal{T}_{unr}(\Sigma)$ and $\overline{u} \in (\mathrm{Dom}(t))^n$ such that $\overline{u} \in \mathcal{Q}_L(q)(t)$

If for all $q \in \mathbb{Q}_L$ and all $n \in \mathbb{N}$, we can decide whether $SAT(q, n)$ holds, we often say that L is decidable. If L is closed by projection, satisfiability reduces to satisfiability of 0-ary queries. Indeed, $SAT(q, n)$ holds iff $SAT(\pi_{\varnothing} q, 0)$ holds.
We will also consider the following problems in the next sections:

- **inclusion $q \subseteq q'$:** on input $q, q' \in \mathbb{Q}$, output yes iff for all trees t, $\mathcal{Q}(q)(t) \subseteq \mathcal{Q}(q')(t)$;
- **equivalence $q = q'$:** on input $q, q' \in \mathbb{Q}$, output yes iff $q \subseteq q'$ and $q' \subseteq q$.

2.3 Finite Tree Automata

This section introduces finite tree automata. Tree automata are a tree acceptor model which defines the robust class of regular tree languages. It is a procedural formalism (while logics are more declarative) which has found many applications in program analysis, verification, logic programming, linguistic and databases (CDG*07). It has also many connections with logic. Logical formulas can define tree languages (see Sections 2.4 and 2.5) and there is often a correspondence between logical formulas and automata (TW68, Don70, Tho97). In particular, it is good practice to find, for any logical formalism over trees, its corresponding class of tree automata. The models of a formula define a tree languages which should be

definable by a (computable) automaton, and conversely. Decidability of the logic then reduces to emptiness of automata, making automata a very useful tool. Moreover, as a computational formalism, tree automata benefit from good complexity properties, and are often used as a tool for evaluating declarative queries written in a logical formalism (Lib06).
With the development of XML, several automata models for unranked trees have risen. In this context, unranked tree automata models turn out to be useful to compare expressiveness of logics for trees, to evaluate queries, for static analysis of transformation languages, to describe schema languages with a clear semantics (Nev02, Sch04, Sch07). The widely accepted automaton model for unranked trees is *hedge automata* introduced in (BKWM01) based on early works (PQ68, Tak75). They extend ranked tree automata with regular word languages of states in transitions.
Finite (ranked) tree automata are defined in Subsection 2.3.1 and hedge automata are defined in Subsection 2.3.2, based on (CDG*07). Other models of unranked tree automata are also briefly overviewed. Only constructions needed for the rest of this thesis are presented but a full description of (ranked and unranked) tree automata can be found in (CDG*07).

2.3.1 Tree Automata for Ranked Trees

A (bottom-up and finite) tree automaton (FTA) A over Σ is a 4-tuple $A = (\Sigma, Q, F, \Delta)$ where:

- Σ is a **ranked** alphabet;
- Q is a finite set of states;
- $F \subseteq Q$ is a set of *final* (or *accepting*) states;
- $\Delta \subseteq \Sigma \times (\bigcup_{n\in\mathbb{N}} Q^n) \times Q$ is a finite set of rules such that if $(f, (q_1, \ldots, q_k), q) \in \Delta$, then $k = \mathsf{ar}(f)$.

Rules $\delta = (f, (q_1, \ldots, q_k), q) \in \Delta$ are denoted $f(q_1, \ldots, q_k) \to q$, and if $k = 0$, δ is called *initial*. $f(q_1, \ldots, q_k)$ is the left-hand side (lhs) of δ, denoted $\mathrm{lhs}(\delta)$, while q is its right-hand side (rhs), denoted $\mathrm{rhs}(\delta)$. The automaton A is called *deterministic* if for all different rules $\delta, \delta' \in \Delta$, $\mathrm{lhs}(\delta) \neq \mathrm{lhs}(\delta')$.
FTA run on ranked trees in a bottom-up way, starting from the leaves and going up to the root. A tree $f(t_1, \ldots, t_k)$, where $k \in \mathbb{N}$, evaluates to some state $q \in Q$, if there is a rule $f(q_1, \ldots, q_k) \to q$ in Δ such that t_i evaluates to q_i, for all $i \in \{1, \ldots, k\}$. A tree is accepted by A if it evaluates to a final state. This leads to the notion of run. Formally, we let $t \in \mathcal{T}_{ran}(\Sigma)$ and view Q as a weakly ranked alphabet such that $\mathsf{ar}(Q) = max_{f\in\Sigma}\ \mathsf{ar}(f)$. A *run* of A on t is a tree $r \in \mathcal{T}_{ran}(Q)$ such that r is edge-isomorphic to t and r satisfies, for all $k \in \mathbb{N}$ and all $u, u_1, \ldots, u_k \in \mathrm{Dom}(t)$ such that $u_1, \ldots, u_k$ are the children of u (given in order):

$$\mathsf{lab}^t(u)(\mathsf{lab}^r(u_1), \ldots, \mathsf{lab}^r(u_k)) \to \mathsf{lab}^r(u) \in \Delta$$

A run r is *successful* (or *accepting*) if the root of r is labeled by a final state, *ie* $\mathsf{lab}^r(\mathsf{root}^r) \in F$. For all states q, the set of trees which evaluate to q is defined by:

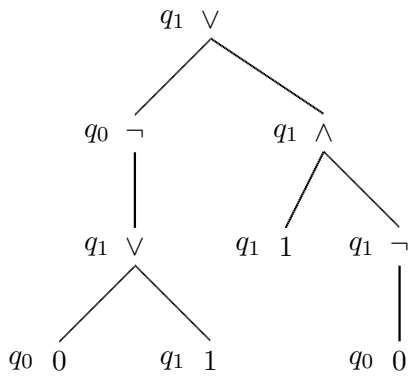


Figure 2.4: A tree over Σ_b and a successful run of A_b on it.

$$\mathcal{L}_q(A) = \{t \in \mathcal{T}_{ran}(\Sigma) \mid \text{there is a run } r \text{ of } A \text{ on } t \text{ such that } \mathsf{lab}^r(\mathsf{root}^r) = q\}$$

The *language* $\mathcal{L}(A)$ *accepted* (or *recognized,defined*) by A is the set $\mathcal{L}(A) = \bigcup_{q\in F} \mathcal{L}_q(A)$. A tree $t \in \mathcal{L}(A)$ is said to be accepted. Consequently, there exists (at least) a successful run of A on each accepted tree. Finally, a language $L \subseteq \mathcal{T}_{ran}(\Sigma)$ is *recognizable* if $L = \mathcal{L}(A)$ for some FTA A.

Example 2.3.1 *Let Σ_b be the ranked alphabet consisting of the binary symbols $\wedge, \vee$, the unary symbol $\neg$, and the constants $0, 1$. Trees of $\mathcal{T}_{ran}(\Sigma_b)$ represent Boolean formulas. We define an automaton A_b on Σ_b which accepts only Boolean formulas logically equivalent to 1. We let its set of states (resp. final states) being equal to $Q_b = \{q_0, q_1\}$ (resp. $F_b = \{q_1\}$), and its set of rules Δ_b is defined by, $\forall b, b_1, b_2 \in \{0, 1\}$, $\forall \oplus \in \{\wedge, \vee\}$:*

$$\begin{array}{rcl@{\qquad\qquad}rcl} b & \to & q_b & \neg(q_b) & \to & q_{\neg b} \\ \oplus(q_{b_1}, q_{b_2}) & \to & q_{b_1 \oplus b_2} & & & \end{array}$$

Figure 2.4 shows an accepted tree over Σ_b together with a successful run of A_b on it.

It is known (CDG*07) that FTA are determinizable, *ie* for each FTA A, there is a deterministic FTA A' such that $\mathcal{L}(A) = \mathcal{L}(A')$ (the number of states of A' however may be exponential in the number of states of A). Moreover (CDG*07):

Proposition 2.3.2 *The class of recognizable ranked tree languages is closed by union, intersection, complement, inverse homomorphisms and linear homomorphisms.*

The closure by intersection is proved by constructing the *product automaton*. This construction is needed in the next chapters, so we define it formally (for binary trees).

product automaton Let $A_1 = (\Sigma, Q_1, F_1, \Delta_1)$ and $A_2 = (\Sigma, Q_2, F_2, \Delta_2)$ be two FTA. The product automaton $A_1 \times A_2$ is defined by: $(\Sigma, Q_1 \times Q_2, F_1 \times F_2, \Delta)$, where:

- $a \to (p, q) \in \Delta$ iff $a \to p \in \Delta_1$ and $a \to q \in \Delta_2$;

- $f((p_1, q_1), (p_2, q_2)) \to (p, q) \in \Delta$ iff $\begin{cases} f(p_1, p_2) \to p \in \Delta_1 \\ f(q_1, q_2) \to q \in \Delta_2 \end{cases}$

decision problems and complexity We briefly define common decision problems for FTA. In order to define their complexities, the size of an FTA has to be defined. Let A be an FTA, and $\delta = f(q_1, \ldots, q_n) \to q$ one of its rules. Every state is of size 1, so that the size of δ, denoted $\|\delta\|$, is $n + 2$. The size of A, denoted $\|A\|$, is $|Q| + \sum_{\delta \in \Delta} \|\delta\|$. Let A, B be two FTA, and $t \in \mathcal{T}_{ran}(\Sigma)$. The maximal arity of a symbol of Σ is denoted by $\mathsf{ar}(\Sigma)$. The following decision problems are standard:

Problem	Input	Output	Complexity
membership	t	$t \in \mathcal{L}(A)$?	$O(\|t\|)$
uniform membership	t,A	$t \in \mathcal{L}(A)$?	- $O(\|t\| + \|A\|)$ if A is deterministic - $O(\|t\| \times \|A\|)$ otherwise
emptiness	A	$\mathcal{L}(A) = \varnothing$?	$O(\|A\|)$
inclusion	A, B	$\mathcal{L}(A) \subseteq \mathcal{L}(B)$?	- $O(\|A\|.\|B\|)$ if B is deterministic (CGLN08) - EXPTIME-complete otherwise
universality	A	$\mathcal{T}_{ran}(\Sigma) = \mathcal{L}(A)$?	- $O((\mathsf{ar}(\Sigma)+1).\|\Sigma\|.\|A\|)$ if A is deterministic - EXPTIME-complete otherwise
finiteness	A	Is $\mathcal{L}(A)$ finite ?	$O(\|Q\| \times \|A\|)$
equivalence	A, B	$\mathcal{L}(A) = \mathcal{L}(B)$?	- PTIME if A, B are deterministic and Σ is fixed - EXPTIME-complete otherwise

More precisely, it is known that the membership problem is ALOGTIME-complete [3], and the uniform membership problem is LOGCFL-complete[4] (Loh01).

Other automaton models for ranked trees

Many other automata models exist for ranked trees. For instance, *top-down tree automata* (CDG*07) starts from the root and go down the leaves. Rules have the form $(f, q) \to (q_1, \ldots, q_k)$. If the current node has k children, is labeled f, and has been evaluated to some state q, the rule can be applied and its children evaluate to $q_1, \ldots, q_k$ respectively. A top-down tree automaton is deterministic if every pair of rules with different lhs have necessarily different rhs. Top-down tree automata are equally expressive as bottom-up tree automata, but deterministic top-down tree automata are strictly weaker than bottom-up FTA. *Tree-walking automata* (TWA)

[3] ALOGTIME is the class of languages decidable in logarithmic time by an alternating Turing machine (Pap94)

[4] LOGCFL is the class of languages logspace reducible to a context-free language. In particular, ALOGTIME$\subseteq$ LOGCFL and this inclusion is conjectured to be strict.

are automata which run in the tree in a sequential way. In particular, they have only one single head which move along the edges of the tree. They are introduced in (AU69), and have known a growing interest with XML issues. TWA are strictly weaker than FTA, as they cannot define all regular languages (BC05) and cannot be determinized (BC06). Extensions of TWA with *pebbles* are considered in (EH99, EHB99, BSSS06a). Different pebble policies lead to classes of TWA of different expressiveness. Classes of TWA with pebbles are strongly connected to fragments of first-order logic extended with transitive closure (EH07). They have been used for instance to separate positive FO+TC$_1$ and MSO (BSSS06a), and recently FO+TC$_1$ and MSO (tCS08).
We refer the reader to (CDG*07) and to (Sch07) for a full overview of tree automata for ranked and unranked trees.

2.3.2 Tree Automata for Unranked Trees

Hedge automata are the model used in this thesis. For a complete survey of automata for unranked trees see the last chapter of (CDG*07) or (Sch07).
Hedge automata have been introduced in (BKWM01) based on early works (PQ68, Tak75).
A *finite hedge automaton* (FHA) A over Σ is a 4-tuple (Σ, Q, F, Δ) where:

- Σ is an unranked alphabet;
- Q is a finite set of states;
- $F \subseteq Q$ is a set of final states;
- $\Delta \subseteq \Sigma \times 2^{Q^*} \times Q$ is a set of rules of the form (a, L, q) – denoted $a(L) \to q$– , where $a \in \Sigma$, $q \in Q$, and L is a regular word language over Q.

A runs on unranked trees in a bottom-up way, starting from the leaves and moving up to the root. In particular, a run of A on a hedge $h \in \mathcal{H}(\Sigma)$ is a hedge $r \in \mathcal{H}(Q)$ over Q which has the same shape has r, and satisfies: for all nodes $u \in \mathrm{Dom}(t)$, there is a rule $a(L) \to q \in \Delta$ such that:

- $a = \mathsf{lab}^t(u)$;
- $q = \mathsf{lab}^r(u)$;
- if u is an inner-node and $u_1, \ldots, u_n$ are its children (given in order), then $\mathsf{lab}^r(u_1) \ldots \mathsf{lab}^r(u_n) \in L$;
- if u is a leaf, then $\epsilon \in L$ (where ϵ is the empty word)

The run r is successful if $\mathsf{lab}^r(\mathsf{root}^r) \in F$. A tree is *accepted* by A if there is a successful run of A on it. The language $\mathcal{L}(A)$ recognized by A is the set of unranked trees accepted by A. An unranked tree language $L \subseteq \mathcal{T}_{unr}(\Sigma)$ is *recognizable* if there is an FHA A such that $L = \mathcal{L}(A)$. The languages for transitions are called *horizontal languages*. Any formalism defining regular word languages can be used to specify the horizontal languages, such as regular expressions or finite automata. Of course the size of a hedge automaton depends on the sizes of the representations of the horizontal languages. In particular, if n is the size of the representation of the horizontal language of some transition $f(L) \to q$, then $n + 2$ is the size of this transition.

A FHA A is *deterministic* if for all rules $a(L) \to q \in \Delta$ and $a(L') \to q' \in \Delta$, either $L \cap L' = \varnothing$ or $q = q'$. An extension of the determinization procedure of FTA allows one to construct from any FHA a deterministic FHA which recognizes the same language (its number of states however can be exponential in the initial number of states).
The class of languages recognized by FHA also enjoys good closure properties:

Proposition 2.3.3 ((CDG*07)) *The class of unranked tree languages recognized by FHA is closed by union, intersection, and complement.*

Other automaton models for unranked trees

In (CNT04), unranked trees over an alphabet Σ are viewed as their curryfied representation, *ie* as binary trees over the ranked alphabet $\Sigma_@ = \Sigma \cup \{@\}$, where symbols from Σ are constant symbols and $@$ is a binary function symbol. For instance the curryfied representation of $a(b(c,c),d)$ is $@(@(a,@(@(b,c),c)),d)$.
Stepwise automata are a direct formulation on unranked trees of classical ranked tree automata on the curryfied version of unranked trees (CNT04). Hence stepwise automata can define all recognizable unranked tree languages.
Automata for streams are automata that process trees represented as sequences of opening and closing tags. In the context of XML for instance, they just read XML trees linearly in the same order as their textual representations (this is often called the *document order*). This is relevant when web applications exchange large XML documents for instance. More generally such streams are modeled by a 3-partite alphabet with opening, stay, and closing symbols. Such streams are processed by *visibly pushdown automata* (AM04). VPA are pushdown automata for which the stack operation is determined by the symbol type (in particular, it pushes one symbol onto the stack when reading an opening symbol, pops the stack when reading a closing symbol, and let the stack unchanged for stay symbols). VPA can define exactly all stream representations of recognizable unranked tree languages and thus enjoy good closure properties.

2.4 First Order Logic (FO)

We start by defining formally the first-order logic (FO) and several notions needed for the rest of this thesis. In a second part, we survey the main works on FO, especially in the context of trees.

2.4.1 Syntax, Semantics, and Examples

Let σ be a signature consisting of relational symbols. We let $\mathcal{X}$ be a set of variables ranged over by x, y. FO-formulas ϕ over σ are inductively defined by the following grammar:

$$\phi \ ::= \ R_i(x_1, \ldots, x_n) \mid \phi \vee \phi \mid \neg\phi \mid \exists x \phi$$

where R_i is an n-ary relation symbol of σ and $x_1, \ldots, x_n \in \mathcal{X}$. As usual, we define conjunction $\phi_1 \wedge \phi_2$ by $\neg(\neg\phi_1 \vee \neg\phi_2)$, and universal quantification $\forall x \phi$ by $\neg\exists x \neg\phi$. The set of FO-formulas over σ is denoted FO$[\sigma]$.
The set of *free variables* FVar(ϕ) of an FO$[\sigma]$ formula ϕ is inductively defined by:

$$\begin{array}{lcl} \text{FVar}(R(x_1,\ldots,x_n)) & = & \{x_1,\ldots,x_n\} \\ \text{FVar}(\phi_1 \vee \phi_2) & = & \text{FVar}(\phi_1) \cup \text{FVar}(\phi_2) \\ \text{FVar}(\neg\phi) & = & \text{FVar}(\phi) \\ \text{FVar}(\exists x\phi) & = & \text{FVar}(\phi) - x \end{array}$$

A linear order on $\mathcal{X}$ is often assumed and we shall write $\phi(x_1,\ldots,x_n)$ to mean that $\text{FVar}(\phi) \subseteq \{x_1,\ldots,x_n\}$. A *sentence* or *closed formula* ϕ is a formula such that $\text{FVar}(\phi) = \varnothing$.
Similarly, the set of variables $\text{Var}(\phi)$ of ϕ is inductively defined by:

$$\begin{array}{lcl} \text{Var}(R(x_1,\ldots,x_n)) & = & \{x_1,\ldots,x_n\} \\ \text{Var}(\phi_1 \vee \phi_2) & = & \text{Var}(\phi_1) \cup \text{Var}(\phi_2) \\ \text{Var}(\neg\phi) & = & \text{Var}(\phi) \\ \text{Var}(\exists x\phi) & = & \text{Var}(\phi) \cup \{x\} \end{array}$$

Finally the *quantifier depth* of ϕ, denoted $\text{qd}(\phi)$, is the maximal number of quantifiers on a path from the root to a leaf in the term representation of ϕ.
$\text{FO}[\sigma]$ formulas ϕ are interpreted over σ-structures M under a (total) *assignment* (also called *valuation*) ρ from a superset of $\text{FVar}(\phi)$ into $\text{Dom}(M)$. The satisfiability relation $M, \rho \models \phi$ (say ϕ holds under M and ρ) is inductively defined by:

$$\begin{array}{lll} M,\rho \models R(x_1,\ldots,x_n) & \text{if} & (\rho(x_1),\ldots,\rho(x_n)) \in R^M \\ M,\rho \models \phi_1 \vee \phi_2 & \text{if} & M,\rho \models \phi_1 \text{ or } M,\rho \models \phi_2 \\ M,\rho \models \neg\phi & \text{if} & M,\rho \not\models \phi \\ M,\rho \models \exists x\phi & \text{if} & \text{there is } u \in \text{Dom}(M) \text{ such that } M,\rho[x \mapsto u] \models \phi \end{array}$$

where $\rho[x \mapsto u]$ denotes the valuation ρ extended with the mapping $x \mapsto u$. Given a formula $\phi(x_1,\ldots,x_n)$ and an n-tuple $\overline{u} \in (\text{Dom}(M))^n$, we often write $M \models \phi(\overline{u})$ to mean that ϕ holds under M and the valuation which maps x_i to the i-th component of $\overline{u}$, for all $i \in \{1,\ldots,n\}$.
Let $n \in \mathbb{N}$, let A a set of pairs $(M,\overline{u})$ such that M is a σ-structure and let $\overline{u} \in (\text{Dom}(M))^n$.

Definition 2.4.1 *The set A is FO$[\sigma]$-definable if there is an FO$[\sigma]$-formula $\phi(x_1,\ldots,x_n)$ such that for all σ-structures M and for all $\overline{u}$:*

$$(M,\overline{u}) \in A \textit{ iff } M \models \phi(\overline{u})$$

A can also be viewed as the interpretation of an n-ary predicate symbol R. In this case we also say that R is $\text{FO}[\sigma]$-definable (its interpretation A is let implicit).
If $n = 0$ and A is definable by ϕ, then ϕ is a sentence, and in this case we say that the set $\{M \mid (M,()) \in A\}$ is definable by ϕ. Star-free word languages are a famous example of logically definable sets. In particular, a word language over a finite alphabet Σ is star-free *if and only if* it is definable by an $\text{FO}[\sigma_w]$-formula, where σ_w contains the label predicates lab_a, $a \in \Sigma$, and a binary predicate $<$ interpreted as a linear order on letter positions (MP71).

Signatures and Examples

Let Σ be a finite alphabet. When speaking about trees (binary or unranked), we often omit the alphabet in the signature. For instance we write $FO[\prec_{ns^*}, \prec_{ch^*}]$ to denote FO formulas over the signature $\{\prec_{ns^*}, \prec_{ch^*}, (\mathsf{lab}_a)_{a\in\Sigma}\}$.
For unranked trees, we consider FO$[\prec_{ns^*}, \prec_{ch^*}]$ formulas, as other predicates are FO$[\prec_{ns^*}, \prec_{ch^*}]$-definable. For instance:

$$
\begin{array}{lcl}
x = y & = & x \prec_{ch^*} y \wedge y \prec_{ch^*} x \\
x \prec_{ch} y & = & x \prec_{ch^*} y \wedge x \neq y \wedge \neg\exists z\ z \neq x \wedge z \neq y \wedge x \prec_{ch^*} z \wedge z \prec_{ch^*} y \\
x \prec_{ns} y & = & x \prec_{ns^*} y \wedge x \neq y \wedge \neg\exists z\ z \neq x \wedge z \neq y \wedge x \prec_{ns^*} z \wedge z \prec_{ns^*} y \\
x \prec_{fc} y & = & x \prec_{ch} y \wedge \neg\exists z\ z \prec_{ns} y \\
x \prec_{ch_k} y & = & \exists z_1 \ldots \exists z_k\ x \prec_{fc} z_1 \wedge z_k = y \wedge \bigwedge_{i=1}^{k-1} z_i \prec_{ns} z_{i+1} \\
\mathsf{root}^x & = & \neg\exists y\ y \prec_{ch} x
\end{array}
$$

On the other hand, the transitive closure of a definable binary relation is not definable in first-order logic in general (EF05). In particular, $\prec_{ch^*}$ and $\prec_{ns^*}$ are not FO$[\prec_{ch}, \prec_{fc}]$-definable.
For binary trees, we consider FO$[\prec_{ch_1}, \prec_{ch_2}, \prec_{ch^*}]$ formulas.

FO as a query language

As already seen, formulas can define relations on domains of σ-structures. Consequently first-order logic can also be seen as a query language. On unranked trees for instance, $(\text{FO}[\prec_{ch^*}, \prec_{ns^*}], \mathsf{ar}(.), \|.\|, \mathcal{Q}(.))$ is the query language where query expressions are FO$[\prec_{ch^*}, \prec_{ns^*}]$-formulas, $\|\phi\|$ is the number of symbols of ϕ, and if $\text{FVar}(\phi) = \{x_1, \ldots, x_n\}$, then $\mathsf{ar}(\phi)$ is defined by $\mathsf{ar}(\phi) = n$ and $\mathcal{Q}(\phi(x_1, \ldots, x_n))$ is defined by:

$$
\begin{array}{c}
\mathcal{Q}(\phi(x_1, \ldots, x_n))(t) \\
= \\
\{(u_1, \ldots, u_n) \mid t, [x_1 \mapsto u_1] \ldots [x_n \mapsto u_n] \models \phi(x_1, \ldots, x_n)\}
\end{array}
$$

for all trees $t \in \mathcal{T}_{unr}(\Sigma)$.
For instance the following formula defines a query which selects all pairs of sibling-nodes labeled a and b respectively:

$$\phi(x, y) \ = \ x \prec_{ns} y \ \wedge \ \mathsf{lab}_a(x) \wedge \mathsf{lab}_b(y)$$

Depending on the context, we often denote FO$[\prec_{ch^*}, \prec_{ns^*}]$ the query language $(\text{FO}[\prec_{ch^*}, \prec_{ns^*}], \mathsf{ar}(.), \|.\|, \mathcal{Q}(.))$. Finally, we say that a query is FO-definable if it is equal to $\mathcal{Q}(\phi)$ for some FO-formula ϕ.

2.4.2 FO: State of the Art

The study of FO benefits from a long history of research and we cannot survey all the results related to it. We refer the reader to (EF05, Lib04a) for more details.

Model-checking The model-checking of FO is PSPACE-complete on finite structures (even on trees) (Sto74, Var82), while it is in PTIME for the k-variable fragment FO^k (the k-variable fragment consists of all formulas ϕ which uses at most k bound variables) (Sto74). Recent work on the k-variable fragment can be found in (Var95).

Satisfiability The satisfiability problem of FO was considered as the main problem in mathematical logic by many mathematicians (Hilbert, Ackermann, Herbrand, von Neumann). Since 1936, satisfiability of FO is known to be undecidable (Chu36, Tur37), even on finite structures (Tra75). Since 1915, many fragments of FO have been proved to be decidable. More recently, the two-variable fragment FO^2 has been proved to be decidable in 2NEXPTIME (Mor75). A NEXPTIME lower-bound has been established later (Lew80). Recently, a NEXPTIME upper-bound has been proved (GKV97), making satisfiability of FO^2 formulas NEXPTIME-complete over arbitrary structures. However, FO^k is undecidable for $k \geq 3$.
Over unranked trees, FO is also decidable (TW68, Don70) by reduction to emptiness of tree automata. Actually those papers prove the stronger result that *Monadic Second Order Logic* is decidable over unranked trees, as described in the next section. Although the relationship between regular tree languages (defined by automata) and logically defined tree languages is well-known for MSO over trees, it is not so clear for FO-definable tree languages.

FO-definability of recognizable tree languages It is well-known that over strings (which can be viewed as structures over the signature $\{(\mathsf{lab}_a)_{a\in\Sigma}, \prec_{ns}\}$), $\text{FO}[\prec_{ns^*}]$-definable languages are exactly the class of star-free languages (MP71, Sch65).
Over trees, several papers attempt to give algebraic characterizations of FO-definable regular languages (or subclasses of). This has been considered for instance in the thesis (Boj04), and the most recent paper (BS05b) gives an algebraic characterization of $\text{FO}[\prec_{ns}, \prec_{ch}]$-definable ranked tree languages. This characterization yields a decision procedure to test whether a regular ranked tree language is definable in $\text{FO}[\prec_{ns}, \prec_{ch}]$. This also holds for unordered unranked trees, but it is still open for ordered unranked trees or even ranked trees with the descendant relation.

Data values Recently, satisfiability of FO has been considered in (unranked) trees with data coming from an infinite alphabet (BDM*06). In particular, the author consider a new predicate $\sim$ which holds between a node u and a node v if u and v carry the same data-values. This is particularly relevant when considering XML documents without ignoring data-values. The authors prove that the two-variable fragment $\text{FO}^2[\sim, \prec_{ch}, \prec_{ns}]$ is decidable in 3NEXPTIME, and gives a NEXPTIME lower bound. Allowing more than three variables leads to undecidability. It is still open whether $\text{FO}^2[\sim, \prec_{ch^*}, \prec_{ch}, \prec_{ns}]$ is decidable, but it is already known that $\text{FO}^2[\sim, \prec_{ch^*}, \prec_{ch}]$ is decidable on strings (viewed here as unary trees) (BMS*06).

k-variable fragments On (possibly infinite) unordered trees of rank at most d, it is known that $\text{FO}[\prec_{ch^*}]$ is equivalent, with respect to closed formulas, to $\text{FO}^{max\ (d,4)}[\prec_{ch^*}]$ if $d > 1$, and to $\text{FO}^3[\prec_{ch^*}]$ if $d = 1$ (IK89). Over the first-order vocabulary of unranked trees $\{\prec_{ch^*}, \prec_{ns^*}\}$, every $\text{FO}[\prec_{ch^*}, \prec_{ns^*}]$ formula with two free variables is equivalent to an $\text{FO}[\prec_{ch^*}, \prec_{ns^*}]$ formula which uses at most three free and bound variables, i.e. such that $|\text{var}(\phi)| \leq 3$ (Mar05b).

2.5 Monadic Second Order Logic (MSO)

Monadic Second-Order Logic (MSO) extends FO with set quantification. MSO has established as a benchmark logic in the context of XML (NS02a, Nev02, NS00, Lib06). Over trees, it is an expressive logic which is captured by the robust class of regular tree languages (Tho97, TW68, Don70), node-selecting tree automata (NS02a, NPTT05), attributed grammars (NV02), XML Schemas (MNSB06).

2.5.1 Syntax and Semantics

Let σ be a signature and $\mathcal{X}$ a set of first-order variables x, y and second-order variables X, Y. MSO$[\sigma]$ formulas are inductively defined by the following grammar:

$$\phi \ ::= \ R(x_1, \ldots, x_n) \mid x \in X \mid \exists x \phi \mid \exists X \phi \mid \neg\phi \mid \phi \vee \phi$$

As usual, $\forall X \phi$ stands for $\neg\exists X \neg\phi$. The set of free-variables FVar(ϕ) is defined like for FO$[\sigma]$-formulas, but second-order order variables are also taken into account.
Let M be a σ-structure. A valuation ρ maps first-order variables into elements of Dom(M), and second-order variables into subsets of Dom(M). The satisfiability relation $M, \rho \models \phi$ defined for FO$[\sigma]$-formulas naturally extends to MSO$[\sigma]$-formulas, where $M, \rho \models x \in X$ if $\rho(x) \in \rho(X)$, and $M, \rho \models \exists X \phi$ if there exists a set $U \subseteq$ Dom(M) such that $M, \rho[X \mapsto U] \models \phi$.
When speaking about unranked trees, we consider MSO formulas over the signature $\sigma_{unr}(\Sigma) = \{\prec_{fc}, \prec_{ns}, (\mathsf{lab}_a)_{a\in\Sigma}\}$, as other predicates are all definable in MSO as for instance:

$$x \prec_{ns^*} y \ = \ \forall X \, (x \in X \wedge (\forall z, z' \, z \in X \wedge z \prec_{ns} z' \implies z' \in X)) \implies y \in X$$

In other words, it means that all subsets X closed by the next-sibling relation and containing x also contains y. $x \prec_{ch^*} y$ is defined similarly.
As for FO, MSO can be viewed as a query language, where queries are defined by formulas whose free-variables are first-order. Those queries are often referred as MSO-queries or regular queries.
Free second-order variables can also be used to define n-ary queries, by taking the cartesian products of the selected sets. For instance, the binary query defined by $\phi(X, Y)$ is equivalently definable by $\phi'(x, y) = \exists X \exists Y, x \in X \wedge y \in Y \wedge \phi(X, Y)$. However in this thesis queries are always assumed to be defined by using first-order variables.

2.5.2 Correspondence between MSO and recognizable languages

The first correspondence between MSO and recognizable languages was first established by Büchi on strings (Büc60). It states that the class of MSO-definable string languages is exactly the class of string languages recognizable by a finite automaton. This result was extended to binary trees by Thatcher, Wright (TW68), and Doner (Don70). In those paper, MSO is called WS2S, for *weak second order logic with two successors*. *Weak* means that set quantification is over finite sets, and the two successors are the first-child and second-child relations. The correspondence also holds for unranked trees via a binary encoding (see, for example, (CDG*07)).

Theorem 2.5.1 *The class of MSO[$\sigma_{unr}(\Sigma)$]-definable unranked tree languages is effectively equal to the class of recognizable unranked tree languages.*

In other words, MSO[$\sigma_{unr}(\Sigma)$] and hedge automata are equally expressive, and the back and forth translations are effectively computable. In particular, for all MSO[$\prec_{ch}, \prec_{ns}$] sentence ϕ, there exists an FHA A_ϕ such that:

$$\{t \in \mathcal{T}_{unr}(\Sigma) \mid t \models \phi\} \;=\; \mathcal{L}(A_\phi)$$

Conversely, for all FHA A, there exists an MSO[$\prec_{ch}, \prec_{ns}$] sentence ϕ_A such that:

$$\mathcal{L}(A) \;=\; \{t \in \mathcal{T}_{unr}(\Sigma) \mid t \models \phi_A\}$$

The size of A_ϕ however might be non-elementary[5] in the size of ϕ, and this complexity is sometimes unavoidable (SM73a). Since emptiness of hedge automata is in PTIME, decidability of MSO[$\prec_{ch}, \prec_{ns}$] is obtained as a corollary of Theorem 2.5.1. It is also known that this problem requires non-elementary time as worst case, and this is a lower bound (Mey73, Sto74).
Actually, the theorem is also proved for formulas with free variables. The usual way to represent instances of free node variables in a tree is to extend the alphabet with Boolean tuples. Let $t \in \mathcal{T}_{unr}(\Sigma)$, and $U \subseteq \mathrm{Dom}(t)$. We let $\chi_{t,U}$ the tree over $\{0,1\}$ which has the same shape as t and such that all nodes of $\mathrm{Dom}(\chi_{t,U}) - U$ are labeled 0, and all other nodes are labeled 1. We denote by $t \times \chi_{t,U}$ the tree over $\Sigma \times \{0,1\}$ whose labels are obtained by concatenation of the respective labels of t and $\chi_{t,U}$. This operation is obviously associative, and $\chi_{t,U_1,U_2,\ldots,U_n}$ stands for $\chi_{t,U_1} \times \chi_{t,U_2} \times \cdots \times \chi_{t,U_n}$. This is again obtained from (TW68, Don70) and extended to unranked trees via a binary encoding:

Theorem 2.5.2 *For all MSO[$\sigma_{unr}(\Sigma)$] formula $\phi(x_1, \ldots, x_n, X_{n+1}, \ldots, X_m)$ with m free first-order and second-order variables, there is a (computable) FHA A_ϕ over $\Sigma \times \{0,1\}^m$ such that:*

$$\mathcal{L}(A_\phi) \;=\; \{t \times \chi_{t,\{u_1\},\ldots,\{u_n\},U_{n+1},\ldots,U_m} \mid t \models \phi(u_1, \ldots, u_n, U_1, \ldots, U_m)\}$$

Conversely, for all FHA A over $\Sigma \times \{0,1\}^m$, there is a (computable) MSO[$\sigma_{unr}(\Sigma)$]-formula $\phi_A(X_1, \ldots, X_m)$ with m free second-order variables such that:

$$\mathcal{L}(A) \;=\; \{t \mid t \models \phi(\pi_1 t, \ldots, \pi_m t)\}$$

where for all $i \in \{1, \ldots, m\}$, $\pi_i t = \{u \in \mathrm{Dom}(t) \mid \pi_{i+1}(\mathsf{lab}^t(u)) = 1\}$.

2.5.3 MSO: State of the Art

Satisfiability of MSO over finite trees have been presented in the previous section. Relationship between MSO and automata on other structures (finite and infinite word and trees) is surveyed by Thomas (Tho97).

[5] A function $f : \mathbb{N} \to \mathbb{N}$ is *elementary* if it can be formed by several elementary operations in a bounded manner. The operations are not detailed here but elementary functions were introduced in (Grz53). You can also see (FG02) for a more recent definition. What is important however is that a function is elementary iff it is bounded by a tower of exponentials of fixed height.

Beyond trees A natural question is whether satisfiability of MSO is still decidable on finite graph structures. This question has been investigated in several papers by Courcelle and Engelfriet (Cou94, CE95, Cou97, EvO97). In particular, they consider graph decompositions that yield classes of graphs that are representable by terms over particular graph operations (in particular, graphs of *bounded tree-width* and *bounded clique-width*). Courcelle introduces transductions defined by MSO formulas (called MSO-transductions) (Cou94, Cou97). In essence, to define a transduction from a σ-structure to a σ'-structure, one creates k copies of the input domain (for a fixed k) filtered by k MSO[σ]-formulas $\psi_1(x), \ldots, \psi_k(x)$ with one free variable. This defines the domain of the output structure. Relations R' of the output σ'-structure are also defined by MSO[σ]-formulas $\phi_{R',\bar{i}}$ with $\mathsf{ar}(R')$ free first-order variables, where $\bar{i} \in \{1, \ldots, k\}^{\mathsf{ar}(R')}$ is a tuple that specifies the correspondence between copies of the input domain and free variables of $\phi_{R',\bar{i}}$. This is a rough description of MSO-transductions, as in particular parameters (by means of additional free variables to formulas) can be used to make the transduction non-functional. MSO-transductions are closed by composition, and the image of any class of structures on which MSO is decidable by an MSO-transduction forms a class of structures on which MSO is also decidable (Cou94, Cou97). The class of graphs of bounded clique-width is the image of a regular set of (binary) trees by an MSO-transduction (CE95, EvO97). This also holds for trees of bounded tree-width. Hence MSO (over the vocabulary of graphs, *ie* with a binary edge relation E) is decidable on the classes of graphs of bounded tree-width and graphs of bounded clique-width respectively. The converse has been conjectured by Seese: any set of graphs with decidable MSO theory has a bounded clique-width (See91). Recently, Courcelle and Oum have proved this conjecture for the extension of MSO with a counting modulo 2 predicate $Even(X)$ which holds if X has an even cardinality (CO07). This results weakens the Seese's conjecture as the extension of MSO with counting modulo 2 is strictly more expressive than MSO.

Model-checking The model-checking of MSO is known to be PSPACE-complete, even on trees (Sto74, Var82).
However, it is linear on trees if the formula is fixed (with a non-elementary constant factor). It suffices to construct from a formula ϕ an FHA A_ϕ equivalent to ϕ: model-checking of ϕ on a tree t reduces to membership of t to $\mathcal{L}(A_\phi)$. Model-checking of MSO can be viewed as a *parameterized problem*, where $\|\phi\|$ is the parameter. Parameterized problems are usually problems with two inputs p, i, the size of p ($\|p\|$, called the parameter) being very small comparing to the size of i ($\|i\|$), so that even an exponential in $\|p\|$ would still be tractable (See96). This is particularly relevant when for instance ϕ is a small query and t is a very large XML document. A problem is *fixed-parameter tractable* if it can be solved in time $f(\|p\|)\text{poly}(\|i\|)$, for some computable function f and some polynomial poly. Model-checking of MSO on trees is fixed-parameter tractable (where $\|\phi\|$ is the parameter), but in this case f is non-elementary. Frick and Grohe prove in (FG02) that it is unavoidable, even on strings. In particular, unless $P = NP$, there is no model-checking algorithm for MSO on strings (with a linear order predicate) whose time complexity is bounded by $f(\|\phi\|)\text{poly}(\|t\|)$, for an elementary function f and a polynomial poly. Courcelle proves that it is still fixed-parameter tractable (and even linear in the size of the structure) on the class of graphs of bounded tree-width (Cou90a).
A logic closed to MSO, called *ETL* (*Efficient Tree Logic*) is introduced in (NS00). The main ingredients of ETL are guarded quantification and new constructors

(MSO-definable) that allow vertical and horizontal navigation by means of regular expressions of ETL formulas. ETL has the same expressive power as MSO (w.r.t. unary queries) over unranked trees, while its model-checking problem is linear in $\|t\|$ and doubly exponential in $\|\phi\|$. This can be reduced to a single exponential when considering an equally expressive fragment of ETL.

FO with transitive closure FO + TC is the extension of FO with a transitive closure operator. In particular, for all FO + TC-formulas $\phi(\overline{x}, \overline{y}, \overline{z})$ where $\overline{x}, \overline{y}$ are tuples of variables of the same length n, for all tuples of variables $\overline{\alpha}, \overline{\beta}$ of length n, $\text{TC}_{\overline{x},\overline{y}}[\phi(\overline{x}, \overline{y}, \overline{z})](\overline{\alpha}, \overline{\beta})$ is interpreted as follows: once the interpretation of $\overline{z}$ is fixed by some tuple of elements $\overline{w}$, the formula $\phi(\overline{x}, \overline{y}, \overline{w})$ defines a binary relation between tuples of elements of size n, and the TC operator allows one to take its transitive closure, starting from a tuple denoted by $\overline{\alpha}$ and going to a tuple denoted by $\overline{\beta}$. Formally, on a structure M modulo an assignment ρ, if we denote by $R^*_{\phi(\overline{x},\overline{y},\rho(\overline{z}))}$ the transitive closure of the binary relation defined by $\phi(\overline{x}, \overline{y}, \rho(\overline{z}))$ on $\text{Dom}(M)^n$, we have:

$$M, \rho \models \text{TC}_{\overline{x},\overline{y}}[\phi(\overline{x}, \overline{y}, \overline{z})](\overline{\alpha}, \overline{\beta}) \quad \text{iff} \quad (\rho(\overline{\alpha}), \rho(\overline{\beta})) \in R^*_{\phi(\overline{x},\overline{y},\rho(\overline{z}))}$$

DTC is the *deterministic transitive closure*, meaning that the transitive closure can be defined on functional binary relations only. TC_i is the transitive closure of FO + TC_i-definable binary relations on tuples of size i. posTC is the positive transitive closure, meaning that the TC operator cannot occur below an odd number of negations. FO+posTC can express all properties expressible by a nondeterministic Turing machine running in logarithmic space (Imm87). Therefore, posTC is more expressive than MSO on unranked trees and is even undecidable. FO + posTC_1 is strictly less expressive than MSO over ranked trees (BSSS06b). It has been recently proved that it also holds for FO + TC_1 over unranked trees, *ie* FO + TC_1 is strictly less expressive than MSO over unranked trees (tCS08). However it is still unknown whether the inclusion of FO + posTC_1 into FO + TC_1 is strict (tCS08).

Answer Enumeration for MSO-queries As the number of answers to an n-ary query defined by an MSO-formula might be very large, instead of outputting all the answer tuples of the query, it might be convenient to output all answer tuples one by one, with a reasonable delay between two consecutive answers. A preprocessing phase is often needed to precompute a data-structure from which enumeration can be done efficiently. Of course, this preprocessing phase has to be done more efficiently than computing directly all the answers. Enumeration for MSO queries in trees has been investigated in two papers (Bag06, Cou07). In (Bag06), it is proved that the answer tuples to a query defined by a **fixed** formula $\phi(x_1, \ldots, x_n)$ on a ranked tree t can be enumerated with preprocessing phase in time $O(\|t\|)$ and a delay $O(n)$ between two consecutive answers. If the query is given by a FTA A_ϕ on $\Sigma \times \{0, 1\}^n$ (see Section 2.6 for the relationship between n-ary queries and tree automata), the preprocessing phase is in $O(\|A_\phi\|^3\|t\|)$ and the delay is in $O(n)$. This result still holds for graphs of bounded tree-width (hence for unranked trees).

2.6 Tree Automata as a Query Language

By Theorem 2.5.2, all $\text{MSO}[\sigma_{unr}(\Sigma)]$-formula $\phi(x_1, \ldots, x_n)$ defines a recognizable unranked tree languages over $\Sigma \times \{0, 1\}^n$. Conversely, an unranked tree

language L over $\Sigma \times \{0,1\}^n$ represents an n-ary query $\mathcal{Q}(L)$ which associates with every tree $t \in \mathcal{T}_{unr}(\Sigma)$, the set $\mathcal{Q}(L)(t)$ defined by:

$$\mathcal{Q}(L)(t) \;=\; \bigcup_{U_1,\ldots,U_n \subseteq \mathrm{Dom}(t),\; t \times \chi_{U_1,\ldots,U_n} \in L} U_1 \times \cdots \times U_n$$

Theorem 2.5.2 implies that L is recognizable iff $\mathcal{Q}(L)$ is definable by an MSO-formula $\phi(X_1,\ldots,X_n)$ with n free second-order variables. This is again equivalent to say that $\mathcal{Q}(L)$ is definable by an MSO-formula $\phi'(x_1,\ldots,x_n)$ with n free first-order variables. It suffices to take $\phi'(x_1,\ldots,x_n) = \exists X_1 \ldots \exists X_n,\ \phi(X_1,\ldots,X_n) \wedge \bigwedge_{i=1}^n x_i \in X_i$. It means in particular that the representation of a query by a language over $\Sigma \times \{0,1\}$ is not unique: there might be two different languages $L, L' \subseteq \mathcal{T}_{unr}(\Sigma \times \{0,1\})$ such that $\mathcal{Q}(L) = \mathcal{Q}(L')$. Let L be the language over $\Sigma \times \{0,1\}^n$ associated with $\phi'(x_1,\ldots,x_n)$, as in Theorem 2.5.2. For all $t \in L$, and all $i \in \{1,\ldots,n\}$, there exists a unique node $u \in \mathrm{Dom}(t)$ such that $\pi_{i+1}\mathsf{lab}^t(u) = 1$. We say that L is *canonical*.

FHA over $\Sigma \times \{0,1\}^n$ have the same expressive power as MSO-queries. This leads to another formalization of query by tree automata, by putting the Boolean values into the states. Some states are then used to select components of the output tuples. These states are called *selecting states*. More formally, given an FHA $A = (\Sigma, Q, F, \Delta)$, A is extended with a set of selection states $S \subseteq Q^n$. (A,S) defines the n-ary query $\mathcal{Q}(A,S)$ by:

$$\mathcal{Q}(A,S)(t) \;=\; \{(u_1,\ldots,u_n) \;\mid\; \begin{array}{l}\text{there exists a successful run } r \text{ on } t\\ \text{such that } (\mathsf{lab}^r(u_1),\ldots,\mathsf{lab}^r(u_n)) \in S\}\end{array}$$

The tuple $(u_1,\ldots,u_n)$ is said to be *selected* by r. Those queries are called *existential run-based queries* in (NPTT05). Another notion of queries exist, where it is required that the tuple is selected by all successful runs (called *universal run-based queries*). This distinction is also made in (NS02a), for attributed grammars (described further). However, existential and universal run-based queries have the same expressive power. The restriction where A is deterministic, although it enjoys good complexity properties, is strictly less expressive than MSO for n-ary queries.

The idea to add selecting states to tree automata to turn them into query languages appears in several papers. In (FGK03), FTA are extended with selecting states to get the so called (monadic) *selecting tree automata* and they are extended to selecting hedge automata. Selecting tree automata need to be non-deterministic to get the power of monadic MSO-queries. The (monadic) query evaluation problem for a selecting tree automata A on a tree t is in time $O(|Q|^3\|t\|)$, where Q is the set of states of A. Actually this framework is used to query *compressed trees*, *ie* DAG representations of trees with maximal sharing of common subtrees, and it is proved that the (monadic) query evaluation problem is in time $2^{O(|Q|)}\|t^*\|$, where t^* is a compressed tree.

Selecting tree automata (STA) are extended to n-ary queries (and proved to capture n-ary MSO-queries) in (NPTT05), and the authors propose *unambiguous STA* which allow one to evaluate the query in polynomial-time, both in the size of the input and the output. An STA is unambiguous if there is at most one successful run per tree. Unambiguous STA have the same expressive power as STA with respect to unary queries, but form a strict subclass of STA with respect to more general n-ary queries. The complexity of query evaluation for STA queries is improved in (Bag06). In particular, it is proved that the answers to an n-ary

query defined by an STA A on a tree t can be enumerated with a delay $O(n)$ and preprocessing $O(\|A\|^3\|t\|)$. This gives a query evaluation algorithm in time $O(\|A\|^3\|t\| + n|\mathcal{Q}(A)(t)|)$, where $\mathcal{Q}(A)(t)$ is the answer set.
To express unary queries on unranked trees, a deterministic tree automata model is proposed in (NS02a), called *query automata*, and are equally expressive as (monadic) selecting tree automata. They consist in two-way automata (Mor94) with selecting states which can move up and down in the tree along cuts of the tree. Two-way deterministic string automata are used for horizontal traversing of the children of a node u, in order to assign a new state to u. These kind of transitions cannot be used more than once per sequences of children. The authors prove EXPTIME upper and lower bounds for the satisfiability, the containment, and the equivalence problems for unary queries defines by query automata.

Other n-ary query formalisms inspired by language theory *Attribute Grammars* were proposed in (NV02) as an n-ary query language. Attribute values are used to select the nodes. *Boolean Attribute Grammars* (BAG) are proposed for unary queries, and are equally expressive as unary MSO-queries. *Relation Attribute Grammars* extends BAG with the ability to query n-tuples of nodes. They are strictly more expressive than n-ary MSO-queries. Attribute grammars are extended to unranked trees in (Nev00) by allowing regular expressions in right-hand sides of production rules. There are studied in their unary setting only. The emptiness and equivalence problems are proved to be EXPTIME-complete.
Streaming tree automata (STA) are proposed in (GCNT08) as a querying formalism for XML streams. These are visibly pushdown automata, already described in Section 2.3.2, which run on stream representations of unranked trees. An extended alphabet $\Sigma \times \{0,1\}^n$ is considered to add a querying power to VPA. Motivated by XML streaming applications, the authors give a query evaluation algorithm which output the query results as soon as possible (optimality is proved) when reading the stream linearly. STA over $\Sigma \times \{0,1\}^n$ captures n-ary MSO-queries. This idea of earliest query answering has also been investigated in (Ber06) and in (BJ07) in the context of XPath queries.
Forest grammars are proposed in (BS04a) as a formalism for n-ary queries in unranked trees. Rules have the form $X \rightarrow < a > r < /a >$, where r is a regular expression of non-terminal symbols. Such a grammar generates well-balanced sequences of opening and closing tags, or equivalently, linear representations of unranked trees. Special non-terminal symbols are used to select nodes. An evaluation algorithm for binary queries is presented which runs in worst-case time complexity $O(\|F\|(|A| + s.\|t\|)$ for a binary query defined by a forest grammar F on a tree t, where A is the set of answers, and s is the number of first and second components of solution tuples, *ie* $s = |\pi_1(A)| + |\pi_2(A)|$.

2.7 Schema languages

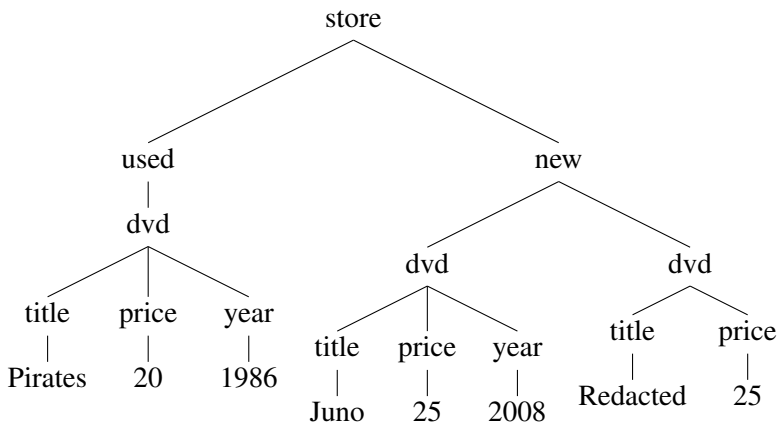
This section describes the main schema languages. The following overview is not exhaustive since this thesis is mainly concerned with n-ary query languages for unfixed n. We only formally define what is needed in the rest of this thesis. However a complete overview can be found in the last chapter of (CDG*07), or in (Sch07).

2.7.1 Document Type Definition

Schema languages express constraints on the structure of XML documents. There are many schema languages. One of the most widely used is the W3C standard *Document Type Definition* (DTD) (BPSM*06). Basically DTDs are context-free grammars whose right-hand sides are regular expressions over terminal and non-terminal symbols. The set of derivation trees define the set of XML documents satisfying the DTD. For instance, the following DTD d constraints the XML documents to describe a (simplified) DVD store (store is the start symbol):

$$\begin{array}{lcl} store & \rightarrow & used\ new \\ used & \rightarrow & dvd^* \\ new & \rightarrow & dvd^* \\ dvd & \rightarrow & title\ price\ year? \\ title & \rightarrow & PCDATA \\ price & \rightarrow & PCDATA \\ year & \rightarrow & PCDATA \end{array}$$

The terminal PCDATA means that under the tag Title for instance, any chain of characters is allowed. Trees that satisfies d have the following form:



DTDs cannot express all recognizable unranked tree languages. This is because the type of the children of a node is determined by the label of the parent, regardless its context. In particular, DTDs have the exchange property: if a tree t satisfies a DTD, and two of its subtrees have the same root label, one of the subtree can be substituted by the other while remaining satisfied by the DTD. Languages having this property are called *local languages* (MLM01).

A lot of works have been done on DTDs, for instance to lower the complexity of validating a document (BKW98) with respect to a DTD. In (MNS04), a complete picture of the complexities of inclusion, equivalence, and intersection problems for different fragments of DTDs is given. In particular, inclusion of DTDs is PSPACE-complete but in PTIME if the regular expressions of the production rules are *one-unambiguous*.

As it is needed in this thesis, we formalize DTDs. A DTD d over a finite alphabet Σ is a pair (s, δ) where $s \in \Sigma$ is the start symbol and δ maps any symbol $a \in \Sigma$ to a regular expression e_a over Σ. Regular expressions over Σ are generated by

the grammar $e ::= a \in \Sigma \mid e + e \mid e.e \mid e^* \mid \epsilon$. The set of (unranked) trees $\mathcal{L}(d)$ accepted by d is $\mathcal{L}(s)$, where $\mathcal{L}$ is defined by:

$$\begin{array}{lcl} \mathcal{L}(\epsilon) & = & \mathsf{0} \\ \mathcal{L}(a) & = & \{a(h) \mid h \in \mathcal{L}(\delta(a))\} \\ \mathcal{L}(e_1 + e_2) & = & \mathcal{L}(e_1) \cup \mathcal{L}(e_2) \\ \mathcal{L}(e_1.e_2) & = & \mathcal{L}(e_1) \mid \mathcal{L}(e_2) \\ \mathcal{L}(e^*) & = & \mathsf{0} \cup \bigcup_{i>0} \underbrace{\mathcal{L}(e)|\ldots|\mathcal{L}(e)}_{i \text{ times}} \end{array}$$

2.7.2 Extended DTD

In the DTD representing a dvd store given previously, one may want to constraint the used dvds to have a field year, which is useless for new dvds. This is not expressible with DTDs. In (PV00), DTDs are extended to overcome this lack of expressiveness to the so called *extended DTDs*. The idea is to add an (optional) type to the non-terminals in order to take the context into account (of course types do not appear in the derivation trees). The previous requirement is now expressible:

$$\begin{array}{lcl} \mathit{store} & \rightarrow & \mathit{used}\ \mathit{new} \\ \mathit{used} & \rightarrow & (\mathit{dvd}, \mathsf{UsedDvdType})^* \\ \mathit{new} & \rightarrow & (\mathit{dvd}, \mathsf{NewDvdType})^* \\ (\mathit{dvd}, \mathsf{UsedDvdType}) & \rightarrow & \mathit{title}\ \mathit{price}\ \mathit{year} \\ (\mathit{dvd}, \mathsf{NewDvdType}) & \rightarrow & \mathit{title}\ \mathit{price} \\ \mathit{title} & \rightarrow & \mathit{PCDATA} \\ \mathit{price} & \rightarrow & \mathit{PCDATA} \\ \mathit{year} & \rightarrow & \mathit{PCDATA} \end{array}$$

Formally, an extended DTD d' over Σ is a pair (d, T) where T is a finite set (of types) and d is a DTD over $\Sigma \times T$. The language accepted by d', denoted $\mathcal{L}(d')$, is:

$$\mathcal{L}(d') \ = \ \{\pi_1 t \in \mathcal{T}_{unr}(\Sigma) \mid t \in \mathcal{L}(d')\}$$

where $\pi_1 t$ is the first projection of t, obtained by projecting away each second component of its labels. Note that an extended DTD where T is a singleton is a DTD. From (PV00) we know that:

Theorem 2.7.1 *Extended DTDs and hedge automata can define exactly the same unranked tree languages.*

2.7.3 Other schema languages

Several schema languages extend the expressive power of DTDs with types, like for extended DTDs, to name just a few: XML Schemas (FW04), Relax NG (CM01). They are studied in particular in (MNSB06, MLM01).

2.8 XPath 1.0 and 2.0

The W3C standard XPath (BBC*07) is a language for the description of paths in XML documents. It is used in the node selecting core of several other W3C standards: for instance in the query language XQuery (BCF*07), in the transformation language XSLT (Cla99), in the schema language XML Schema (FW04), or in the

addressing language XPointer (DMJ01). Some of these standards are based on the first version of XPath, XPath 1.0 (CD99). The second version of XPath, XPath 2.0 (BBC*07) extend XPath 1.0 with richer data-types, variables, quantifiers, and Boolean operators on path expressions. One of the motivation of the W3C was to turn XPath into a query language, and to make its navigational core expressive power closer to first-order logic (Kay08). Consider now an example. The following path expressions,

$$//dvd[/year]/title$$

produces the titles of DVDs for which the year is specified. '//' denotes descendant, '/' denotes child, and '[]' denotes a test expression. When a path expression is applied from the root, it produces a set of nodes, so that XPath is viewed as a unary query language. However, path expressions can be applied from any starting node, making XPath a *navigational language* to navigate in the tree from starting nodes to ending nodes. Therefore XPath can be viewed more generally as a binary query language. XPath has a lot of features that renders it undecidable, such as manipulation of arithmetical data-values. Moreover its description proposed by the W3C is very large. Hence clean fragments of XPath, which in essence consists of its navigational core, have been extracted.
Gottlob, Koch, and Pichler (GKP05) define *CoreXPath* or synonymously *CoreXPath1.0*, the navigational core of XPath 1.0.
Ten Cate and Marx (tCM07) distinguish the counterpart *CoreXPath2.0*. Since variables are allowed in *CoreXPath2.0*, it can be viewed as an n-ary query language. Moreover, quantifiers are provided as primitives, so that *CoreXPath2.0* is equally expressive as n-ary FO queries modulo linear time transformations (see Section 4.2 for the translation). This implies PSPACE completeness of model checking for *CoreXPath2.0*, so that one cannot hope for polynomial time query answering except if P=PSPACE.
In this thesis, we distinguish a polynomial time fragment of *CoreXPath2.0* which still captures the class of all n-ary FO queries, while allowing for polynomial time query evaluation in the size of the answer set, the query, the tree t, and tuple width n.

2.8.1 Syntax and Semantics

CoreXPath1.0 is first defined and then extended to *CoreXPath2.0*. In *CoreXPath1.0*, '/' is interpreted as the composition of path expression. Basic paths expressions relate nodes to their parent, child, descendant, ancestor, etc... These expressions are called *axis*. All XPath axis and their interpretation are depicted in Fig. 2.5. We denote by Axis the set of XPath axis. Every axis $a \in$ Axis is interpreted in an unranked tree t over an alphabet Σ as a binary relation $[\![a]\!]^t_{axis} \subseteq \text{Dom}(t) \times \text{Dom}(t)$ of nodes. It relates a starting node from which an ending node can be reached by following a. For instance in an unranked tree t:

$$\begin{array}{lcl} [\![\text{self}]\!]^t_{axis} & = & \{(u,u) \mid u \in \text{Dom}(t)\} \\ [\![\text{child}]\!]^t_{axis} & = & \{(u,v) \mid u,v \in \text{Dom}(t) \ \wedge \ u \prec^t_{ch} v\} \\ [\![\text{descendant}]\!]^t_{axis} & = & \{(u,v) \mid u,v \in \text{Dom}(t) \ \wedge \ u \prec^t_{ch^+} v\} \end{array}$$

CoreXPath1.0 expressions consist of *path expressions* P to navigate in the tree and *test expressions* T to test properties of nodes. Similarly to axis, path expressions P

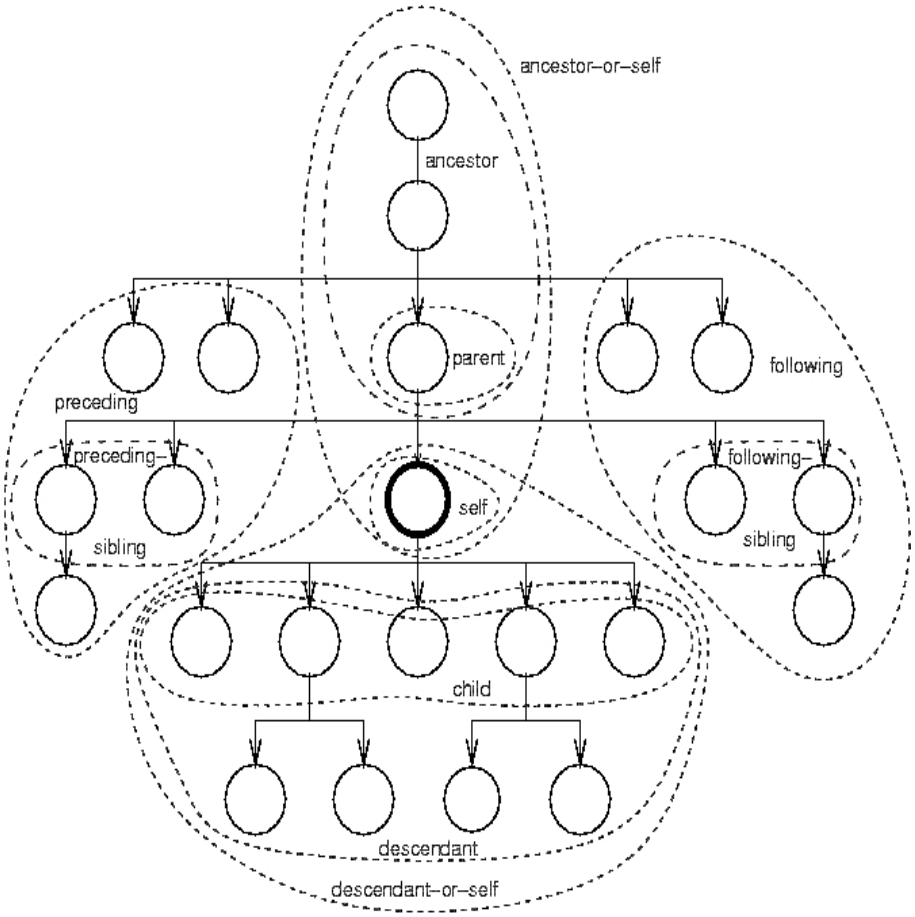


Figure 2.5: XPath axes starting from the bold node (picture taken from (Shi08))

are interpreted as binary relations $[\![P]\!]^t_{path} \subseteq \mathrm{Dom}(t) \times \mathrm{Dom}(t)$ while test expressions T are interpreted as unary relations $[\![T]\!]^t_{test} \subseteq \mathrm{Dom}(t)$. The syntax of *CoreXPath1.0* is given in Fig. 2.6 and its semantics in Fig. 2.7. XPath path expressions P can be viewed as binary queries with the interpretation $\mathcal{Q}_{path,2}(P) = t \mapsto [\![P]\!]^t_{path}$, but they are generally considered as unary queries $\mathcal{Q}_{path,1}(P)$, by taking the root as starting node: $\mathcal{Q}_{path,1}(P)(t) = \{u \mid (\mathsf{root}^t, u) \in [\![P]\!]^t_{path}\}$. Each time the interpretation of XPath expressions is ambiguous we specify whether we take the unary or binary query point of view.
The following unary *CoreXPath1.0* query

$$\text{descendant} :: dvd[?\text{child} :: year]/\text{child} :: title$$

selects all nodes labeled by titles of DVDs for which the year is provided.
The following binary query allows one to reach any node wherever it starts:

$$\text{nodes} = \text{ancestor\_or\_self} :: */\text{descendant\_or\_self} :: *$$

CoreXPath2.0 extends *CoreXPath1.0* with node variables $\$x$ (from a countable set of variables $\mathcal{X}$), quantified path expression for $\$x$ in P_1 return P_2, relative complements P_1 except P_2 and intersections P_1 intersect P_2. The syntax of *CoreXPath2.0* expressions is given in Fig. 2.8. A variable path expression $\$x$ requires to move from the current node to the node pointed by $\$x$. This value can be constrained further by node equality constraints in test expressions $[\$x$ is . $]$ to test

```
Axis       := self | child | parent | descendant | descendant_or_self
            | ancestor | ancestor_or_self | following_sibling
            | following | preceding_sibling | preceding
NameTest   := a | *    where a ∈ Σ
Step       :=  Axis::NameTest
PathExpr   := Step | PathExpr/PathExpr | PathExpr union PathExpr
            | PathExpr[TestExpr]
TestExpr   := ?PathExpr | not TestExpr | TestExpr and TestExpr
```

Figure 2.6: Syntax of Core XPath 1.0

$$
\begin{array}{l}
[\![\text{ax}::a]\!]^t_{axis} = \{(v_1, v_2) \in [\![\text{ax}]\!]^t_{axis} \mid \mathsf{lab}^t(v_2) = a\} \\
[\![\text{ax}::*]\!]^t_{axis} = [\![\text{ax}]\!]^t_{axis} \\
[\![P_1/P_2]\!]^t_{path} = [\![P_1]\!]^t_{path} \circ [\![P_2]\!]^t_{path} \\
[\![P_1 \text{ union } P_2]\!]^t_{path} = [\![P_1]\!]^t_{path} \cup [\![P_2]\!]^t_{path} \\
[\![P[T]]\!]^t_{path} = \{(v_1, v_2) \in [\![P]\!]^t_{path} \mid v_2 \in [\![T]\!]^t_{test}\}
\end{array}
$$

$$
\begin{array}{l}
[\![?P]\!]^t_{test} = \{v \mid \exists v' \in \text{Dom}(t),\ (v, v') \in [\![P]\!]^t_{path}\} \\
[\![\text{not } T]\!]^t_{test} = \text{Dom}(t) - [\![T]\!]^t_{test} \\
[\![T_1 \text{ and } T_2]\!]^t_{test} = [\![T_1]\!]^t_{test} \cap [\![T_2]\!]^t_{test}
\end{array}
$$

Figure 2.7: Semantics of Core XPath 1.0

whether the node bound to $\$x$ is equal to the current node, or $[\$x \text{ is } \$y]$, to test node equalities. The size of path and test expressions $\|P\|$ and $\|T\|$ is the number of symbols in these expressions. We write $\text{Var}(P)$ and $\text{Var}(T)$ for the set of free variables in the respective expressions (defined similarly as for FO, where $\text{for}\$x$ is viewed as a binder).

Since free variables may occur in expressions, we need to adapt the interpretation: *CoreXPath2.0* expressions P and T are interpreted on a tree t modulo some assignment of the tree variables of P and T respectively into the nodes of t. The interpretations of P and T are respectively denoted $[\![P]\!]^{t,\alpha}_{path}$ and $[\![T]\!]^{t,\alpha}_{test}$. Hence $[\![P]\!]^{t,\alpha}_{path}$ is a subset of node pairs of t while $[\![T]\!]^{t,\alpha}_{test}$ is a subset of nodes of t. The interpretation of *CoreXPath1.0* expressions is naturally extended to interpretation modulo an assignment. Hence we only give the semantics of the new expressions of *CoreXPath2.0*, in Fig. 2.9. *CoreXPath2.0* can be viewed as a navigational language which selects nodes with variables along the navigation.

CoreXPath2.0 path expressions P with n free variables $x_1, \ldots, x_n$ can define n-ary queries $\mathcal{Q}(P)$ by, for all $t \in \mathcal{T}_{unr}(\Sigma)$:

$$\mathcal{Q}(P)(t) \ = \ \{(u_1, \ldots, u_n) \mid [\![P]\!]^{t,[x_1 \mapsto u_1] \ldots [x_n \mapsto u_n]}_{path} \neq \varnothing\}$$

2.8.2 Expressiveness and Complexity of *CoreXPath1.0*

A lot of works have been done on *CoreXPath1.0*. See (BK07) for an overview or the survey (GKP03b).

Gottlob, Koch, and Pichler (GKP03a, GKP05) present an efficient algorithm for answering monadic queries by path expressions P on trees t in time $O(\|P\|.\|t\|)$. This gives a quadratic binary query answering algorithm for *CoreXPath1.0*, in time $O(\|P\|.\|t\|^2)$. The main evaluation trick of *CoreXPath1.0* lies in that the set of

```
NameTest   := a | *    where a ∈ Σ
Step       :=  Axis::NameTest
NodeRef    := . | $x       where x ∈ X
PathExpr   :=
                Step | PathExpr/PathExpr | PathExpr union PathExpr
              | PathExpr[TestExpr]
              | NodeRef
              | PathExpr intersect PathExpr
              | PathExpr except PathExpr
              | for $x in PathExpr return PathExpr
TestExpr   := ?PathExpr | not TestExpr | TestExpr and TestExpr | CompTest
CompTest   := NodeRef is NodeRef
```

Figure 2.8: Syntax of *CoreXPath2.0*

$$[\![.]\!]^{t,\rho}_{path} = \{(v,v) \mid v \in \mathrm{Dom}(t)\}$$
$$[\![\$x]\!]^{t,\rho}_{path} = \{(v,\rho(x)) \mid v \in \mathrm{Dom}(t)\}$$

$$[\![P_1 \text{ intersect } P_2]\!]^{t,\rho}_{path} = [\![P_1]\!]^{t,\rho}_{path} \cap [\![P_2]\!]^{t,\rho}_{path}$$
$$[\![P_1 \text{ except } P_2]\!]^{t,\rho}_{path} = [\![P_1]\!]^{t,\rho}_{path} - [\![P_2]\!]^{t,\rho}_{path}$$
$$[\![\text{for } \$x \text{ in } P_1 \text{ return } P_2]\!]^{t,\rho}_{path} = \{(v_1,v_3) \mid \exists v_2 \in \mathrm{Dom}(t).$$
$$(v_1,v_2) \in [\![P_1]\!]^{t,\rho}_{path} \text{ and } (v_1,v_3) \in [\![P_2]\!]^{t,\rho[x\mapsto v_2]}_{path}\}$$
$$[\![. \text{ is } \$x]\!]^{t,\rho}_{test} = \{\rho(x)\}$$
$$[\![. \text{ is } .]\!]^{t,\rho}_{test} = \mathrm{Dom}(t)$$
$$[\![\$x \text{ is } \$y]\!]^{t,\rho}_{test} = \{\rho(x) \mid \rho(x) = \rho(y)\}$$

Figure 2.9: Semantics of the new expressions of *CoreXPath2.0*

successors $S_a(N) = \{u' \mid \exists u \in N,\ a(u,u')\}$ of a set of nodes N by a standard axis a, in a tree t, is computable in linear time $O(\|t\|)$. This can be extended to full *CoreXPath1.0* expressions, so that computing $S_P(N)$, for some *CoreXPath1.0* expression P, is in linear time $O(\|P\|.\|t\|)$. The query evaluation complexity of *CoreXPath1.0* and several fragments of it are precisely characterized in (GKPS05), where it is proved in particular that *CoreXPath1.0* is PTIME-hard.
CoreXPath1.0, with respect to binary queries, has the same expressiveness as the two-variable fragment of FO over unranked trees (MdR05). It is also shown in (Mar05b, Mar05a) that closure of *CoreXPath* under path complementation can express all binary FO queries (on unranked trees).
Static analysis problems for *CoreXPath1.0* (such as inclusion, satisfiability) are considered in (Gen06, GL06, MS04, Woo03, NS02b). In particular, inclusion is EXPTIME-complete.

2.8.3 Expressiveness and Complexity of *CoreXPath2.0*

CoreXPath2.0 is introduced in (tCM07), where query equivalence is shown to be decidable via a complete axiomatization. Complexities of query inclusion for several expressive fragments are given in (tCL07). Query evaluation complexity of an FO-expressive *CoreXPath2.0* fragment is studied in (FNTT07), and fully described in Chapter 4. As we will see in Section 4.2, there is a linear embedding of FO in unranked trees into *CoreXPath2.0*, and conversely.

Before the introduction of *CoreXPath2.0*, the satisfiability problem for several fragments of the full language XPath 2.0 have been considered in (Hid03), where a tractability frontier is established.

2.8.4 XPath-like languages

In (Mar05a), Marx proves that a slight extension of *CoreXPath1.0*, called *Conditional XPath*, that allows transitive closure $(\text{axis} :: a[\text{test}])^+$ leads to FO expressiveness, with respect to binary queries. This is a kind of 'until' operator to express things like "*do an* axis *step until* test *is false*". Like *CoreXPath1.0* unary queries, Conditional XPath unary queries P can be evaluated in time $O(\|P\|.\|t\|)$ on trees t (Mar04).
Regular XPath, the extension of *CoreXPath1.0* with path equalities[6] and transitive closure of arbitrary path expressions is introduced in (tC06). It is shown that over unranked trees, Regular XPath has the same expressive power with respect to binary queries as FO*, the extension of FO with transitive closure of formulas with exactly two free variables (it is a fragment of FO+TC_1 as exactly two free variables are allowed). As a fragment of FO+TC_1, which is strictly less expressive than MSO, FO* is a strictly less expressive than MSO (tCS08). Another open problem is to know whether the path equality is necessary (it is conjectured that it is).
Fragments of *CoreXPath1.0* that allow comparison of data-values are considered in (BDM*06), where it is shown that satisfiability and containment are decidable.

2.8.5 Caterpillars

XPath is basically a language to specify sequences of basic axis steps (given by the axis), with the possibility to perform tests. This makes XPath a two-sorted language, with *path* and *test* expressions. Another paradigm is to take regular expressions of basic axis steps. Hence there is only one sort of expressions. This is the foundation of *regular path queries*, defined on edge labeled graphs in (ABS00), which on trees are exactly the same as *caterpillar expressions* (BKW00). Caterpillar expressions C are generated by the grammar:

$$\begin{array}{lcl} axis & := & self \mid child \mid parent \mid left \mid right \\ test & := & a \in \Sigma \mid root \mid leaf \mid first \mid last \\ C & := & 0 \mid 1 \mid axis \mid test \mid C \cup C \mid C \circ C \mid C^* \end{array}$$

Caterpillar expressions C are interpreted as binary relations $[\![C]\!](t)$ on a tree t (or equivalently as binary queries $[\![C]\!]$), as follows:

[6] The path equality $p \approx p'$ of two path expressions p, p' is a test expression that holds at some node u if there exists a node that can be reached from u both by p and p'

$$
\begin{array}{lcl@{\qquad}lcl}
[\![self]\!](t) & = & \{(u,u) \mid u \in \mathrm{Dom}(t)\} & [\![child]\!](t) & = & \{(u,v) \mid u \prec^t_{ch} v\} \\
[\![parent]\!](t) & = & \{(u,v) \mid v \prec^t_{ch} u\} & [\![right]\!](t) & = & \{(u,v) \mid u \prec^t_{ns} v\} \\
[\![left]\!](t) & = & \{(u,v) \mid v \prec^t_{ns} u\} & [\![a]\!](t) & = & \{(u,u) \mid \mathsf{lab}^t(u) = a\} \\
[\![root]\!](t) & = & \{(\mathsf{root}^t, \mathsf{root}^t)\} & [\![leaf]\!](t) & = & \{(u,u) \mid u \text{ is a leaf}\} \\
[\![first]\!](t) & = & \{(u,u) \mid \neg\exists v, v \prec^t_{ns} u\} & [\![last]\!](t) & = & \{(u,u) \mid \neg\exists v, u \prec^t_{ns} v\} \\
[\![0]\!](t) & = & \varnothing & [\![1]\!](t) & = & \mathrm{Dom}(t) \times \mathrm{Dom}(t) \\
[\![C_1 \cup C_2]\!](t) & = & [\![C_1]\!](t) \cup [\![C_2]\!](t) & & & \\
[\![C_1 \circ C_2]\!](t) & = & [\![C_1]\!](t) \circ [\![C_2]\!](t) & & & \\
[\![C^*]\!](t) & = & ([\![C]\!](t))^* & & &
\end{array}
$$

where $([\![C]\!](t))^*$ is the reflexive and transitive closure of $[\![C]\!](t)$. Note that the caterpillar expression 1 is useless since it is expressible by $parent^* \circ child^*$. Caterpillar expressions are closely related to tree-walking automata (BKW00). As tree-walking automata, caterpillar expressions are strictly less expressive than MSO over unranked trees, with respect to binary queries. Moreover, their expressiveness is incomparable to FO, as caterpillar expressions somehow loose their way: for instance, the query which selects all node pairs (u,u) such that the parent of u is labeled a is not expressible. Goris and Marx add a loop operator $loop(C)$ that allow one to come back to the starting node (GM05). In particular, $loop(C)$ is interpreted by $[\![loop(C)]\!](t) = [\![C]\!](t) \cap [\![self]\!](t)$. This results in an extension called *looping caterpillars*. Looping caterpillars captures FO on unranked trees, with respect to binary queries. Their query evaluation problem is still polynomial both in the size of the tree and in the size of the caterpillar. The authors also propose an MSO-complete extension with Monadic Datalog tests which still have a tractable query evaluation problem. Finally, they characterize looping caterpillars by tree walking automata with pebbles, with a restricted pebbling policy.
The idea to combine binary relations by the use of union, composition, and (possibly) transitive closure, but without variables, is not new. It is known in its most recent version as the Tarski's relation algebra (Tar41, NT77, Ng84, Mar05c). Before, several calculus of (binary) relations and their implications in mathematics have been introduced by De Morgan, Peirce, and Schröder during the 19th Century. The first paper on the topic has been written by De Morgan (dM60). See also (Giv06) for the relationship to mathematics and (Pra92) for an historical overview.

2.9 Temporal Logics

Temporal logics are used to describe properties of systems and their evolution in time. Systems are usually modeled by possibly infinite labeled transition graphs. Temporal logics are widely used in formal verification to check properties of software and hardware systems for instance. Temporal logics include temporal operators to express things like "there is a node reachable from the current node which satisfies some property" (eventuality), or "all nodes reachable from the current node satisfies some property" (globality), or "there is path from the current node to a node v such that some property P holds for all the inner-nodes of the path, and v is the first node such that some other property P' holds" . On labeled transition systems, the nodes are actually states, and edges represent state evolution in time. Hence, the three last time operators can be described as follows: eventuality is the fact that some property holds in a future state, globality is the fact that some property holds in all future states, and the last operator is an *until* operator, meaning

that the property P holds until P' holds. The existing temporal logics differ on how they can describe the future.

Linear Temporal Logic (LTL) is a temporal logic where the future is viewed as sequence of nodes, *ie* paths. Over strings, LTL is known to have the same expressive power as FO$[\prec_{ns^*}]$ (Kamp's Theorem). An analogous temporal logic for trees is *Computation Tree Logic* (CTL) (and its extension CTL*) (CE82). CTL* include a next operator, an until operator and an operator for eventuality, but, contrarily to LTL, more complex paths can be defined, in particular by exploiting the branching structure of the tree. Let us denote by CTL$^*[\sigma]$ the CTL* formulas where the relations E belongs to σ. On finite binary trees, it is known that CTL$^*[\prec_{ch_1}, \prec_{ch_2}]$ = FO$[\prec_{ch^*}, \prec_{ch_1}, \prec_{ch_2}]$ (HT87), with respect to Boolean queries. Over unranked trees, past operators by which to follow inverses of the binary relations of the signature are needed to get FO-completeness. This is called CTL$^*_{past}$ in (BL05), where it is proved that over unranked trees, and with respect to both Boolean and unary queries[7], FO$[\prec_{ch^*}, \prec_{ns^*}]$ and CTL$^*_{past}[\prec_{ch}, \prec_{ns}]$ are equally expressive. Satisfiability of CTL$^*_{past}[\prec_{ch}, \prec_{ns}]$ is in EXPTIME.

The modal μ-calculus L_μ is a modal logic with (greatest and least) fixed points. Although it has been introduced to describe properties of labeled transition graphs (Koz83), we shall consider it in the context of trees. Over infinite binary trees, it has been shown to capture MSO (Niw88, EJ91) (with respect to Boolean queries). Over unranked trees, the μ-calculus has been studied in (BL05). See also (Lib06) for a recent survey. With respect to Boolean queries, MSO$[\prec_{ch}, \prec_{ns}]$ = $L_\mu[\prec_{fc}, \prec_{ns}]$. If instead of the first-child relation the signature contains the child relation, one has to add backward modalities (analogous to the past for CTL) to get MSO-completeness. This expressiveness result also holds for unary queries. The model-checking problem of formulas ϕ of the μ-calculus on acyclic labeled transition systems S is in time $O(\|\phi\|^2\|S\|)$ (Mat02). Hence it also applies for tree structures over $\{\prec_{ns}, \prec_{fc}\}$. This complexity can be reduced to $O(\|\phi\|\|S\|)$ when considering the alternation-free μ-calculus, which is equally expressive as the μ-calculus on acyclic structures (Mat02). The satisfiability problem is in EXPTIME, even with backward modalities (Var98). In particular, the best known upper-bound is in $2^{O(\|\phi\|^4 log\ \|\phi\|)}$, over arbitrary transition systems (GTW02). A logic based on the μ-calculus, but designed for trees, is proposed in (Gen06, GLS07). It matches the expressiveness of MSO over unranked trees, while allowing satisfiability testing in $2^{O(\|\phi\|)}$.

The *Propositional Dynamic Logic* (PDL) is considered in (ABD*05) in the context of unranked trees. Its satisfiability is in EXPTIME but a characterization of its expressiveness is still unknown.

2.10 Monadic Datalog and Conjunctive Queries

Monadic Datalog has been proposed as a query language for unranked trees (GK04). This is the restriction of Datalog (EF05) where lhs of rules are monadic. By distinguishing a particular lhs, call the goal, monadic datalog programs can defined unary queries. For instance, the following datalog program, whose goal is Q, selects all nodes which are at an even distance from the root of the tree:

[7] Temporal logics describe properties of nodes in a labeled transition graphs. When restricted to tree structures, formulas can be viewed as unary queries, and as Boolean queries if one considers the root only (in particular, a tree is in the language defined by some formula ϕ iff its root satisfies the property defined by ϕ) (BL05).

$$\begin{array}{lll} Q(x) & :- & \mathsf{root}(x) \\ Q(x) & :- & z \prec_{ch} y,\ y \prec_{ch} x,\ Q(z) \end{array}$$

It is proved in (GK04) that Monadic Datalog over the predicates $\prec_{fc}, \prec_{ns}$, leaf, root, last-child[8], and $\mathsf{lab}_a, a \in \Sigma$ has the same expressiveness as MSO[$\prec_{ch}, \prec_{ns}$], w.r.t. unary queries. Indeed, they can simulate tree automata, thanks to the fixpoint interpretation of the rules. Moreover, query evaluation of a monadic datalog program p on an unranked tree t is in time $O(\|p\|.\|t\|)$. Monadic datalog is used as the core language of Elog, a language for visual data extraction on the web (GKB*04).

Conjunctive queries (CQ) over a signature σ are FO[σ] formulas of the form:

$$\exists \overline{x}\ \phi(\overline{x}, \overline{y})$$

where ϕ is a conjunction of atoms over σ. The set of conjunctive queries over σ is denoted CQ[σ]. Thanks to free variables, conjunctive queries can – indeed – define queries. Complexity of conjunctive queries over arbitrary structures have been extensively studied, as they are very closed to SQL queries (AHV95). Model-checking for CQ is NP-hard, but several tractable subclasses has been introduced, for instance, *acyclic conjunctive queries* (ACQ) (Yan81), which are defined through a notion of acyclicity of their query hypergraphs. The main result for acyclic conjunctive queries ϕ is that they can be evaluated in time $O(\|D\|.\|\phi\|.\|\mathcal{Q}(\phi)(D)\|)$, on a database D (Yan81). In (DG06), an extension of ACQ with disequalities is introduced. The model-checking problem becomes NP-complete in combined complexity, but can be evaluated in time $O(f(\|\phi\|).\|D\|.\|\mathcal{Q}(\phi)(D)\|.log^2\|D\|)$, for some exponential f. See also (BDG07) for recent result on answer enumeration for ACQs.

Tree-pattern queries are tree-shaped queries that intuitively correspond to acyclic conjunctive queries over descendant and child axis, and label tests. They are strictly less expressive than first-order logic and therefore less expressive than the query languages proposed in this thesis. Many algorithms exist to evaluate tree-pattern queries but none of them is linear in the size of the output (BKS02, JWLY03, Che06, ZXM07). However, evaluation of tree-pattern queries is still an active research topic as in many practical cases they are sufficiently expressive.

In the binary setting, the acyclicity of a conjunctive query ϕ corresponds to the acyclicity of its *query graph*[9]. CQ in trees over XPath axis are considered in (GKS04), where a frontier of tractability is established.

Concerning expressiveness, union of ACQ whose atomic predicates are FO[$\sigma_{unr}(\Sigma)$] with two free variables captures FO[$\sigma_{unr}(\Sigma)$], with respect to n-ary queries (Mar05a). Actually, we state and prove a similar result in this thesis, but with different techniques, and the proof also holds for MSO.

[8] Leaf is a unary predicate which holds for all leaves, and last-child is a binary predicate relating a node to its last-child

[9] The query graph of ϕ is the graph where vertices are variables of ϕ and there is an undirected edge between two vertices x and y if some atom $P(x, y)$ or $P(y, x)$ occurs in ϕ

2.11 Unordered Trees

XML documents are naturally ordered by the (total) order induced by their linear serialization (often called document order). This order is used to order the sequences of children of the nodes. Although XML documents are usually modeled by unranked ordered trees, this order might be sometimes irrelevant, for instance when the tree represents records. Less work have been done on unordered unranked trees. One way to measure the impact of ordering is by considering *order-invariance*. An order-invariant query is a query which exploits an arbitrary order, but the choice of the order does not matter (in particular, the result remains the same when choosing another order). Order-invariance is studied in (BL05, Cou91, GS00, Lib04b, BS05a) (non-exhaustive).
While logics for ordered trees can use the order to define some counting properties of the nodes (for instance, counting the number of nodes labeled a modulo in MSO), this cannot be done in unordered trees, so that usually counting constructions are natively added to logics. For instance, *Counting MSO* extends MSO with counting modulo quantifiers to count cardinalities of definable sets modulo (Cou90b). Other extensions or restriction of CMSO exist and their correspondence with tree automata model for unordered trees are considered in (BT05). See also (Lib06) for a survey.
Spatial logics have been used to express properties of structures such as unordered trees (CG04), but also graphs (CGG02, DGG04) and heaps (Rey02). The main ingredient of spatial logics are spatial connectives: basically, these connectives are derived from operators that can be used to generate the domain of interpretation. In the context of unordered trees, the *Tree Query Logic* (TQL) have been introduced in (CG04) as a query language for semi-structured data. Its expressiveness and satisfiability with respect to Boolean queries are studied in (BTT05, Bon06). The variant of TQL on ordered trees – which is defined and studied in the second part of this thesis – is mentioned as an interesting issue in (CG04, BTT05, Gen06).

2.12 n-Ary Query Languages

We can conclude from the previous sections that the literature is dominated by query languages for Boolean or unary queries. Some of the formalisms previously introduced however allows one to define n-ary queries, such as attributed grammars (NV02), tree automata (NPTT05), *CoreXPath2.0*, conjunctive queries (GKS04). In this section, we survey other formalisms for n-ary queries.
XQuery is a W3C standard to – indeed – query XML documents (BCF*07), by which to select tuples of nodes and rearrange them into output trees. Its underlying selection formalism is XPath. XPath (2.0) is now a subset of XQuery whose core has been cleanly formalized, so that this thesis mainly concentrates on XPath 2.0 instead of XQuery. From a practical point of view, a lot of work attempt to optimize XQuery queries, but its logical core has been formalized only recently (Koc05).
Several paper attempt to combine Boolean or unary queries to define n-ary queries. In (Sch00), it is shown how to define n-ary queries by combining unary queries. This is done by using regular path expressions over an alphabet of unary formulas from some logic L. If L is unary FO (resp. unary MSO), then the full combination language is FO (resp. MSO) expressive, wrt to n-ary queries. This idea is similar to the logic ETL (NS00).
(ABL07) proposes a composition language to combine Boolean and unary queries.

Sufficient expressiveness conditions on the basic Boolean and unary query languages are given to get an FO (resp. MSO)-complete query language. Model-checking complexity of the combined language is then related to the model-checking complexities of the basic query languages. In this thesis, we propose a simpler, minimal and expressive composition language and mainly focus on query evaluation for n-ary queries, which was left open by (ABL07), and was not considered in (Sch00).

XDuce (HP03a) and *CDuce* (BCF03b) are ML-like programming languages for XML applications, and both include a type-checking system. CDuce extends XDuce in that it is a real functional programming language including a richer pattern-matching language, a richer type-system (with type functions), and parametric polymorphism. *Regular expression patterns* are the basis of their pattern-matching languages. A regular expression pattern is built over tree variables, constructors of a hedge algebra and regular expressions. A linearity condition on variables occurring in a pattern imposes that subtree equality tests cannot be performed. In the second part of this thesis, we study the spatial logic TQL for ordered trees which can also be viewed as an extension of the (recursive) pattern-language of XDuce. The main difference here is that we allow Boolean operators and drop the linearity condition for variables.

Part I

From Binary to Arbitrary Arity Queries

COMPOSING BINARY QUERIES

3

CONTENTS

THIS chapter introduces a composition language that combines binary queries taken from some binary query language in order to define n-ary queries. Binary queries are used to navigate from a starting node to an ending node, while first-order variables are used to define output tuples. The language is then closed by disjunction and a form of conjunction. Expressiveness on unranked trees and complexity of query evaluation are investigated. In particular, the composition language is proved to be FO-complete (resp. MSO-complete) with respect to n-ary queries as soon as the underlying binary query language is FO-complete (resp. MSO-complete) with respect to binary queries.

3.1 INTRODUCTION

Boolean and monadic queries have been intensively studied from theoretical and practical point of views over last years. Several other formalisms, such as XPath, can also be used to define binary queries. Less work has been done however on n-ary queries (see Section 2.12 for an overview of the existing query languages).
Binary queries can be viewed as a way to relate a starting node to an ending node, making possible some kind of navigation. This is particularly true for XPath path expressions for instance, by which a sequence of basic navigation steps combined with monadic tests allow one to define more complex binary relations. By adding variables, a binary query language can naturally be extended into an full n-ary query language: the idea is to use binary queries to navigate in the tree, and to select output tuple components along the navigation thanks to variables.
In this chapter, we formalize this way of defining n-ary queries. In particular, we propose a simple and foundational composition language $\mathcal{C}(L)$ that allows one to combine binary queries – from an arbitrary query language L – to define n-ary queries. The choice of the underlying binary query language L is parametric, making the composition language generic.
Basically, this language turns any binary query language into a full n-ary query language, by the use of composition, variables, disjunction and conjunction. Variables are only used to select components of the output tuple, and there is no quantifier. The semantics is given by binary relations of nodes, modulo some assignment of the variables. Hence composition is naturally interpreted as composition of relations, and other Boolean connectives by the usual set operations on binary relations. The conjunction however is defined by means of filters, as in XPath. A filter in the navigation is like a branching which splits the navigation. We introduce a syntactic restriction, called the *non-variable sharing*, which forbids to repeat the same variable on both sides of a composition. This restriction is crucial to get a polynomial-time query evaluation algorithm (as soon as there is a polynomial-time query evaluation algorithm for the underlying binary query language).
Concerning expressiveness, with respect to n-ary queries, we prove that the composition language is equally expressive as FO (resp. MSO) on unranked trees, even with the non-variable sharing restriction, as soon as the binary query language is equally expressive as FO (resp. MSO) with respect to binary queries. The proof is standard and uses folklore results in the spirit of (Mar05a, Sch00, ABL07, Tho84, MR03). It is based on Shelah's composition method (She). The non-variable sharing fragment without disjunction is closely related to acyclic conjunctive queries. Adding disjunction to conjunctive queries is usually done by taking union of disjunction-free conjunctive queries. Our composition language allows one to use disjunction at arbitrary positions in the formula, while keeping the same complexity bound for query evaluation as acyclic conjunctive queries over graphs.
Unary queries can also be used to navigate in the tree thanks to a domain restriction called *subhedge restriction*. The nodes selected by a unary query are either outputted as components of the answer or used to define a subdomain in which other queries can be done. All the expressiveness and complexity results obtained for binary query composition also apply to unary query composition.
Finally, in Chapter 4, we apply the composition language to study XPath fragments that allow one to define n-ary queries.

Related Work The idea of combining Boolean, unary, or binary queries to define n-ary queries is not new.
Schwentick (Sch00) shows how to define n-ary queries by combining unary queries by means of regular path expressions of unary formulas of several fragments of FO. He states a decomposition lemma similar to Lemma 3.5.4, from which he proves that combination of unary FO-queries is complete for n-ary FO (Proposition 4.2 of (Sch00)). Our language however is simpler and has led to extensions of XPath with variables (as shown in the next Chapter). Moreover, we were interested in query evaluation which was not considered in (Sch00).
Arenas, Barcelo and Libkin also present a composition language in (ABL07). While we combine binary queries, they provide a mechanism to combine Boolean and unary queries. Basically it consists of taking a Boolean query language L_1 to define words over an alphabet of sets of formulas taken from a unary query language L_2. Consider the shortest path from some node u to some node v. Every node of this path satisfies in the tree a set of L_2 unary formulas. Hence this path can be mapped to a word w of sets of L_2-formulas. The pair (u, v) is said to satisfy $\chi(x, y)$ if w satisfies χ, where $\chi \in L_1$. This mechanism allows one to define binary queries. On the other hand, terms t with variables are used to point nodes of the tree. They are built over the signature containing a function symbol to take the least common ancestor, and a root constant to point the root of the tree. Finally, n-ary queries are defined by taking Boolean combinations of atoms $\chi(t, t')$, where t, t' are terms and $\chi \in L_1$, as well as atoms of the form $t \prec_{ch^*} t'$ and $t \prec_{ns^*} t'$. The authors give sufficient expressiveness conditions on the basic Boolean and unary query languages to get an FO (resp. MSO)-complete query language (Theorem 2, (ABL07)). In Theorem 3 of (ABL07), they relate the model-checking complexity of the combined language to the model-checking complexities of the basic query languages.
In the present chapter, we focus on query evaluation for n-ary queries, which was left open by (ABL07), and our composition language is simpler. They investigate model-checking only, where non-variable sharing does not matter. It is not clear whether some fragment of the combination language of (ABL07) does correspond to our fragment without variable sharing. It may be definable by introducing a notion of acyclicity as in conjunctive queries, but it not as simple as our syntactic restriction.
Proving FO and MSO-completeness for fragments of our composition language is done via the Shelah's decomposition method (She). The main part of the proof is a folkore results about the decomposition of FO or MSO formulas into existentially quantified Boolean combinations of binary FO or MSO formulas in the spirit of (Mar05a, Sch00, ABL07, Tho84, MR03).

Organization of the chapter The composition language is introduced in Section 3.2. It is then related to FO over binary query atoms and conjunctive queries in Section 3.3. Query evaluation of composition queries is investigated in Section 3.4. FO and MSO completeness are proved in Section 3.5 where a composition language for unary queries is proposed.

3.2 COMPOSITION LANGUAGE

In this section, we first define the composition language and its non-variable sharing fragment, and then give examples.

3.2.1 Syntax and Semantics

We start from a query language $L = (\mathbb{Q}_L, \mathsf{ar}, \|.\|, \mathcal{Q}_L(.))$ for binary queries – also called *basic queries* – and a countable set $\mathcal{X}$ of variables. Remind that $\mathbb{Q}_L$ is a set of query expressions interpreted by $\mathcal{Q}_L(.)$ as queries whose arity is given by ar. The size of the query expressions is given by $\|.\|$. The language L is extended into a full n-ary query language by using a composition operator denoted by $\circ$, disjunction, a form of conjunction, and free variables from $\mathcal{X}$. Conjunctions are simulated by test expressions like in XPath. Binary queries are used to navigate in the tree, and free variables are used to select components of output tuples along the navigation.
Formally, compositions are modeled by *composition formulas*. The set of composition formulas over binary queries from L is denoted by $\mathcal{C}(L)$, and composition formulas ϕ are defined by the following abstract syntax:

ϕ	$::=$		*composition formula*
		$\mathfrak{q}$	$\mathfrak{q} \in \mathbb{Q}_L$, *binary query*
	$\mid$	x	$x \in \mathcal{X}$, *variable*
	$\mid$	$\phi \circ \phi$	*composition*
	$\mid$	$[\phi]$	*test*
	$\mid$	$\phi \vee \phi$	*disjunction*

$\mathrm{FVar}(\phi)$ denotes the set of free variables of ϕ (there are no bound variables), and we write $\phi(\overline{x})$ to mean that the free variables of ϕ are exactly those forming the tuple $\overline{x}$. The set of subformulas of ϕ is denoted by $\mathrm{Sub}(\phi)$.
Composition formulas are interpreted on unranked trees modulo some valuation of their variables. In particular, as a navigation language, and similarly to *CoreXPath2.0*, a composition formula ϕ denotes a binary relation of the nodes of the tree on which it is interpreted.
Let $\phi \in \mathcal{C}(L)$, $t \in \mathcal{T}_{unr}(\Sigma)$ and $\rho : \mathrm{FVar}(\phi) \to \mathrm{Dom}(t)$ be a valuation of the variables of ϕ. The formula ϕ denotes on t a binary relation $[\![\phi]\!]^{t,\rho}$ inductively defined by:

$$
\begin{array}{lcl}
[\![x]\!]^{t,\rho} & = & \{(\rho(x), \rho(x)\} \\
[\![\mathfrak{q}]\!]^{t,\rho} & = & \mathcal{Q}_L(\mathfrak{q})(t) \\
[\![\phi_1 \circ \phi_2]\!]^{t,\rho} & = & [\![\phi_1]\!]^{t,\rho} \circ [\![\phi_2]\!]^{t,\rho} \\
[\![[\phi]]\!]^{t,\rho} & = & \{(u,u) \mid \exists v,\ (u,v) \in [\![\phi]\!]^{t,\rho}\} \\
[\![\phi_1 \vee \phi_2]\!]^{t,\rho} & = & [\![\phi_1]\!]^{t,\rho} \cup [\![\phi_2]\!]^{t,\rho}
\end{array}
$$

Note that $\circ$ is associative, since the composition of binary relations is associative. Hence we write $\phi_1 \circ \phi_2 \circ \phi_3$ instead of $(\phi_1 \circ \phi_2) \circ \phi_3$ or $\phi_1 \circ (\phi_2 \circ \phi_3)$.
Composition formulas ϕ with n-free variables[1] $x_1, \ldots, x_n$ (given in order), when starting the navigation from the root node, define n-ary queries as follows:

$$\mathcal{Q}_{\mathcal{C}(L)}(\phi)(t) = \{(u_1, \ldots, u_n) \mid \exists u \in \mathrm{Dom}(t), (\mathsf{root}^t, u) \in [\![\phi]\!]^{t,[x_1 \mapsto u_1, \ldots, x_n \mapsto u_n]}\}$$

[1] As usual, we assume a linear order on the variables

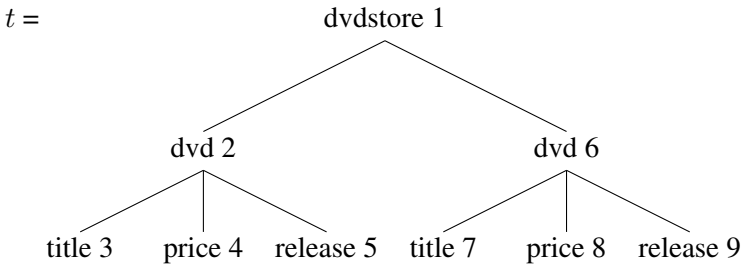


Figure 3.1: The tree t and their node identifiers

The size of a composition formula is the number of nodes of the term by which it is represented, where basic query expressions are considered to be of constant size. More formally, it is inductively defined by:

$$\begin{aligned}
\|\mathfrak{q}\| &= \|x\| = 1 \\
\|\phi_1 \circ \phi_2\| &= \|\phi_1\| + \|\phi_2\| + 1 \\
\|\phi_1 \vee \phi_2\| &= \|\phi_1\| + \|\phi_2\| + 1 \\
\|[\phi]\| &= \|\phi\| + 1
\end{aligned}$$

The arity $\mathsf{ar}(\phi)$ of a composition formula ϕ is the number of free variables of ϕ. Hence, $\mathcal{C}(L)$ can be viewed as the query language $(\mathcal{C}(L), \mathsf{ar}(.), \|.\|, \mathcal{Q}_{\mathcal{C}(L)}(.))$.

3.2.2 The non-variable sharing fragment $\mathcal{C}^{\text{nvs}}(L)$

As it will be shown in Section 3.5, the use of variables in composition formulas can be restricted in such a way that variables are not shared by members of compositions. This can be done while keeping FO or MSO-completeness. This fragment is called the *non-variable sharing fragment*, denoted $\mathcal{C}^{\text{nvs}}(L)$, for any binary query language L. As shown in Section 3.4, this restriction is crucial for the complexity of query evaluation.
Formulas of $\mathcal{C}^{\text{nvs}}(L)$ must satisfy the following property:

$$\forall \phi \in \mathcal{C}^{\text{nvs}}(L), \text{ if } \phi_1 \circ \phi_2 \in \text{Sub}(\phi), \text{ then } \text{FVar}(\phi_1) \cap \text{FVar}(\phi_2) = \varnothing$$

The non-variable sharing restriction is also informally denoted by NVS($\circ$). For instance $x \circ \mathfrak{q} \circ y \circ \mathfrak{q}' \circ z$ and $(x \circ \mathfrak{q} \circ y) \vee (x \circ \mathfrak{q}' \circ [\mathfrak{q}'' \circ z])$ satisfies NVS($\circ$) while $x \circ \mathfrak{q} \circ y \circ \mathfrak{q}' \circ x$ and $x \circ [\mathfrak{q} \circ y \circ \mathfrak{q}' \circ x]$ do not.

3.2.3 Examples

Consider the tree t of Fig. 3.1 representing the XML document of a DVD store (data values are omitted for the sake of clarity). The nodes of the tree are represented by natural numbers, *ie* $\text{Dom}(t) = \{1, \ldots, 9\}$. Let $\gamma_{\text{dvd}}(x,y), \gamma_{\text{title}}(x,y), \gamma_{\text{price}}(x,y), \gamma_{\text{release}}(x,y)$ be four FO$[\sigma_{unr}]$-formulas defined by:

$$\begin{aligned}
\gamma_{\text{dvd}}(x,y) &= x \prec_{ch} y \wedge \mathsf{lab}_{\text{dvd}}(y) & \gamma_{\text{title}}(x,y) &= x \prec_{ch} y \wedge \mathsf{lab}_{\text{title}}(y) \\
\gamma_{\text{price}}(x,y) &= x \prec_{ch} y \wedge \mathsf{lab}_{\text{price}}(y) & \gamma_{\text{release}}(x,y) &= x \prec_{ch} y \wedge \mathsf{lab}_{\text{release}}(y)
\end{aligned}$$

The formulas $\gamma_{\text{dvd}}, \gamma_{\text{title}}, \gamma_{\text{price}}, \gamma_{\text{release}}$ define the binary queries $\mathcal{Q}(\gamma_{\text{dvd}})$, $\mathcal{Q}(\gamma_{\text{title}})$, $\mathcal{Q}(\gamma_{\text{price}})$, $\mathcal{Q}(\gamma_{\text{release}})$ respectively. Consider the composition formula

$$\phi(x,y,z) = \gamma_{\text{dvd}} \circ [\gamma_{\text{title}} \circ x] \circ [\gamma_{\text{price}} \circ y] \circ [\gamma_{\text{release}} \circ z]$$

Evaluation of ϕ is as follows. It starts from the root, and then goes down to nodes labeled dvd. Then three filters are composed, each one selects nodes labeled title, price and release respectively. Hence ϕ defines a ternary query which outputs every triple of respective title, price and release date. In particular, $\mathcal{Q}(\phi)(t) = \{(3,4,5),(7,8,9)\}$. Formally, if we let ρ_1, ρ_2 be the valuations defined respectively by:

$$\begin{array}{lll} \rho_1(x) = 3 & \rho_1(y) = 4 & \rho_1(z) = 5 \\ \rho_2(x) = 7 & \rho_2(y) = 8 & \rho_2(z) = 9 \end{array}$$

we have:

$$\begin{array}{l} [\![\phi]\!]^{t,\rho_1} = \{(1,2)\}, \text{ hence } (\rho_1(x),\rho_1(y),\rho_1(z)) \in \mathcal{Q}(\phi(x,y,z))(t) \\ [\![\phi]\!]^{t,\rho_2} = \{(1,6)\}, \text{ hence } (\rho_2(x),\rho_2(y),\rho_2(z)) \in \mathcal{Q}(\phi(x,y,z))(t) \end{array}$$

The query defined by ϕ could equivalently be defined without tests. The idea is to start from the root, and then go down to some node labeled title, select it with the variable x, go to the right of its next-sibling (which is labeled price), select it with the variable y, and so on. More formally, $\phi(x,y,z)$ is equivalent to

$$\begin{array}{ll} & \phi'(x,y,z) \;=\; \gamma^*_{\text{title}} \circ x \circ \gamma_{\prec_{ns}} \circ y \circ \gamma_{\prec_{ns}} \circ z \\ \text{where} & \gamma_{\prec_{ns}}(x,y) = x \prec_{ns} y \\ \text{and} & \gamma^*_{\text{title}}(x,y) = x \prec^{y}_{ch^*} \wedge \mathsf{lab}_{\text{title}}(y) \end{array}$$

The following table details the interpretation of ϕ':

$$\begin{array}{lcl} [\![\phi'(x,y,z)]\!]^{t,\rho_1} & = & \{(1,5)\} \\ \\ [\![\gamma^*_{\text{title}}]\!]^{t,\rho_1} & = & \{(1,3),(1,7)\} \\ [\![x]\!]^{t,\rho_1} & = & \{(3,3)\} \\ [\![\gamma_{\prec_{ns}}]\!]^{t,\rho_1} & = & \{(3,4),(4,5),(7,8),(8,9),(2,6)\} \\ [\![y]\!]^{t,\rho_1} & = & \{(4,4)\} \\ [\![\gamma_{\prec_{ns}}]\!]^{t,\rho_1} & = & \{(3,4),(4,5),(7,8),(8,9),(2,6)\} \\ [\![z]\!]^{t,\rho_1} & = & \{(5,5)\} \end{array}$$

Observe that the composition of the relations on the right is equal to $\{1,5\}$. Equivalently, ϕ' can be expressed as a composition of *CoreXPath1.0* path expressions with variables, as follows:

$$(\mathsf{descendant} :: \text{title}) \circ x \circ (\mathsf{nextsibling} :: *) \circ y \circ (\mathsf{nextsibling} :: *) \circ z$$

This is again equivalent to the *CoreXPath2.0* expression:

$$\mathsf{descendant} :: \text{title}[.\ \mathsf{is}\ \$x]/\mathsf{nextsibling} :: *[.\ \mathsf{is}\ \$y]/\mathsf{nextsibling} :: *[.\ \mathsf{is}\ \$z]$$

We will see in Section 4.2 that *CoreXPath2.0* and composition of *CoreXPath1.0* expressions are closely related.

3.3 Relation to FO and Conjunctive Queries

In this section, we first relate $\mathcal{C}(L)$ and FO over binary atoms from L, and prove that only two bound variables are needed to translate $\mathcal{C}(L)$ into FO. The given translation can also be viewed as a reformulation of the semantics of composition formulas by means of FO logic. Then we relate $\mathcal{C}^{\text{nvs}}(L)$ with and without disjunction to acyclic conjunctive queries (ACQs) or union of ACQs. Consequently we can use known results about evaluation of ACQ to evaluate disjunction-free $\mathcal{C}^{\text{nvs}}(L)$ formulas.

3.3.1 Relation to FO

Let $L = (\mathbb{Q}, \mathsf{ar}, \|.\|, \mathcal{Q}(.))$ be a binary query language. We denote by FO$[L]$ the first-order logic over atoms of the form $\mathbb{q}(x, y)$, where $\mathbb{q} \in \mathbb{Q}$. The size of an FO$[L]$ formula ϕ is the size of ϕ where query expressions are assumed to be of size 1. For all $i \in \mathbb{N}$, FO$^i[L]$ denotes its fragment which uses at most i bound but possibly reused variables (a free variable which is bound elsewhere is therefore also considered as a bound variable).
Remind that root is a constant symbol interpreted in a tree structure by the root of the tree. It is needed in the translation from $\mathcal{C}(L)$ to FO$[L]$ since $\mathcal{C}(L)$ formulas express navigations starting from the root.

Proposition 3.3.1

1. *FO$[L \cup \mathsf{root}]$ captures $\mathcal{C}(L)$, wrt n-ary queries, modulo a linear time transformation which produces a formula of linear size.*
2. *FO$^2[L \cup \mathsf{root}]$ captures $\mathcal{C}(L)$, wrt n-ary queries, modulo an exptime transformation which produces a formula of exponential size.*

Proof. 1. Let $\gamma(\overline{x})$ be a $\mathcal{C}(L)$-formula. It allows one to navigate from the root to a node while selecting nodes by using variables from $\overline{x}$. If one wants to define an FO$[L \cup \mathsf{root}]$ which defines the same query as $\mathcal{Q}(\gamma(\overline{x}))$, one needs to start from a node and test that there is a node that can be reached from this node. Let x, y be two variables which do not occur in $\overline{x}$. We define the translation $(\!|\gamma(\overline{x})|\!)_{x,y}$ which outputs an FO$[L]$ formula with free variables $x, y, \overline{x}$, and such that, for all trees $t \in \mathcal{T}_{unr}(\Sigma)$, all node tuples $\overline{u}$ and nodes u, v:

$$t, [x \mapsto u, y \mapsto v, \overline{x} \mapsto \overline{u}] \models (\!|\gamma(\overline{x})|\!)_{x,y} \quad \text{iff} \quad (u, v) \in [\![\gamma(\overline{x})]\!]^{t,\overline{u}}$$

The translation is inductively defined by:

$$\begin{array}{lcll}
(\!|z|\!)_{x,y} & = & x = z = y & \\
(\!|\mathbb{q}|\!)_{x,y} & = & \mathbb{q}(x, y) & \\
(\!|[\gamma]|\!)_{x,y} & = & x = y \ \wedge \ \exists z \ (\!|\gamma|\!)_{x,z} & z \text{ fresh} \\
(\!|\gamma_1 \vee \gamma_2|\!)_{x,y} & = & (\!|\gamma_1|\!)_{x,y} \ \vee \ (\!|\gamma_2|\!)_{x,y} & \\
(\!|\gamma_1 \circ \gamma_2|\!)_{x,y} & = & \exists z \ (\!|\gamma_1|\!)_{x,z} \wedge (\!|\gamma_1|\!)_{z,y} & z \text{ fresh}
\end{array}$$

Fresh means that the introduced variable is new and does not occur elsewhere. Finally, we let $\phi(\overline{x}) = \exists y \ (\!|\gamma(\overline{x})|\!)_{\mathsf{root},y}$, and we get $\mathcal{Q}(\phi(\overline{x})) = \mathcal{Q}(\gamma(\overline{x}))$.

2. In the latter translation we had to introduce a variable denoting the starting node and another variable for the ending node, and we had to memorize those variables along the translation. This is needed to translate compositions $\gamma_1 \circ \gamma_2$, as γ_1 might be complex. This results in the introduction of a lot of bound variables. However, we can assume a normal form which allows one to use only two bound variables in

the translation. In particular, we say that $\gamma \in \mathcal{C}(L)$ is in *normal form* if for every subformula $\gamma_1 \circ \gamma_2 \in \text{Sub}(\gamma)$, γ_1 is either a query expression of $\mathbb{Q}$, a variable, or a test. The normal formal can be obtained by applying exhaustively the following rewriting rules (which preserve the semantics):

$$\begin{array}{lcl} (\gamma_1 \vee \gamma_2) \circ \gamma_3 & \rightarrow & (\gamma_1 \circ \gamma_3) \vee (\gamma_2 \circ \gamma_3) \\ (\gamma_1 \circ \gamma_2) \circ \gamma_3 & \rightarrow & \gamma_1 \circ (\gamma_2 \circ \gamma_3) \end{array}$$

The size of the normal form however might be exponential in the size of the original formula.

Assuming a formula $\gamma(\overline{x})$ in normal form, we now give a new translation of it into an $\text{FO}^2[L \cup \mathsf{root}]$ formula defining the same query. Let x, y be two variables which do not occur in $\overline{x}$. For all $\alpha \in \{x, y\}$, we denote by $\widehat{\alpha}$ the variable y if $\alpha = x$ and the variable x if $\alpha = y$. Obviously, $\widehat{\widehat{\alpha}} = \alpha$. We translate $\gamma(\overline{x})$ into $(\!|\gamma|\!)_\alpha$, where α intuitively denotes the starting node of the navigation. It is defined inductively as follows:

$$\begin{array}{lcl} (\!|z|\!)_\alpha & = & z = \alpha \\ (\!|\mathbb{q}_1|\!)_\alpha & = & \exists \widehat{\alpha}\, \mathbb{q}_1(\alpha, \widehat{\alpha}) \\ (\!|\gamma_1 \vee \gamma_2|\!)_\alpha & = & (\!|\gamma_1|\!)_\alpha \ \vee \ (\!|\gamma_2|\!)_\alpha \\ (\!|[\gamma]|\!)_\alpha & = & (\!|\gamma|\!)_\alpha \\ (\!|z \circ \gamma|\!)_\alpha & = & z = \alpha \wedge (\!|\gamma|\!)_\alpha \\ (\!|\mathbb{q}_1 \circ \gamma|\!)_\alpha & = & \exists \widehat{\alpha}\, \mathbb{q}_1(\alpha, \widehat{\alpha}) \ \wedge \ (\!|\gamma|\!)_{\widehat{\alpha}} \\ (\!|[\gamma_1] \circ \gamma_2|\!)_\alpha & = & (\!|\gamma_1|\!)_\alpha \ \wedge \ (\!|\gamma_2|\!)_\alpha \end{array}$$

Note that this translation only introduces at most two bound variables. We now have the following: for all $\alpha \in \{x, y\}$, all trees $t \in \mathcal{T}_{unr}(\Sigma)$, all tuples $\overline{u}$, and all nodes u,

$$t, \alpha \mapsto u, \overline{x} \mapsto \overline{u} \models (\!|\gamma(\overline{x})|\!)_\alpha \quad \text{iff} \quad \exists v\ (u, v) \in [\![\gamma \overline{x}]\!]^{t, \overline{u}}$$

The proof goes by induction on formulas, but we only prove two cases, as the others are either easy or similar:

- $t, \alpha \mapsto u, \overline{x} \mapsto \overline{u} \models (\!|\mathbb{q}_1 \circ \gamma|\!)_\alpha$

  iff $t, \alpha \mapsto u, \overline{x} \mapsto \overline{u} \models \exists \widehat{\alpha} \mathbb{q}_1(\alpha, \widehat{\alpha}) \ \wedge \ (\!|\gamma|\!)_{\widehat{\alpha}}$ (by definition)

  iff there is $v \in \text{Dom}(t)$ such that $(u, v) \in \mathcal{Q}(\mathbb{q}_1)(t)$ and $t, \widehat{\alpha} \mapsto v, \overline{x} \mapsto \overline{u} \models (\!|\gamma|\!)_{\widehat{\alpha}}$ (since α is not free in $(\!|\gamma|\!)_{\widehat{\alpha}}$)

  iff there is $v \in \text{Dom}(t)$ such that $(u, v) \in \mathcal{Q}(\mathbb{q})(t)$ and there is $w \in \text{Dom}(t)$ such that $(v, w) \in [\![\gamma]\!]^{t, \overline{u}}$ (by induction hypothesis)

  iff there is $w \in \text{Dom}(t)$ such that $(u, w) \in [\![\mathbb{q}_1 \circ \gamma]\!]^{t, \overline{u}}$ (by definition of the semantics of composition formulas)

- $t, \alpha \mapsto u, \overline{x} \mapsto \overline{u} \models (\!|[\gamma_1] \circ \gamma_2|\!)_\alpha$

  iff $t, \alpha \mapsto u, \overline{x} \mapsto \overline{u} \models (\!|\gamma_1|\!)_\alpha$ and $t, \alpha \mapsto u, \overline{x} \mapsto \overline{u} \models \wedge (\!|\gamma_2|\!)_\alpha$ (by definition)

  iff there are $v_1, v_2 \in \text{Dom}(t)$ such that $(u, v_1) \in [\![\gamma_1]\!]^{t, \overline{u}}$ and $(u, v_2) \in [\![\gamma_2]\!]^{t, \overline{u}}$ (by induction hypothesis)

  iff there are $v_2 \in \text{Dom}(t)$ such that $(u, u) \in [\![[\gamma_1]]\!]^{t, \overline{u}}$ and $(u, v_2) \in [\![\gamma_2]\!]^{t, \overline{u}}$ (by induction hypothesis)

  iff there are $v_2 \in \text{Dom}(t)$ such that $(u, v_2) \in [\![[\gamma_1] \circ \gamma_2]\!]^{t, \overline{u}}$

□

3.3.2 Relation to Conjunctive Queries

We study $\mathcal{C}^{\text{nvs}}(L)$ the fragment of $\mathcal{C}(L)$ where variable sharing is forbidden. In particular, we show that union-free expressions of this language can be identified with acyclic conjunctive queries (ACQs) over signature L. Adding union to CQs is usually done by taking unions of CQs, but it is not clear how to add union to CQ at arbitrary positions in the query. In particular, what would be the notion of acyclicity of conjunctive queries with union? However, the composition language allows one to manage unions without any restriction on the position where they occur, while providing a rather simple notion of acyclicity (non-variable sharing between the members of composition). In this section, we make this notions clearer.
As already said in section 2.10, since we consider binary atoms for conjunctive queries, the notion of acyclicity of a CQ corresponds to the usual notion of acyclicity of their (undirected) query graph.
Let ACQ$[L]$ be the language of ACQs over L (defined similarly as FO$[L]$). We also denote by ACQ$^\vee[L]$ the set of disjunctions of ACQ$[L]$ formulas. As for FO$[L]$ and $\mathcal{C}(L)$, we assume that the size of the query expressions does not amount in the size of the formulas.

Proposition 3.3.2 *Let L be a binary query language.*

1. *Every disjunction-free $\mathcal{C}^{nvs}(L)$ formula γ is equivalent to some ACQ$[L \cup \mathsf{root}]$ formula ψ, modulo a linear-time transformation. Moreover, the transformation preserves the query expressions, ie $\{\mathfrak{q} \mid \mathfrak{q} \in Sub(\gamma)\} = \{\mathfrak{q} \mid \mathfrak{q} \in Sub(\psi)\}$;*

2. *Every $\mathcal{C}^{nvs}(L)$ formula γ is equivalent to some ACQ$^\vee[L \cup \mathsf{root}]$ formula ψ, modulo an exptime transformation which produces a formula of (at most) exponential size. Moreover, the transformation preserves the query expressions, ie $\{\mathfrak{q} \mid \mathfrak{q} \in Sub(\gamma)\} = \{\mathfrak{q} \mid \mathfrak{q} \in Sub(\psi)\}$.*

Proof. For the first item, note that the translation of a composition formula γ in the proof of item (1) of Proposition 3.3.1 produces a conjunctive query when γ is disjunction-free, and this translation needs the root predicate. The produced conjunctive query is acyclic when γ is acyclic. The proof is technical but easy.
For the second item, it suffices to distribute disjunctions upward by applying the following rules exhaustively: $(\gamma_1 \vee \gamma_2) \circ \gamma \rightarrow (\gamma_1 \circ \gamma) \vee (\gamma_2 \circ \gamma)$ and $[\gamma_1 \vee \gamma_2] \rightarrow [\gamma_1] \vee [\gamma_2]$. Note that the resulting formula might be of exponential size. □

Disjunction-free formulas of $\mathcal{C}^{\text{nvs}}(L)$ can be identified with ACQs over L, providing L is closed by intersection and inverse[2] (which is the case for instance for FO binary queries):

Proposition 3.3.3 *Let L be a binary query language closed by inverse and intersection. $\mathcal{C}^{nvs}(L \cup \prec_{ch^*})$ captures ACQ$^\vee[L]$.*

Proof. Since $\mathcal{C}^{\text{nvs}}(L \cup \prec_{ch^*})$ is closed by disjunction, one only needs to prove the result for ACQ$[L]$ formulas.

[2]The inverse of a binary query q is the binary query $q^{-1} : t \mapsto \{(v,u) \mid (u,v) \in q(t)\}$

Informally acyclic conjunctive queries are viewed as trees whose vertices are variables and edges are labeled by relations. Those trees are naturally translated as follows: a branching node corresponds to a conjunction, and hence to the composition of two filter expressions, while the succession of two edges correspond to a composition. Only free variables are kept in the composition formula. Actually existential quantifications are just replaced by compositions. An example of the full transformation is given in Fig. 3.2.
Formally, one needs a notion of *extended query graph* $G(\phi)$ of a query $\phi \in \text{ACQ}[L]$, that includes the atomic predicates as label of edges. Vertices of this graph are variables, and there is a directed edge from x to y labeled by some query expression $\mathbb{q}$ if $\mathbb{q}(x,y)$ occurs in ϕ. See Fig. 3.2 for an example. The translation is intuitively as follows: branching in the query graph is translated into compositions of filter expressions and the sucession of two edges is translated into a composition. Only free variables are kept in the translation. For instance, the succession of two edges (x,y) and (y,z) labeled respectively by $\mathbb{q}$ and $\mathbb{q}'$ corresponds to the composition $\mathbb{q} \circ \mathbb{q}'$ (if x,y,z are existentially quantified in ϕ). However, their might be several edges between two vertices, for instance in $G(\mathbb{q}_1(x,y) \wedge \mathbb{q}_2(x,y))$. Hence, we first use the closure under intersection of L to replace multiple edges between two vertices x,y labeled $\mathbb{q}_1,\ldots,\mathbb{q}_n$ by a single edge labeled $\cap_i \mathbb{q}_i$, where $\cap_i \mathbb{q}_i$ is a query expression such that $\mathcal{Q}_L(\cap_i \mathbb{q}_i) = \cap_i \mathcal{Q}_L(\mathbb{q}_i)$. The graph, at this point, is a tree (since ϕ is acyclic), if we do not consider the orientation of edges. In order to be able to translate ϕ into a composition formula, the last step is to choose a root of the tree and an orientation, so that it becomes a tree when considering the chosen orientation. This can be done by using closure by inverse of L, which allows one to reverse the orientation of an edge (x,y) labeled $\mathbb{q}$ to an edge (y,x) labeled $\mathbb{q}^{-1}$, where $\mathbb{q}^{-1}$ is a query expression such that $\mathcal{Q}_L(\mathbb{q}^{-1}) = \mathcal{Q}_L(\mathbb{q})^{-1}$. In the translated composition formula, one uses $\prec_{ch^*}$ to reach the root of this tree.

□

The complexity of the translation of Proposition 3.3.3 is in linear-time if for all query expressions $\mathbb{q} \in \mathbb{Q}$, there exists a query expression $\mathbb{q}^{-1} \in \mathbb{Q}$ computable in linear-time such that $\mathcal{Q}(\mathbb{q}^{-1})$ is the inverse of $\mathcal{Q}(\mathbb{q})$ and for all query expressions $\mathbb{q},\mathbb{q}' \in \mathbb{Q}$, there exists a query expression $\mathbb{q}''$ computable in linear time such that $\mathcal{Q}(\mathbb{q}'') = \mathcal{Q}(\mathbb{q}) \cap \mathcal{Q}(\mathbb{q}')$.
We now suppose that there is a fixed algorithm to evaluate the binary queries of L, which runs in time $p(\|\mathbb{q}\|, \|t\|)$, for query expressions $\mathbb{q}$ on trees $t \in \mathcal{T}_{unr}(\Sigma)$.
By Proposition 3.3.2, the query evaluation of a disjunction-free $\mathcal{C}^{\text{nvs}}(L)$ formula $\gamma(\overline{x})$, over a tree t, can be reduced to the query evaluation of an ACQ formula over a particular database db. The relational database db consists of the set of binary relations $db = \{\mathcal{Q}(\mathbb{q})(t) \mid \mathbb{q} \in \text{Sub}(\phi)\}$. The size of this database is $O(\|\phi\|.\|t\|^2)$, since the queries occuring in ϕ are binary; it can be computed by evaluating the binary queries $\mathcal{Q}(\mathbb{q})(t)$, for all $\mathbb{q} \in \text{Sub}(\phi)$. It can be done in time $\sum_{\mathbb{q} \in \text{Sub}(\phi)} p(\|\mathbb{q}\|, \|t\|))$.
From Yannakakis's algorithm (Yan81) that solves ACQs for relational databases D in time $O(\|D\|.\|\phi\|.\|\mathcal{Q}(\phi)(D)\|)$ where ϕ is an ACQ on a relational database D, we get:

Proposition 3.3.4 *Query evaluation of n-ary queries defined by disjunction-free $\mathcal{C}(L)$ formulas γ can*

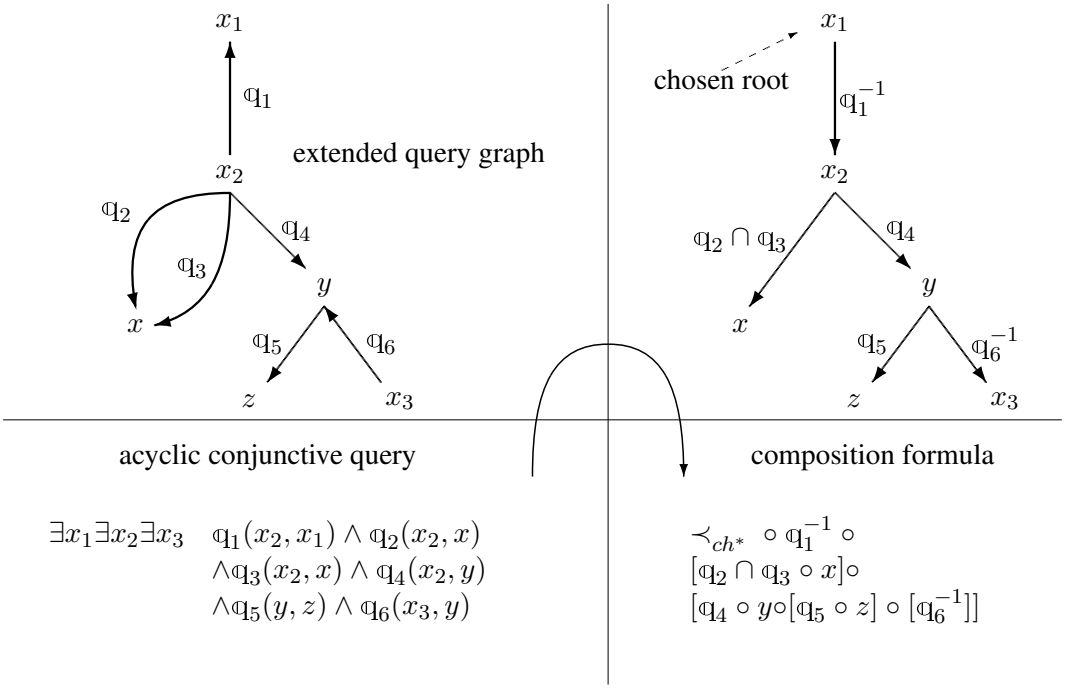


Figure 3.2: From ACQs to Composition Formulas

be done it time complexity

$$O(\|t\|^2.\|\gamma\|.\|\mathcal{Q}(\phi)(t)\| \; + \sum_{\mathbb{q} \in Sub(\phi)} p(\|\mathbb{q}\|, \|t\|))$$

where $p(\|\mathbb{q}_1\|, \|t\|)$ *is the time complexity to evaluate binary queries* $\mathbb{q}_1$ *from* L *on* t.

The next section extends this result to composition formulas with disjunctions.

3.4 Query Non-Emptiness and Query Evaluation

In this section, we give an algorithm to answer queries by composition formulas of $\mathcal{C}^{\text{nvs}}(L)$. The algorithm is generic since the choice of the underlying binary query language L is parametric. We will see that the non-variable sharing restriction is crucial to get polynomial-time query evaluation complexity. In particular, we first give complexity results for the query non-emptiness problem, which we prove to be NP-hard for $\mathcal{C}(L)$, but polynomial for $\mathcal{C}^{\text{nvs}}(L)$ (Section 3.4.1). As a consequence, we cannot hope to have a polynomial-time query evaluation algorithm for $\mathcal{C}(L)$ (even if the query evaluation for L is polynomial). We give a query evaluation algorithm for $\mathcal{C}^{\text{nvs}}(L)$ in Section 3.4.2, and prove that it is polynomial. This algorithm is similar to the Yannakakis's algorithm (Yan81) for acyclic conjunctive queries, but our relations are binary (since they are given by binary queries), and composition formulas allow disjunction. A detailed comparison to acyclic conjunctive queries is given in Section 3.3. In particular, the main difficulty is to manage disjunctions on the left of compositions $(\phi_1 \vee \phi_2) \circ \phi_3$. We cannot distribute them as it would result in an exponential blow-up. Instead we introduce parameters in formulas which intuitively can be viewed as pointers to formulas.

3.4.1 Query Non-Emptiness and Model-Checking

We prove several results on query non-emptiness and model-checking for composition formulas. In particular, we give a polynomial-time query non-emptiness algorithm for $\mathcal{C}^{\text{nvs}}(L)$ formulas, that can also be used for model-checking.

Proposition 3.4.1 *Let $\Sigma = \{0, 1, \#\}$ be an alphabet, and $L = (\mathbb{Q}, \|.\|, \mathsf{ar}., \mathcal{Q}(.))$ a binary query language such that there are $\mathbb{q}_0, \mathbb{q}_1$ in $\mathbb{Q}$ and for all $b \in \{0, 1\}$, and all trees t in $\mathcal{T}_{unr}(\Sigma)$, $\mathcal{Q}(\mathbb{q}_b)(t) = \{(\mathsf{root}^t, u) \mid \mathsf{lab}^t(u) = b\}$.*
The query non-emptiness problem is NP-Hard for $\mathcal{C}(L)$ on unranked trees, even if the tree is fixed.

Proof. Let $t =$  .

We give a polynomial reduction of CNF satisfiability into our problem. The idea is to associate with a given CNF formula $\Psi = \bigwedge_{1 \leq i \leq p} C_i$ a composition formula $\phi = [\phi_1] \circ \ldots \circ [\phi_p]$ over L, such that Ψ is satisfiable iff $\mathcal{Q}(\phi)(t) \neq \varnothing$.
Each ϕ_i is a composition formula associated to the i-th clause C_i. It is defined by associating to each literal x_j the composition formula $\mathbb{q}_1 \circ x_j$ and to $\neg x_j$ the formula $\mathbb{q}_0 \circ x_j$, and to a disjunction of literals a disjunction of those formulas. For example, if we consider $\Psi = (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$, then $\phi = [\mathbb{q}_1 \circ x_1 \vee \mathbb{q}_0 \circ x_2] \circ [\mathbb{q}_1 \circ x_2 \vee \mathbb{q}_0 \circ x_3]$. Note that $\|\phi\| = O(\|\Psi\|)$ since basic queries are considered of constant size in $\|\phi\|$. □

However, it becomes polynomial for $\mathcal{C}^{\text{nvs}}(L)$, providing a polynomial query evaluation algorithm for L exists, as we next show.
We first have to simplify expressions of $\mathcal{C}^{\text{nvs}}(L)$ so that no disjunctions appear on the left of compositions. The naive idea would be to replace $(\phi_1 \vee \phi_2) \circ \phi$ by $\phi_1 \circ \phi \vee \phi_2 \circ \phi$. Unfortunately, rewriting the formula with this rule may lead to an exponential explosion, since the subformula ϕ is copied. We solve this problem by introducing sharing formulas.
Let $p \in \mathsf{Par}$ be a parameter in a finite set, which refers to expressions. A *sharing formula* ξ is similar to an expression of $\mathcal{C}^{\text{nvs}}(L)$ except that it may contain parameters $p \in \mathsf{Par}$, while excluding disjunctions as well as parameters on the left of compositions. We also use a special formula $\mathbb{id}$ interpreted as the identity relation on nodes. This will simplify the query evaluation algorithm.

$$\begin{array}{lllllll} \eta & ::= & x & \mid & [\xi] & \mid & \mathbb{q} \\ \xi & ::= & p & \mid & \xi \vee \xi' & \mid & \eta \circ \xi \quad \mid \quad \mathbb{id} \end{array}$$

We denote by $\mathsf{Par}(\xi)$ the set of parameters occurring in ξ. An equation system $\Delta = [p_1 \mapsto \xi_1, \ldots, p_n \mapsto \xi_n]$ is a finite acyclic mapping from parameters to sharing formulas. This means that all p_i are pairwise distinct and that $\mathsf{Par}(\xi_i) \subseteq \{p_{i+1}, \ldots, p_n\}$ for all $1 \leq i \leq n$. Pairs (ξ, Δ) satisfying $\mathsf{Par}(\xi) \subseteq \mathsf{Par}(\Delta)$ define composition formulas ξ_Δ inductively as follows:

$$\begin{array}{lll} \mathbb{id}_\Delta = \mathbb{id} & p_\Delta = \Delta(p)_\Delta & x_\Delta = x \\ \mathbb{q}_\Delta = \mathbb{q} & [\xi]_\Delta = [\xi_\Delta] & \\ (\eta \circ \xi)_\Delta = \eta_\Delta \circ \xi_\Delta & (\xi \cup \xi')_\Delta = \xi_\Delta \cup \xi'_\Delta & \end{array}$$

The size of a sharing formula is the number of its nodes (viewed as a term). The size of an equation system is the sum of the sizes of its formulas. Queries by shar-

ing formulas are defined as before: $\mathcal{Q}(\xi, \Delta) = \mathcal{Q}(\xi_\Delta)$. For instance, a composition formula of the form $(\phi_1 \vee \phi_2) \circ (\phi_3 \vee \phi_4) \circ (\phi_5 \vee \phi_6)$ if equivalent to the sharing formula $\phi_1 \circ p \vee \phi_2 \circ p$ together with the mapping $[p \mapsto \phi_3 \circ p' \vee \phi_4 \circ p' \,;\, p' \mapsto \phi_5 \vee \phi_6]$. More formally:

Lemma 3.4.2 *Every composition formula ϕ can be transformed in linear time into a pair (ξ, Δ) so that $\phi = \xi_\Delta$. Moreover, $\|\xi\| + \|\Delta\| = O(\|\phi\|)$.*

Proof. This can be done by applying the following rewrite rule exhaustively, always with fresh parameters p:

$$(\phi_1 \vee \phi_2) \circ \phi \;\rightarrow\; \phi_1 \circ p \;\vee\; \phi_2 \circ p \quad \text{where} \quad \Delta(p) = \phi$$

Once this is done, we will have to replace formulas η on the right of equations by $\eta \circ \mathtt{id}$. The introduction of $\mathtt{id}$ is not essential but will reduce the number of cases in our algorithm. □

We present an algorithm deciding query non-emptiness on a tree t for a sharing formula ξ and a mapping Δ such that ξ_Δ does not share variables in compositions. It computes all truth values $\mathrm{NE}(\xi, u)$ for all $u \in \mathrm{Dom}(t)$ and an input formula ξ:

$$\mathrm{NE}(\xi, u) = 1 \text{ iff } \exists \rho, \exists u' \in \mathrm{Dom}(t).(u, u') \in [\![\xi_\Delta]\!]^{t,\rho}$$

By using dynamic programming, we will compute the values $\mathrm{NE}(\xi, u)$ at most once for all subformulas of ξ and Δ. There is only a linear number of them by Lemma 3.4.2.

$$\begin{array}{lcl}
\mathrm{NE}(\mathtt{id}, u) & = & 1 \\
\mathrm{NE}(\mathtt{q} \circ \xi, u) & = & \bigvee_{(u,u') \in \mathcal{Q}(\mathtt{q})(t)} \mathrm{NE}(\xi, u') \\
\mathrm{NE}(p, u) & = & \mathrm{NE}(\Delta(p), u) \\
\mathrm{NE}([\xi] \circ \xi', u) & = & \mathrm{NE}(\xi, u) \wedge \mathrm{NE}(\xi', u) \\
\mathrm{NE}(x \circ \xi, u) & = & \mathrm{NE}(\xi, u) \\
\mathrm{NE}(\xi \vee \xi', u) & = & \mathrm{NE}(\xi, u) \vee \mathrm{NE}(\xi', u)
\end{array}$$

The rule for $\mathrm{NE}(x \circ \xi, u)$ is correct since $x \notin \mathrm{Var}(\xi_\Delta)$ by the non-variable sharing assumption, so that $\mathrm{NE}(\xi, u)$ can be satisfied independently of the value of x. An analogous argument applies to $\mathrm{NE}([\xi] \circ \xi', u)$.

Proposition 3.4.3 *Let ξ be a sharing formula, Δ be an equation system, and t be a tree. Computing the table NE on inputs ξ, Δ and t is in time*

$$O\Big(\sum_{\mathtt{q} \in \mathit{Sub}(\xi_\Delta)} p(\|\mathtt{q}\|, \|t\|) + \|t\|^2(\|\xi\| + \|\Delta\|)\Big)$$

if query evaluation for L is in time $p(\|\mathtt{q}\|, \|t\|)$ for some function p.

Proof. For $u \in \mathrm{Dom}(t)$, and $\mathtt{q} \in \mathbb{Q}$, let $S_{u,\mathtt{q}} = \{u' \mid (u, u') \in \mathcal{Q}(\mathtt{q})(t)\}$. As a first step, all binary queries occurring in ξ_Δ are precompiled in a data structure that returns in time $|S_{u,\mathtt{q}}|$ the set $S_{u,\mathtt{q}}$, for any $u \in \mathrm{Dom}(t)$ and $\mathtt{q} \in \mathbb{Q}$. This can be done in time $O(\sum_{\mathtt{q} \in \mathrm{Sub}(\xi_\Delta)} p(\|b\|, \|t\|))$. Computing NE with memoization can be done in time $O(\|t\|^2.\|\xi_\Delta\|)$, i.e. $O(\|t\|^2.(\|\xi\| + \|\Delta\|))$. □

Combining Lemma 3.4.2 and Proposition 3.4.3 gives:

Corollary 3.4.4 *Query non-emptiness on inputs ϕ, t, where $\phi \in \mathcal{C}^{nvs}(L)$ and $t \in \mathcal{T}_{unr}(\Sigma)$, can be tested in time*

$$O(\sum_{\mathbb{q}\in Sub(\phi)} p(\|\mathbb{q}\|, \|t\|) + \|t\|^2.\|\phi\|)$$

if query evaluation for L is in time $p(\|\mathbb{q}\|, \|t\|)$ for some function p.

From a composition formula $\phi \in \mathcal{C}^{\text{nvs}}(L)$ with free variables $\overline{x} = x_1, \ldots, x_n$, a tree t, and a tuple of nodes $\overline{u} = u_1, \ldots, u_n \in \text{Dom}(t)$, one can construct a formula $\phi_{t,\overline{u}}$ such that $\phi_{t,\overline{u}}$ is non-empty on t iff $\overline{u} \in \mathcal{Q}(\phi)(t)$. It suffices to replace each occurrence of a variable x_i by a special query expression $\mathbb{q}_{u_i}$, interpreted on t by: $\mathcal{Q}(\mathbb{q}_{u_i})(t) = \{(u_i, u_i)\}$, and by $\varnothing$ on other trees. Note that on t those new queries are evaluable in constant time. By using Corollary 3.4.4, it gives a model-checking algorithm. However, by definition of the model-checking problem, the input tuple $\overline{u}$ may have a length $|\overline{u}|$ different from $|\overline{x}|$, hence we first have to check it, and if the lengths are equal, then transform ϕ into $\phi_{t,\overline{u}}$. Since the size of $\phi_{t,\overline{u}}$ is linear in the size of ϕ, we finally get:

Proposition 3.4.5 *Model-checking on inputs $\phi, t, \overline{u}$, where $\phi \in \mathcal{C}^{nvs}(L)$, $t \in \mathcal{T}_{unr}(\Sigma)$, and $\overline{u}$ is a tuple of nodes of t, can be done in time:*

$$O(\sum_{\mathbb{q}\in Sub(\phi)} p(\|\mathbb{q}\|, \|t\|) \ + \ |\overline{u}| \ + \ \|t\|^2.\|\phi\|)$$

if query evaluation for L is in time $p(\|\mathbb{q}\|, \|t\|)$ for some function p.

3.4.2 Query Evaluation

In this section, we give a polynomial-time query evaluation algorithm for $\mathcal{C}^{\text{nvs}}(L)$-definable n-ary queries. It is similar to the Yannakakis's algorithm (Yan81), but works on databases which contain only binary relations. The query however may have disjunction, occurring anywhere in the formula: in this setting it is an extension of Yannakakis's algorithm. As for query non-emptiness, we work with sharing formulas ξ together with mappings Δ.

The query evaluation algorithm in Figure 3.3 processes the input formula ξ recursively, while filtering unsatisfiable cases (in constant time by using the table NE, see Proposition 3.4.3), eliminating duplicates, and memoizing auxiliary results by dynamic programming so that they are never recomputed. The non-variable sharing restriction allows, in a composition, to evaluate each member independently and combine the results with a simple cartesian product (and not a complex join). We also assume that all binary queries $\mathbb{q}$ occurring in ξ have been precomputed in a data-structure which returns in time $|S_{t,u,\mathbb{q}}|$ the set $S_{t,u,\mathbb{q}} = \{v \mid (u, v) \in \mathcal{Q}(\mathbb{q})(t)\}$, on inputs u and $\mathbb{q}$.

A partial valuation is a function of type $V \to \text{Dom}(t)$ for some subset $V \subseteq \text{Var}(\xi)$. Let ϵ be the partial valuation with empty domain, and $\nu[x \mapsto u]$ the extension of ν mapping x to u. If ν and ν' are valuations with different domains, then we write $\nu \cdot \nu'$ for disjoint union of both functions. In a disjunction $\xi \vee \xi'$, variables of ξ may not occur in ξ'. When computing the union, those variables have to be bound to arbitrary nodes. Function $\mathsf{extend}_{t,X}$ takes care of this. It extends a partial valuation in all possible ways, so that it becomes total on a finite set of variables X.

The algorithm stores sets of partial valuations $\texttt{vals}(\xi_0, u)$ as auxiliary results (the node u denotes the starting node of the valuation). All these partial valuations can

```
1  q(ξ, Δ, t) =
2   let vals(ξ_0, u) =
3     if NE(ξ_0, u)
4     then case ξ_0
5
6       of id:
7          return {ε}
8
9       of p:
10         return vals(Δ(p), u)
11
12      of q∘ξ':
13         return ⋃_{(u,u')∈Q(q)(t)} {ν | ν ∈ vals(ξ', u')}
14
15      of x∘ξ':
16         return {ν[x ↦ u] | ν ∈ vals(ξ', u)}
17
18      of [ξ']∘ξ'':
19         return {ν' · ν'' | ν' ∈ vals(ξ', u), ν'' ∈ vals(ξ'', u)}
20
21      of ξ' ∨ ξ'':
22         return extend_{t,Var((ξ'∨ξ'')_Δ)}(vals(ξ', u))
23                ∪ extend_{t,Var((ξ'∨ξ'')_Δ)}(vals(ξ'', u))
24    else return {}
25  in
26    return {(ν(x_1), ..., ν(x_n)) | ν ∈ vals(ξ, root^t)
27                                    and Var(ξ) = {x_1, ..., x_n}}
```

Figure 3.3: Computing of the answer set $\mathcal{Q}(\xi_\Delta)(t)$ with implicit memoization.

be extended to some complete solution, since unsatisfiable cases are filtered by using NE table.

Proposition 3.4.6 *Given a sharing formula ξ and a mapping Δ, the algorithm in Figure 3.3 computes the answer set $A = \mathcal{Q}(\xi_\Delta)(t)$ of the n-ary query ξ_Δ in time $O((\|\xi\| + \|\Delta\|).\|t\|^2.n.|A|)$, if ξ_Δ belongs to $\mathcal{C}^{nvs}(L)$.*

Proof. Since every recursive call to vals filters all unsatisfiable cases, every intermediate result can be extended to a whole solution. Duplicates generated by union are eliminated, so that every recursive call returns at most $|A|$ tuples. The worst case time complexity occurs for the $\mathrm{q}\circ\xi'$ case, and is in $O(\|t\| + n\ \|t\|\ |A|)$. Indeed, there are at most $\|t\|$ successors u by q, hence they are returned in $\|t\|$ steps at most (since every binary query has been first precomputed in a data structure which returns all the successors of a node, by a particular binary query, in linear time). Hence, the algorithm performs at most $\|t\|$ unions of sets of size at most $n|A|$, which can be done in time $O(\|t\|.n.|A|)$.
Finally, since we use memoization, intermediate results are never recomputed, so that there are at most $O(\|t\|.(\|\xi\| + \|\Delta\|))$ recursive calls to vals which costs at most $O(\|t\|.n.|A|)$ steps. Hence, line 15 requires at most $O((\|\xi\| + \|\Delta\|).n.\|t\|^2.|A| + \|t\||A|) = O((\|\xi\| + \|\Delta\|).n.\|t\|^2.|A|)$ steps. □

If we also take into account the complexities of the precomputation and the trans-

formation of the composition formula into a sharing composition formula, as a corollary of Lemma 3.4.2, Propositions 3.4.3 and 3.4.6, we get:

Theorem 3.4.7 ***Query Evaluation*** *Answer sets of n-ary queries $A = \mathcal{Q}(\phi)(t)$ defined in $\mathcal{C}^{nvs}(L)$ can be computed in time*

$$O(\sum_{\mathfrak{q} \in Sub(\phi)} p(\|\mathfrak{q}\|, \|t\|) + n.\|\phi\|.\|t\|^2.|A|)$$

if query evaluation for L is in time $p(\|\mathfrak{q}\|, \|t\|)$ for some function p.

3.5 Expressiveness of the Composition Language

In this section, we prove the following theorem:

Theorem 3.5.1 ***Expressiveness***
Let L be a query language. Over unranked trees,

if L is equally expressive as $FO_{bin}[\prec_{ns^*}, \prec_{ch^*}]$ (resp. $MSO_{bin}[\prec_{ns^*}, \prec_{ch^*}]$), then $\mathcal{C}^{nvs}(L)$ is equally expressive as $FO[\prec_{ns^*}, \prec_{ch^*}]$ (resp $MSO[\prec_{ns^*}, \prec_{ch^*}]$).

$$\begin{array}{lcl} FO[\prec_{ns^*}, \prec_{ch^*}] & = & \mathcal{C}^{nvs}(FO_{bin}[\prec_{ns^*}, \prec_{ch^*}]) \\ MSO[\prec_{ns^*}, \prec_{ch^*}] & = & \mathcal{C}^{nvs}(MSO_{bin}[\prec_{ns^*}, \prec_{ch^*}]) \end{array}$$

In order to prove Theorem 3.5.1, we use a standard technique of *Finite Model Theory* called the *composition method* (She) used in several papers such as (Tho84, Sch00, MR03, Mar05a). Basically, a decomposition theorem relates the theory (for some logic) of a compound structure to the theories of its components. In particular, the set of formulas of fixed quantifier depth n satisfied in the compound structure can be computed from the sets of formulas of quantifier depth n satisfied by the respective components of the structure. On trees for instance it gives an alternative proof of the correspondence between MSO and FTA (Zei94). In the first section, we briefly introduce the classical notions of Finite Model Theory used to prove composition results: logical types and Ehrenfeucht-Fraïssé games (see (EF05) or (Lib04b) for more details). Then we state a well-known composition lemma for unranked trees with distinguished nodes and its corollary on the decomposition of FO and MSO formulas with n free variables. This corollary is then used to prove Theorem 3.5.1.

3.5.1 Brief reminder on fundamental properties of finite model theory

Let σ be a signature. Let $k \in \mathbb{N}$ be a natural number, M a σ-structure, and $v_1, \ldots, v_k \in \text{Dom}(M)$. We denote by $(M, v_1, \ldots, v_k)$ the extension of the structure M by k distinguished nodes $v_1, \ldots, v_k$: it is a structure on the signature $\sigma \cup \{c_1, \ldots, c_k\}$, where $c_1, \ldots, c_k$ are new constant symbols interpreted respectively by $v_1, \ldots, v_k$ in M. Similarly, it is also possible to extend structures with sets of nodes.
Let $\mathcal{L}$ denote either MSO$[\sigma]$ or FO$[\sigma]$. Let $n \in \mathbb{N}$. The term $\text{type}_n^{\mathcal{L}}(M, v_1, \ldots, v_k)$ stands for the set of $\mathcal{L}$-formulas $\phi(x_1, \ldots, x_k)$ of quantifier depth at most n that are satisfied when mapping x_i to v_i for all $1 \leq i \leq n$. In other words,

$$\text{type}_n^{\mathcal{L}}(M, v_1, \ldots, v_k) = \{\phi(x_1, \ldots, x_k) \mid M \models \phi(v_1, \ldots, v_k) \text{ and } \text{qd}(\phi) \leq n\}$$

A well-known result of finite model theory states that for any $n \in \mathbb{N}$, the set $\{\text{type}_n^{\mathcal{L}}(M, v_1, \ldots, v_k) \mid (M, v_1, \ldots, v_k) \text{ is an extended structure}\}$ is finite[3]. This is because there are finitely many $\mathcal{L}$ formulas of quantifier depth at most n, modulo logical equivalence. Since the number of extended structures is infinite, there are necessarily structures which satisfies exactly the same $\mathcal{L}$-formulas. This is formalized next.
Given two extended structures $(M, \overline{v})$ and $(M', \overline{v}')$, we write $(M, \overline{v}) \equiv_n^{\mathcal{L}} (M', \overline{v}')$ if $\text{type}_n^{\mathcal{L}}(M, \overline{v}) = \text{type}_n^{\mathcal{L}}(M', \overline{v}')$. Intuitively, it means that $\mathcal{L}$-formulas of quantifier depth at most n are not expressive enough to distinguish $(M, \overline{v})$ from $(M', \overline{v}')$, *ie* there is no $\mathcal{L}$-formula $\phi(\overline{x})$ such that $\text{qd}(\phi) \leq n$, $M \models \phi(\overline{v})$ and $M' \models \neg\phi(\overline{v}')$.
The sets $\text{type}_n^{\mathcal{L}}(M, v_1, \ldots, v_k)$ can be characterized by n-*Hintikka formulas*. These are $\mathcal{L}$-formulas $\tau^{\mathcal{L},n}_{M,v_1,\ldots,v_k}(x_1, \ldots, x_k)$ with k free variables and quantifier depth at most n that satisfy, for all extended structures $(M', v'_1, \ldots, v'_k)$:

$$M' \models \tau^{\mathcal{L},n}_{M,v_1,\ldots,v_k}(v'_1, \ldots, v'_k) \text{ iff } (M, v_1, \ldots, v_k) \equiv_n^{\mathcal{L}} (M', v'_1, \ldots, v'_k)$$

Finally, every $\mathcal{L}$ formula of quantifier depth at most n is equivalent to a (computable) finite disjunction of n-Hintikka formulas.
The equivalence relation $\equiv_n^{\mathcal{L}}$ can also be characterized by Ehrenfeucht-Fraïssé games (*EF-games*) . In the following we explain EF-games on arbitrary extended σ-structures for MSO[σ]. Let $(M, \overline{u})$ and $(N, \overline{v})$ be two extended σ-structures such that $\text{length}(\overline{u}) = \text{length}(\overline{v})$. Let $n \in \mathbb{N}$. The n-*round MSO game* on $(M, \overline{u})$ and $(N, \overline{v})$ is played by two players, called the *Spoiler* and the *Duplicator*. At each round Spoiler chooses an element or a set of elements of one of the structures, and Duplicator chooses an element or a set in the opposite structure. Spoiler begins and at each round he has the choice of the structure. After n rounds, there are a elements $o_1, \ldots, o_a \in \text{Dom}(M)$ and a elements $p_1, \ldots, p_a \in \text{Dom}(N)$, b sets $O_1, \ldots, O_b \subseteq \text{Dom}(M)$ and b sets $P_1, \ldots, P_b \subseteq \text{Dom}(N)$ chosen by the two players. Duplicator *wins* the game if the mapping which maps o_i to p_i, $i = 1, \ldots, a$ is a *partial isomorphism* from $(M, \overline{u}, O_1, \ldots, O_b)$ to $(N, \overline{v}, P_1, \ldots, P_b)$, *ie* (i) the substructures M' and N' obtained respectively by restricting their domain to $\overline{u}, o_1, \ldots, o_a$ and $\overline{v}, p_1, \ldots, p_a$ satisfy the same atomic formulas, and, (ii) for all i, j, $o_i \in O_j$ iff $p_i \in P_j$. The duplicator has a *winning strategy* if he can win independently of the choices made by Spoiler.
The main result concerning EF-games is the following:

Proposition 3.5.2 *Duplicator has a winning strategy for the n-round MSO game on $(M, \overline{u})$ and $(N, \overline{v})$ iff $(M, \overline{u}) \equiv_n^{MSO} (N, \overline{v})$.*

FO games are defined similarly, but only elements (and not sets) can be chosen, so that condition (ii) of the definition of partial isomorphisms is removed.
We first give a brief and informal example of a composition lemma on strings.

Example 3.5.3 (Composition Method on Finite Strings) *We consider $\sigma_{str} = \{<, (\mathsf{lab}_a)_{a\in\Sigma}\}$ the signature of strings ($<$ being interpreted as a linear order on the string positions). A string over the alphabet Σ is naturally viewed as a σ_{str}-structure. The set of strings over Σ is denoted Σ^*.*
Let $w_1, w_2 \in \Sigma^*$, and $\alpha \in \Sigma$. Let $n \in \mathbb{N}$.

$$\text{if } w_1 \equiv_n^{MSO} w_2, \text{ then } w_1\alpha \equiv_n^{MSO} w_2\alpha$$

[3] Its size however is a tower of exponentials of height n

In order to prove it, it suffices to show, by Proposition 3.5.2, that Duplicator has a winning strategy for the n-round MSO game on $w_1\alpha$ and $w_2\alpha$. His strategy, called S', is as follows. Let u_α (resp. v_α) be the last node of $w_1\alpha$ (resp. $w_2\alpha$), and assume that $\mathrm{Dom}(w_1\alpha) = \mathrm{Dom}(w_1) \cup \{u_\alpha\}$ and $\mathrm{Dom}(w_2\alpha) = \mathrm{Dom}(w_2) \cup \{v_\alpha\}$. By hypothesis, Duplicator has a winning strategy S for the MSO game on w_1 and w_2. It can be extended to a winning strategy for the MSO game on $w_1\alpha$ and $w_2\alpha$: if Spoiler chooses $v \in \mathrm{Dom}(w_1)$, then Duplicator answers according to S, and if Spoiler chooses u_α, then Duplicator chooses v_α. If Spoiler chooses $V \subseteq \mathrm{Dom}(w_1\alpha)$, then let $U_1 \subseteq \mathrm{Dom}(w_1)$ and $U_2 \subseteq \{u_\alpha\}$ such that $U = U_1 \cup U_2$. Duplicator's answer is then $V_1 \cup V_2$, where V_1 is the answer to U_1 according to S and $V_2 = \{v_\alpha\}$ if $U_2 = \{u_\alpha\}$, and $\varnothing$ otherwise. The strategy is defined similarly when spoiler chooses elements or sets in $w_2\alpha$.
To conclude, it remains to prove that S' is a winning strategy, by showing that the elements or sets chosen during the n-round MSO game on $w_1\alpha$ and $w_2\alpha$ induce a partial isomorphism with respect to $<$, lab_a, and the chosen sets. Informally, if some chosen element $u \in \mathrm{Dom}(w_1\alpha)$ belongs to some chosen set $U \subseteq \mathrm{Dom}(w_1\alpha)$, then its corresponding element v in $\mathrm{Dom}(w_2\alpha)$ is also in the set $V \subseteq \mathrm{Dom}(w_2\alpha)$ corresponding to U: indeed, if u is in U_1, then v is in V_1, since S is a winning strategy, hence $v \in V$. If u is in U_2, then $u = u_a$ and $v = v_a$, so that $v \in V_2$, and $v \in V$. This is proved similarly for atomic formulas, by using the fact that S is a winning strategy.
Let ϕ an MSO$[<]$-sentence of quantifier depth n. There is a (computable) finite set of strings W such that ϕ is equivalent to a finite disjunction of n-Hintikka formulas $\bigvee_{w\in W} \tau_w^{MSO[<],n}$. It is possible to associate with ϕ a finite (string) automaton[4] $A_\phi = (\Sigma, Q, F, q_0, \Delta)$ which defines the same string language as ϕ:

- $Q = \{type_n^{MSO[<]}(w) \mid w \in \Sigma^*\}$ *(it is finite upon logical equivalence);*
- $F = \{type_n^{MSO[<]}(w) \mid w \in W\}$;
- $q_0 = type_n^{MSO[<]}(\epsilon)$;
- $\Delta = \{type_n^{MSO[<]}(w) \xrightarrow{\alpha} type_n^{MSO[<]}(w\alpha) \mid w \in \Sigma^* \text{ and } \alpha \in \Sigma\}$

Note that by the composition result proved before, A_ϕ is deterministic. Indeed, suppose that $type_n^{MSO[<]}(w) \xrightarrow{\alpha} type_n^{MSO[<]}(w\alpha) \in \Delta$ and $type_n^{MSO[<]}(w) \xrightarrow{\alpha} q \in \Delta$, for some $q \in Q$. By definition of Δ, q is necessarily of the form $type_n^{MSO[<]}(w'\alpha)$, for some $w' \in \Sigma^*$, and $type_n^{MSO[<]}(w') = type_n^{MSO[<]}(w)$. By the composition result we get $q = type_n^{MSO[<]}(w\alpha)$.
Then it follows that for any string w, the computation of A_ϕ on w leads to the state $type_n^{MSO[<]}(w)$. This proves in particular that deterministic finite (string) automata capture MSO$[<]$ over strings.
Another version of this example can also be found in (NS02a).

3.5.2 FO and MSO completeness

Since the proof of expressiveness relies on decomposition technique which deals with hedges, we state all the results for hedges.

[4] We use the standard notion of string automata where Σ is an alphabet, Q a finite set of states, F a set of final states, q_0 an initial state, and Δ a set of rules of the form $q \xrightarrow{a} q'$

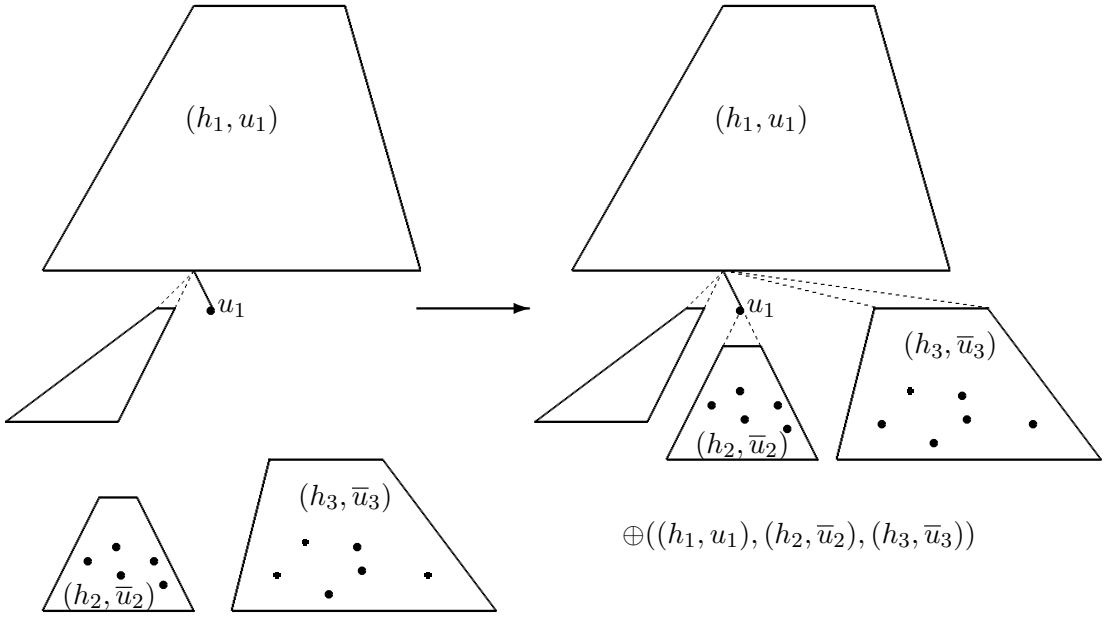


Figure 3.4: Operation on extended hedges

We first define an operation on hedges extended with distinguished nodes, which combines independent extended hedges to form a single extended hedge.
Let $h_1, h_2, h_3 \in \mathcal{H}(\Sigma)$. Let u_1 a node of $\mathrm{Dom}(h_1)$, $\overline{u}_2$ a tuple of nodes of $\mathrm{Dom}(h_2)$, and $\overline{u}_3$ a tuple of nodes of $\mathrm{Dom}(h_3)$ such that u_1 is a leaf of h_1 without right siblings[5]. We let $\oplus((h_1, u_1), (h_2, \overline{u}_2), (h_3, \overline{u}_3)))$ be the hedge defined by adding $(h_2, \overline{u}_2)$ as the child of u, and $(h_3, \overline{u}_3)$ as next-siblings of u. In particular, the domain of this hedge is the union of the domains of h_1, h_2 and h_3 and this hedge is extended by the nodes $u_1, \overline{u}_2, \overline{u}_3$. Fig. 3.4 illustrates this operation. In particular, u_1 is used to connect the roots of h_2 and h_3, but the nodes of $\overline{u}_2$ and $\overline{u}_3$ do not play any role in this connection process.
The next composition lemma intuitively states that the logical type of $\oplus((h_1, u_1), (h_2, \overline{u}_2), (h_3, \overline{u}_3)))$ only depends on the logical types of (h_1, u_1), $(h_2, \overline{u}_2)$ and $(h_3, \overline{u}_3)$.

Lemma 3.5.4 (Folklore Composition Lemma) *Let $\mathcal{L}$ denote either MSO$[\prec_{ns^*}, \prec_{ch^*}]$ or FO$[\prec_{ns^*}, \prec_{ch^*}]$. Let $n \in \mathbb{N}$. Let $(h_1, u_1), (h'_1, u'_1), (h_2, \overline{u}_2), (h'_2, \overline{u}'_2)$, and $(h_3, \overline{u}_3), (h'_3, \overline{u}'_3)$ be extended hedges, such that u_1 and u'_1 are leaves without next-siblings.*

$$\begin{array}{ll} \textit{If} & (h_1, u_1) \equiv^{\mathcal{L}}_n (h'_1, u'_1) \\ & (h_2, \overline{u}_2) \equiv^{\mathcal{L}}_n (h'_2, \overline{u}'_2) \\ & (h_3, \overline{u}_3) \equiv^{\mathcal{L}}_n (h'_3, \overline{u}'_3) \\ \textit{then} & \\ & \oplus((h_1, u_1), (h_2, \overline{u}_2), (h_3, \overline{u}_3)) \equiv^{\mathcal{L}}_n \oplus((h'_1, u_1), (h'_2, \overline{u}'_2), (h'_3, \overline{u}'_3)) \end{array}$$

Proof. As for the case of strings in Example 3.5.3, we combine the strategies of Duplicator on the substructures to define the strategy of Duplicator on the whole structure. We prove the lemma for MSO$[\prec_{ns^*}, \prec_{ch^*}]$ only but the proof

[5] I.e. there is no $w \in \mathrm{Dom}(h_1)$ such that $u_1 \prec^{h_1}_{ns^+} w$

for FO$[\prec_{ns^*}, \prec_{ch^*}]$ is simply obtained by omitting the part of the proof which concerns set choices and membership to sets. In the rest of the proof, MSO stands for MSO$[\prec_{ns^*}, \prec_{ch^*}]$.

Let us call S_1, S_2, S_3 the strategies of Duplicator on the MSO games on (h_1, u_1) and (h'_1, u'_1), $(h_2, \overline{u}_2)$ and $(h'_2, \overline{u}'_2)$, $(h_3, \overline{u}_3)$ and $(h'_3, \overline{u}'_3)$ respectively. Note that the notion of strategies has been formalized by Fraïssé by sequences of sets of partial isomorphisms (EF05), but we prefer to keep it informal to avoid unnecessary heavy notations. We also assume that Spoiler chooses an element or a set in the first structure, as the proof is similar when she makes its choices in the second structure.

Let $(h, u_1, \overline{u}_2, \overline{u}_3) = \oplus((h_1, u_1), (h_2, \overline{u}_2), (h_3, \overline{u}_3))$ and $(h', u'_1, \overline{u}'_2, \overline{u}'_3) = \oplus((h'_1, u'_1), (h'_2, \overline{u}'_2), (h'_3, \overline{u}'_3))$. We now describe the strategy of Duplicator, called S, for the MSO game on $(h, u_1, \overline{u}_2, \overline{u}_3)$ and $(h', u'_1, \overline{u}'_2, \overline{u}'_3)$. Let $i \in \{1, 2, 3\}$. If Spoiler chooses an element in $\mathrm{Dom}(h_i)$, then Duplicator answers by choosing an element in $\mathrm{Dom}(h'_i)$ according to S_i. If Spoiler chooses u_1, then Duplicator answers by choosing u'_1, and if spoiler chooses a node of $\overline{u}_2$ or $\overline{u}_3$, then Duplicator answers by the element at the same rank in $\overline{u}'_2$ or $\overline{u}'_3$ respectively.

If Spoiler chooses a set of elements P, then we let P_1, P_2, P_3 such that: $P_i \subseteq \mathrm{Dom}(h_i)$, $i = 1, 2, 3$, and $P_1 \cup P_2 \cup P_3 = P$. For all $i \in \{1, 2, 3\}$, we let Q_i be the answer of Duplicator to P_i according to S_i. Finally, Duplicator answers to P by choosing $Q_1 \cup Q_2 \cup Q_3$.

It remains to prove that S is a winning strategy. It can be done by showing that the respective elements selected by the players together with the distinguished solutions form a partial isomorphism from $(h, u_1, \overline{u}_2, \overline{u}_3)$ to $(h', u'_1, \overline{u}'_2, \overline{u}'_3)$.

It is clear when the selected elements come from the same substructure, because S_1, S_2 and S_3 are winning strategies. If they come from different substructures, we have to check that the selected elements define a partial isomorphism wrt $\prec_{ns^*}$ and $\prec_{ch^*}$. Suppose for instance that $v_1, v_2 \in \mathrm{Dom}(h)$ and $v'_1, v'_2 \in \mathrm{Dom}(h')$ their corresponding nodes have been chosen. If $v_1 \in \mathrm{Dom}(h_1)$ and $v_2 \in \mathrm{Dom}(h_2)$, then by definition of S we have $v'_1 \in \mathrm{Dom}(h'_1)$ and $v'_2 \in \mathrm{Dom}(h'_2)$. It cannot be the case that $v_1 \prec_{ns^*} v_2$. Suppose that $v_1 \prec_{ch^*} v_2$, then necessarily $v_1 \prec_{ch^*} u_1 \prec_{ch^*} v_2$. Since S_1 is a winning strategy, we also have $v'_1 \prec_{ch^*} u'_1$, and since $u'_1 \prec_{ch^*} v'_2$, we get $v'_1 \prec_{ch^*} v'_2$. The converse is similar, so that finally $v_1 \prec_{ch^*} v_2$ iff $v'_1 \prec_{ch^*} v_2$. The other cases are proved similarly.

Now suppose that some chosen node u belongs to some choosen set P, and let u' and Q their corresponding element and set in the other structure. We can divide P into three parts P_1, P_2, P_3 accordingly to the substructures h_1, h_2, h_3 so that $P = P_1 \cup P_2 \cup P_3$. By definition of the strategy, $Q = Q_1 \cup Q_2 \cup Q_3$ where P_i is mapped to Q_i, according to S_i, for all $i \in \{1, 2, 3\}$. Now, let $i \in \{1, 2, 3\}$ such that $u \in P_i$. Since S_i is a winning strategy, we get $u \in Q_i$, and $u \in Q$.

The converse is similar, so that finally $u \in P$ iff $u' \in Q$. □

We now state a standard consequence of the composition lemma on the existence of a normal form for FO and MSO formulas.

Remind that the extended query graph of a conjunctive query ϕ over a set of predicates L (defined in the proof of Prop. 3.3.3) is the directed graph whose vertices are variables of ϕ and such that there is a directed edge between x and y labeled by some predicate $R \in L$ if $R(x, y)$ is an atom of ϕ. We say that ϕ is a *directed tree query* if its extended query graph is a (directed) tree. In particular, every vertex has exactly one incoming edge except exactly one node (the root).

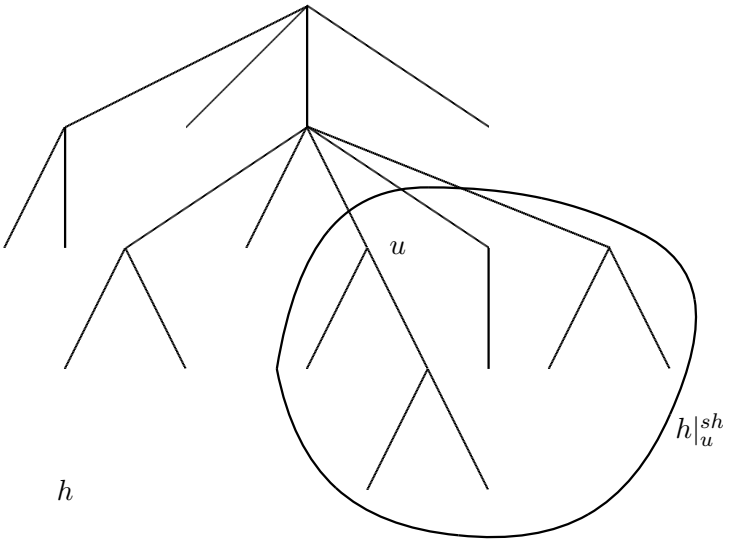


Figure 3.5: A hedge h and its subhedge $h|_u^{sh}$ at node u

Given a hedge h, and a node $u \in \mathrm{Dom}(h)$, the *subhedge* of h at node u, denoted $h|_u^{sh}$, is obtained by restricting the domain of h to the set of nodes v such that $u \prec_{ns^*}^h \circ \prec_{ch^*}^h v$. Informally, it is the hedge induced by u, its descendants, and the descendants of its next-siblings, as illustrated by Fig. 3.5.
Let $\mathcal{L}$ denote either MSO$[\prec_{ns^*}, \prec_{ch^*}]$ or FO$[\prec_{ns^*}, \prec_{ch^*}]$, and $\mathcal{L}_{\text{bin}}$ be their respective fragments with exactly two free variables. An $\mathcal{L}_{\text{bin}}$ formula $\phi(x, y)$ is *subhedge-restricted* to x if for all hedges h and nodes $u, v \in \mathrm{Dom}(h)$,

$$h \models \phi(u, v) \text{ iff } h|_u^{sh} \models \phi(u, v)$$

A standard consequence of the composition lemma is given by the following corollary.

Corollary 3.5.5 (folklore) *Over hedges, every $\mathcal{L}$ formula is equivalent to a disjunction of directed tree queries over $\mathcal{L}_{bin}$ formulas subhedge-restricted to their first component.*

Proof. (Sketch) As already mentioned, the decomposition of an $\mathcal{L}$ formula ϕ thanks a composition lemma is standard. Therefore we only sketch the proof.
First, it suffices to guess how the nodes selected by ϕ in a hedge h relate to each other and their least left common ancestor[6]. This is expressed by a directed tree query (called a *configuration*) over the predicates $\prec_{fc}$, $\prec_{ns}$, and following$(x, y) = \exists z\ x \prec_{ns^+} z \wedge z \prec_{ch^*} y$, whose free variables are exactly the free variables of ϕ. A configuration basically partitions the tree into subparts which are either *subhedge contexts* (subhedges with one hole) or subhedges. A straightforward consequence of the composition lemma is that if two extended hedges satisfies the same configuration, and if each of their respective subparts satisfies the same $\mathcal{L}$ formulas of quantifier rank lesser than some $m \in \mathbb{N}$, then the whole extended hedges satisfy also the same $\mathcal{L}$ formulas of quantifier rank lesser than m.

[6] A node u_a is a *left common ancestor* of some nodes $v_1, \ldots, v_n$ of a hedge h if for all $i \in \{1, \ldots, m\}$, $u_a \prec_{ns^*}^h \circ \prec_{ch^*}^h v_i$. It is the *least left common ancestor*, denoted llca$(v_1, \ldots, v_m)$, if there is no left common ancestor v such that $u_a \prec_{ns^*}^h \circ \prec_{ch^*}^h v$ and $u_a \neq v$. Actually, the least leaft common ancestor corresponds to the least common ancestor in the first-child next-sibling encoding

Suppose that the quantifier rank of ϕ is equal to m. The final step is to extend the configuration with formulas with two free variables, so that two extended hedges satisfy the resulting directed tree query iff they respect the same configuration and their respective subparts satisfy the same $\mathcal{L}$ formulas of quantifier rank lesser than m. This is done as follows (we only describe the case of subhedge context as it is similar to the other case): every edge of the configuration corresponding to a subhedge context is extended by taking the conjunction of its label with a formula $\psi_\tau(x, y)$ intended to describe the subhedge context rooted at x and with hole y. This formula is derived from an arbitrary m-Hintikka formula $\tau(y)$ with one free variable (denoting the hole), related to the domain of nodes which are in the subhedge rooted at node x but are not descendant of y. In particular, $\psi(x, y)$ is subhedge restricted wrt to x.
For every configuration, we consider the extensions of its edges by formulas of the form $\psi_\tau(x, y)$, for all m-Hintikka formulas τ. This gives a finite set of tree queries which have the same shape as the configuration. But for the construction to be correct, we only keep the tree queries which are satisfied by some extended hedge which also satisfies ϕ.
Finally, the resulting tree query is a (finite) disjunction over all configurations and all m-Hintikka formulas used to describe the subparts of the extended hedges. □

The proof of Theorem 3.5.1 is a straigthforward consequence of Corollary 3.5.5 and the construction of Proposition 3.3.3 which associates with any disjunction of acyclic conjunctive queries an equivalent composition formula satisfying the non-variable sharing property. Moreover, note that given a directed tree query, the construction of Proposition 3.3.3 does not need to inverse the binary queries (that was only needed to orient the graph). Therefore, every binary query occuring in the resulting composition formula is still subhedge-restricted wrt to their first component. This founds the unary query language proposed in the nextion section.

3.5.3 Composition of monadic queries over hedges

.
Based on the proof of Theorem 3.5.1 (and especially on Corollary 3.5.5), we propose a language to compose monadic queries only, while capturing FO or MSO wrt n-ary queries. This done via the *subhedge-restriction* defined in the previous section.
To define the composition language with subhedge restriction, we start from a unary query language L over hedges. The new composition language, denoted $\mathcal{C}_{\text{sh}}(L)$, has the same syntax has $\mathcal{C}(L)$, and the same semantics, except that unary query expressions $\mathfrak{q}$ are interpreted with respect to subhedges:

$$\text{for all hedges } h,\ [\![\mathfrak{q}]\!]^h = \{(u, v) \mid v \in \mathcal{Q}(\mathfrak{q})(h|_u^{sh}\}$$

We denote $\mathcal{C}_{\text{sh}}^{\text{nvs}}(L)$ the non-variable sharing fragment of $\mathcal{C}_{\text{sh}}(L)$. The proof of Theorem 3.5.1 associates with any FO or MSO formula an equivalent composition formula over binary FO or MSO formulas subtree-restricted wrt to their first component. In $\mathcal{C}_{\text{sh}}(L)$, we enforce this restriction in the semantics, so that the expressiveness results of Theorem 3.5.1 naturally extends to composition with subhedge restriction. Moreover, subhedge restriction is expressible in FO, hence:

Theorem 3.5.6 *Let $FO_1[\prec_{ns^*}, \prec_{ch^*}]$ be the set of first-order formulas with exactly one free variable. Over hedges,*

$$\mathcal{C}^{nvs}_{sh}(FO_1[\prec_{ns^*}, \prec_{ch^*}]) \;=\; FO[\prec_{ns^*}, \prec_{ch^*}]$$

3.6 Conclusion

We have presented a simple and foundational composition language $\mathcal{C}(L)$ and its syntactically restricted fragment $\mathcal{C}^{\text{nvs}}(L)$ that allows one to combine binary queries to define queries of arbitrary arity, by the use of variables, composition, union, and some kind of conjunction (tests). The syntactic restriction is quite simple as it consists only in disallowing variable sharing between the members of compositions. $\mathcal{C}^{\text{nvs}}(L)$ is equally expressive as FO (or MSO) over unranked trees, wrt to n-ary queries, as soon as L is equally expressive as FO (or MSO), wrt to binary queries. In contrast to FO or MSO, in the composition language, the variables are only used to select components of the tuple, but quantification is not explicitly needed to get full FO (or MSO) expressiveness. This is because existential quantifications are hidden in compositions ($x \circ \mathbb{q} \circ \mathbb{q}' \circ y$ corresponds for instance to the FO-formula $\exists z\ \mathbb{q}(x,z) \wedge \mathbb{q}(z,y)$).

On the other hand, we have proved that if L admits a polynomial-time query evaluation algorithm, then $\mathcal{C}^{\text{nvs}}(L)$ also admits a polynomial-time query evaluation algorithm. The complexity however is quadratic in the size of the tree, which may be intractable for large trees (and indeed, there are XML documents whose size is several gigabytes (Wik)). Consequently it would interesting to know whether there are still reasonably efficient fragments of the composition language (even with a sub-FO expressiveness) which admits a very efficient query evaluation algorithm (both linear in the size of the tree, the query and the output). The model-checking problem of several fragments of conjunctive queries over XPath axis is studied in (GKS04), where a dichotomy result between linear and NP-Hard fragments is established. Similarly, one could study the query evaluation problem for composition formulas over XPath axis.

In the proof of expressiveness, the basic binary queries were subhedge restricted, meaning that they relate a node u to another node which is either descendant or descendant of one of the next-sibling of u. From an XML point of view, it means that binary queries can be constrained to relate a node to another that can be reached by following the document order (or equivalently the order given by the serialization of the XML tree). This asks the question whether there is a generic algorithm which evaluates composition of (document order restricted) binary queries in a streaming fashion. More generally, is it possible to extend any streaming evaluation algorithm for binary queries to a streaming evaluation algorithm for their composition.

On a logical perspective, it could be interesting to do a deep investigation of the composition method on finite structures to understand why a composition lemma exists. In particular, if some operation on the structures is known, is it possible to give sufficient conditions on this operation for a composition lemma to hold. This asks the question whether the composition language can be generalized to other class of structures, such as graphs, while keeping good expressiveness properties.

Finally, *hybrid logics* extend modal and temporal logics with *nominals*, which are propositional predicates interpreted by singletons (while general propositional predicates may hold in more than one state of a labeled transition system). It

would be interesting to see how hybrid logics can be used to define n-ary queries (for example by considering nominals as variables). Then, what would be their expressiveness and how they would relate to the composition language? For instance, the Hybrid μ-Calculus (SV01) could be first investigated, as it is known the μ-calculus with past has the same expressiveness as MSO on unranked trees, wrt monadic queries (BL05).

4 Application to XPath Fragments with Variables

Contents

This chapter gives two applications of the composition language to define XPath-based n-ary query languages that match the expressiveness of first-order logic but admit polynomial-time query evaluation algorithms.
The first language is an extension of the Conditional XPath language of Marx (Mar05a) with variables, while the second language is a fragment of *CoreXPath2.0*.

```
Axis       := self | child | parent | descendant | descendant_or_self
            | ancestor | ancestor_or_self | following_sibling
            | following | preceding_sibling | preceding
NameTest   := a | *    where a ∈ Σ
Step       :=  Axis::NameTest | (Axis::NameTest[TestExpr])+
PathExpr   := Step | PathExpr/PathExpr | PathExpr union PathExpr
            | PathExpr[TestExpr] | x where x ∈ X
TestExpr   := ?PathExpr | not TestExpr | TestExpr and TestExpr
```

Figure 4.1: Syntax of *CXPath*$^{\text{nary}}$

4.1 Conditional XPath with Variables

Conditional XPath (CXPath) is the extension of *CoreXPath1.0* with path expressions of the form $(\text{axis} :: a[\text{test}])^+$. It captures FO$[\prec_{ns^*}, \prec_{ch^*}]$ on unranked trees with respect to binary queries. Based on the composition language, we propose an extension *CXPath*$^{\text{nary}}$ of CXPath with variables from a set $\mathcal{X}$ that captures FO$[\prec_{ns^*}, \prec_{ch^*}]$ with respect to n-ary queries. Its syntax is given in Figure 4.1. The set of variables occurring in a *CXPath*$^{\text{nary}}$ (test or path) expression ϕ is denoted FVar(ϕ). Path expressions P are interpreted by binary relations $[\![P]\!]^{t,\rho}$ over a tree t and modulo an assignment ρ of the variables of P. We do not define the semantics formally as it very similar to the semantics of *CoreXPath2.0*. Conditional steps $(ax :: \alpha[T])^+$ are interpreted as the transitive closure of $[\![ax :: \alpha[T]]\!]^{t,\rho}$, where $\alpha \in \Sigma \cup \{*\}$.
CXPath is the restriction of *CXPath*$^{\text{nary}}$ without variables. As next shown, in order to ensure the existence of a polynomial-time query evaluation algorithm, we also require that *CXPath*$^{\text{nary}}$ path expressions satisfies the following conditions:

1. if P/P' is a *CXPath*$^{\text{nary}}$ path expression, then FVar$(P) \cap$ FVar$(P') = \varnothing$;
2. if $P[T]$ is a *CXPath*$^{\text{nary}}$ path expression, then FVar$(P) \cap$ FVar$(T) = \varnothing$;
3. if $(ax :: \alpha[T])^+$ is a *CXPath*$^{\text{nary}}$ path expression, then FVar$(T) = \varnothing$;
4. if (not T) is a *CXPath*$^{\text{nary}}$ test expression, then FVar$(T) = \varnothing$;
5. if (T and T') is a *CXPath*$^{\text{nary}}$ test expression, then FVar$(T) \cap$FVar$(T') = \varnothing$.

It is known that CXPath captures FO:

Proposition 4.1.1 ((Mar05a)) *Every binary query defined by an FO$[\prec_{ns^*}, \prec_{ch^*}]$-formula $\phi(x, y)$ is definable by a CXPath path expression P: for all unranked trees t and all nodes $u, v \in$ Dom(t)*

$$t \models \phi(u, v) \quad \textit{iff} \quad (u, v) \in [\![P]\!]^t$$

Let *CXPath*$_{\text{path}}$ denote the set of *CXPath* path expressions. As *CoreXPath1.0* path expressions, *CXPath*$_{\text{path}}$ can also be viewed as a binary query language. *CXPath*$^{\text{nary}}$ path expressions can easily be embedded into $\mathcal{C}^{\text{nvs}}$(*CXPath*$_{\text{path}}$), via the following translation $(\!|.|\!)$:

$$\begin{array}{lcllcl}
(\!|step|\!) & = & step & (\!|P/P'|\!) & = & (\!|P|\!) \circ (\!|P'|\!) \\
(\!|P \text{ union } P'|\!) & = & (\!|P|\!) \vee (\!|P'|\!) & (\!|P[T]|\!) & = & (\!|P|\!) \circ [(\!|T|\!)] \\
(\!|x|\!) & = & x & (\!|?P|\!) & = & [(\!|P|\!)] \\
(\!|\text{not } T|\!) & = & (\text{self} :: *[\text{not } T]) & (\!|T \text{ and } T'|\!) & = & [(\!|T|\!)] \circ [(\!|T'|\!)]
\end{array}$$

In this translation, only $step$ and not T becomes basic binary queries in the resulting composition formula. It is possible since $step$ and T do not contain variables. Thanks to conditions 1 to 5, the non-variable sharing condition is satisfied by $(\!|P|\!)$, for all *CXPath*$^{\text{nary}}$ path expressions P.
The back translation $(\!|\,|\!)^{-1}$ from $\mathcal{C}^{\text{nvs}}(CXPathpath)$ to *CXPath*$^{\text{nary}}$ path expressions is defined by:

$$\begin{array}{rclrcl} (\!|x|\!) & = & x & (\!|P|\!) & = & P \\ (\!|\phi_1 \circ \phi_2|\!) & = & (\!|\phi_1|\!)/(\!|\phi_2|\!) & (\!|\phi_1 \vee \phi_2|\!) & = & (\!|\phi_1|\!) \text{ union } (\!|\phi_2|\!) \\ (\!|[\phi]|\!) & = & \text{self::*}[(\!|\phi|\!)] & & & \end{array}$$

where P is a *CXPath*$_{\text{path}}$ expression.
Note that back and forth translations are in linear time. From the back translation, Proposition 4.1.1 that states FO-completeness of *CXPath*$_{\text{path}}$ wrt binary queries, and the expressiveness result Theorem 3.5.1 of the composition language, one gets:

Proposition 4.1.2 *CXPath*^{nary} *path expressions captures* $FO[\prec_{ns^*}, \prec_{ch^*}]$*-definable* n*-ary queries on unranked trees.*

From the forth translation, the complexity result Theorem 3.4.7 of query evaluation of the composition language, and the fact that binary queries defined by *CXPath*$_{\text{path}}$ expressions P can evaluated in time $O(\|P\|\|t\|^2)$ ((Mar05a)), we have:

Proposition 4.1.3 n*-ary queries defined by CXPath*^{nary} *path expressions* P *can be evaluated on trees* t *in polynomial time:*

$$O(\|P\|.\|t\|^2(1 + n.|\mathcal{Q}(P)(t)|))$$

4.2 A Polynomial-Time Fragment of *CoreXPath2.0*

In this section, we study *CoreXPath2.0* and proposes a fragment of it which captures FO on unranked trees, while admitting a polynomial-time query evaluation algorithm. The fragment is obtained by disallowing the use of variables in particular subexpressions, and variable sharing between members of subexpressions (as for instance composition). Whithout those restrictions, the query non-emptiness problem is hard. The proofs of the expressiveness and query evaluation results are done by back and forth translations to $\mathcal{C}^{\text{nvs}}$(*CoreXPath1.0* $\cup$ except), where except is the complement operator on path expressions defined in *CoreXPath2.0*.
The syntax and semantics of *CoreXPath2.0* have been defined in Section 2.8.1.

4.2.1 XPath 2.0 and FO

XPath 2.0 has been designed to feature the expressiveness of FO. To see this, we start with an auxiliary path expression by which to reach all nodes in a tree:

```
nodes =
   (ancestor::* union .) / (descendant::* union .)
```

Now we translate FO$[\prec_{ns^*}, \prec_{ch^*}]$ formulas as follows into *CoreXPath2.0* path expressions whose semantics on a tree t modulo some valuation is the set of all node pairs or the empty set. The translation does not care much about the current position during navigation. At each time, it simply jumps to the node denoted by some

variable and tests some literal there.

$(\!|\exists x.\phi|\!) = \mathsf{for}\ \$x\ \mathsf{in\ nodes\ return}\ (\!|\phi|\!)$
$(\!|\neg\phi|\!) = \mathsf{nodes\ except}\ (\!|\phi|\!)$
$(\!|\phi \wedge \phi'|\!) = \mathsf{nodes/self::*[?}(\!|\phi|\!)\ \mathsf{and}\ \ \mathsf{?}(\!|\phi'|\!)\mathsf{]}$
$(\!|x \prec_{ns^*} y|\!) = \mathsf{nodes/}\$x\mathsf{/(following\_sibling :: *\ union\ .)/[.\ is}\ \$y\mathsf{]/nodes}$
$(\!|x \prec_{ch^*} y|\!) = \mathsf{nodes/}\$x\mathsf{/(descendant :: *\ union\ .)/[.\ is}\ \$y\mathsf{]/nodes}$

The encoding is correct in the following sense.

Lemma 4.2.1 *For all FO$[\prec_{ns^*}, \prec_{ch^*}]$-formulas ϕ, all trees t and assignments ρ from FVar(ϕ) into* Dom(t)*:*

$$t, \rho \models \phi \textit{ iff } [\![(\!|\phi|\!)]\!]^{t,\rho} \neq \emptyset$$

Proof. First remark that $[\![(\!|\phi|\!)]\!]^{t,\rho}$ is either empty or equal to $\mathrm{Dom}(t)^2$. The proof goes then by induction on the structure of FO-formulas. We only elaborate the most interesting cases, which are that of quantification and negation. The following statements are equivalent: $t, \rho \models \exists x.\phi$ iff there exists $v \in \mathrm{Dom}(t)$ st $t, \rho[x \mapsto v] \models \phi$ iff $\cup_{v \in \mathrm{Dom}(t)} [\![(\!|\phi|\!)]\!]^{t,\rho[x \mapsto v]} \neq \emptyset$ (by induction hypothesis) iff $[\![\mathsf{for}\ \$x\ \mathsf{in\ nodes\ return}\ (\!|\phi|\!)]\!]^{t,\rho} \neq \emptyset$ iff $[\![(\!|\exists x \phi|\!)]\!]^{t,\rho} \neq \emptyset$.
For negations, we have $t, \rho \models \neg\phi$ iff $t, \rho \not\models \phi$ iff (by induction hypothesis) $[\![(\!|\phi|\!)]\!]^{t,\rho} = \varnothing$ iff $[\![(\!|\neg\phi|\!)]\!]^{t,\rho} = \mathrm{Dom}(t)^2$. □

This translation maps conjunction to path compositions. Alternatively, one could use the intersect operator, or conjunctions in tests. Thus, there is some redundancy in the language. In particular, we can remove the intersect operator by using the following equivalence:

```
P1 intersect P2 = P1 except (nodes except P2)
```

Observe that existential quantifiers of FO are mapped to for-loops, as for-loops are a kind of existential quantification in *CoreXPath2.0*.

Proposition 4.2.2 *CoreXPath2.0 and FO$[\prec_{ns^*}, \prec_{ch^*}]$ define the same n-ary queries modulo linear time transformations, on unranked trees.*

$$\textit{CoreXPath2.0} = \textit{FO}[\prec_{ns^*}, \prec_{ch^*}]$$

Proof. The latter transformation of FO to *CoreXPath2.0* is in linear time and preserves queries by Lemma 4.2.1. Conversely, it is quite obvious that all n-ary queries expressible in *CoreXPath2.0* are FO definable. It is sufficient to introduce existentially quantified variables systematically for all positions in path expressions. Note that existential quantifiers cannot be dropped when below negation. Note also that all axis of *CoreXPath2.0* are definable in FO, so that the back transformation is in linear time too. □

Model checking for FO is PSPACE complete (Sto74). By Proposition 4.2.2, this result carries over to *CoreXPath2.0*.

Corollary 4.2.3 *Model checking for CoreXPath2.0 is PSPACE-complete.*

4.2.2 Towards a polynomial-time fragment of *CoreXPath2.0*

We present restrictions on *CoreXPath2.0* in order to define a polynomial-time fragment (wrt query evaluation) which still captures FO on unranked trees, wrt n-ary queries.

First of all, we disallow for-loops in order to avoid quantifier alternation, which raises PSPACE-hardness of model checking. This can be done without loss of expressiveness:

Proposition 4.2.4 (Quantifier elimination)
CoreXPath2.0 without for *loops, CoreXPath2.0, and FO* $[\prec_{ns^*}, \prec_{ch^*}]$ *have the same expressiveness wrt* n*-ary queries, over unranked trees.*

Proof. From Proposition 4.2.2, we already know that *CoreXPath2.0* and FO have the same expressive power. It can be proved that *CoreXPath2.0* without for-loops still matches the expressiveness of FO. Indeed, Marx has shown that every extension of *CoreXPath1.0* that is closed under complementation (of binary relations) is complete with respect to binary FO-queries (Theorem 2 of (Mar05b)). This is the case for instance for *CoreXPath1.0* extended with the except operator, which we denote by *CoreXPath1.0* $\cup$ except. Now, the result follows from Theorem 3.5.1, since $\mathcal{C}$(*CoreXPath1.0* $\cup$ except) can easily be expressed in *CoreXPath2.0*, as shown by Proposition 4.2.8 (which can be read independently). Note that the transformation of Theorem 3.5.1 creates a formula of non-elementary size. □

While quantifiers (for-loops) can be removed from *CoreXPath2.0* without loss of expressiveness (wrt n-ary queries), it cannot be done in FO, even if existential quantifiers which occurs under an even number of negations are still allowed (called the existential fragment):

Lemma 4.2.5 *The existential fragment of FO* $[\prec_{ns^*}, \prec_{ch^*}]$ *is strictly less expressive than FO* $[\prec_{ns^*}, \prec_{ch^*}]$.

Proof. The inclusion is given by Lemma 4.2.4. To prove that it is strict, we consider the following negative formula expressing that if x is an ancestor of y, no single ns step occurs on the firstchild/nextsibling path from x to y:

$$\neg(\exists z.\exists z'.(x \prec_{ch^*} z \wedge z \prec_{ns^*} z' \wedge z' \prec_{ch^*} y))$$

It can be expressed in *CoreXPath2.0* without for-loops:

$$.[\text{not } (\$x/\text{descendant::}*/\text{nextsibling::}*/\text{descendant::}*[. \text{ is } \$y])]$$

The translation $(\!|\ .\ |\!)$ of this path expression to an FO formula creates existentially quantified variables below negation. This quantifier cannot be dropped or eliminated, even if we add firstchild to the signature.
To see this, let $\phi_0(x, y)$ be the counter example above and suppose that it can be expressed by some existential formula of the form $\exists x_1 \ldots \exists x_n.\psi$ where ψ is a quantifier free FO formula with free variables $x_1, \ldots, x_n, x, y$.
Let t be a tree with nodes u, v satisfying $(u, v) \in \text{firstchild}(t)^{n+2}$. This means that one can descend from u to v over a path of $n+2$ subsequent firstchild steps. Since $t, u, v \models \phi_0(x, y)$ there exists a variable assignment ρ with $\rho(x) = u, \rho(y) = v$ such that $t, \rho \models \psi$. Since the number of intermediate nodes on the path from u to v is $n+1$, there exists at least one node w on this path that does not belong to $\rho(\{x_1, \ldots, x_n, x, y\})$. Let us transform the tree t into a tree t' by adding a new node on the left of the previous first child of w. One can prove that $t', \rho \models \psi$, since on the nodes of t we only changed the relation firstchild for w. This contradicts $t', u, v \not\models \phi_0$ since the path from u to v contains a nextsibling step now. □

Removing for-loops, and even variables below negations, from *CoreXPath2.0*, however, does not suffice to reduce the complexity of query non-emptiness.

Proposition 4.2.6 *Query non-emptiness for CoreXPath2.0 without* for*-loops is NP-complete.*

Proof. upper-bound Let t be an unranked tree and P a path expression of *CoreXPath2.0* without for-loops, and let $\overline{x}$ the variables (necessarily free) contained in P. We want to decide whether $\mathcal{Q}(P)(t) \neq \emptyset$. This can be done as follows. First guess a variable assignment $\alpha : \{\overline{x}\} \to \mathrm{Dom}(t)$. Secondly, we translate P into a formula of *Propositional Dynamic Logic (PDL)*, over the *CoreXPath2.0* axis (i.e. *CoreXPath2.0* axis viewed as basic programs). Variables $\$x$ are translated into propositional symbols p_x which are true at position $\alpha(x)$, i.e. $t, u \models p_x$ iff $u = \alpha(x)$, for $u \in \mathrm{Dom}(t)$. Finally, it is known that PDL has linear time combined complexity for model-checking (ADdR03), which concludes the proof.
lower-bound This is proved by reducing SAT. The proof is the same as the proof of Proposition 3.4.1. Disjunctions are expressed by using union and conjunction by using composition of filters. □

In the proof of Proposition 4.2.6, the encoding of Sat relies on using variable sharing between different branches of compositions. Similar effects can arise with filters and conjunctions in tests, so we have to forbid variable sharing in all these cases.
The next restriction that we want to impose on test expressions is that they do not contain variables below negation. Otherwise, we would have to treat variable sharing in conjunction and disjunction symmetrically.
Variables on the right of exceptions are also forbidden, since this is a form of negation. Variables are also forbidden on the left of exceptions as well as in intersections as it is needed in to relate *CoreXPath* to our composition language, but we still do not know if this is necessary.
Variables sharing in filters would amount to variable sharing in composition.
There are no restrictions on the union and or operators, so variables can be shared there. It is crucial to forbid variables below all kinds of negation in except and not . Otherwise, one could encode intersect and and expressions with variable sharing.
We finally come to the polynomial-time fragment of *CoreXPath2.0*:

Definition 1. *The n-ary query language CoreXPath2.0^{nvs} is the restriction of CoreXPath2.0 whose expressions satisfy the following conditions:*

N(for) *no* for *loops and thus no explicit quantifiers.*

NV(intersect) *no variables in intersections: all subexpressions P_1* intersect *P_2 satisfy FVar$(P_1) =$ FVar$(P_2) = \varnothing$.*

NV(except) *no variables in exceptions: all subexpressions P_1* except *P_2 satisfy FVar$(P_1) =$ FVar$(P_2) = \varnothing$.*

NV(not) *no variables below negation, i.e., subexpressions* not*(T) satisfy FVar$(T) = \varnothing$*

NVS(/) *no variable sharing in composition: all subexpressions P_1/P_2 satisfy FVar$(P_2) = \varnothing$*

NVS([]) *no variable sharing in filters: all subexpressions $P[T]$ satisfy $FVar(P) \cap FVar(T) = \varnothing$*

NVS(and) *no variable sharing in conjunctions: all tests in subexpressions T_1 and T_2 satisfy $FVar(T_1) \cap FVar(T_2) = \varnothing$*

Theorem 4.2.7

Expressiveness. *CoreXPath2.0^{nvs} can express all n-ary first-order queries.*

Complexity. *Answers sets of n-ary queries $\mathcal{Q}(P)$ by path expressions P can be answered in polynomial time $O(\|P\|.\|t\|^3 + n.\|P\|.\|t\|^2.|A|)$ where $|A|$ is the cardinality of the answer set.*

The next two sections are devoted to the proof of this Theorem, by relation to composition language. In particular, it is proved that *CoreXPath2.0*$^{\text{nvs}}$ and composition of variable-free *CoreXPath2.0* expressions $\mathcal{C}^{\text{nvs}}$(*CoreXPath2.0*$^{\text{varfree}}$) are equally expressive, modulo linear-time back and forth translations.

4.2.3 The variable-free fragment

We denote by *CoreXPath2.0*$^{\text{varfree}}$ the variable-free fragment of *CoreXPath2.0*. As we next show, this language can be identified with the extension of *CoreXPath1.0* by the except operator.
In *CoreXPath2.0* variables are used to define n-ary queries but are not present in *CoreXPath2.0*$^{\text{varfree}}$. Instead we naturally use path expressions $P \in$ *CoreXPath2.0*$^{\text{varfree}}$ to define binary queries, which relate starting nodes of navigation to ending nodes, as for *CoreXPath1.0*.

Proposition 4.2.8 *The following languages define the same binary queries modulo linear time translations:*

$$\textit{CoreXPath1.0} \cup \textsf{except} \;=\; \textit{CoreXPath2.0}^{\textit{varfree}}$$

Proof. The inclusion from the left to the right is obvious by syntactic identity. For the converse, we encode *CoreXPath2.0*$^{\text{varfree}}$ into *CoreXPath1.0*∪except by a linear time translation $(\!|.|\!)$. We only define it for the additional features of *CoreXPath2.0*, compared to *CoreXPath1.0*:

$$\begin{aligned}
&(\!|.|\!) = \textsf{self}\\
&(\!|P \textsf{ intersect } P'|\!) = \textsf{nodes except } ((\textsf{nodes except } (\!|P|\!)) \textsf{ union}\\
&\qquad\qquad\qquad\qquad\qquad\qquad (\textsf{nodes except} (\!|P'|\!)))\\
&(\!|P \textsf{ except } P'|\!) = (\!|P|\!) \textsf{ except } (\!|P'|\!)\\
&(\!|. \textsf{ is } .|\!) = ?\textsf{self}
\end{aligned}$$

□

Theorem 4.2.9 *CoreXPath2.0^{varfree} satisfies the following two properties:*

Expressiveness. *All binary FO-queries can be expressed in CoreXPath2.0^{varfree}.*

Complexity *Query evaluation of binary queries expressed by CoreXPath2.0^{varfree} path expressions P on trees t is in time $O(\|P\|.\|t\|^3)$.*

The completeness result is due to Proposition 4.2.8 and Theorem 2 of Marx (Mar05b), which says that every extension of *CoreXPath1.0* that is closed under complementation is equally expressive as FO on unranked trees wrt binary queries.

For the algorithmic part, we reason with *CoreXPath1.0* $\cup$ except expressions, thanks to Proposition 4.2.8. However, we view the except operator as a unary operator, meaning that except P = nodes except P. This can be assumed wlog since P except P' = except ((except P) union P').
It is well known that unary queries in *CoreXPath1.0* can be evaluated in linear time (GKP05). This gives a quadratic binary query evaluation algorithm for *CoreXPath1.0*, in the size of the input tree. The main evaluation trick of *CoreXPath1.0* lies in that the set of successors $S_a(N) = \{u' \mid \exists u \in N,\ a(u,u')\}$ of a set of nodes N by a standard axis a, in a tree t, is computable in linear time $O(|t|)$. This can be extended to full *CoreXPath1.0* expressions, so that computing $S_P(N)$, for some *CoreXPath1.0* expression P, is in linear time $O(|P||t|)$. However, it is not clear whether this trick can be used for evaluating *CoreXPath2.0*$^{\text{varfree}}$, since except operator can occur at any position in the input expression, and in general $S_{\text{except } P}(N) \neq \overline{S_P(N)}$.
We now give an algorithm to answer binary queries by *CoreXPath2.0*$^{\text{varfree}}$ expressions. Given a *CoreXPath2.0*$^{\text{varfree}}$ expression P, and a tree t, we represent $\mathcal{Q}(P)(t)$ as a $|t| \times |t|$ Boolean matrix M_P^t. Naturally, M_P^t is defined by $\forall u,u' \in \text{Dom}(t)$, $M_P^t[u,u'] = 1$ iff $(u,u') \in \mathcal{Q}(P)(t)$. We consider the following operations on matrix: the sum $+$ and the product $.$ over the Boolean algebra $(\{0,1\},\vee,\wedge)$. We also defined $\neg M$ as the matrix M where 0's (resp. 1's) have been replaced by 1's (resp. 0's), and $[M]$ by $\forall u,u' \in \text{Dom}(t)$ $[M][u,u'] = 1$ iff $u = u'$ and $\exists u'' \in \text{Dom}(t)$ s.t. $M[u,u''] = 1$. Similarly, we can easily define operations on matrix which correspond to operators in test expressions. We get the following, for all *CoreXPath2.0*$^{\text{varfree}}$ expressions P_1, P_2, P and all trees t:

$$\begin{array}{lcllcl} M^t_{P_1/P_2} & = & M^t_{P_1}.M^t_{P_2} & M^t_{\text{except } P} & = & \neg M^t_P \\ M^t_{P_1 \text{ union } P_2} & = & M^t_{P_1} + M^t_{P_2} & M^t_{P[T]} & = & M^t_P.[M^t_T] \end{array}$$

A naive implementation of these operations leads to a $O(|P||t|^3)$ time complexity to evaluate a *CoreXPath2.0*$^{\text{varfree}}$ expression P on a tree t. However, from a theoretical point of view, this can be improved to $O(|P||t|^{2.376})$ since the best known algorithm for n-square Boolean matrix multiplication is in time $O(n^{2.376})$, due to Coppersmith and Winograd (CW87).

4.2.4 Relation to the composition language

We have the following relationship between *CoreXPath2.0*$^{\text{nvs}}$ and the composition language:

Proposition 4.2.10 *The following two languages define the same n-ary queries modulo linear time translations:*

$$\mathcal{C}_2^{nvs}(\textit{CoreXPath2.0}^{varfree}) \ = \ \textit{CoreXPath2.0}^{nvs}$$

Proof. The inclusion from the left to the right follows from the following translation:

$$\begin{array}{l} (\!|P|\!) = P \text{ where } P \text{ in } \textit{CoreXPath2.0}^{\text{varfree}} \\ (\!|\phi \circ \phi'|\!) = (\!|\phi|\!)/(\!|\phi'|\!) \\ (\!|x|\!) = .[\ .\ \text{is}\ \$x] \\ (\!|[\phi]|\!) = .[(\!|\phi|\!)] \\ (\!|\phi \vee \phi'|\!) = (\!|\phi|\!) \text{ union } (\!|\phi'|\!) \end{array}$$

It remains to verify that the result of the translation belongs to *CoreXPath2.0*$^{\text{nvs}}$. This holds since *CoreXPath2.0*$^{\text{varfree}} \subseteq$ *CoreXPath2.0*$^{\text{nvs}}$ and since we assume the

non-variable sharing. The inverse inclusion follows by the inverse translation in Figure 4.2.
It can be seen by induction on the size of path expressions that the back translation always maps to $\mathcal{C}^{\text{nvs}}(\mathit{CoreXPath2.0}^{\text{varfree}})$. □

In order to evaluate $\mathit{CoreXPath2.0}^{\text{nvs}}$ expressions, it suffices to translate them into $\mathcal{C}_2^{\text{nvs}}(\mathit{CoreXPath2.0}^{\text{varfree}})$ expressions in linear time (Proposition 4.2.10) and to evaluate the resulting expressions thanks to Theorem 3.4.7 and Theorem 4.2.9. This gives the proof of the complexity part of Theorem 4.2.7.
On the other hand, from Proposition 4.2.10, Theorem 3.5.1 and Theorem 4.2.9, we get the expressiveness result of Theorem 4.2.7.

$$
\begin{array}{lcl}
(\!|A :: L|\!)^{-1} & = & A :: L \\
 & & (\text{in } \textit{CoreXPath2.0}^{\text{varfree}}) \\
(\!|.|\!)^{-1} & = & \mathsf{self} \\
 & & (\text{in } \textit{CoreXPath2.0}^{\text{varfree}}) \\
(\!|\$x|\!)^{-1} & = & \mathsf{nodes} \circ x \\
(\!|P_1/P_2|\!)^{-1} & = & (\!|P_1|\!)^{-1} \circ (\!|P_2|\!)^{-1} \\
 & & (\text{NVS}(/) \text{ implies} \text{NVS}(\circ)) \\
(\!|P_1 \text{ union } P_2|\!)^{-1} & = & (\!|P_1|\!)^{-1} \cup (\!|P_2|\!)^{-1} \\
(\!|P_1 \text{ intersect } P_2|\!)^{-1} & = & P_1 \text{ intersect } P_2 \\
 & & (\text{in } \textit{CoreXPath2.0}^{\text{varfree}} \text{ modulo linear time} \\
 & & \text{by NV(intersect) and Proposition 4.2.8}) \\
(\!|P_1 \text{ except } P_2|\!)^{-1} & = & P_1 \text{ except } P_2 \\
 & & (\text{in } \textit{CoreXPath2.0}^{\text{varfree}} \text{ modulo linear time} \\
 & & \text{by NV(except) and Proposition 4.2.8}) \\
(\!|P_1[P_2]|\!)^{-1} & = & (\!|P_1|\!)^{-1} \circ [(\!|P_2|\!)^{-1}] \\
 & & (\text{NVS}([]) \text{ ensures } \text{NVS}(\circ)) \\
(\!|P[\text{not } T]|\!)^{-1} & = & (\!|P|\!)^{-1} \circ .[\text{not } T] \\
 & & (.[\text{not } T] \text{ is in } \textit{CoreXPath2.0}^{\text{varfree}} \text{ by NV(not)} \\
 & & \text{and Proposition 4.2.8 so the result satisfies NVS}(\circ)\ ) \\
(\!|P[T_1 \text{ and } T_2]|\!)^{-1} & = & (\!|P|\!)^{-1} \circ [(\!|T_1|\!)^{-1}] \circ [(\!|T_2|\!)^{-1}] \\
 & & (\text{NVS(and) guarantees } \text{NVS}(\circ)) \\
(\!|P[T_1 \text{ or } T_2]|\!)^{-1} & = & (\!|P|\!)^{-1} \circ ([(\!|T_1|\!)^{-1}] \ \vee \ [(\!|T_2|\!)^{-1}]) \\
 & & (\text{NVS}([]) \text{ maps to } \text{NVS}(\circ)\ )
\end{array}
$$

Figure 4.2: From *CoreXPath2.0*$^{\text{nvs}}$ to $\mathcal{C}_2^{\text{nvs}}$(*CoreXPath2.0*$^{\text{varfree}}$)

Part II

A Spatial Logic for Trees

5 TREE AUTOMATA WITH GLOBAL CONSTRAINTS

CONTENTS

THIS chapter introduces a novel class of tree automata with constraints, called *tree automata with global equality and disequality constraints* (TAGEDs for short), for which a large subclass is proved to be decidable, with respect to to the emptiness problem. They have been introduced to decide a fragment of TQL (Theorem 6.5.5), but might be of independent interest. A natural correspondence with MSO is investigated. In particular, a decidable extension of MSO with tree equality tests is considered. TAGEDs are studied over binary trees, but lifted to unranked trees in the last section.

5.1 INTRODUCTION

The emergence of XML has strengthened the interest in tree automata, as it is a clean and powerful model for XML tree acceptors (NS02b, Sch07). In this context, tree automata have been used, for example, to define schemas, and queries, but also to decide tree logics, to type XML transformations, and even to learn queries. However, it is known that sometimes, expressiveness of tree automata is not sufficient. This is the case for instance in the context of non-linear rewriting systems, for which more powerful tree acceptors are needed to decide interesting properties of those rewrite systems. For example, the set of ground instances of $f(x, x)$ is not regular.

Tree automata with constraints have been introduced to overcome this lack of expressiveness (BT92, DCC95, JRV06, KL06). In particular, the transitions of these tree automata are fired as soon as the subtrees of the current tree satisfy some structural (dis)equality. But typically, these constraints are kept local to preserve decidability of emptiness and good closure properties – in particular, tests are performed between siblings or cousins –. In the context of XML, and especially to define tree patterns, one needs global constraints. For instance, it might be useful to represent the set of ground instances of the pattern $X(\texttt{author}(x), \texttt{author}(x))$, where X is a binary context variable, and x is an author which occur at least twice (we assume this pattern to be matched against XML trees representing a bibliography). In this example, the two subtrees corresponding to the author might be arbitrarily faraway, making the tree equality tests more global. Patterns might be more complex, by the use of negation (which allow to test tree disequalities), Boolean operations, and regular constraints on variables. The TQL logic, in particular, allows to define such patterns. In this chapter, we introduce *Tree Automata with Global Equalities and Disequalities* (TAGEDs for short), which capture the expressiveness of the guarded TQL fragment with tree variables (assumed to be existentially quantified). These are tree automata A equipped with an equality relation $=_A$ and a disequality relation $\neq_A$ on (a subset of) states. A tree t is accepted by A if there is a computation of A on t which leads to an accepting state, and such that, whenever two subtrees of t evaluate to two states q_1, q_2 (not necessarily equal) such that $q_1 =_A q_2$ (resp. $q_1 \neq_A q_2$), then these two subtrees must be structurally equal (resp. structurally different). TAGEDs may be interesting on their own, since they are somehow orthogonal to usual automata with constraints (BT92). Indeed, if we view equality tests during computations as an equivalence relation on a subset of nodes (two nodes being equivalent if their rooted subtrees have been successfully tested to be equal by A), in the former, there are a bounded number of equivalence classes of unbounded cardinality, while in the latter, there might be an unbounded number of equivalence classes of bounded cardinality.

We prove several properties of TAGED-definable languages: closure by union and intersection, non-closure by complement. We prove other results such as nondeterminization and undecidability of universality. We also prove a useful result which states that we can assume that $=_A \subseteq id_Q$ without loss of generality (where id_Q is the identity relation on the set of states). The main results are decidability of emptiness for several classes of TAGEDs: TAGEDs with equality tests or disequality tests only, and TAGEDs which performs a bounded number of disequality tests along any root-to-leaf path (and arbitrarily many equality tests). This is done by introducing a pumping technique for TAGEDs.

Related Work Extensions of tree automata which allow for syntactic equality and disequality tests between subterms have already been defined by adding constraints to the rules of automata. E.g., adding the constraint $1.2 = 2$ to a rule means that one can apply the rule at position π only if the subterm at position $\pi.1.2$ is equal to the subterm at position $\pi.2$. Testing emptiness of the recognized language is undecidable in general (Mon81) but two classes with a decidable emptiness problem have been emphasized. In the first class, automata are deterministic and the number of equality tests along a path is bounded (DCC95) whereas the second restricts tests to sibling subterms (BT92). This latter class has recently been extended to unranked trees (KL06), the former one has been extended to equality modulo equational theories (JRV06). But, contrarily to TAGEDs, in all these classes, tests are performed locally, typically between sibling or cousin subterms. Automata with local and global equality tests, using one memory, have been considered in (CC05). These are tree automata with a memory which can store a single tree. They run in a bottom-up fashion, and the two trees contained in the respective memories coming from two computations on the two subtrees of the current node can be combined thanks to tree operations and compared with tree equality tests. The emptiness problem is decidable for tree automata with memory, and they can simulate positive TAGEDs (TAGEDs performing only equality tests) which use at most one state per runs to test equalities. However, their ability to perform local tests make them incomparable to TAGEDs. Finally, automata for DAGs are studied in (ANR05, Cha99), they cannot be compared to positive TAGEDs, as they run on DAG representations of trees (with maximal sharing), and in TAGEDs, we cannot impose that every equal subtrees evaluate to the same state in a successful run, as shown by Example 5.2.2 where it is needed to evaluate the leaves to possibly different states.

Organization of the Chapter TAGEDs are defined in Section 5.2. Some closure properties of TAGED-definable languages as well as decision problems are investigated in Section 5.3. Decidability of emptiness is proved in Section 5.4 for two subclasses which can test only equalities or inequalities respectively. An expressive subclass which mixes equality and disequality tests is introduced in Section 5.5, where emptiness is proved. A natural correspondence between TAGEDs and an extension of MSO with tree equality tests is investigated in Section 5.6. Finally, TAGEDs that work on unranked trees over an infinite alphabet are considered in Section 5.7.

5.2 Definition and Examples

For the sake of clarity, we consider binary trees, with constant and binary function symbols from a ranked alphabet Σ, but all the definitions can naturally be extended to ranked trees of arbitrary arity. In the last section of this chapter, we show that all the results can be lift to unranked trees, via a particular binary encoding which preserves the subtrees (a subtree in the encoding corresponds to a subtree in the unranked tree).

Definition 5.2.1 (TAGED) *A TAGED is a 6-tuple $A = (\Sigma, Q, F, \Delta, =_A, \neq_A)$ such that:*

- *(Σ, Q, F, Δ) is a tree automaton;*
- *$=_A$ is a reflexive and symmetric binary relation on a subset of Q;*
- *$\neq_A$ is an irreflexive and symmetric binary relation on Q.*

A TAGED A is said to be positive (resp. negative) if $\neq_A$ (resp. $=_A$) is empty.

The notion of successful run differs from tree automata as we add equality and disequality constraints. A *run* r of the tree automaton (Σ, Q, F, Δ) on a tree t satisfies the equality constraints if whenever two nodes u, v of r are labeled by states $\mathsf{lab}^r(u)$ and $\mathsf{lab}^r(v)$ respectively such that $\mathsf{lab}^r(u) =_A \mathsf{lab}^r(v)$, the two subtrees $t|_u$ and $t|_v$ are equal:

$$\text{for all } u, v \in \mathrm{Dom}(r), \text{ if } \mathsf{lab}^r(u) =_A \mathsf{lab}^r(v) \text{ then } t|_u = t|_v.$$

Similarly, it satisfies the disequality constraints if:

$$\text{for all } u, v \in \mathrm{Dom}(r), \text{ if } \mathsf{lab}^r(u) \neq_A \mathsf{lab}^r(v) \text{ then } t|_u \neq t|_v.$$

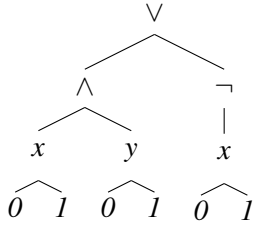
A run is *successful* (or *accepting*) if it is successful for the tree automaton (Σ, Q, F, Δ) and if it satisfies all the constraints. The language accepted (or recognized) by A, denoted $\mathcal{L}(A)$, is the set of trees t having a successful run for A. We denote by $\mathrm{dom}(=_A)$ the domain of $=_A$, i.e. $\{q \mid \exists q' \in Q, q =_A q'\}$. The set $\mathrm{dom}(\neq_A)$ is defined similarly. Two TAGEDs are *equivalent* if they accept the same language. Finally, the size of A, denoted $\|A\|$, is the size of (Σ, Q, F, Δ) plus $2|\mathrm{dom}(=_A) \cup \mathrm{dom}(\neq_A)|$.
TAGEDs are strictly more expressive than tree automata, as illustrated by the next example.

Example 5.2.1 *Let $Q = \{q, q_=, q_f\}$, $F = \{q_f\}$, and let Δ be defined as the set of following rules, for all $a, f \in \Sigma$:*

$$a \to q \quad a \to q_= \quad f(q,q) \to q \quad f(q,q) \to q_= \quad f(q_=, q_=) \to q_f.$$

*The language accepted by the positive TAGED $A_1 = (\Sigma, Q, F, \Delta, \{q_= =_{A_1} q_=\})$ is the set $\{f(t,t) \mid f \in \Sigma, t \in \mathcal{T}_{ran}(\Sigma)\}$, which is known to be non regular (CDG*07). A tree and a successful of A_1 on it are depicted below:*

Example 5.2.2 *Let $\mathcal{X}$ be a finite set of variables. We now define a TAGED $A_{sat} = (\Sigma_{sat}, Q_{sat}, F_{sat}, \Delta_{sat})$ which accepts tree representations of satisfiable Boolean formulas with free variables $\mathcal{X}$. The alphabet Σ_{sat} consists of the binary symbols $\wedge, \vee$ and x, for all $x \in \mathcal{X}$, the unary symbol $\neg$, and the two constant symbols $0, 1$. Every Boolean formula is naturally viewed as a tree, except for variables $x \in \mathcal{X}$ which are encoded as trees $x(0,1)$ over $\Sigma_{\mathcal{X}}$. For instance, the formula $(x \wedge y) \vee \neg x$ is encoded as the tree:*



Let $Q = \{q_x \mid x \in \mathcal{X}\} \cup \{q_0, q_1, p_0, p_1\}$, and $F = q_1$. The idea is to choose non-deterministically to evaluate the leaf 0 or 1 below x to q_x, but not both, for all $x \in \mathcal{X}$. This means that we affect 0 or 1 to a particular occurrence of x. Then, by imposing that every leaf evaluated to q_x are equal, for all $x \in \mathcal{X}$, we can ensure that we have chosen to same Boolean value for all occurrences of x, for all $x \in \mathcal{X}$. This can be done with the following set of rules, for all $b, b_1, b_2 \in \{0, 1\}$, all $\oplus \in \{\wedge, \vee\}$, and all $x \in \mathcal{X}$:

$$\begin{array}{rcl@{\qquad}rcl} b & \to & p_b & b & \to & q_x \\ x(p_0, q_x) & \to & q_1 & x(q_x, p_1) & \to & q_0 \\ \neg(q_b) & \to & q_{\neg b} & \oplus(q_{b_1}, q_{b_2}) & \to & q_{b_1 \oplus b_2} \end{array}$$

Finally, for all $x \in \mathcal{X}$, we let $q_x =_{A_{sat}} q_x$.

The *uniform membership problem* is: given a TAGED A, given a tree t, does t belong to $\mathcal{L}(A)$?

Proposition 5.2.3 *Uniform membership is NP-complete for TAGEDs.*

Proof. Example 5.2.2 gives a polynomial reduction of SAT to the membership of TAGEDs. To show it is in NP, it suffices to guess a labeling of the nodes of the tree by states, and then to verify that it is a run, and that equality and disequality constraints are satisfied. This can be done in linear time both in the size of the automaton and of the tree. Indeed, we can compute in time $O(\|t\|)$ a data structure which allows to test in constant time whether two subtrees (identified by their node roots) are equal or not. □

Note that Proposition 5.2.3 remains true even for positive TAGEDs.

5.3 Closure Properties of TAGEDs and Decision Problems

5.3.1 Closure Properties of TAGED-definable languages

Proposition 5.3.1 (Closure by union and intersection) *TAGED-definable languages are closed by union and intersection.*

Proof. Let $A = (\Lambda, Q, F, \Delta, =_A, \neq_A)$ and $A' = (\Lambda, Q', F', \Delta', =_{A'}, \neq_{A'})$ be two TAGEDs. Wlog, we suppose that $Q \cap Q' = \emptyset$. The TAGED accepting $\mathcal{L}(A) \cup \mathcal{L}(A')$ is defined by $A \cup A' = (\Lambda, Q \cup Q', F \cup F', \Delta \cup \Delta', =_A \cup =_{A'}, \neq_A \cup \neq_{A'})$.
For the closure by intersection, we extend the product automaton $A \times A'$ with the following relations:

- $=_{A \times A'} = \{((q, q'), (p, p') \mid q =_A p \text{ or } q' =_{A'} p'\}$
- $\neq_{A \times A'} = \{((q, q'), (p, p') \mid q \neq_A p \text{ or } q' \neq_{A'} p'\}$

The set of final states is $F \times F'$.
We let $\text{ta}(A)$ and $\text{ta}(A')$ be the "tree automaton" parts of A and A' respectively. Now we prove that $\mathcal{L}(A \times A') = \mathcal{L}(A) \cap \mathcal{L}(A')$.
Let $t \in \mathcal{L}(A \times A')$, then there exists a successful run r'' of $A \times A'$ on t. By construction, this run can be decomposed into a successful run r of $\text{ta}(A)$ on t and a successful run r' of $\text{ta}(A')$ on t. It remains to show that r and r' satisfies the constraints

of A and A' respectively. Let $u, v \in \mathrm{Dom}(t)$ such that $\mathsf{lab}^r(u) =_A \mathsf{lab}^r(v)$. It means that there exist four states $q, p \in Q, q', p' \in Q'$ such that $\mathsf{lab}^{r''}(u) = (q, q')$, $\mathsf{lab}^{r''}(v) = (p, p')$ and $q =_A p$. Hence $(q, q') =_{A \times A'} (p, p')$ and we get $t|_u = t|_v$. This is similar for r' and when considering inequalities.
Conversely, suppose that $t \in \mathcal{L}(A) \cap \mathcal{L}(A')$. Hence there exist two successful runs r and r' of A and A' on t respectively. By construction of $A \times A'$, these two runs can be combined into a run r'' of $A \times A'$ on t. It remains to show that r'' satisfies the constraints of $A \times A'$. Let $u, v \in \mathrm{Dom}(t)$ such that $\mathsf{lab}^{r''}(u) =_{A \times A'} \mathsf{lab}^{r''}(v)$. It means that there exists four states $q, p \in Q$ and $q', p' \in Q'$ such that $\mathsf{lab}^{r''}(u) = (q, q')$, $\mathsf{lab}^{r''}(v) = (p, p')$, and either $q =_A p$ or $q' =_{A'} p'$. If $q =_A p$, we get $t|_u = t|_v$ since r respects the constraints of A, and if $q' =_{A'} p'$, we also get $t|_u = t|_v$ since r' respects the constraints of A. The proof goes similarly when dealing with inequalities. □

These closure properties hold also for the class of languages defined by positive or negative TAGEDs. One can see a TAGED as a computational machine which runs on trees in a bottom-up fashion. A TAGED is therefore said to be *deterministic* if there is at most one possible computation per trees. This notion can be defined as usual with a simple syntactic restriction:

Definition 5.3.1 *A TAGED $A = (\Sigma, Q, F, \Delta, =_A, \neq_A)$ is deterministic if all rules have different left-hand sides.*

Note that for all deterministic TAGED A, it is possible to compute a non-deterministic TAGED accepting the complement of $\mathcal{L}(A)$: we have to check if the tree evaluates in a non-accepting state or in an accepting state but in this case we non-deterministically guess a position where a constraint is not satisfied. However:

Proposition 5.3.2 *TAGEDs are not determinizable.*

Proof. Let $\Sigma = \{f, a\}$ be an alphabet where f is binary and a constant. Consider the language $L_0 = \{f(t, t) \mid t \in \mathcal{T}_{ran}(\Sigma)\}$. It is obvious that L_0 is definable by a non-deterministic TAGED. Suppose that there is a deterministic TAGED $A = (\Sigma, Q, F, \Delta, =_A, \neq_A)$ such that $\mathcal{L}(A) = L_0$. Let t be a tree whose height is strictly greater than $|Q|$. Since $f(t, t) \in L_0$, there are a successful run $q_f(r, r)$ of A on $f(t, t)$ for some final state q_f, two nodes u, v and a state $q \in Q$ such that $t|_v$ is a strict subtree of $t|_u$, and $\mathsf{lab}^r(u) = \mathsf{lab}^r(v) = q$. Since $f(t|_u, t|_u) \in L_0$, and A is deterministic, there is a final state $q'_f \in F$ and a rule $f(q, q) \to q'_f \in \Delta$. Hence $q'_f(r|_u, r|_v)$ is a run of A on $f(t|_u, t|_v)$. Since $q_f(r, r)$ satisfies the constraints, they are also satisfied between $r|_u$ and $r|_v$, so that $q'_f(r|_u, r|_v)$ satisfies the constraints as well. Hence $f(t|_u, t|_v) \in \mathcal{L}(A)$, which contradicts $t|_u \neq t|_v$. □

Deterministic TAGEDs are still strictly more expressive than tree automata, since $\{f(g(t), g(t)) \mid t \in \mathcal{T}_{ran}(\{f, a\})\}$ is definable by a deterministic positive TAGED but not by a tree automaton. It suffices to evaluate every tree of $\mathcal{T}_{ran}(\{f, a\})$ in a state q, and to add the rules $g(q) \to q_=$ and $f(q_=, q_=) \to q_f$, for some state q and final state q_f, where $q_= =_A q_=$. Proposition 5.3.2 is not surprising, since:

Proposition 5.3.3 *The class of TAGED-definable languages is not closed by complement.*

Proof. We exhibit a tree language whose complement is easily definable by a TAGED, but which is not TAGED-definable. The idea is to take a language where at each level, the two subtrees are equal. To check membership to its complement,

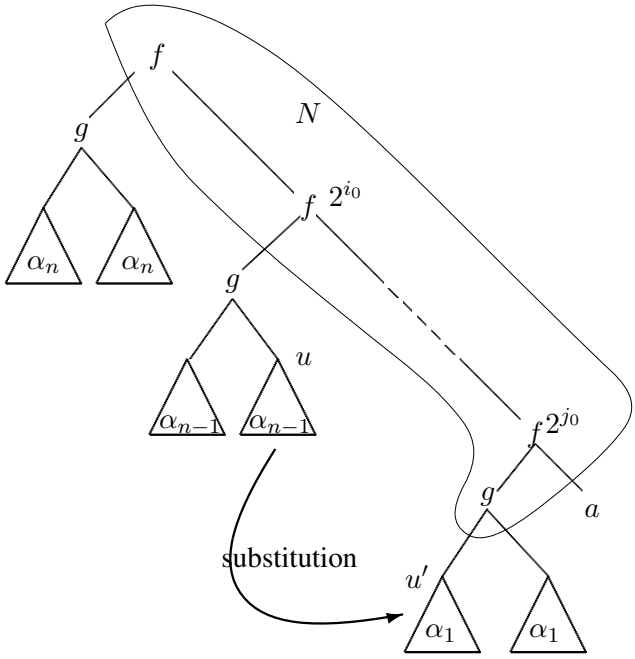


Figure 5.1: Tree t_n

it suffices to guess a position where the two subtrees are different, which can be done by a TAGED.
Let $\Sigma = \{f, g, a\}$ where f, g are binary and a is a constant, and $h \notin \Sigma$ be a binary symbol. Let $T_0 = \{a\}$, and for $n > 0$, $T_n = \{f(g(t,t),t') \mid t \in \mathcal{T}_{ran}(\{h,a\}),\ t' \in T_{n-1}\}$. Let $L = \bigcup_{n\in\mathbb{N}} T_n$. The complement of L is easily definable by a TAGED. Suppose that L is definable by a TAGED $A = (\Sigma \cup \{h\}, Q, F, \Delta, =_A, \neq_A)$. Let $n \geq |Q| + 1$, and let $\alpha_1, \ldots, \alpha_n \in T_{\{h,a\}}$ such that $\forall i < j, \|\alpha_i\| < \|\alpha_j\|$. Now, let $t_0 = a$, and for $i > 0$, $t_i = f(g(\alpha_i, \alpha_i), t_{i-1})$ (see Fig 5.1). It is clear that $t_n \in T_n$. Hence there is a successful run r of A on t_n.
In the rest of the proof, we identify the nodes of t_n and the paths from the root to them.
Since $n \geq |Q| + 1$, there are $b, b' \in \{1, 2\}$, $i_0, j_0 \in \{0, \ldots, n-1\}$, $i_0 < j_0$, two nodes $u, u' \in \mathrm{Dom}(t_n)$, and a state $q \in Q$ such that: (i) $u = 2^{i_0}1b$ and $u' = 2^{j_0}1b'$, (ii) $\mathsf{lab}^r(u) = \mathsf{lab}^r(u') = q$. Let $t'_n = t_n[t_n|_u]_{u'}$, i.e. the tree t_n where the subtree at node u' has been substituted by the subtree at node u. We do the same corresponding substitution in r, which results into a run denoted r' (see Fig 5.1). Note that $t'_n \notin L$ since $\|\alpha_i\| \neq \|\alpha_j\|$ by definition of t_n, for all $i \neq j$. We now prove that r' satisfies the constraints, which will contradict $t_n|_u \neq t_n|_{u'}$. Let $N = \{2^k \mid k = 0, \ldots, j_0\} \cup \{2^{j_0}1\}$ (see Fig 5.1). Intuitively, if a constraint is disturbed by the substitution, *ie* if a constraint is unsatisfied in t'_n, it involves necessarily at least one node of N. Let $v, w \in \mathrm{Dom}(t'_n)$, $v \neq w$. We consider three cases (the others are symmetric):

- if $v, w \notin N$, then necessarily any constraint between v and w is still satisfied, because the subtrees rooted at v and w have not changed;
- suppose that $v \in N$ and $v = 2^k$, for some $k \leq j_0$. We prove that necessarily, $t'_n|_v \neq t'_n|_w$. Indeed, the root of $t'_n|_v$ is necessarily labeled f. If the root of

$t'_n|_w$ is labeled f, then either $t'_n|_v$ is a strict subtree of $t'_n|_w$ or $t'_n|_w$ is a strict subtree of $t'_n|_v$. Otherwise the root of $t'_n|_w$ is labeled by g, h or a, so we obviously have $t'_n|_v \neq t'_n|_w$.

Hence, if $\mathsf{lab}^{r'}(v) \neq_A \mathsf{lab}^{r'}(w)$, the constraints is satisfied. We can easily prove that $\mathsf{lab}^{r'}(v) =_A \mathsf{lab}^{r'}(w)$ is impossible.

- suppose that $v \in N$ and $v = 2^{j_0}1$. The root of $t'_n|_v$ is necessarily labeled g. There are two cases:
  - if $\mathsf{lab}^{r'}(v) =_A \mathsf{lab}^{r'}(w)$, we can prove a contradiction. Indeed:
    If $w = 2^{j_0}1b'w'$, for some $w' \in \mathrm{Dom}(\alpha_{j_0})$, we have $\mathsf{lab}^{r}(v) =_A \mathsf{lab}^{r}(2^{i_0}1b'w')$, and $t_n|_{2^{i_0}1b'w'} = t_n|_v$, which is impossible since the root of $t_n|_{2^{i_0}1bw'}$ is labeled h or a, and the root of $t_n|_v$ is labeled g.
    If $w = 2^k$, for some $k \in \{0, \ldots, n\}$, then we have $t_n|_w = t_n|_v$, which is also impossible for similar reasons.
    If $w = 2^k1$, for some $k \in \{0, \ldots, n-1\}$, then $k \neq j_0$, and necessarily we have $t_n|_w = t_n|_v$. It is impossible since it implies that $t_n|_w = g(\alpha_k, \alpha_k)$ and $t_n|_v = g(\alpha_{j_0}, \alpha_{j_0})$, which contradicts $|\alpha_k| \neq |\alpha_{j_0}|$.
    Suppose that $w = 2^j1cw'$, for some $j \neq j_0$, $c \in \{1, 2\}$, and $w' \in \mathrm{Dom}(\alpha_j)$. Hence we have $\mathsf{lab}^{r}(v) =_A \mathsf{lab}^{r}(w)$, and $t_n|_v = t_n|_w$, which contradicts that their respective roots are labeled by different labels.
  - if $\mathsf{lab}^{r}(v) \neq_A \mathsf{lab}^{r}(w)$, then the constraint is satisfied, i.e. $t'_n|_v \neq t'_n|_w$. Indeed, suppose that $t'_n|_v = t'_n|_w$. Since the root of $t'_n|_v$ is labeled g, w is necessarily equal to 2^k1, for some $k \neq j_0$. Hence $t_n|_w$ is of the form $g(\alpha_{i_0}, \alpha_{j_0}) = t'_n|_v$, which contradicts $t_n \in L$, since $|\alpha_{i_0}| \neq |\alpha_{j_0}|$.

Hence r' is a successful run of A on t'_n, which contradicts $t'_n \notin L$. □

The complement of the tree language L used as a counter-example in the proof of Proposition 5.3.3 is definable by a negative TAGED. This means in particular that the class of languages definable by negative TAGEDs is not closed by complement. One could also use a symmetric counter-example language L' to prove that the class of languages definable by positive TAGEDs is not closed by complement. Instead of requiring an equality between the children of nodes labeled g, it would suffice to require a disequality.

5.3.2 Universality is undecidable

The universality problem is as follows: given a TAGED A over a ranked alphabet Σ, does A accept all trees over Σ?

Proposition 5.3.4 *Testing universality of TAGEDs is undecidable (even over a fixed alphabet).*

Proof. We adapt the proof of (Mon81) of emptiness undecidability for classical tree automata with equality constraints. We start from an instance of the Post Correspondence Problem (PCP). We encode the set of solutions of PCP as a tree language whose complement is easily definable by a TAGED. Hence, the complement is universal iff PCP has no solution.
The Post Correspondence Problem (PCP) is defined as follows: an instance of PCP is given by a finite alphabet Σ and $2m$ words $u_1, \ldots, u_m, v_1, \ldots, v_m$ over

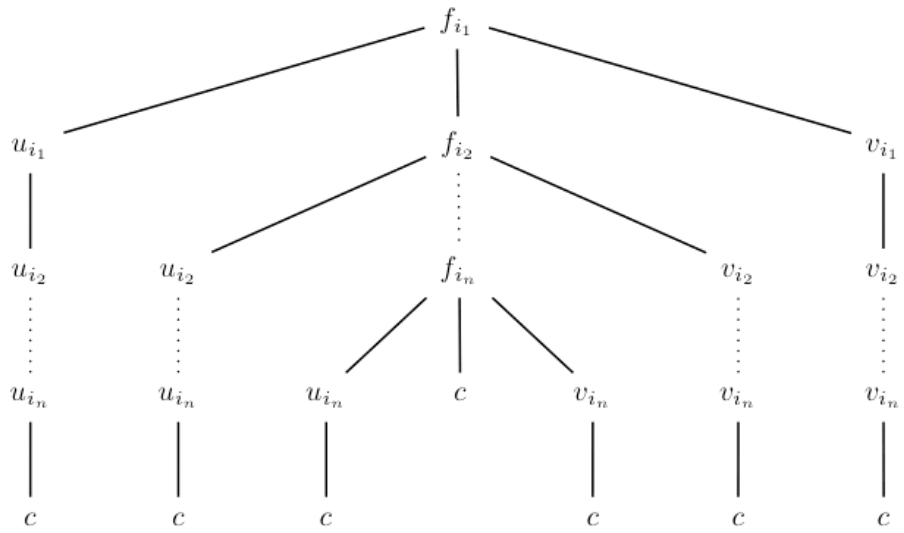


Figure 5.2: representation of a solution of PCP

Σ. A solution to this problem is a finite sequence of indices $i_1, \ldots, i_n$ such that $u_{i_1} \ldots u_{i_n} = v_{i_1} \ldots v_{i_n}$.
Let Σ be an alphabet, and $\mathcal{I} = \{u_1, \ldots, u_m, v_1, \ldots, v_m\}$ be an instance of PCP. We denote by $\Sigma \cup \{f_1, \ldots, f_m, c\}$ the ranked alphabet obtained by extending Σ with fresh ternary function symbols f_i and a constant 0. Symbols from Σ are viewed as unary function symbols.
Let $i_1, \ldots, i_n$ be a solution of $\mathcal{I}$. This solution can be represented as a tree over $\Sigma \cup \{f, c\}$, as in Figure 5.2. For all $1 \leq j \leq m$, the notation $u_j(\circ)$ stands for the context $u_{j,1}(u_{j,2}(\ldots u_{j,k}(\circ))\ldots)$, where $u_{j,1}, \ldots, u_{j,k}$ are symbols from Σ and $u_j = u_{j,1} \ldots u_{j,k}$.
We let S be set of encodings of candidate solutions of $\mathcal{I}$, i.e. S is the set of trees defined by the following grammar:

$$t ::= f_i(t_u, t, t_v) \mid c \quad t_u ::= u_i[t_u] \mid c \quad t_v ::= v_j[t_v] \mid c \quad i, j = 1, \ldots, m$$

The set S can easily be defined by a tree automaton. We can define a TAGED A which checks whether an encoding of a candidate solution is non-valid (i.e. is not a solution of $\mathcal{I}$). In other words, $\mathcal{L}(A) \cap S$ is the set of encodings of non-valid solutions. The TAGED A needs to check if one of the following patterns matches some subtree of the input tree (where X is intended to match a whole subtree, _ matches everything, and $\neg X$ matches any subtree different from the subtree that matches X):

$f_i(X, \_, \neg X)$	$\forall i \in \{1, \ldots, m\}$(to be matched at the root only)
$f_i(u_j(X), f_k(X, \_, \_), \_)$	$\forall i, j, k \in \{1, \ldots, m\},\ i \neq j$
$f_i(\_, f_k(\_, \_, X), v_j(X))$	$\forall i, j, k \in \{1, \ldots, m\},\ i \neq j$
$f_i(u_i(X), f_j(\neg X, \_, \_), \_)$	$\forall i, j \in \{1, \ldots, m\}$
$f_i(\_, f_j(\_, \_, \neg X), v_i(X))$	$\forall i, j \in \{1, \ldots, m\}$

Note that one could not replace $f_i(u_j(X), f_k(X, \_, \_), \_)$ (for $i \neq j$) by $f_i(u_j(X), \_, \_)$ since it could match valid solutions (if u_i is a prefix of u_j for

example). Hence, we have $\overline{\mathcal{L}(A)} \cap S \neq \varnothing$ iff $\mathcal{I}$ has a (valid) solution, iff $\mathcal{T}_{ran}(\Sigma) \not\subseteq \mathcal{L}(A) \cup \overline{S}$. Hence, PCP reduces to (non) universality of TAGEDs, since $\mathcal{L}(A) \cup \overline{S}$ can be defined by a TAGED. Indeed since S is regular, also is $\overline{S}$, and $\overline{S}$ is therefore TAGED-definable. The proof follows from the closure under union of TAGED-definable languages (Proposition 5.3.1).
It is known that the variant of PCP where m is fixed and $m \geq 7$ remains undecidable. The alphabet Σ can also be fixed to $\{a, b\}$. Consequently, testing universality of a TAGED over a fixed alphabet is also undecidable. □

5.3.3 On restricting the equality relation

Even if TAGEDs are not determinizable, we can suppose that testing an equality between subtrees can be done using the same state, as stated by the following lemma:

Lemma 5.3.5 *Every TAGED A is equivalent to a TAGED A' (whose size might be exponential in the size of A) such that $=_{A'} \subseteq id_{Q_{A'}}$, where $id_{Q_{A'}}$ is the identity relation on $Q_{A'}$. Moreover, A' can be built in exponential time.*

In order to prove Lemma 5.3.5, we first introduce useful notions.

Path Isomorphism

Let $t \in \mathcal{T}_{ran}(\Sigma)$, and $u, v \in \mathrm{Dom}(t)$ such that $u \prec^t_{ch^*} v$. We denote by $\mathrm{path}_t(u, v)$ the finite sequence of nodes $u_1, \ldots, u_n$ such that $u_1 = u$, $u_n = v$, and for all $i \in \{1, \ldots, n-1\}$, u_{i+1} is a child of u_i. In particular, $\mathrm{path}_t(u, u) = u$. Given two other nodes u', v' such that $u' \prec^t_{ch^*} v'$, we say that $\mathrm{path}_t(u, v)$ is *edge-isomorphic* to $\mathrm{path}_t(u', v')$, if v and v' are reachable from u and v respectively by the same sequence of first-child or second-child edges. More formally, if $\mathrm{path}_t(u, v) = u_1 \ldots u_n$ and $\mathrm{path}_t(u', v') = u'_1 \ldots u'_n$ for some $u_1, \ldots, u_n, u'_1, \ldots, u'_n$, then they are edge-isomorphic if for all $i \in \{1, \ldots, n-1\}$, for all $\alpha \in \{1, 2\}$, $u_i \prec^t_{ch_\alpha} u_{i+1}$ iff $u'_i \prec^t_{ch_\alpha} u'_{i+1}$.

Node Equivalence

Given a tree $t \in \mathcal{L}(A)$ and a run r of A on t which satisfies the equality constraints, we define an equivalence relation $\sim_{t,r}$ on $\mathrm{Dom}(t)$ as follows[1]. The relation $\sim_{t,r}$ is the transitive and reflexive closure of the relation $\leftrightarrow_{t,r}$ defined as follows: for all $u, v \in \mathrm{Dom}(t)$, $u \leftrightarrow_{t,r} v$ if there exist two nodes u', v' above u, v resp. such that the downward path from u' to u is edge-isomorphic to the downward path from v' to v and $\mathsf{lab}^r(u') =_A \mathsf{lab}^r(v')$.
For instance, Fig 5.3 shows a tree where the two subtrees have been evaluated to some state q such that $q =_A q$, and the corresponding equivalence relation (reflexivity is not depicted in the figure but all nodes are equivalent to themselves). The following propositions are widely used in the rest of this chapter:

Proposition 5.3.6 *For all $u, v \in \mathrm{Dom}(t)$, if $u \sim_{t,r} v$, then $t|_u = t|_v$.*

Proof. If $u \leftrightarrow_{t,r} v$, there are u', v' such that $u' \prec^t_{ch^*} v'$ and $\mathrm{path}_t(u', v')$ is edge-isomorphic to $\mathrm{path}_t(u, v)$, and $\mathsf{lab}^r(u') =_A \mathsf{lab}^r(v')$. Hence $t|_{u'} = t|_{v'}$ and therefore $t|_u = t|_v$. By transitivity, we also get $t|_u = t|_v$ whenever $u \sim_{t,r} v$. □

[1] when it is clear from the context, we omit the subscript t, r and write $\sim$

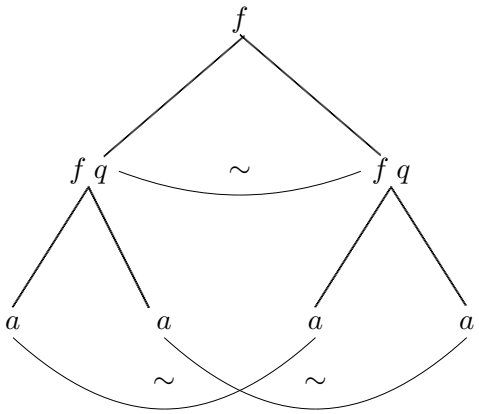


Figure 5.3: Equivalence relation where $q =_A q$

Proposition 5.3.7 *For all $u, v, u', v' \in \mathrm{Dom}(t)$ such that $u \prec^t_{ch^*} u'$ and $v \prec^t_{ch^*} v'$, if $u \sim_{t,r} v$ and $path_t(u, u')$ is edge-isomorphic to $path_t(v, v')$, then $u' \sim_{t,r} v'$.*

Proof. Suppose that $u \leftrightarrow_{t,r} v$. By definition, there are $u'', v'' \in \mathrm{Dom}(t)$ such that $u'' \prec^t_{ch^*} u$ and $v'' \prec^t_{ch^*} v$, $\mathrm{path}_t(u'', u)$ is edge-isomorphic to $\mathrm{path}_t(v'', v)$, and $\mathsf{lab}^r(u'') =_A \mathsf{lab}^r(v'')$. Since $\mathrm{path}_t(u, u')$ is edge-isomorphic to $\mathrm{path}_t(v, v')$, $\mathrm{path}_t(u'', u')$ is also edge-isomorphic to $\mathrm{path}_t(v'', v')$. By definition of $\leftrightarrow_{t,r}$, $u' \leftrightarrow_{t,r} v'$.
Now, by transitivity, it also holds whenever $u \sim_{t,r} v$. □

Proof of Lemma 5.3.5

Proof. Intuitively, we can view an accepting run r of A on a tree t as a DAG structure. Let $U \subseteq \mathrm{Dom}(t)$ such that all subtrees $t|_u$, $u \in U$, have been successfully tested to be equal by A in the run r (i.e. $\forall u, v \in U$, $\mathsf{lab}^r(u) =_A \mathsf{lab}^r(v)$). Let $t_0 = t|_u$, for some $u \in U$. We replace all nodes of U by a single node u_0 which enroots t_0. The parent of any node of U points to u_0. We maximally iterate this construction to get the DAG. Note that this DAG is not maximal sharing[2], since only subtrees which have been successfully tested to be equal are shared. We construct A' s.t. it simulates a run on this DAG, obtained by overlapping the runs on every equal subtrees for which a test has been done.
Formally, we first describe the construction of the automaton A' and then prove its correctness.

Automaton Construction We define $Q' = 2^Q \times \mathcal{C}$ where $\mathcal{C}$ is a set of choices depending on A. Intuitively, when a choice has been made, it enforces the subtrees successfully tested to be equal to run in the same state. This is done by grouping the states which are used at equivalent positions in the tree.

[2]There might be two isomorphic subgraphs occurring at different positions.

A *choice* is a (partial) function $c : \mathrm{dom}(=_A) \to 2^Q$ such that for all $q, q' \in \mathrm{dom}(c)$, $q =_A q'$ implies $c(q) = c(q')$. Note that $\mathrm{dom}(c)$ may be strictly included in $\mathrm{dom}(=_A)$ and that for all $q \in \mathrm{dom}(c)$, $c(q)$ may not be included in $\mathrm{dom}(=_A)$.
Let $c \in \mathcal{C}$. We let $Q_c = \{P \subseteq Q \mid \forall q.(q \in P \cap \mathrm{dom}(=_A) \implies P = c(q))\}$ (it imposes that P must respect the choice c). We let Δ_c be the set of rules defined as follows: (i) for all states $P, P', P'' \in Q_c$, $f(P, P') \to P'' \in \Delta_c$ iff for all $p'' \in P''$, there are $p \in P$ and $p' \in P'$ such that $f(p, p') \to p'' \in \Delta$; (ii) for all $P \in Q_c$, $a \to P \in \Delta_c$ iff for all $p \in P$, we have $a \to p \in \Delta$. The set of final states F_c is defined by $F_c = \{P \in Q_c \mid P \cap F \neq \emptyset\}$. Finally, we define $=_c, \neq_c$ by:

$$\begin{array}{lll} P =_c P' & \text{if} & \exists p \in P,\ \exists p' \in P',\ p =_A p' \quad (\text{ hence } P = P') \\ P \neq_c P' & \text{if} & \exists p \in P, \exists p' \in P', p \neq_A p' \end{array}$$

We let $A_c = (\Sigma, Q_c, F_c, \Delta_c, =_c, \neq_c)$, and let A' be the TAGED accepting $\bigcup_{c \in \mathcal{C}} \mathcal{L}(A_c)$ (we can construct it thanks to Proposition 5.3.1, and its size is exponential in the size of A). We now prove that $\mathcal{L}(A') = \mathcal{L}(A)$.

Correctness Let $q \in Q$. We let $\mathcal{L}(q, A)$ be the set of trees t such that there exists a q-run[3] of A on t which satisfies the constraints. $\mathcal{L}(q, A')$ is defined similarly.

$\mathcal{L}(A') \subseteq \mathcal{L}(A)$. Let $c \in \mathcal{C}$, $P \in Q_c$, and $t \in \mathcal{T}_{ran}(\Sigma)$. such that $t \in \mathcal{L}(P, A_c)$. By definition of Δ_c, we can easily prove by induction on t that if there is a P-run r_c of A_c on t, then for all $p \in P$, there exists a p-run r of A on t. The run r is constructed inductively in a top-down fashion. One first chooses a final state in the root of r_c. Then one chooses two states in the respective two subtrees of the root according to the existence of a rule of Δ, and so on until reaching the leaves of r_c. Moreover, if r_c satisfies the constraints, then r satisfies the constraints too. Indeed, let $u, v \in N_t$ such that $\mathsf{lab}^r(u) =_A \mathsf{lab}^r(v)$. By construction of r, we have $\mathsf{lab}^r(u) \in \mathsf{lab}^{r_c}(u)$ and $\mathsf{lab}^r(v) \in \mathsf{lab}^{r_c}(v)$. By definition of $=_c$, $\mathsf{lab}^{r_c}(u) =_c \mathsf{lab}^{r_c}(v)$ and we get $t|_u = t|_v$. This is done similarly for inequalities.

$\mathcal{L}(A) \subseteq \mathcal{L}(A')$. Let $t \in \mathcal{L}(A)$ and r be a successful run of A on t. We let $\mathrm{states}(u) = \{q \mid \exists v \sim u, q = \mathsf{lab}^r(v)\}$. We let c be defined as follows: for all $p \in \mathrm{dom}(=_A)$, $u \in \mathrm{Dom}(t)$, $c(p) = \mathrm{states}(u)$ if $p \in \mathrm{states}(u)$. Hence, for every pair of equivalent nodes $u \sim v$, if $\mathsf{lab}^r(u), \mathsf{lab}^r(v) \in \mathrm{dom}(c)$, then we have $c(\mathsf{lab}^r(u)) = c(\mathsf{lab}^r(v))$.
We show that there exists a successful run r_c of A_c on t. We define it by: $\forall u \in \mathrm{Dom}(t)$, $\mathsf{lab}^{r'}(u) = \mathrm{states}(u)$. Let us now show that r_c is a run of A_c, and that it satisfies the constraints.

- *for all* $u \in \mathrm{Dom}(t)$, $\mathrm{states}(u) \in Q_c$.

  This holds by definition of $\mathrm{states}(u)$ and Q_c;

- r_c *is a run of* A_c *on* t.

  Let $u, u_1, u_2 \in \mathrm{Dom}(t)$ such that u_1 and u_2 are the sons of u. We show that the transition $f(\mathrm{states}(u_1), \mathrm{states}(u_2)) \to \mathrm{states}(u)$ is in Δ_c. Let $p \in \mathrm{states}(u)$. There is some node $v \in \mathrm{Dom}(t)$ such that $u \sim v$ and $\mathsf{lab}^r(v) = p$. Let v_1, v_2 be the two sons of v respectively (they exist since $t|_u = t|_v$). Let $p_1 = \mathsf{lab}^r(v_1)$ and $p_2 = \mathsf{lab}^r(v_2)$. Hence there is a rule of the form $f(p_1, p_2) \to p$ in Δ. By definition of $\sim$, we have $u_1 \sim v_1$ and $u_2 \sim v_2$,

[3] A q-run is a run whose root is labeled by q

hence $p_1 \in \text{states}(u_1)$ and $p_2 \in \text{states}(u_2)$, which concludes the proof (this goes similarly for leaf nodes).

- *r_c satisfies the equality constraints.*

  Let $u_1, u_2 \in \text{Dom}(t)$ and let $P_1, P_2 \in Q_c$ such that $\mathsf{lab}^{r_c}(u_1) = P_1$ and $\mathsf{lab}^{r_c}(u_2) = P_2$. Suppose that $P_1 =_c P_2$. It means that there are two states $p_1 \in P_1$ and $p_2 \in P_2$ such that $p_1 =_A p_2$. Hence, $c(p_1) = c(p_2) = P_1 = P_2 = \text{states}(u_1) = \text{states}(u_2)$.

  Hence, there are $u_1', u_2' \in \text{Dom}(t)$ (not necessarily different from u_1 and u_2) such that $u_1' \sim u_1$ and $u_2' \sim u_2$, and $\mathsf{lab}^r(u_1') = p_1$, $\mathsf{lab}^r(u_2') = p_2$. By definition of $\sim$, we also get $u_1' \sim u_2'$, since $p_1 =_A p_2$. Finally, as $\sim$ is transitive, we get $u_1 \sim u_2$, and $t|_{u_1} = t|_{u_2}$.

- *r_c satisfies the disequality constraints*

  It is similar to the previous case. Let $u_1, u_2 \in \text{Dom}(t)$. If $\mathsf{lab}^{r_c}(u_1) \neq_c \mathsf{lab}^{r_c}(u_2)$, it means that there are two nodes $u_1' \sim u_1$ and $u_2' \sim u_2$ such that $\mathsf{lab}^r(u_1') \neq_A \mathsf{lab}^r(u_2')$. Since $t|_{u_1} = t|_{u_1'}$ and $t|_{u_2} = t|_{u_2'}$, and $t|_{u_1'} \neq t|_{u_2'}$, we get $t|_{u_1} \neq t|_{u_2}$.

□

5.3.4 A Normal Form for the Runs when $=_A \subseteq id_Q$

If some TAGED satisfies $=_A \subseteq id_Q$, some normal form for its runs can be assumed. The idea is to put the same subruns at nodes labeled by an equality state in a run. This can be done if the run satisfies the equality constraints. This result is divided into two intermediate lemmas needed independently in the rest of the chapter. The full result is given by Lemma 5.3.10.

Lemma 5.3.8 *Let A be a TAGED such that $=_A \subseteq id_Q$. Let $t \in \mathcal{T}_{ran}(\Sigma)$. If there is a run r of A on t which satisfies the equality constraints, then there is a run r' of A on t such that:*

- *r' satisfies the equality constraints;*
- *if r satisfies the disequality constraints, then r' satisfies the disequality constraints;*
- *for all $u, v \in \text{Dom}(r)$, if $\mathsf{lab}^{r'}(u) =_A \mathsf{lab}^{r'}(v)$, then $r'|_u = r'|_v$;*
- $\mathsf{lab}^r(\mathsf{root}^r) = \mathsf{lab}^{r'}(\mathsf{root}^{r'})$.

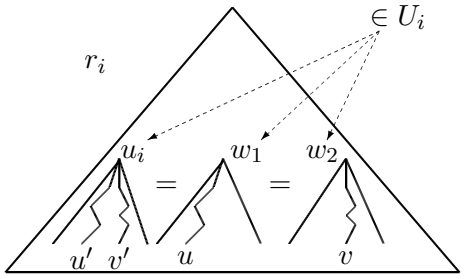
Proof. The construction of r' is done via the following rewriting algorithm:

$q_1, \ldots, q_n \leftarrow \text{dom}(=_A)$ (the order is chosen arbitrarily)
$r_0 \leftarrow r$
for $i = 1$ to n **do**
  $U_i \leftarrow \{u \in \text{Dom}(r_{i-1}) \mid \mathsf{lab}^{r_{i-1}}(u) = q_i\}$
  $u_i \leftarrow$ some node of U_i
  $r_i \leftarrow$ for all $v \in U_i$, replace in r_{i-1} the subrun at node v by $r_{i-1}|_{u_i}$
end for
$r' \leftarrow r_n$
return r'

As we next show, every run r_i satisfies the equality constraints, so that the substitution is well-defined, since all nodes of U_i are disjoint.
We prove the following invariant (called $\mathcal{I}$):
For all $i \in \{0, \ldots, n\}$, r_i is a run of A on t which satisfies the equality constraints and such that for all j such that $1 \leq j \leq i$, and all $u, v \in \mathrm{Dom}(r_i)$, if $\mathsf{lab}^{r_i}(u) = \mathsf{lab}^{r_i}(v) = q_j$, then $r_i|_u = r_i|_v$.
It is clearly true at rank 0 since $r_0 = r$. Let $i > 0$ and suppose that it holds at rank $i-1$. Since r_{i-1} satisfies the equality constraints, for all $v \in U_i$, $t|_{u_i} = t|_v$, hence $r_{i-1}|_{u_i}$ is also a run of A on $t|_v$, so that r_i is still a run of A on t. Since r_{i-1} satisfies the equality constraints, it basically maps every state q of $\mathrm{dom}(=_A)$ to at most one tree t_q (which is a subtree of t). The substitution preserves this property (no new mappings are created). Hence, since $=_A \subseteq id_Q$, the equality constraints are still satisfied in r_i. Finally, let $u, v \in \mathrm{Dom}(r_i)$ such that $\mathsf{lab}^{r_i}(u) = \mathsf{lab}^{r_i}(v) = q_j$, for some $j \in \{1, \ldots, i\}$. If $i = j$, then by definition of r_i, we have $r_i|_u = r_i|_v$. If $j < i$, then we consider several cases (it is not exhaustive as other cases are symmetric):

- u is above (at least) one node of U_i. Hence v cannot be below U_i (otherwise $\|t|_u\| > \|t|_v\|$, since the equality constraints are satisfied). By induction hypothesis, $r_{i-1}|_u = r_{i-1}|_v$. Hence their respective positions labeled q_i are isomorphic, so that the substitution is made at isomorphic positions, and we get $r_i|_u = r_i|_v$;

- u is below some $w_1 \in U_i$ and v is below some element $w_2 \in U_i$. Since $r_i|_{w_1} = r_i|_{w_2} = r_i|_{u_i}$, we can define u' the nodes below u_i such that $\mathrm{path}_t(w_1, u)$ is edge-isomorphic to $\mathrm{path}_t(u_i, u')$. Similarly, we can define v' such that $\mathrm{path}_t(w_2, v)$ is edge-isomorphic to $\mathrm{path}_t(u_i, v')$. This situation is depicted below:



  Hence $r_i|_{u'} = r_i|_u$ and $r_i|_{v'} = r_i|_v$. Therefore u' and v' are labeled q_j in r_i and r_{i-1}. Hence by induction hypothesis, $r_{i-1}|_{u'} = r_{i-1}|_{v'}$, and by definition of the substitution, $r_{i-1}|_{u'} = r_i|_{u'}$ and $r_{i-1}|_{v'} = r_i|_{v'}$. Consequently $r_i|_{u'} = r_i|_{v'}$, and we get $r_i|_u = r_i|_v$;

- u is below some $w \in U_i$ but v is incomparable to any node of U_i. In this case the argument is similar the latter case. We can define u' the node below u_i such that $\mathrm{path}_t(u_i, u')$ is edge-isomorphic to $\mathrm{path}_t(w, u)$, and get $r_i|_u = r_i|_v$ since by induction hypothesis, $r_{i-1}|_{u'} = r_{i-1}|_v$;

- v is below some $w \in U_i$ but u is incomparable to any node of U_i. This case is symmetric to the latter;

- u and v are both incomparable to any node of U_i. In this case the subruns rooted at u and v remain unchanged by the substitution, *ie* $r_{i-1}|_u = r_i|_u$ and $r_{i-1}|_v = r_i|_v$. Hence by induction hypothesis, we get $r_i|_u = r_i|_v$.

□

Lemma 5.3.9 *Let A be a TAGED such that $=_A \subseteq id_Q$. Let $t \in \mathcal{T}_{ran}(\Sigma)$ and r a run of A on t which satisfies the equality constraints and such that for all $u, v \in \mathrm{Dom}(r)$, if $\mathsf{lab}^r(u) =_A \mathsf{lab}^r(v)$, then $r|_u = r|_v$. The following holds:*

$$\text{for all } u, v \in \mathrm{Dom}(r),\ u \sim_{t,r} v \implies r|_u = r|_v$$

Proof. By definition of $\leftrightarrow_{t,r}$, for all nodes u, v such that $u \leftrightarrow_{t,r} v$, there are u' and v' above u and v respectively such that $\mathrm{path}_t(u', u)$ is edge-isomorphic to $\mathrm{path}_t(v', v)$, and $\mathsf{lab}^r(u') =_A \mathsf{lab}^r(v')$. By hypothesis, $r|_{u'} = r|_{v'}$, from which we deduce $r|_u = r|_v$. Since $\sim_{t,r}$ is the reflexive and transitive closure of $\leftrightarrow_{t,r}$, by transitivity, we also get $r|_u = r|_v$ for all nodes u, v such that $u \sim_{t,r} v$. □

As a consequence of Lemma 5.3.8 and Lemma 5.3.9, we get the following:

Lemma 5.3.10 *Let A be a TAGED such that $=_A \subseteq id_Q$. Let $t \in \mathcal{T}_{ran}(\Sigma)$. If there is a run r of A on t which satisfies the equality constraints, then there is a run r' of A on t such that:*

- *r' satisfies the equality constraints;*
- *if r satisfies the disequality constraints, then r' satisfies the disequality constraints;*
- *for all $u, v \in \mathrm{Dom}(r)$, if $u \sim_{t,r'} v$, then $r'|_u = r'|_v$;*
- $\mathsf{lab}^r(\mathsf{root}^r) = \mathsf{lab}^{r'}(\mathsf{root}^{r'})$.

5.4 Positive and Negative TAGEDs

In this section we prove decidability of emptiness of positive and negative TAGEDs respectively. For positive TAGEDs, it uses the fact that we can assume that $=_A \subseteq id_Q$, and the classical reachability method for tree automata. For negative TAGEDs, we reduce the problem to testing satisfiability of set constraints.

5.4.1 Emptiness of Positive TAGEDs

Theorem 5.4.1 *Deciding emptiness of a positive TAGED A is EXPTIME-complete, and in linear time if $=_A \subseteq id_Q$. Moreover, if $\mathcal{L}(A) \neq \varnothing$, then a tree $t \in \mathcal{L}(A)$ is computable in EXPTIME, and in linear time if $=_A \subseteq id_Q$.*

Proof. **upper bound** Let A be a positive TAGED such that $=_A \subseteq id_Q$ (otherwise we transform A modulo an exponential blow-up, thanks to Lemma 5.3.5). Let A^- be its associated tree automaton (i.e. A without the constraints). We have $\mathcal{L}(A) \subseteq \mathcal{L}(A^-)$.
Then it suffices to apply a slightly modified version of the classical reachability method used to test emptiness of a tree automaton (CDG*07). In particular, we can make this procedure associate with any state q a unique tree t_q. When a new

state is reached, it can possibly activate many rules $f(q_1, q_2) \to q$ whose rhs are the same state q. The algorithm has to make a choice between this rules in order to associate with q a unique tree $t_q = f(t_{q_1}, t_{q_2})$. This choice can be done for instance by giving an identifier to each rule and choosing the rule with the least identifier.
We prove that $\mathcal{L}(A) = \varnothing$ iff $\mathcal{L}(A^-) = \varnothing$. If $\mathcal{L}(A^-)$ is empty, then $\mathcal{L}(A)$ is also empty. If $\mathcal{L}(A^-)$ is non-empty, then the procedure described previously outputs a tree t and a run r which obviously satisfies the equality constraints, since any state q is mapped to unique tree t_q (if q is reachable).
lower bound We reduce the problem of testing emptiness of the intersection of the languages defined by n tree automata $A_1, \ldots, A_n$, which is known to be EXPTIME-complete (CDG*07). We assume that their sets of states are pairwise disjoint ($F_i \cap F_j = \varnothing$ whenever $i \neq j$), and for all $i = 1, \ldots, n$, A_i has exactly one final state q_{f_i}, and q_{f_i} does not occur in lhs of rules of A_i (otherwise we slightly modify A_i, modulo a factor 2 in the size of A_i). We let $L = \{f(t_1, \ldots, t_n) \mid f \in \Sigma, \forall i,\ t_i \in \mathcal{L}(A_i), \forall i, j,\ t_i = t_j\}$. It is clear that L is empty iff $\mathcal{L}(A_1) \cap \ldots \mathcal{L}(A_n)$ is empty. It is not difficult to construct a TAGED A (with $\|A\| = O(\sum_i \|A_i\|)$), such that $L = \mathcal{L}(A)$: it suffices to take the union of $A_1, \ldots, A_n$ and to add the rule $f(q_{f_1}, \ldots, q_{f_n}) \to q_f$, where q_f is a fresh final state of A. Then we add the following equality constraints: $\forall i, j,\ q_{f_i} =_A q_{f_j}$. □

5.4.2 Pumping Lemma for Positive TAGEDs

If $=_A \subseteq id_Q$, then in a successful run, we can assume that the subruns rooted at states q such that $q =_A q$ are the same, by Lemma 5.3.10. Hence, we can introduce a pumping technique for positive TAGEDs satisfying this property. The idea is to pump similarly, in parallel, below all states q such that $q =_A q$, while keeping the equality constraints satisfied.
Let $t \in \mathcal{T}_{ran}(\Sigma)$ and r a successful run of A on t. Remind that $\sim_{t,r}$ is defined in Section 5.3.3. Since $=_A \subseteq id_Q$, by Lemma 5.3.10, we can assume that for all nodes $u, v \in \mathrm{Dom}(t)$ such that $u \sim_{t,r} v$, $r|_u = r|_v$. Suppose that there are two nodes $u_1, v_1 \in \mathrm{Dom}(t)$ such that $u_1 \prec^t_{ch^+} v_1$ and there is a state $q \in Q$ such that $\mathsf{lab}^r(u_1) = \mathsf{lab}^r(v_1) = q$. In general, we cannot pump the loop u_1, v_1 because after pumping, an equality constraint might be unsatisfied (in particular, if u is below a state from $\mathrm{dom}(=_A)$ which occurs twice in the run). Instead of pumping only below u_1, we pump in parallel below all nodes equivalent to u_1 by $\sim_{t,r}$. This can be done since for all node u such that $u_1 \sim_{t,r} u$, we have $r|_u = r|_{u_1}$. Now, let $u_2, v_2 \in \mathrm{Dom}(t)$ such that $u_2 \sim_{t,r} u_1$ and v_2 is the node below u_2 such that the downward path from u_1 to v_1 is edge-isomorphic to the downward path from u_2 to v_2 (it exists since $t|_{u_1} = t|_{u_2}$, by Prop 5.3.6). This situation is depicted in Fig. 5.4 where all nodes equivalent to u_1 are also represented. Since $r|_{u_1} = r|_{u_2}$, we have $\mathsf{lab}^r(v_2) = \mathsf{lab}^r(u_2) = q$, so that we can also pump the loop u_2, v_2. More generally, we can pump in parallel below all nodes equivalent to u_1 while keeping the equality constraints satisfied, as shown in Fig. 5.4.
Formally, the pumping technique is described by the following lemma:

Lemma 5.4.2 (Pumping Lemma for Positive TAGEDs) *Let $t \in \mathcal{L}(A)$, and r a successful run of A on t such that $\forall u, v \in \mathrm{Dom}(t)$, if $\mathsf{lab}^r(u) =_A \mathsf{lab}^r(v)$, then $r|_u = r|_v$. Let $\{u_1, \ldots, u_n\} \subseteq \mathrm{Dom}(t)$ be an $\sim_{t,r}$-equivalence class. Suppose that there are some state $q \in Q$ and some node v_1 such that $u_1 \prec^t_{ch^+} v_1$ and $\mathsf{lab}^r(u_1) = \mathsf{lab}^r(v_1) = q$. Let $v_2, \ldots, v_n$ such that for all $i \in \{2, \ldots, n\}$, $path_t(u_i, v_i)$ is*

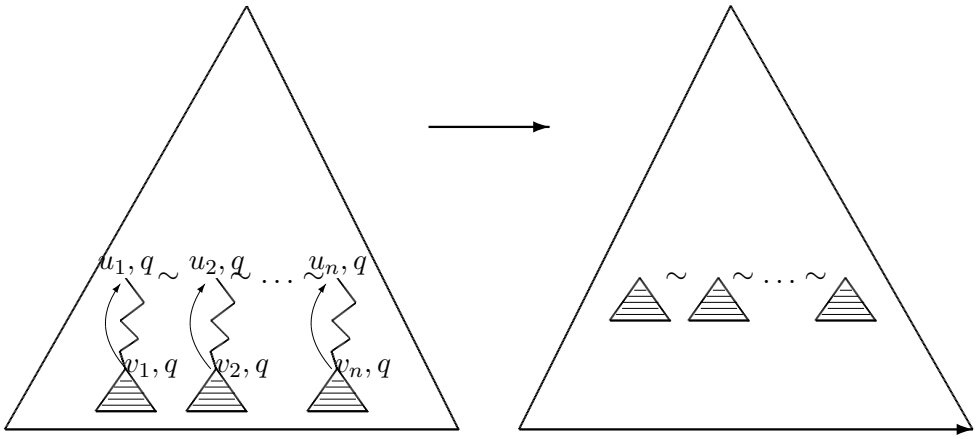


Figure 5.4: Parallel pumping of state q

edge-isomorphic to path$_t(u_1, v_1)$. Let C (resp. R) be the n-ary context over Σ (resp. Q) such that $t = C[t|_{u_1}, \ldots, t|_{u_n}]$ (resp. $r = R[r|_{u_1}, \ldots, r|_{u_n}]$).
Then $R[r|_{v_1}, \ldots, r|_{v_n}]$ is a successful run of A on $C[t|_{v_1}, \ldots, t|_{v_n}]$.

Proof. Fig. 5.4 illustrates this pumping. Let $r' = R[r|_{v_1}, \ldots, r|_{v_n}]$ and $t' = C[t|_{v_1}, \ldots, t|_{v_n}]$. It is clear that r' is a run on t' whose root is labeled by a final state. It remains to show that it satisfies the equality constraints. Let $u, v \in \mathrm{Dom}(t')$ such that $\mathsf{lab}^{r'}(u) =_A \mathsf{lab}^{r'}(v)$. If neither v nor u is above one of the v_is, we have $t|_u = t'|_u$, $t|_v = t'_v$, and $\mathsf{lab}^r(u) =_A \mathsf{lab}^r(v)$. Since r satisfies the equality constraints, we also have $t'|_u = t'|_v$.
Suppose now that $u \prec^t_{ch^*} v_i$, for some $i \in \{1, \ldots, n\}$. By definition of $\sim_{t,r}$, we have $u \sim_{t,r} v$. We cannot have $u = v_i$, otherwise it would mean that q is an equality state, which is not possible since it would imply $t|_{u_1} = t|_{v_1}$, but $t|_{v_1}$ is a strict subtree of $t|_{u_1}$. Hence $u \prec^t_{ch^+} v_i$, and $u \prec^t_{ch^+} u_i$ (since we have pumped the loop u_i, v_i, and u is still present in t'). By hypothesis, we have $r|_u = r|_v$. Let $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ (resp. $\{j_1, \ldots, j_{k'}\} \subseteq \{1, \ldots, n\}$) be the maximal set of indices such that there is a k-ary context C_1 and the k'-ary context C_2 such that $t|_u = C_1[t|_{u_{i_1}}, \ldots, t|_{u_{i_k}}]$ and $t|_v = C_2[t|_{u_{j_1}}, \ldots, t|_{u_{j_{k'}}}]$. By definition of $\sim_{t,r}$, since $u \sim_{t,r} v$, we have $k = k'$, $C_1 = C_2$, and $t|_{u_{i_l}} = t|_{u_{j_l}}$, for all $l \in \{1, \ldots, k\}$. By definition of the pumping, we have $t'|_u = C_1[t|_{v_{i_1}}, \ldots, t|_{v_{i_k}}]$ and $t'|_v = C_2[t|_{v_{j_1}}, \ldots, t|_{v_{j_k}}]$. Hence we get $t'_u = t'_v$, which ends up the proof. □

As a consequence, we have:

Corollary 5.4.3 *If $\mathcal{L}(A) \neq \varnothing$, there is a tree $t \in \mathcal{L}(A)$ whose height is bounded by $|Q|$.*

We can also use this technique to prove that:

Lemma 5.4.4 *If there is a tree $t \in \mathcal{L}(A)$ whose height is strictly greater than $|Q|$, then $\mathcal{L}(A)$ is infinite.*

Proof. If there is a loop in a run involving a node u and a node v such that $u \prec^t_{ch^+} v$, we can iterate this loop in parallel below all nodes equivalent to v, while keeping the constraints satisfied. □

As a consequence of this lemma, we get the following theorem:

Theorem 5.4.5 *Let A be a positive TAGED. It is decidable whether $\mathcal{L}(A)$ is infinite or not, in $O(\|A\|.|Q|^2)$ if $=_A \subseteq id_Q$, and in EXPTIME otherwise.*

Proof. If $=_A \subseteq id_Q$, we only have to test if there is a tree in $\mathcal{L}(A)$ whose height is strictly greater than $|Q|$. This can be done by adding a counter c to A, bounded by $|Q|+1$. Intuitively, if a tree t evaluates to (q, c), it means that the height of t is equal to c. A special counter value denoted by $c_{>|Q|}$ is reached when the height of the tree is strictly greater than $|Q|$. Let A' be the automaton we obtain. We let $(q, c) =_{A'} (q, c')$ iff $q =_A q$ and $c = c'$, so that we have $=_{A'} \subseteq id_{Q'}$. Emptiness is tested in linear time, as in the proof of Theorem 5.4.1.
If A does not satisfies $=_A \subseteq id_Q$, we first have to transform it, modulo an exponential blow-up, thanks to Lemma 5.3.5. □

5.4.3 Emptiness of Negative TAGEDs

We now prove decidability of emptiness of negative TAGEDs, by reduction to positive and negative set constraints (PNSC for short). Set expressions are built over set variables, function symbols, and Boolean operations. Set constraints are either positive, $e_1 \subseteq e_2$, or negative, $e_1 \not\subseteq e_2$, where e_1, e_2 are set expressions. Set expressions are interpreted in the Herbrand structure (where set variables are therefore interpreted by sets of terms) while set constraints are interpreted by Booleans 0,1. Testing the existence of a solution of a system of set constraints has been proved to be decidable in several papers (CP94, AKW95, Ste94, GTT94). In particular, it is known to be NEXPTIME-complete. We do not formally define set constraints and refer the reader to (CP94, AKW95, Ste94, GTT94).
Consider for instance the constraint $f(X, X) \subseteq X$. It has a unique solution which is the empty set. Consider now $X \subseteq f(X, X) \cup a$, where a is a constant symbol. Every set of terms over $\{f, a\}$ closed by the subterm relation is a solution of this equation. More generally, we can encode the emptiness problem of tree automata as a system of set constraints. Let $A = (\Sigma, Q, F, \Delta)$ be a tree automaton. Wlog, we assume that every state $q \in Q$ occurs in the rhs of some rule. We associate with A the system S_A defined by:

$$(S_A) \quad \begin{cases} X_q \subseteq \bigcup_{f(q_1,q_2)\to q\in\Delta} f(X_{q_1}, X_{q_2}) \cup \bigcup_{a\to q\in\Delta} a & \text{for all } q \in Q \\ \bigcup_{q\in F} X_q \not\subseteq \varnothing \end{cases}$$

Proposition 5.4.6 *$\mathcal{L}(A)$ is non-empty iff S_A has a solution.*

Proof. We sketch correctness of the system S_A. Suppose that S_A has a solution given by a set of trees T_q, for all $q \in Q$. Let $q \in Q$ such that there is $t \in T_q$, $t \in \mathcal{L}(A)$. We construct a run on t inductively.

- if $t = a \in \Sigma$, then we take the successful run reduced to the leaf q;
- if $t = f(t_1, t_2)$, for some $f \in \Sigma$, $t_1, t_2 \in T_\Sigma$. By definition of S_A, there is a rule $f(q_1, q_2) \to q$ such that $t_1 \in T_{q_1}$ and $t_2 \in T_{q_2}$. By induction hypothesis, there are runs r_1 and r_2 on t_1 and t_2 respectively, such that $\mathsf{lab}^{r_1}(\mathsf{root}^{r_1}) = q_1$ and $\mathsf{lab}^{r_2}(\mathsf{root}^{r_2}) = q_2$. Hence $q(r_1, r_2)$ is a run of A on t.

Since there is $q \in F$ such that $T_q \neq \varnothing$, for all $t \in T_q$, we can construct a successful run r of A on t, so that $t \in \mathcal{L}(A)$.

Conversely, if $\mathcal{L}(A) \neq \varnothing$, there is a tree $t \in \mathcal{L}(A)$ and a successful run r of A on t. For all $q \in Q$, we let $T_q = \{t|_u \mid u \in \mathrm{Dom}(t), \mathsf{lab}^r(u) = q\}$. The set $\{T_q \mid q \in Q\}$ is a solution of S_A. □

Let $(A, \neq_A)$ be a negative TAGED, and consider the system S'_A consisting in S_A extended with the constraints $X_q \cap X_p = \varnothing$, for all $q, p \in Q$ such that $q \neq_A p$. It is easy to extend the proof of Proposition 5.4.6 to prove that $\mathcal{L}(A, \neq_A) \neq \varnothing$ iff S'_A has a solution. Since deciding existence of a solution of a system of PNSC is in NEXPTIME, we get:

Theorem 5.4.7 *Emptiness of negative TAGEDs is decidable in NEXPTIME.*

5.5 Vertically Bounded TAGEDs

In this section, we define a subclass of TAGEDs, called *vertically bounded TAGEDs*, which mixes equality and disequality constraints. More precisely, vb-TAGEDs allow an unbounded number of positive tests, but boundedly many negative tests along root-to-leaf paths, *ie* branches. While this class subsumes positive TAGEDs, the upper-bound for testing emptiness is bigger than the bound obtained in Section 5.4. Formally:

Definition 5.5.1 *Let Σ be a ranked alphabet. A vertically bounded TAGED over Σ is a pair (A, k) where A is a TAGED over Σ, and $k \in \mathbb{N}$. A run r of (A, k) on a tree $t \in \mathcal{T}_{ran}(\Sigma)$ is a run of A on t. It is successful if r is successful for A and the number of states from dom$(\neq_A)$ occurring along any root-to-leaf path is bounded by k:*

$$\textit{for all root-to-leaf path } u_1 \prec_{ch}^t \ldots \prec_{ch}^t u_n,\ |\{i \mid \mathsf{lab}^r(u_i) \in \textit{dom}(\neq_A)\}| \leq k$$

The notion of recognized language $\mathcal{L}(A, k)$ is defined similarly as for TAGEDs. Another way to define them would be to suppose the existence of a bound k, without given it explicitly ("a TAGED A is bounded if there is some non-negative integer k such that ..."). This would result in a more semantical definition of bounded TAGEDs, since we do not know how to decide if for some input TAGED, there exists an equivalent bounded TAGED. Moreover, deciding emptiness of TAGEDs is still open, and is as much difficult as testing whether a TAGED A can be bounded by some $k \in \mathbb{N}$:

Proposition 5.5.1 *The problem of deciding emptiness of a TAGED is polynomial-time reducible to the following problem: given a TAGED A, and $k \in \mathbb{N}$, does $\mathcal{L}(A) = \mathcal{L}(A, k)$ hold?*

Proof. Let Σ be an alphabet which contains a binary symbol f. Let $A = (\Sigma, Q, F, \Delta, =_A, \neq_A)$ be a TAGED. The language $\{f(t, t') \mid t \in \mathcal{L}(A) \text{ and } t' \in \mathcal{T}_{ran}(\Sigma) - t\}$ is recognizable by the TAGED $A' = (\Sigma, Q', F', \Delta', =_A, \neq_{A'})$, where $Q' = Q \cup \{q, q', q_f, q_f^A\}$, for some new states $q, q', q_f, q_f^A \notin Q$, $F' = \{q_f\}$, $\neq_{A'} = \neq_A \cup \{(q_f^A, q'),\ (q', q_f^A)\}$ and

$$\begin{aligned}\Delta' \;=\; & \Delta \cup \{g(q,q) \to q \mid g \in \Sigma\} \cup \{a \to q \mid a \in \Sigma\} \\ & \cup \{g(q,q) \to q' \mid g \in \Sigma\} \cup \{a \to q' \mid a \in \Sigma\} \\ & \cup \{g(q_1,q_2) \to q_f^A \mid \exists p \in F,\ g(q_1,q_2) \to p \in \Delta\} \\ & \cup \{a \to q_f^A \mid \exists p \in F,\ a \to p \in \Delta\} \cup \{f(q_f^A, q') \to q_f\}\end{aligned}$$

.

Note that $\|A'\| = O(\|A\|)$. We prove that $L(A) = \varnothing$ iff $L(A') = L(A', 0)$. If $L(A) = \varnothing$, then $L(A') = \varnothing = L(A', 0)$. Conversely $L(A', 0)$ is necessarily equal to $\varnothing$ since A' needs to fire the rule $f(q_f^A, q') \to q_f$ in order to go in state q_f. Therefore at least one disequality test is done in every successful run. Hence $L(A') = \varnothing$, which implies $L(A) = \varnothing$ by definition. □

We now come to the main result of this section:

Theorem 5.5.2 *Let (A, k) be a vbTAGED. Testing emptiness of (A, k) can be done in 2NEXPTIME. It is in NEXPTIME if*

- $=_A \subseteq id_Q$ *and*
- $k \leq |Q|$ *(or k is represented by a unary encoding)*

The proof is based on a technical pumping method, sketched in the next paragraph. All the following subsections are devoted to the formal proof of Theorem 5.5.2.

Sketch of Proof of Theorem 5.5.2 We first transform A so that it satisfies $=_A \subseteq id_Q$, thanks to Lemma 5.3.5 (modulo an exponential blow-up). Let $t \in \mathcal{T}_{ran}(\Sigma)$, and r a run of A on it which satisfies the equality constraints (but not necessarily the disequality constraints), and such that its root is labeled by a final state. We introduce sufficient conditions on t and r (which can be verified in polynomial-time, in $\|t\|$, $\|r\|$ and $\|A\|$) to be able to repair the unsatisfied inequality constraints in t in finitely many rewriting steps. This rewriting can be done while keeping the equality constraints satisfied. In particular, since $=_A \subseteq id_Q$, we can assume that for all $u, v \in \mathrm{Dom}(t)$ such that $u \sim_{t,r} v$, $r|_u = r|_v$ (by Lemma 5.3.10). Hence, we can use a "parallel" pumping technique in the spirit of the pumping technique for positive TAGED. The pumping is a bit different however: indeed, if t and r satisfies the sufficient conditions, we increase the size of some contexts of t and r, called *elementary contexts*, in order to repair all the unsatisfied inequality constraints. The repairing process is inductive. In particular, we introduce a notion of frontier below which all inequality constraints have been repaired. The process stops when the frontier reach the top of the tree (and in this case the repaired tree is in the language). If a tree and a run satisfy the sufficient conditions, they are also said to be *repairable*. We first prove a lemma which from a repairable tree and run, and a frontier F, creates a new repairable tree and run, and a new frontier which is strictly contained in F. Then we prove a lemma stating that if $\mathcal{L}(A, k) \neq \varnothing$, there is a repairable tree t and run r such that the height of t is smaller than $2(k+|Q|)|Q|$ (and by $(k+2^{|Q|})2^{|Q|+1}$ if $=_A \not\subseteq id_Q$). Hence it suffices to guess such a repairable tree and run to get decidability of emptiness. The characterization of the emptiness problem by means of repairable trees and runs is formalized through a predicate $\mathbf{P}$ defined in the next section, and this characterization is stated by Theorem 5.5.4. Sections 5.5.2 and 5.5.3 are devoted to the proof of Theorem 5.5.4. □

5.5.1 A Characterization of the Non-Emptiness Problem

In this subsection, we define several notions used to prove Theorem 5.5.2. Let (A, k) be a vbTAGED such that $=_A \subseteq id_Q$.

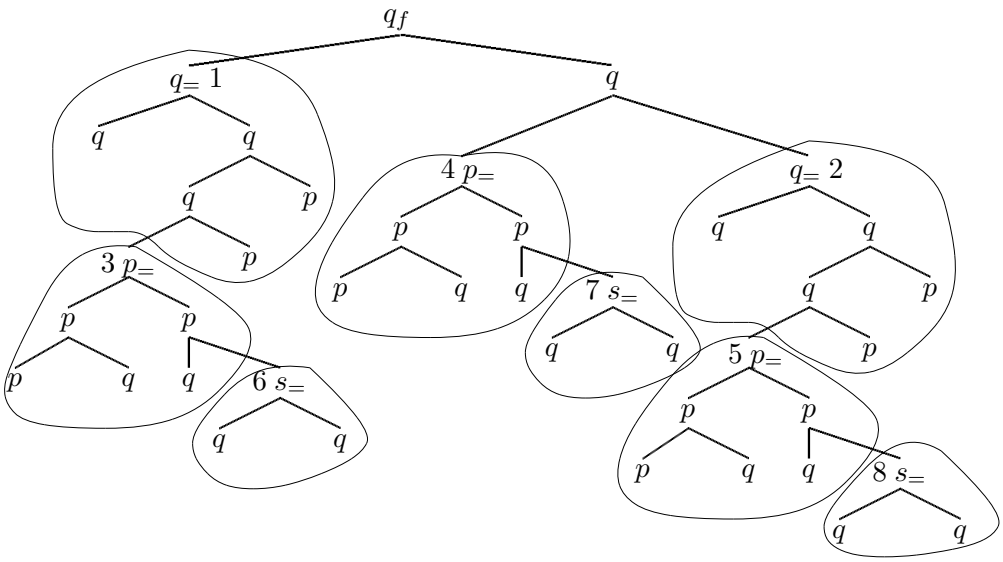


Figure 5.5: Elementary contexts of a run r over $\{q, p, q_f, q_=, p_=, s_=\}$ where $q_= =_A q_=$, $p_= =_A p_=$ and $s_= =_A s_=$. Their root nodes are identified by natural numbers. The set $\mathcal{C}(r)$ is equal to $\{1, 2, 3, 4, 5, 6, 7, 8\}$. The maximal frontier is equal to $\{\{1, 2\},\ \{3, 4, 5\},\ \{6, 7, 8\}\}$.

Elementary Contexts

Given two multi-ary contexts C_1, C_2, we say that C_1 is *included* in C_2 if C_1 occurs in C_2, i.e. if there are some unary context C'_0 and some contexts $C'_1, \ldots, C'_n$ such that n is the arity of C_1, and $C_2 = C'_0[C_1[C'_1, \ldots, C'_n]]$. Let $t \in \mathcal{L}(A)$ and r a successful run of A on t. A context C of r is *elementary* if it is a maximal context (w.r.t. context inclusion) included in r, such that (i) all its nodes (except the root) are labeled in $Q - (\text{dom}(=_A) \cup \text{dom}(\neq_A))$, (ii) the root is labeled in $\text{dom}(=_A) \cup \text{dom}(\neq_A)$, (iii) there is a loop in C, i.e. two descendant nodes of C are labeled by the same state in r. We denote by $\mathcal{C}(r)$ the set of nodes which enroot an elementary context. For all nodes $u \in \mathcal{C}(r)$, we denote by $\text{cxt}_r(u)$ the elementary context over Q rooted at u in r, and by $\text{cxt}_t(u)$ the context of t over Σ rooted at u and edge-isomorphic to $\text{cxt}_r(u)$. Fig. 5.5 represents a run r on some tree t, and its elementary contexts.

Partial Order on Equivalence Classes and Frontiers

Let $t \in \mathcal{L}(A)$ and r a run of A on t which respects the equality constraints. Remind that $\sim_{t,r}$ is defined in Section 5.3.3. For all nodes $u \in \text{Dom}(t)$, we denote by $[u]_{t,r}$ its $\sim_{t,r}$-equivalence class. We define a strict partial order $\prec_{\sim_{t,r}}$ on equivalence classes as follows. For all nodes $u, v \in \text{Dom}(t)$,

$$[u]_{t,r} \ \prec_{\sim_{t,r}} \ [v]_{t,r} \text{ iff } \exists u' \in [u]_{t,r},\ \exists v' \in [v]_{t,r},\ u' \prec^t_{ch^+} v'$$

Lemma 5.5.3 *$\prec_{\sim_{t,r}}$ is a strict partial order on $\sim_{t,r}$-equivalence classes.*

Proof. The proof uses Prop 5.3.6, *ie* for all $u, v \in \text{Dom}(t)$, if $u \sim_{t,r} v$, then $t|_u = t|_v$.

- *irreflexivity*. Suppose that there is $u \in \text{Dom}(t)$ such that $[u]_{t,r} \prec_{\sim_{t,r}} [u]_{t,r}$.

Thus there are $v \in [u]_{t,r}$, and $w \in [u]_{t,r}$ such that $v \prec^t_{ch^+} w$, which contradicts $t|_v = t|_w$;

- *transitivity*. Let $u, v, w \in \mathrm{Dom}(t)$ such that $[u]_{t,r} \prec_{\sim_{t,r}} [v]_{t,r}$ and $[v]_{t,r} \prec_{\sim_{t,r}} [w]_{t,r}$. Thus there are $u' \in [u]_{t,r}$ and $v' \in [v]_{t,r}$ such that $u' \prec^t_{ch^+} v'$, and there are $v'' \in [v]_{t,r}$ and $w'' \in [w]_{t,r}$ such that $v'' \prec^t_{ch^+} w''$. Let $w' \in \mathrm{Dom}(t)$ such that $v' \prec^t_{ch^+} w'$ and $\mathrm{path}_t(v', w')$ is edge-isomorphic to $\mathrm{path}_t(v'', w'')$ (it exists since $t|_{v'} = t|_{v''}$). Since $u' \prec^t_{ch^+} w''$, we get $[u]_{t,r} \prec_{\sim_{t,r}} [w]_{t,r}$.

- *asymmetry*. It is a consequence of irreflexivity and transitivity.

□

We denote by $\preceq_{\sim_{t,r}}$ the reflexive closure of $\prec_{\sim_{t,r}}$. For instance in Fig. 5.5, we have $\{1,2\} \prec_{\sim_{t,r}} \{3,4,5\}$.

Definition 5.5.2 *A set $F \subseteq 2^{\mathrm{Dom}(t)}$ is a frontier if there is $A \subseteq \mathcal{C}(r)$ such that:*

1. $F = \{[a]_{t,r} \mid a \in A\}$
2. *for all $a, a' \in \mathcal{C}(r)$, if $[a]_{t,r} \in F$ and $[a']_{t,r} \prec_{t,r} [a]_{t,r}$, then $[a']_{t,r} \in F$.*

The *maximal frontier* $F_{max}(t, r)$ of t and r is defined by

$$F_{max}(t, r) = \{[u]_{t,r} \mid u \in \mathcal{C}(r)\}$$

In Fig. 5.5, the maximal frontier is $\{\{1,2\},\{3,4,5\}\}$. If for all nodes $u, v \in \mathrm{Dom}(t)$ such that $u \sim_{t,r} v$ one has $r|_u = r|_v$, then all classes of F are subsets of $\mathcal{C}(r)$, *ie* enroot elementary contexts, for all frontier F.
We now introduce a predicate which holds in a tree t, a run r and a frontier F, if r satisfies the equality constraints, but not necessarily all the inequality constraints. However if this predicate holds, all unsatisfied inequality constraint can be repaired by increasing the size of the elementary contexts rooted at the nodes of some class of the frontier, intuitively. This is done by iterating a loop contain in the elementary contexts, and we do it in parallel below all equivalent nodes in order to preserve the equality constraints. We will see in the next two sections that the inequality constraints can be repaired whenever the predicate holds.

Predicate P Let $t \in T_\Sigma$, $r \in T_Q$, $F \subseteq 2^{\mathrm{Dom}(t)}$. We let $\mathbf{P}(t, r, F)$ holds if all the following conditions are satisfied:

1. r is a run of A on t which satisfies the equality constraints, whose root is labeled by a final state, and such that in any $\prec^r_{ch^*}$-ordered chain, there are at most k nodes whose labels in r belong to $\mathrm{dom}(\neq_A)$;
2. for all $u, v \in \mathrm{Dom}(r)$, if $u \sim_{t,r} v$ then $r|_u = r|_v$;
3. F is a frontier;
4. for all $v_1, v_2 \in \mathrm{Dom}(t)$, if $t|_{v_1} = t|_{v_2}$ and $\mathsf{lab}^r(v_1) \neq_A \mathsf{lab}^r(v_2)$, then there are $c \in F$ and $u \in c$ such that $\mathrm{Rep}(t, r, F, v_1, v_2, u)$ holds, *ie*:

   (i) either $v_1 \prec^t_{ch^*} u$ or $v_2 \prec^t_{ch^*} u$;

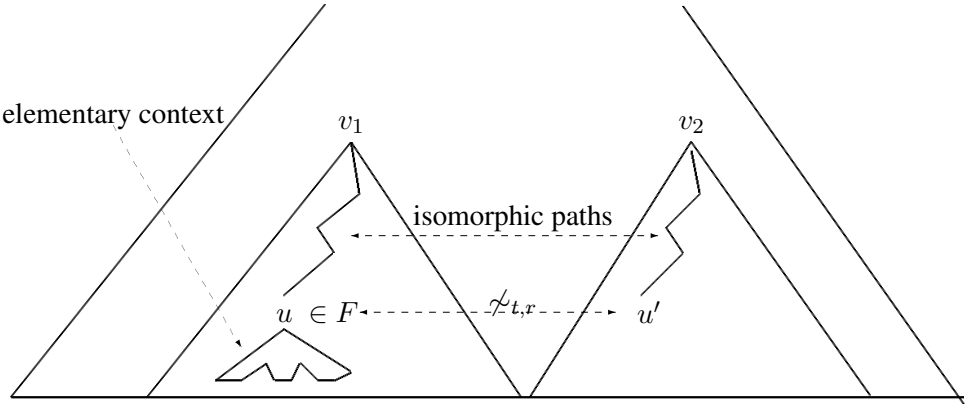


Figure 5.6: A configuration where Rep(t, r, F, v_1, v_2, u) holds.

(ii) if $v_1 \prec^t_{ch^*} u$, then if $u' \in \mathrm{Dom}(t)$ is the node such that $v_2 \prec^t_{ch^*} u'$ and $\mathrm{path}_t(v_1, u)$ is edge-isomorphic to $\mathrm{path}_t(v_2, u')$, then $u \not\sim_{t,r} u'$;

(iii) if $v_2 \prec^t_{ch^*} u$, we define the condition symmetrically as (iii).

(it is illustrated in Fig. 5.6)

The characterization of the non-emptiness problem for vbTAGEDs is given by the following theorem:

Theorem 5.5.4 *Let (A, k) be a vbTAGED, and $B = 2(k + |dom(=_A)|)|Q|$ if $=_A \subseteq id_Q$, and $B = (k + 2^{|Q|})2^{|Q|+1}$ otherwise.*
$\mathcal{L}(A, k)$ is non-empty iff there are a tree t and a run r of A on t such that:

- *the height of t is bounded by B*
- $\mathbf{P}(t,\ r,\ F_{max}(t, r))$ *holds.*

The next two sections are devoted to the proof of Theorem 5.5.4.
The next paragraph shows how to prove Theorem 5.5.2 by using Theorem 5.5.4

Proof of Theorem 5.5.2 Thanks to Theorem 5.5.4, it suffices to guess a tree and a run of heights at most B (and sizes at most 2^{B+1}), such that $\mathbf{P}(t, r, F_{max}(t', r'))$ holds. Moreover, $\mathbf{P}(t, r, F_{max}(t', r'))$ can be verified in PTIME in $\|t\|$, $\|r\|$, and $\|A\|$. If $B = (k + 2^{|Q|})2^{|Q|+1}$, this gives a non-deterministic algorithm doubly exponential in $|Q|$ and $\|k\|$, since $\|k\| = log_2(k)$. However, if $B = 2(k + |\mathrm{dom}(=_A)|)|Q|$ and the encoding of k is unary, or k is polynomial in $|Q|$, the non-deterministic algorithm is simply exponential in $|Q|$ and $\|k\|$.

5.5.2 Proof of the Forth Direction of Theorem 5.5.4

From now on, we fix a vbTAGED (A, k) and suppose that $=_A \subseteq id_Q$. Given an elementary context C in r, we say that C contains a 3-loop if there are a chain of nodes of C which contains at least 3 nodes labeled by the same state.

Lemma 5.5.5 *For all t, r, if $\mathbf{P}(t, r, F_{max}(t, r))$ holds and there is an elementary context of r which contains a 3-loop, then there are t', r' such that $\mathbf{P}(t', r', F_{max}(t', r'))$ and $|\mathrm{Dom}(t')| < |\mathrm{Dom}(t)|$.*

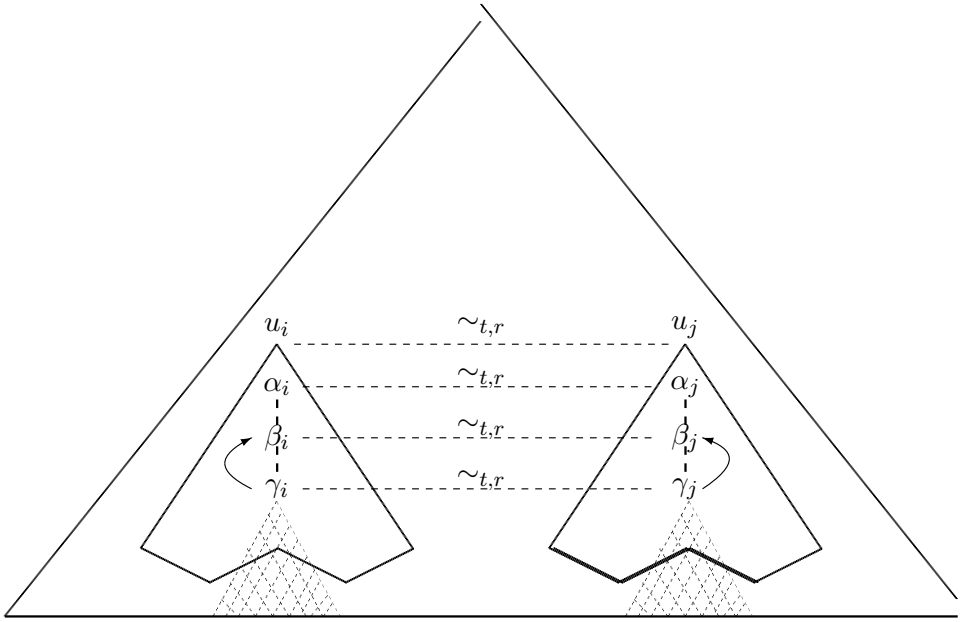


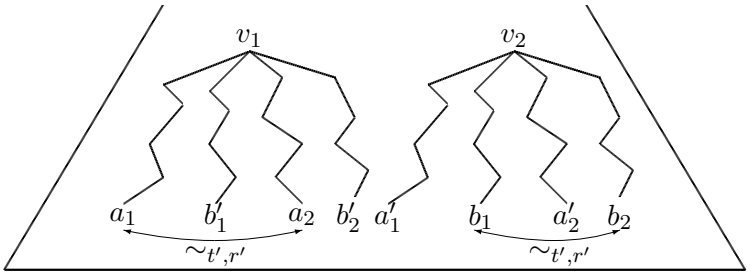
Figure 5.7: parallel pumping in elementary contexts

Proof. The proof is divided in several parts. We first show how to construct t' and r' and then prove the correctness of the construction.

Construction of t' and r' We use a similar technique as the pumping technique presented in the proof of Lemma 5.4.2 (for positive TAGEDs). But, instead of pumping maximally, we pump in parallel in elementary contexts which contain a 3-loop. In each elementary context, we pump the two greatest nodes of the 3-loop (for $\prec^t_{ch^*}$). This ensures that there is still a loop in the elementary context after pumping.
First note that by hypothesis, $\mathcal{C}(r)$ is non-empty, and there is $u \in \mathcal{C}(r)$ such that $\text{cxt}_r(u)$ contains a 3-loop. By hypothesis, we have $r|_v = r|_u$, for all $v \in [u]_{t,r}$ (condition 2 of $\mathbf{P}$). Hence $v \in \mathcal{C}(r)$ for all $v \in [u]_{t,r}$. By definition of $\sim_{t,r}$, all the nodes of $[u]_{t,r}$ are incomparable. Let $n = |[u]_{t,r}|$ and $\{u_1, \ldots, u_n\} = [u]_{t,r}$. Let C be the n-ary context over alphabet Q such that $r = C[r|_{u_1}, \ldots, r|_{u_n}]$. For all $i \in \{1, \ldots, n\}$, there are three nodes $\alpha_i, \beta_i, \gamma_i \in \text{Dom}(\text{cxt}_r(u_i))$ and a state $q \notin \text{dom}(=_A) \cup \text{dom}(\neq_A)$ such that $\alpha_i \prec^t_{ch^+} \beta_i \prec^t_{ch^+} \gamma_i$, and $\mathsf{lab}^r(\alpha_i) = \mathsf{lab}^r(\beta_i) = \mathsf{lab}^r(\gamma_i) = q$. Moreover, we can take $\alpha_i, \beta_i, \gamma_i$ such that for all $i, j \in \{1, \ldots, n\}$, we have $\text{path}_t(u_i, \alpha_i)$ edge-isomorphic to $\text{path}_t(u_j, \alpha_j)$, $\text{path}_t(\alpha_i, \beta_i)$ edge-isomorphic to $\text{path}_t(\alpha_j, \beta_j)$, and $\text{path}_t(\beta_i, \gamma_i)$ edge-isomorphic to $\text{path}_t(\beta_j, \gamma_j)$. We let $r' = C[r_1, \ldots, r_n]$, where for all $i \in \{1, \ldots, n\}$, r_i is the tree $r|_{u_i}$ in which the subtree rooted at β_i has been substituted by $r|_{\gamma_i}$. We do the corresponding substitution in t and obtain a tree t'. This pumping is described in Fig. 5.7 (the subtrees $t|_{\gamma_i}$ are represented in dashed style). It is technical but not difficult to prove that r' is a run of A on t' such that its root is labeled by a final state and it respects the equality constraints.

Proof of Correctness We prove that conditions 1, 2, 3, and 4 of $\mathbf{P}$ hold for t',r', and $F_{max}(t', r')$.

- conditions 1, 2. The equality constraints are still satisfied since we pump in parallel below equivalent nodes. The other conditions obviously hold;
- condition 3 is obvious;
- condition 4. Let $v_1, v_2 \in \mathrm{Dom}(t')$ such that $\mathsf{lab}^{r'}(v_1) \neq_A \mathsf{lab}^{r'}(v_2)$ and $t'|_{v_1} = t'|_{v_2}$. We consider two cases:
  - if $t|_{v_1} \neq t|_{v_2}$, it means that the pumping has "broken" this disequality. Let $[u]_{t,r} = \{u_1, \ldots, u_n\}$ be the nodes defined in the definition of the pumping, i.e. the nodes which enroot elementary contexts in which we have pumped. Note that we still have $[u]_{t,r} \subseteq \mathrm{Dom}(t')$ and $[u]_{t,r} = [u]_{t',r'}$ since we have pumped below the nodes of $[u]_{t,r}$. Let $\{a_1, \ldots, a_{n_1}\} \subseteq [u]_{t,r}$ (resp. $\{b_1, \ldots, b_{n_2} \subseteq [u]_{t,r}$) the nodes of $[u]_{t,r}$ which are below v_1 (resp. v_2). For all $\ell \in \{1, \ldots, n_1\}$, let a'_ℓ of $\mathrm{Dom}(t')$ such that $\mathrm{path}_{t'}(v_2, a'_\ell)$ is edge-isomorphic to $\mathrm{path}_{t'}(v_1, \alpha_\ell)$ (it exists since $t'|_{v_1} = t'|_{v_2}$. Similarly, for all $\ell \in \{1, \ldots, n_2\}$, let b'_ℓ the node below v_1 such that $\mathrm{path}_{t'}(v_1, b'_\ell)$ is edge-isomorphic to $\mathrm{path}_{t'}(v_2, b_\ell)$. This is depicted by the following figure (where $n_1 = n_2 = 2$):



    We now prove that there is a node w in $\{a_1, \ldots, a_{n_1}, b_1, \ldots, b_{n_2}\}$ such that $\mathrm{Rep}(t', r', F_{max}(t', r'), v_1, v_2, w)$ holds. Suppose that for all $\ell \in \{1, \ldots, n_1\}$, we have $a_\ell \sim_{t',r'} a'_\ell$, and for all $\ell \in \{1, \ldots, n_2\}$, we have $b_\ell \sim_{t',r'} b'_\ell$. By definition of the pumping, the maximal context of t which have nodes of $[u]_{t,r}$ as holes has not changed during pumping. Intuitively it means that we have pumped similarly at isomorphic positions, so that the two trees $t|_{v_1}$ and $t|_{v_2}$ were already equal. Formally we are in the situation of Lemma 5.5.6, which is stated and proved at the end of this subsection, and we get $t|_{v_1} = t|_{v_2}$, which is impossible by hypothesis. Hence there is $\ell_1 \in \{1, \ldots, n_1\}$ or $\ell_2 \in \{1, \ldots, n_2\}$ such that $a_{\ell_1} \not\sim_{t',r'} a'_{\ell_1}$ or $b_{\ell_2} \not\sim_{t',r'} b'_{\ell_2}$. Suppose it is a_{ℓ_1}: $a_{\ell_1} \in [u]_{t,r} = [u]_{t',r'}$, and $[u]_{t',r'} \in F_{max}(t', r')$ (since $u \in \mathcal{C}(r')$), hence $\mathrm{Rep}(t', r', F_{max}(t', r'), v_1, v_2, a_{\ell_1})$ holds;
  - if $t|_{v_1} = t|_{v_2}$, since condition 4 holds for $t, r, F_{max}(t, r)$, there is $c \in F_{max}(t, r)$ and $v \in c$ such that $\mathrm{Rep}(t, r, F, v_1, v_2, v)$ holds. Suppose that $v_1 \prec^t_{ch^*} v$, the other case being symmetric. Let v' be the node below v_2 such that $\mathrm{path}_t(v_1, v)$ is edge-isomorphic to $\mathrm{path}_t(v_2, v')$. By definition of the predicate Rep, $v \not\sim_{t,r} v'$. We now consider three cases:
    - Suppose there is a node $w \in [u]_{t,r}$ such that $v_1 \prec^t_{ch^*} w \prec^t_{ch^*} v$, and let w' below v_2 such that $\mathrm{path}_t(v_1, w)$ is edge-isomorphic to $\mathrm{path}_t(v_2, w')$. This situation is depicted in Fig. 5.8. If $w \sim_{t,r} w'$, then by Proposition 5.3.7, we also have $v \sim_{t,r} v'$, which is

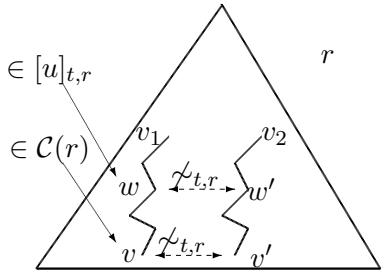


Figure 5.8:

impossible. Hence $w \not\sim_{t,r} w'$, and since we pump only below nodes of $[u]_{t,r}$, w is still a node of t'. Since $w \in \mathcal{C}(r)$, and their is in the elementary context rooted at w in r' (by definition of the pumping), $w \in \mathcal{C}(r')$. By definition of $F_{max}(t', r')$, $w \in F_{max}(t', r')$. Hence $\text{Rep}(t', r', F', v_1, v_2, w)$ holds;

- Suppose there is a node $w \in [u]_{t,r}$ such that $v_2 \prec^t_{ch^*} w \prec^t_{ch^*} v'$. With exactly the same arguments we can prove that $\text{Rep}(t', r', F', v_1, v_2, w)$ holds;
- Suppose that there is no node $w \in [u]_{t,r}$ such that $v_1 \prec^t_{ch^*} w \prec^t_{ch^*} v$ or $v_2 \prec^t_{ch^*} w \prec^t_{ch^*} v'$. Since we pump inside elementary contexts rooted at nodes of $[u]_{t,r}$, and $\mathsf{lab}^r(v_1), \mathsf{lab}^r(v_2) \in \text{dom}(\neq_A)$, by definition of elementary contexts, neither v_1 nor v_2 can be a node of some context $\text{cxt}_t(v)$, for $v \in [u]_{t,r}$. By definition of the pumping, $\text{path}_{t'}(v_1, v)$ is still edge-isomorphic to $\text{path}_{t'}(v_2, v')$. Moreover, $v \in \mathcal{C}(r')$, so that $[v]_{t',r'} \in F_{max}(t', r')$, by definition of $F_{max}(t', r')$. Finally, thanks to Fact 1 (next stated), since $v \not\sim_{t,r} v'$, we also get $v \not\sim_{t',r'} v'$. Hence $\text{Rep}(t', r', F_{max}(t', r'), v_1, v_2, v)$ holds.

Fact 1. $\text{Dom}(t') \subseteq \text{Dom}(t)$ and for all $v_1, v_2 \in \text{Dom}(t')$, $v_1 \sim_{t',r'} v_2$ iff $v_1 \sim_{t,r} v_2$.

Suppose that $v_1 \sim_{t,r} v_2$ and $v_1 \neq v_2$ (the case $v_1 = v_2$ is obvious). We can prove[4] that there are $w_1, w_2 \in \text{Dom}(t)$ such that $w_1 \prec^t_{ch^*} v_1$, $w_2 \prec^t_{ch^*} v_2$ and $\mathsf{lab}^r(w_1) =_A \mathsf{lab}^r(w_2)$.
Suppose that $w_1 \notin \text{Dom}(t')$ or $w_2 \notin \text{Dom}(t')$. It means that w_1, w_2 have been removed by the pumping. Since the elementary contexts in which we pump do not contain nodes labeled by states from $\text{dom}(=_A)$ (except at their root), it means that w_1 and w_2 are below elementary contexts, hence their whole subtrees have

[4]It can be shown by induction: it is obvious if $v_1 = v_2$ or $v_1 \leftrightarrow_{t,r} v_2$, by definition of $\leftrightarrow_{t,r}$. Suppose that there is v_3 such that $v_1 \sim_{t,r} v_3$ and $v_3 \leftrightarrow_{t,r} v_2$, and there are some nodes w_1, w_3 above v_1, v_3 such that $\mathsf{lab}^r(w_3) =_A \mathsf{lab}^r(w_1)$. By definition of $\leftrightarrow_{t,r}$, there are w'_3 and w'_2 such that $\text{path}_t(w'_3, v_3)$ is edge-isomorphic to $\text{path}_t(w'_2, v_2)$, and $\mathsf{lab}^r(w'_3) =_A \mathsf{lab}^r(w'_2)$.

By definition of $\sim_{t,r}$, we have $w'_3 \sim_{t,r} w'_2$, and by condition 2, which holds for $t, r, F_{max}(t, r)$, we get $r|_{w'_3} = r|_{w'_2}$, and, similarly, we get $r|_{w_1} = r|_{w_3}$. Suppose that $w_3 \prec^t_{ch^*} w'_3$, and let w'_1 above v_1 such that $\text{path}_t(w_1, w'_1)$ is edge-isomorphic to $\text{path}_t(w_3, w'_3)$: it exists since $r|_{w_1} = r|_{w_3}$, moreover, $\mathsf{lab}^r(w'_1) =_A \mathsf{lab}^r(w'_3)$. Since $\mathsf{lab}^r(w'_3) =_A \mathsf{lab}^r(w'_2)$ and $=_A \subseteq id_Q$, we also have $\mathsf{lab}^r(w'_1) =_A \mathsf{lab}^r(w'_2)$.

The case $w'_3 \prec^t_{ch^+} w_3$ is proved similarly.

been removed by the pumping. In particular, v_1 and v_2 are removed, which is impossible. Hence $w_1, w_2 \in \text{Dom}(t')$.
We consider two cases:

- If there is no node $u' \in [u]_{t,r}$ such that $w_1 \prec^t_{ch^*} u' \prec^t_{ch^*} v_1$ or $w_2 \prec^t_{ch^*} u' \prec^t_{ch^*} v_2$, then $\text{path}_{t'}(w_1, v_1)$ is edge-isomorphic to $\text{path}_{t'}(w_2, v_2)$. Since $\mathsf{lab}^{r'}(w_1) = \mathsf{lab}^r(w_1)$ and $\mathsf{lab}^{r'}(w_2) = \mathsf{lab}^r(w_2)$, we get $\mathsf{lab}^{r'}(w_1) =_A \mathsf{lab}^{r'}(w_2)$ and therefore $w_1 \sim_{t',r'} w_2$, from which we get $v_1 \sim_{t',r'} v_2$.

- If there is $u' \in [u]_{t,r}$ such that $w_1 \prec^t_{ch^*} u' \prec^t_{ch^*} v_1$. Let $u'' \in \text{Dom}(t)$ such that $w_2 \prec^t_{ch^*} u'' \prec^t_{ch^*} v_2$ and $\text{path}_t(w_1, u)$ is edge-isomorphic to $\text{path}_t(w_2, u')$. Since $w_1 \sim_{t,r} w_2$, we also have $u'' \sim_{t,r} u'$. Hence $u'' \in [u]_{t,r}$. Since we pump in parallel in $\text{cxt}_t(u')$ and $\text{cxt}_t(u'')$, $\text{path}_{t'}(w_1, v_1)$ is still edge-isomorphic to $\text{path}_{t'}(w_2, v_2)$, so that $v_1 \sim_{t',r'} v_2$.

Conversely, suppose that $v_1 \sim_{t',r'} v_2$ and $v_1 \neq v_2$ (the case $v_1 = v_2$ is obvious). For the same reason as before, there are $w_1, w_2 \in \text{Dom}(t')$ such that $w_1 \prec^{t'}_{ch^*} v_1$, $w_2 \prec^{t'}_{ch^*} v_2$ and $\mathsf{lab}^{r'}(w_1) =_A \mathsf{lab}^{r'}(w_2)$. Since $\text{Dom}(t') \subseteq \text{Dom}(t)$, we necessarily have $w_1, w_2, v_1, v_2 \in \text{Dom}(t)$. By definition of the pumping, we also have $w_1 \prec^t_{ch^*} v_1$ and $w_2 \prec^t_{ch^*} v_2$. Suppose that $\text{path}_t(w_1, v_1)$ is not edge-isomorphic to $\text{path}_t(w_2, v_2)$. We show a contradiction (hence this will prove $v_1 \sim_{t,r} v_2$). Since $\text{path}_t(w_1, v_1)$ and $\text{path}_t(w_2, v_2)$ are not edge-isomorphic, and after pumping $\text{path}_{t'}(w_1, v_1)$ and $\text{path}_{t'}(w_2, v_2)$ are isomorphic, necessarily a pumping has occurred in an elementary context rooted at some node $u' \in [u]_{t,r}$ such that $w_1 \prec^t_{ch^*} u' \prec^t_{ch^*} v_1$ or $w_2 \prec^t_{ch^*} u' \prec^t_{ch^*} v_2$. Suppose that $w_1 \prec^t_{ch^*} u'$, and let u'' such that $\text{path}_t(w_1, u')$ is edge-isomorphic to $\text{path}_t(w_2, u'')$ (it exists since $w_1 \sim_{t,r} w_2$ and $t|_{w_1} = t|_{w_2}$). If u'' does not belong to the path from w_2 to v_2, then after pumping, we would still have $\text{path}_{t'}(w_1, v_1)$ non-isomorphic to $\text{path}_{t'}(w_2, v_2)$, since we pump below u' and u''. Hence u'' belongs to $\text{path}_t(w_2, v_2)$. It is not difficult to show that since we pump in parallel at equivalent positions, then we still have $\text{path}_{t'}(w_1, v_1)$ non-isomorphic to $\text{path}_{t'}(w_2, v_2)$, which is a contradiction.
End of Proof of Fact 1. □
End of Proof of Lemma 5.5.5. □

We are now able to prove the forth direction of Theorem 5.5.4. We first assume that $=_A \subseteq id_Q$.
If $\mathcal{L}(A, k) \neq \varnothing$, there is $t \in \mathcal{L}(A, k)$ and r a successful run of (A, k) on t. By Lemma 5.3.10, we can suppose that for all nodes $u, v \in \text{Dom}(t)$, if $u \sim_{t,r} v$, then $r|_u = r|_v$. In other words, condition 2 of $\mathbf{P}$ is satisfied for t and r.
Let C be the maximal context of r (for context inclusion) such that the root of r is the root of C, and no nodes of C (except possibly the root) are labeled in $\text{dom}(=_A) \cup \text{dom}(\neq_A)$. We call this context the *top context*. If the root of C is not labeled by a state of $\text{dom}(=_A) \cup \text{dom}(\neq_A)$, we first pump in C, while there is a loop. We do the corresponding pumping in t. Note that this pumping preserves satisfiability of the constraints, since we pump above the nodes where a test occurs. We obtain a successful run r' of (A, k) on a tree t' such that the height of the top context of r' is at most $|Q|$, if the root of C is not labeled in $\text{dom}(=_A) \cup \text{dom}(\neq_A)$. We call this property P_1. It is not difficult to prove that $\mathbf{P}(t', r', F_{max}(t', r'))$ holds. For instance, condition 4 of $\mathbf{P}$ obviously holds since all the constraints are satisfied, condition 2 has been proved before, and other conditions are straightforward consequences of the definition of t', r' and $F_{max}(t', r')$.

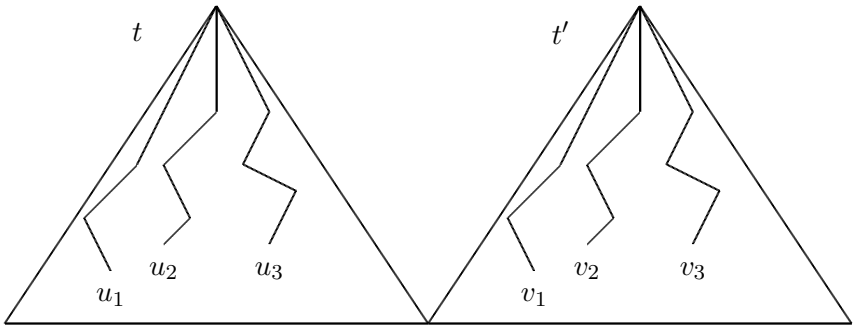


Figure 5.9: Example satisfying the hypothesis of Lemma 5.5.6

Now, suppose that the height of t' is strictly greater than B. In any $\prec^{t'}_{ch^+}$-ordered chain, they are at most k nodes labeled by a state of $\text{dom}(\neq_A)$ in r', and $|\text{dom}(=_A)|$ nodes labeled by a state of $\text{dom}(=_A)$ (otherwise there would be two different descendant nodes enrooting equal subtrees in t', which is impossible). The property P_1, the fact that the height of t' is strictly greater than $B = 2(k + |\text{dom}(=_A)|)|Q|$, and the fact that there are at most $|\text{dom}(=_A)|$ nodes labeled by an equality state in r' along a descending path imply that there is an elementary context containing a 3-loop. Hence the hypothesis of Lemma 5.5.5 are satisfied, so that there are a tree t'' and a run r'' such that $\mathbf{P}(t'', r'', F_{max}(t'', r''))$ holds and $\|t''\| < \|t'\|$. We iterate the reasoning while there is an elementary context containing a 3-loop. At the end, we get a tree t^* and a run r^* such that the height of t^* is bounded by B, and $\mathbf{P}(t^*, r^*, F_{max}(t^*, r^*))$ holds. Hence, t^* has at most 2^{B+1} nodes, and $\mathbf{P}(t^*, r^*, F_{max}(t^*, r^*))$ holds.
If the height of t is lesser than B, $\mathbf{P}(t, r, F_{max}(t, r))$ directly holds (the size does not matter here).
If A does not satisfy $=_A \subseteq id_Q$, we first have to transform it (modulo an exponential blow-up), thanks to Lemma 5.3.5.

Auxiliary Lemma

Lemma 5.5.6 *Let $t, t' \in \mathcal{T}_{ran}(\Sigma)$, $u_1, \ldots, u_n \in \text{Dom}(t)$ and $v_1, \ldots, v_n \in \text{Dom}(t')$ such that: u_i and u_j are incomparable by $\prec^t_{ch^*}$, for $i \neq j$, and $path_t(\mathsf{root}^t, u_i)$ is edge-isomorphic to $path_{t'}(\mathsf{root}^{t'}, v_i)$ for all $i \in \{1, \ldots, n\}$ (see Fig. 5.9).*
Then for all $t_1, \ldots, t_n \in \mathcal{T}_{ran}(\Sigma)$, if $t = t'$, then $t[u_1 \leftarrow t_1] \ldots [u_n \leftarrow t_n] = t'[v_1 \leftarrow t_1] \ldots [v_n \leftarrow t_n]$, where $t[u_1 \leftarrow t_1] \ldots [u_n \leftarrow t_n]$ is the tree t where the subtree at position u_i has been substituted by t_i, for all $i \in \{1, \ldots, n\}$. The tree $t'[v_1 \leftarrow t_1] \ldots [v_n \leftarrow t_n]$ is defined similarly.

Proof. This is because if $t = t'$, then $t|_{u_i} = t'|_{v_i}$, for all $i \in \{1, \ldots, n\}$. □

5.5.3 Proof of the Back Direction of Theorem 5.5.4

The back direction is proved by induction on the frontiers, partially ordered by their cardinalities.

Lemma 5.5.7 (Base Lemma) *If $\mathbf{P}(t, r, \varnothing)$ holds, then $t \in \mathcal{L}(A, k)$.*

Proof. This is due to condition 4 of $\mathbf{P}$. Indeed, suppose that there is an unsatisfied disequality constraint between a node v_1 and a node v_2. By condition 4, there is

$u \in F$ such that $\text{Rep}(t, r, F, v_1, v_2, u)$ holds, which is impossible since $F = \varnothing$. Hence the disequality constraints are satisfied, and together with condition 1, we get that r is a successful run of A on t, which concludes the proof. □

Lemma 5.5.8 (Induction Lemma) *Let t, r, F such that $F \neq \varnothing$ and $\mathbf{P}(t, r, F)$ holds, there are t', r', F' such that $\mathbf{P}(t', r', F')$ holds and $|F'| < |F|$.*

Proof. **Construction of t', r', F'.**
Intuition. In order to obtain t' and r', we choose a maximal (for $\prec_{\sim_{t,r}}$) $\sim_{t,r}$-equivalence class contained in the frontier F (it exists since $F \neq \varnothing$). Let $[u_1]_{t,r} = \{u_1, \ldots, u_n\}$ be this class. By condition 2, for all $u \in [u_1]_{t,r}$, $u \in \mathcal{C}(r)$. We let $D = \{v \in \text{Dom}(t) \mid \exists i,\ v \prec^t_{ch^*} u_i\}$. There might be unsatisfied inequality constraints between the nodes of t and the nodes of D. By increasing (in parallel) the size of the elementary contexts rooted at nodes of $[u_1]_{t,r}$, we can repair some of them. This can be done by pumping in parallel a loop contained in the elementary contexts, while preserving the equality constraints, as illustrated by Fig. 5.10 (see the next paragraph for a formal description). Of course, pumping elementary contexts may also create new unsatisfied constraints, but we show that there is a way to pump which does not create new unsatisfied constraints. Some unsatisfied constraints cannot be repaired though, but thanks to condition 4 of $\mathbf{P}$, those constraints will be repaired later in the induction.
Parallel Growth of The Elementary Contexts. We first define formally how to increase the size of the contexts, while keeping the equality constraints satisfied. By definition of elementary contexts, there are two descendant nodes $\alpha_1 \prec^t_{ch^+} \beta_1$ contained in $\text{cxt}_t(u_1)$, and a state $q \in Q - (\text{dom}(=_A) \cup \text{dom}(\neq_A))$ such that $\mathsf{lab}^r(\alpha_1) = \mathsf{lab}^r(\beta_1) = q$. For all $i \in \{2, \ldots, n\}$, we define α_i as the node below u_i such that $\text{path}_t(u_i, \alpha_i)$ is edge-isomorphic to $\text{path}_t(u_1, \alpha_1)$ (it exists since $u_1 \sim_{t,r} u_i$ and $t|_{u_1} = t|_{u_i}$ by Prop. 5.3.6). Note that by Prop. 5.3.7, $\alpha_1 \sim_{t,r} \alpha_i$. We define the nodes β_i similarly, and also get $\beta_1 \sim_{t,r} \beta_i$. By condition 2, for all $i \in \{1, \ldots, n\}$, $\mathsf{lab}^r(\alpha_i) = \mathsf{lab}^r(\beta_i) = q$. Hence there is a unary context C over Σ such that for all $i \in \{1, \ldots, n\}$, $t|_{\alpha_i} = C[t|_{\beta_i}]$. We let $t^1 = t$, $t^2 = t\{\alpha_i \leftarrow C[C[t|_{\beta_i}]],\ i = 1, \ldots, n\}$ the tree t where the subtree at node α_i has been substituted by $C[C[t|_{\beta_i}]]$, for all $i \in \{1, \ldots, n\}$. Similarly, for all $j \in \mathbb{N}$, we define t^j by iterating this substitution j times. We define r^j similarly. First note that r^j is a run of A on t^j. Moreover, since we make the substitution in parallel at isomorphic positions in the elementary contexts rooted at $[u_1]_{t,r}$, and by definition of $\sim_{t,r}$, the equality constraints are still satisfied by r^j on t^j, for all $j \in \mathbb{N}$. This pumping is illustrated in Fig. 5.10.
Existence of a Repairing Run We let $\uparrow([u_1]_{t,r})$ the ancestors of the nodes of $[u_1]_{t,r}$, *ie* the set $\{w \mid \exists w' \in [u_1]_{t,r},\ w \prec^t_{ch^*} w'\}$. The pumpings r^i, t^i, $i \in \mathbb{N}$ may create unsatisfied disequality constraints which involve a node (or two) of $\uparrow([u_1]_{t,r})$. On the contrary, pumpings can also repair unsatisfied disequality constraints in r and t. We define a set of pairs of nodes $(u, v) \in \text{Dom}(t) \times \text{Dom}(t)$ between which there is an unsatisfied constraint in t and r, or which may create a disequality constraint when pumping, but for which there is some $i \in \mathbb{N}$ such that the constraint is satisfied in t^i and r^i. These pairs are called candidates. We denote by $\text{Cand}([u_1]_{t,r})$ this set. For all $u, v \in \text{Dom}(t)$, $(u, v) \in \text{Cand}([u_1]_{t,r})$ if

(i) $u \in \uparrow([u_1]_{t,r})$,

(ii) $\mathsf{lab}^r(u) \neq_A \mathsf{lab}^r(v)$,

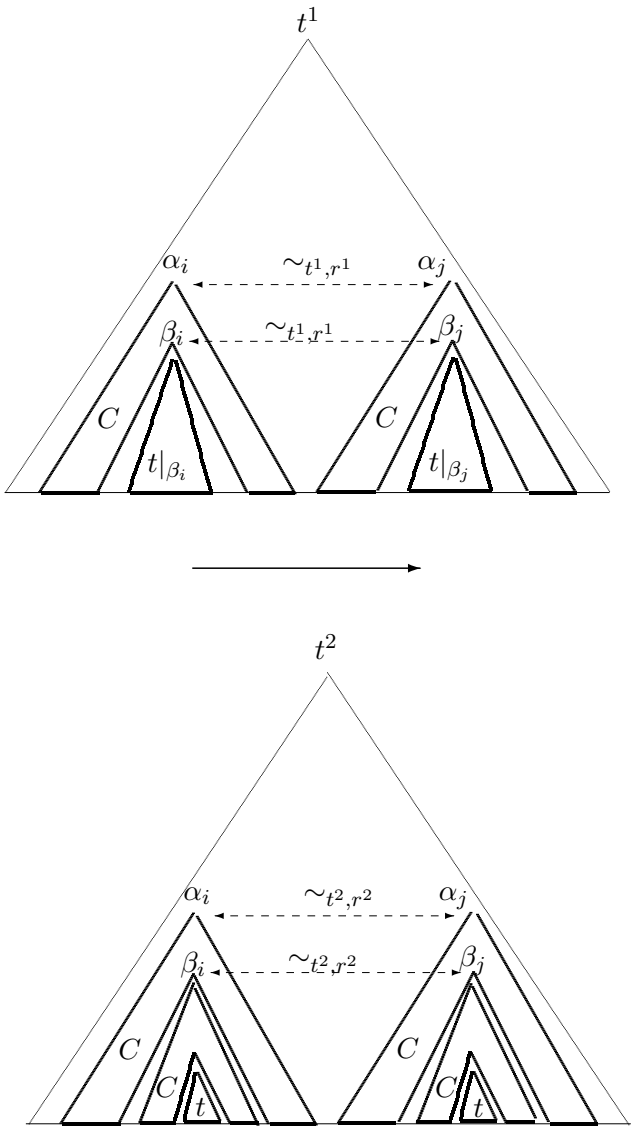


Figure 5.10: parallel growth in elementary contexts

(iii) if $t|_u = t|_v$, then there is $\ell \in \{1, \ldots, n\}$ such that $\text{Rep}(t, r, F, u, v, u_\ell)$ holds.

Note that by definition of the pumping, for all $i > 0$, for all nodes $u \in \text{Dom}(t)$ such that $\mathsf{lab}^r(u) \in \text{dom}(=_A) \cup \text{dom}(\neq_A)$, u is still a node of t^i. For all $i > 0$, and all pairs $(u, v) \in \text{Cand}([u_1]_{t,r})$, we say that i is *incompatible* with (u, v) if $t^i|_u = t^i|_v$.

Claim 1*. For all pairs $(u, v) \in \text{Cand}([u_1]_{t,r})$, there is at most one $i > 0$ such that i is incompatible with (u, v)*

It is proved at the end of section 5.5.3. From *Claim 1*, we deduce that there is $i_0 \in \mathbb{N}$ such that i_0 is compatible with all pairs of $\text{Cand}([u_1]_{t,r})$. We let $t' = t^{i_0}$ and $r' = r^{i_0}$. Note that by definition of the pumping, condition 2 of $\mathbf{P}$ still holds for $\sim_{t',r'}$, since we make the elementary contexts grow in parallel below all equivalent nodes of $[u_1]_{t,r}$, both in r and t. Moreover, the equality constraints are still satisfied (thanks to parallel pumping), the root of r' is labeled by a final state, and the

pumping do not increase the number of nodes ordered by $\prec_{ch^+}$ which are labeled in dom($\neq_A$). Hence condition 1 of $\mathbf{P}$ holds for r'.
Remark 1 By definition of the pumping, all inequality constraints between pairs of nodes $(u, v) \in \text{Cand}([u_1]_{t,r})$ are satisfied in r'.
Finally, we let $F' = F - [u_1]_{t,r}$ (note that $|F'| < |F|$).

Correctness We now prove that $\mathbf{P}(t', r', F')$ holds:

- Conditions 1 and 2 have already been proved;
- Condition 3. By definition of frontiers, there is a set of nodes $A \subseteq \mathcal{C}(r)$ such that $F = \{[a]_{t,r} \mid a \in A\}$. By definition of the pumping, $\mathcal{C}(r) \subseteq \mathcal{C}(r')$, hence $F' \subseteq \mathcal{C}(r') \cup \mathsf{root}^{r'}$. Moreover since we pump only below nodes of the frontier, for all $a \in A$, $[a]_{t,r} = [a]_{t',r'}$. Hence $[u_1]_{t,r}$ is also a maximal element of $\{[a]_{t',r'} \mid a \in A\}$. Therefore F' is a frontier;
- Condition 4. Let $v_1, v_2 \in \text{Dom}(t')$ such that $t'|_{v_1} = t'|_{v_2}$ and $\mathsf{lab}^{r'}(v_1) \neq_A \mathsf{lab}^{r'}(v_2)$.

  We consider several cases:

  - $v_1 \notin \uparrow([u_1]_{t,r})$ and $v_2 \notin \uparrow([u_1]_{t,r})$. Since $\mathsf{lab}^r(v_1), \mathsf{lab}^r(v_2) \in \text{dom}(\neq_A)$, neither v_1 nor v_2 are in the elementary contexts rooted at the nodes of $[u_1]_{t,r}$. Hence the subtrees at positions v_1 and v_2 have not changed during the pumping. Hence $t|_{v_1} = t|_{v_2}$. Since $\mathbf{P}(t, r, F)$ holds, there is $c \in F$ and $u \in c$ such that Rep(t, r, F, v_1, v_2, u) holds. Necessarily, $c \in F'$, otherwise it would mean that $u \in [u_1]_{t,r}$, which would contradict $v_1 \notin \uparrow([u_1]_{t,r})$ or would contradict $v_2 \notin \uparrow([u_1]_{t,r})$. Hence Rep$(t', r', F', v_1, v_2, u)$ holds;
  - $v_1 \in \uparrow([u_1]_{t,r})$. In this case $(v_1, v_2) \notin \text{Cand}([u_1]_{t,r})$, otherwise $t'|_{v_1} \neq t'|_{v_2}$. By definition of Cand$([u_1]_{t,r})$, $t|_{v_1} = t|_{v_2}$ and for all $i \in \{1, \ldots, n\}$, Rep(t, r, v_1, v_2, u_i) does not hold. By condition 4 of $\mathbf{P}(t, r, F)$, there is are $c \in F$ and $u \in c$ such that Rep(t, r, F, v_1, v_2, u) holds. Hence $u \notin [u_1]_{t,r}$, and $c \neq [u_1]_{t,r}$. Hence $c \in F'$ and Rep$(t', r', F', v_1, v_2, u)$ holds.
  - the last case is subsumed by the latter.

End of proof of Lemma 5.5.8 □

The combination of Lemma 5.5.7 and Lemma 5.5.8 proves the back direction of Theorem 5.5.4.

Remaining Proofs

Proof of Claim 1. Suppose that there are two indices $i < j$ incompatible with (u, v). We consider the following two cases:

- $v \notin \uparrow([u_1]_{t,r})$. By hypothesis, $\mathsf{lab}^{r^i}(v) \neq_A \mathsf{lab}^{r^i}(u)$, $t^i|_v = t^i|_u$, $\mathsf{lab}^{r^j}(v) \neq_A \mathsf{lab}^{r^j}(u)$ and $t^j|_v = t^j|_u$. Since $\mathsf{lab}^r(v) \in \text{dom}(\neq_A)$, and $v \notin \uparrow([u_1]_{t,r})$, v is below, or incomparable, to any node which belongs to an elementary context rooted at a node of $[u_1]_{t,r}$. Hence the subtree at node v remains unchanged during the pumping. Therefore $t|_v = t^i|_v = t^j|_v$.

Since by hypothesis, $t^i|_v = t^i|_u$ and $t^j|_v = t^j|_u$, we also get $t^j|_u = t^i|_u$. Since $i < j$, and $u \in \uparrow([u_1]_{t,r})$, by definition of the pumping, we have $\|t^j|_u\| > \|t^i|_u\|$, which contradicts $t^j|_u = t^i|_u$;

- $v \in \uparrow([u_1]_{t,r})$. We consider two cases:
  - $t|_u = t|_v$. By definition of the set of candidates, there is $\ell \in \{1, \ldots, n\}$ such that $\text{Rep}(t, r, F, u, v, u_\ell)$ holds. Suppose that $u \prec^t_{ch^*} u_\ell$ and let v_ℓ be the node below v such that $\text{path}_t(u, u_\ell)$ is edge-isomorphic to $\text{path}_t(v, v_\ell)$ (it exists since $t|_u = t|_v$). By definition of the predicate Rep, $u_\ell \not\sim_{t,r} v_\ell$. Since $\text{path}_t(u, u_\ell)$ and $\text{path}_t(v, v_\ell)$ are edge-isomorphic, and $t|_u = t|_v$, we also get $t|_{u_\ell} = t|_{v_\ell}$. This implies that there is no node of $[u_1]_{t,r}$ comparable to v_ℓ (by $\prec^t_{ch^*}$): suppose that there is some m such that u_m ($u_m \sim_{t,r} u_1$) is comparable to v_ℓ and $u_m \neq v_\ell$. By Prop. 5.3.6, we have $t|_{u_m} = t|_{u_1}$, which contradicts $t|_{u_1} = t|_{v_\ell}$.
    Therefore the subtree at position v_ℓ does not change during the pumping. In particular, $t^i|_{v_\ell} = t^j|_{v_\ell}$. Since $u_\ell \in [u_1]_{t,r}$, we pump below u_ℓ. Hence we have $t^i|_{u_\ell} \neq t^j|_{u_\ell}$. Therefore, either $t^i|_{u_\ell} \neq t^i|_{v_\ell}$ or $t^j|_{u_\ell} \neq t^j|_{v_\ell}$. Since $\text{path}_t(u, u_\ell)$ and $\text{path}_t(v, v_\ell)$ are edge-isomorphic, and there is no node of $[u_1]_{t,r}$ comparable to v_ℓ, by definition of the pumping, we also have that $\text{path}_{t^i}(u, u_\ell)$ and $\text{path}_{t^i}(v, v_\ell)$ are edge-isomorphic, and $\text{path}_{t^j}(u, u_\ell)$ and $\text{path}_{t^j}(v, v_\ell)$ are edge-isomorphic. Therefore, either $t^i|_v \neq t^i|_u$ or $t^j|_v \neq t^j|_u$;
  - $t|_u \neq t|_v$. By hypothesis, $t^i|_u = t^i|_v$. We first prove that there is necessarily $\ell \in \{1, \ldots, n\}$ such that $\text{Rep}(t^i, r^i, F, u, v, u_\ell)$ holds. Suppose the contrary. It means for all $\ell \in \{1, \ldots, n\}$ such that $u \prec^t_{ch^*} u_\ell$, and for all v' such that $\text{path}_{t^i}(u, u_\ell)$ is edge-isomorphic to $\text{path}_{t^i}(v, v')$, we have $v' \sim_{t,r} u_\ell$. And symmetrically, for all $\ell \in \{1, \ldots, n\}$ such that $v \prec^t_{ch^*} u_\ell$, and for all u' such that $\text{path}_{t^i}(u, u')$ is edge-isomorphic to $\text{path}_{t^i}(v, u_\ell)$, we have $u' \sim_{t,r} v_\ell$. Since $t|_u \neq t|_v$, and thanks to Lemma 5.5.6, this inequality is preserved during pumping. Hence $t^i|_u \neq t^i|_v$, which contradicts $t^i|_u = t^i|_v$.
    Therefore there is $\ell \in \{1, \ldots, n\}$ such that $\text{Rep}(t^i, r^i, F, u, v, u_\ell)$. Suppose that $u \prec^t_{ch^*} u_\ell$ (the case $v \prec^t_{ch^*} u_\ell$ is symmetric). Let v' such that $v \prec^t_{ch^*} v'$ and $\text{path}_{t^i}(v, v')$ is edge-isomorphic to $\text{path}_{t^i}(u, u_\ell)$. By definition of Rep, we have $u_\ell \not\sim_{t^i,r^i} v'$. Since $t^i|_u = t^i|_v$, we also get $t^i|_{u_\ell} = t^i|_{v'}$. For the same reasons as the previous case, there is no node $w \in [u_1]_{t,r}$ which is comparable to v'. Hence, since we pump only in the elementary contexts rooted at nodes of $[u_1]_{t,r}$, the subtree rooted at v' does not change during pumping. Since the subtree rooted at u_ℓ changes during pumping and $\text{path}_{t^i}(u, u_\ell)$ is edge-isomorphic to $\text{path}_{t^i}(v, v')$, we necessarily have $t^j|_u \neq t^j|_v$, which contradicts the hypothesis.

□

5.6 MSO WITH TREE EQUALITY TESTS

As already seen in Chapter 2, there is well-known correspondence between finite tree automata and MSO (Section 2.5.2). It is quite natural to wonder whether such a correspondence exists between TAGEDs and an extension of MSO. For the sake of clarity, we investigate this question on binary trees. We define an extension of MSO$[\prec_{ch_1}, \prec_{ch_2}]$ with the tree equality predicate $x \sim y$, which holds in a tree t, between a node u and a node v if $t|_u$ and $t|_v$ are equal. In the rest of this section, MSO$[\sim]$ stands for MSO$[\prec_{ch_1}, \prec_{ch_2}, \sim]$.

Proposition 5.6.1 *For all TAGEDs A over a (binary) alphabet Σ, there is a computable closed MSO$[\sim]$-formula ϕ_A such that for all trees $t \in \mathcal{T}_{ran}(\Sigma)$:*

$$\mathcal{L}(A) \;=\; \{t \mid t \models \phi_A\}$$

Proof. Let $A = (\Sigma, Q, F, \Delta)$ a TAGED. It is already known (see (CDG*07) or (Gen06)) that the tree automaton (Σ, Q, F, Δ) is equivalent to an MSO-formula of the form: $\phi \;=\; \exists X_{q_1} \ldots \exists X_{q_n}\ \phi_\Delta(X_{q_1}, \ldots, X_{q_n})$
where $\{q_1, \ldots, q_n\} = Q$. Set variables X_{q_i}s are intended to capture the set of nodes labeled by states q_i in a successful run of A (hence the set denoted by $\{X_{q_1}, \ldots, X_{q_n}\}$ forms a partition of the set of nodes, with possibly empty blocks). The formula $\phi_\Delta(X_{q_1}, \ldots, X_{q_n})$ describes the behavior of the tree automaton, in terms of runs. The constraints are naturally added by taking ϕ in conjunction with $\bigwedge_{q \neq_A p} \forall x \in X_q \forall y \in X_p,\ \neg(x \sim y)$ and $\bigwedge_{q =_A p} \forall x \in X_q \forall y \in X_p,\ x \sim y$. Finally ϕ_A is defined by:

$$\phi_A \;=\; \left\{ \begin{array}{ll} \exists X_{q_1} \ldots \exists X_{q_n} & \phi_\Delta(X_{q_1}, \ldots, X_{q_n}) \;\;\wedge \\ & \bigwedge_{q \neq_A p} \forall x \in X_q \forall y \in X_p,\ \neg(x \sim y) \;\;\wedge \\ & \bigwedge_{q =_A p} \forall x \in X_q \forall y \in X_p,\ x \sim y \end{array} \right.$$

□

A TAGED A is universal iff $\neg\phi_A$ is unsatisfiable. Since testing universality of TAGEDs is undecidable (Proposition 5.3.4), and ϕ_A is computable (Proposition 5.6.1):

Theorem 5.6.2 *Testing satisfiability of MSO$[\sim]$-formulas is undecidable.*

We now define a fragment of MSO$[\sim]$ which corresponds to vertically bounded TAGEDs. Then we prove this fragment to be decidable, by reduction to emptiness of vertically bounded TAGEDs. This fragment is defined through new predicates $\mathsf{eq}(X)$ and $\mathsf{diff}_k(X, Y)$, $k \in \mathbb{N}$. The predicate $\mathsf{eq}(X)$ holds in a tree t under some assignment $\rho : X \mapsto U$, for some $U \subseteq \mathrm{Dom}(t)$, if for all $u, v \in U$, $t|_u = t|_v$. For all $k \in \mathbb{N}$, the predicate $\mathsf{diff}_k(X, Y)$ holds in t under some assignment ρ if (i) the size of any subset of $\rho(X)$ or $\rho(Y)$ linearly ordered by $\prec^t_{ch^+}$ is bounded by k, (ii) for all $u \in \rho(X)$ and $v \in \rho(Y)$, the trees $t|_u \neq t|_v$. These predicates are easily definable in MSO$[\sim]$:

$$\begin{array}{lcl} \mathsf{eq}(X) & = & \forall x \forall y,\ x \in X \wedge y \in X \rightarrow x \sim y \\ \mathsf{diff}_k(X, Y) & = & \neg\mathsf{chain}_{k+1}(X) \wedge \neg\mathsf{chain}_{k+1}(Y) \wedge \\ & & \forall x \forall y,\ x \in X \wedge y \in Y \rightarrow \neg(x \sim y) \\ \mathsf{chain}_{k+1}(X) & = & \exists x_1 \ldots \exists x_{k+1},\ \bigwedge_{i=1}^{k+1} x_i \in X \wedge \bigwedge_{i=1}^{k} x_i \prec_{ch} x_{i+1} \end{array}$$

We let $\text{MSO}^{\exists}_{=}$ the extension of MSO whose formulas are of the form $\exists X_1 \dots \exists X_n \phi$, where:

- ϕ is a formula of $\text{MSO}[\mathsf{eq}, (\mathsf{diff}_k)_{k\in\mathbb{N}}]$
- $X_1, \dots, X_n$ are not quantified in ϕ
- if $\mathsf{eq}(X)$ is an atom of ϕ, then $X = X_i$, for some $i \in \{1, \dots, n\}$
- if $\mathsf{diff}_k(X, Y)$ is an atom of ϕ, then $X = X_i$ and $Y = X_j$, for some $i, j \in \{1, \dots, n\}$

$\text{MSO}^{\exists}_{=}$ is a strict fragment of $\text{MSO}[\sim]$, but is strictly more expressive than MSO, as tree isomorphism is not expressible in MSO (CDG*07).

Proposition 5.6.3 *For any closed formula ϕ in $MSO^{\exists}_{=}$, one can compute a vbTAGED, whose size is non-elementary in the size of ϕ, accepting the models of ϕ.*

Proof. First, by moving up the predicates eq and diff_k, we can prove that ϕ is equivalent to a finite disjunction of formulas of the form $\exists \overline{X}\ \gamma(\overline{X}) \wedge \gamma_{test}(\overline{X})$ where eq and diff_k predicates do not occur in γ and γ_{test} is a conjunction of atoms $\mathsf{eq}(X_i)$ or $\mathsf{diff}_k(X_i, X_j)$, or their negations, where $X_i, X_j \in \overline{X}$. Then we remove the negations in γ_{test}. We let $Sing(X)$ the MSO-formula that expresses that X is a singleton set (it exists).
The formula $\neg\mathsf{eq}(X)$ is equivalent to

$$\exists X_1 \exists Y_1, X_1 \subseteq X \wedge X_2 \subseteq X \wedge Sing(X_1) \wedge Sing(X_3) \wedge \mathsf{diff}_1(X_1, X_2)$$

The formula $\neg\mathsf{diff}_k(X, Y)$ holds in a tree t under assignment ρ if either there is a descendant chain of length strictly greater than k in $\rho(X)$ or $\rho(Y)$ (this is expressible in MSO by some formula $desc_{\geq k}(X, Y)$), or there are two nodes $u \in \rho(X)$ and $v \in \rho(Y)$ such that $t|_u = t|_v$. Hence $\neg\mathsf{diff}_k(X, Y)$ is equivalent to:

$$\begin{aligned} \neg\mathsf{diff}_k(X, Y) \quad \leftrightarrow \quad & desc_{\geq k}(X, Y) \wedge \exists X' \exists Y' \exists Z,\ Z = X' \cup Y' \wedge \\ & Sing(X') \wedge Sing(Y') \wedge X' \subseteq X \wedge Y' \subseteq Y \wedge \mathsf{eq}(Z) \end{aligned}$$

Hence we can easily remove the negations, and every $MSO^{\exists}_{=}$ formula ϕ is equivalent to a finite disjunction $\bigvee_i \Phi_i$, where each Φ_i has the following form:

$$\exists \overline{X}\ \psi(\overline{X}) \wedge \psi_{test}(\overline{X})$$

where $\overline{X}$ is a tuple of set variables, the formula ψ is an MSO-formula, and ψ_{test} is a conjunction of atoms $\mathsf{eq}(X_i)$ or $\mathsf{diff}_k(X_i, X_j)$, where $X_i, X_j \in \overline{X}$. We also assume that in ψ_{test}, there is no atom of the form $\mathsf{diff}_k(X_i, X_i)$, otherwise we replace it by the equivalent formula $X_i = \varnothing$. The next construction builts a vbTAGED (A_{Φ_i}, k_i) equivalent to Φ_i, for some $k_i \in \mathbb{N}$ and all i. Moreover, each vbTAGED (A_{Φ_i}, k_i) satisfies $\mathcal{L}(A_{\Phi_i}, k_i) = \mathcal{L}(A_{\Phi_i})$, since a control is added to the states of A_{Φ_i} to check that disequality states are not used more that k_i times along a root-to-leaf path. Hence, ϕ is equivalent to $(\bigcup_i A_{\Phi_i}, max_i\ k_i)$ where $\bigcup_i A_{\Phi_i}$ is obtained as in the proof of Proposition 5.3.1. The bound $max_i\ k_i$ does not really matter here, since the control is done in the states, so that one could take all bound greater than $max_i\ k_i$. Note that if there is no control on the use of inequality states,

the construction is not correct, since in general, $\mathcal{L}(A_{\Phi_i}, k_i) \neq \mathcal{L}(A_{\Phi_i}, p)$, for all $p \neq k_i$.

We now give the construction for a formula $\Phi = \exists \overline{X}, \psi(\overline{X}) \wedge \psi_{test}(\overline{X})$. We let $\overline{X}$ being equal to $X_1, \ldots, X_n$. We use the classical Thatcher and Wright's construction (TW68, CDG*07) to transform $\psi(\overline{X})$ into a tree automaton $A = (\Sigma \times \{0,1\}^n, Q, F, \Delta)$, such that the i-th component of the tuple of any label corresponds to the i-th variable in $\overline{X}$, namely X_i. Then, projecting A on its first component results in a tree automaton recognizing the models of $\exists \overline{X},\ \psi(\overline{X})$. But, instead of projecting the automata as usual, we project the Booleans from the labels into the states like in (NPTT05). We denote by $\mathsf{proj}(A) = (\Sigma, Q_{\mathsf{proj}(A)}, F_{\mathsf{proj}(A)}, \Delta_{\mathsf{proj}(A)})$ the resulting automaton. It is defined by: $Q_{\mathsf{proj}(A)} = Q \times \{0,1\}^n$, $F_{\mathsf{proj}(A)} = F \times \{0,1\}^n$ and $\Delta_{\mathsf{proj}(A)}$ is defined as the union of:

- the set of initial rules $a \to (q, \overline{b})$; such that $(a, \overline{b}) \to q \in \Delta$,
- the set of rules $a((q_1, \overline{b}_1), (q_2, \overline{b}_2)) \to (q, \overline{b})$, such that $\overline{b}_1, \overline{b}_2$ are Boolean tuples, and $(a, \overline{b})(q_1, q_2) \to q \in \Delta$.

For all $\overline{b} \in \{0,1\}^n$, we denote by $\overline{b}(i)$ its i-th projection. Intuitively, every successful run r of $\mathsf{proj}(A)$ on some tree t defines a valuation ρ such that for all i, $\rho(X_i) = \{u \mid \exists (q, \overline{b}), \mathsf{lab}^r(u) = (q, \overline{b}) \text{ and } \overline{b}(i) = 1\}$. This valuation satisfies $t, \rho \models \psi(\overline{X})$. Now, suppose that X_i occurs in an atom $\mathsf{diff}_k(X_i, X_j)$ of ψ_{test}, for some j. One has to add a constraint which imposes that each subtree t_1 evaluated to some state $(q_1, \overline{b}_1)$ with $\overline{b}_1(i) = 1$ and each subtree t_2 evaluated to some state $(q_2, \overline{b}_2)$ with $\overline{b}_2(j) = 1$ satisfy $t_1 \neq t_2$. This can be done with the constraint $(q_1, \overline{b}_1) \neq_{\mathsf{proj}(A)} (q_2, \overline{b}_2)$. However, one must also ensure that there is not more than k nodes in $\rho(X_i)$ linearly ordered by $\prec^t_{ch^+}$. More generally, if X_i occur in several atoms $\mathsf{diff}_{k_1}(X_i, X_{j_1}), \ldots, \mathsf{diff}_{k_p}(X_i, X_{j_p})$ of ψ_{test}, one must ensure that the size of every subset of $\rho(X_i)$ linearly ordered by $\prec^t_{ch^+}$ is lesser than $min_i k_i$. This can be done by adding a counter c_i to the states. More generally, we add a counter for each variable X_i. The final vbTAGED A_Φ equivalent to Φ is then defined as follows. For all $i \in \{1, \ldots, n\}$, we let B_i the minimal bound k such that $\mathsf{diff}_k(X_i, X_j)$ or $\mathsf{diff}_k(X_j, X_i)$ is a atom of ψ_{test}, for some j. The vbTAGED A_Φ is defined as follows:

- $Q_\Phi = Q_{\mathsf{proj}(A)} \times Q_{count}$ where $Q_{count} = \{0, \ldots, B_1\} \times \cdots \times \{0, \ldots, B_n\}$
- $F_\Phi = F_{\mathsf{proj}(A)} \times Q_{count}$
- Δ_ϕ is defined by the inference rules:

$$\frac{\begin{array}{c} a \to (q, \overline{b}) \in \Delta_{\mathsf{proj}(A)} \\ \overline{c} \in Q_{count} \\ \forall i \in \{1, \ldots, n\}, \overline{c}(i) = 1 \text{ if } \overline{b}(i) = 1, \text{ and } \overline{c}(i) = 0 \text{ otherwise} \end{array}}{a \to (q, \overline{b}, \overline{c}) \in \Delta_\Phi}$$

$$\frac{\begin{array}{c} a((q_1, \overline{b}_1), (q_2, \overline{b}_1)) \to (q, \overline{b}) \in \Delta_{\mathsf{proj}(A)} \\ \overline{c}_1, \overline{c}_2, \overline{c} \in Q_{count} \\ \text{for all } i \in \{1, \ldots, n\}, \overline{c}(i) = \overline{b}(i) + max\ (\overline{c}_1(i), \overline{c}_2(i)) \\ \text{for all } i \in \{1, \ldots, n\}, \overline{c}(i) \leq B_i \end{array}}{a((q_1, \overline{b}_1, \overline{c}_1), (q_2, \overline{b}_1, \overline{c}_2)) \to (q, \overline{b}, \overline{c}) \in \Delta_\Phi}$$

- $(q_1, \overline{b}_1, \overline{c}_1) =_{A_\Phi} (q_2, \overline{b}_2, \overline{c}_2)$ if $\exists i \in \{1, \ldots, n\}$ such that $\overline{b}_1(i) = \overline{b}_2(i) = 1$ and $\mathsf{eq}(X_i)$ is an atom of ψ_{test};

- $(q_1, \overline{b}_1, \overline{c}_1) \neq_{A_\Phi} (q_2, \overline{b}_2, \overline{c}_2)$ if $\exists i, j \in \{1, \ldots, n\}$ such that $\overline{b}_1(i) = 1$ and $\overline{b}_2(j) = 1$ and $\mathsf{diff}_k(X_i, X_j)$ is an atom of ψ_{test}, for some k;
- we let $k = max_i\ B_i$ the bound of the vbTAGED A_Φ. The bound does not really matter since the control is already done by A_Φ. In fact, one could take any bound greater than k.

Note that $\neq_{A_\Phi}$ is irreflexive since we assume that there is no atoms of the form $\mathsf{diff}_k(X, X)$ in ψ_{test}. By construction, $L(A_\Phi)$ accepts exactly the models of Φ.

Complexity The size of A is non-elementary in the size of ϕ in the worst-case, as it is already the case without equality and disequality constraints (SM73a). □

The converse of Proposition 5.6.3 also holds:

Proposition 5.6.4 *For any vbTAGED (A, k), one can compute a closed $MSO^{\exists}_{=}$ formula ϕ whose models are the trees accepted by A (modulo an exponential blow-up).*

Proof. Let (A, k) be a vbTAGED. The construction is similar to the construction of Proposition 5.6.1. The 'tree automata' part of A is encoded by an MSO formula:

$$\phi \ = \ \exists X_{q_1} \ldots \exists X_{q_n}\ \phi_\Delta(X_{q_1}, \ldots, X_{q_n})$$

where $\{q_1, \ldots, q_n\}$ are the states of A. The cardinality constraint is expressed by:

$$\phi_{card} \ = \ \neg\exists x_1 \ldots \exists x_{k+1} \bigwedge_{i=1}^{k} x_i \prec_{ch^+} x_{i+1} \ \wedge \ \bigwedge_{i=1}^{k+1} \bigvee_{q \in \mathrm{dom}(\neq_A)} x_i \in X_q$$

The constraints are expressed by $\bigwedge_{q \neq_A p} \mathsf{diff}_k(X_q, X_p) \wedge \bigwedge_{q =_A p} \mathsf{eq}(X_q \cup X_p)$ (set union is easily expressible in MSO). The bound k in $\mathsf{diff}_k(X_q, X_p)$ does not really matter here since the control is done by ϕ_{card}. Hence, one could also take any bound greater than k. □

As a consequence of Theorem 5.5.2 and Propositions 5.6.3 and Propositions 5.6.4:

Theorem 5.6.5 *$MSO^{\exists}_{=}$ and vbTAGEDs effectively define the same tree languages, and satisfiability of $MSO^{\exists}_{=}$ formulas is decidable.*

5.7 TAGEDs for Unranked Trees over an Infinite Alphabet

TAGEDs are used to decide a fragment of the spatial logic TQL. However TQL is interpreted on unranked trees over an infinite alphabet. We first lift the emptiness result on vertically bounded TAGEDs over a finite alphabet to vertically bounded TAGEDs over an infinite alphabet.
Then TAGEDs can naturally be lifted from ranked to unranked trees, on basis of hedge automata, as well as their fragments, positive TAGEDs, negative TAGEDs, and vertically bounded TAGEDs. We call this extension *unranked TAGEDs*. Emptiness results in the ranked case can be lifted to the unranked setting via a particular binary encoding which preserves the subtree equalities.

5.7.1 Extension to an Infinite Alphabet

Let Λ be an infinite alphabet. In order to extend TAGEDs to carry labels from Λ, one needs more complex transitions, with cofinite sets of labels, to keep the Boolean closure properties. In particular, rules of TAGEDs over an infinite alphabet have the form $\alpha(q_1, q_2) \to q$ or $\alpha \to q$, where $\alpha \in \mathcal{P}_{cf}(\Lambda)$ is a (co)finite set of labels. The rule can be applied to a node if its label is contained in α. The size of a rule $\alpha(q_1, q_2) \to q$ is $|\alpha| + 3$ if α is finite, and $|\beta| + 4$ if α is equal to the complement of some finite set β.

Theorem 5.7.1 *Emptiness of vertically bounded TAGEDs over an infinite alphabet is decidable in 2NEXPTIME.*

Proof. Let (A, k) be a vbTAGED over Λ. The proof of Theorem 5.5.4 still holds for vbTAGEDs over an infinite alphabet, since it relies on manipulating runs only. However deciding whether there is a tree t and run r whose height is bounded by B (B is defined in Theorem 5.5.4) and such that $\mathbf{P}(t, r, F_{max}(t, r))$ holds needs another method, as there is an infinite number of trees whose height is bounded by B. However we prove that we can still bound the exploration space.
Let $\mathcal{P}_{cf}(A)$ be the set of cofinite sets occurring in the rules of A. We first transform A such that every finite set of $\mathcal{P}_{cf}(A)$ is a singleton set. This can be done by creating as many rules as there are labels in the finite sets (and therefore in polynomial time in the size of A). We also transform A such that there is no finite set $\alpha \in \mathcal{P}_{cf}(A)$ included in an infinite set $\alpha' \in \mathcal{P}_{cf}(A)$ (property *). This can be done in polynomial-time by creating, for all singleton sets $\{f\} \in \mathcal{P}_{cf}(A)$, all infinite sets $\alpha \in \mathcal{P}_{cf}(A)$ such that $f \in \alpha$ and all rules of the form $\alpha(q_1, q_2) \to q$, the two rules:

$$f(q_1, q_2) \to q \text{ and } (\alpha' - f)(q_1, q_2) \to q$$

We let $\# \in \Lambda$ a symbol such that for all infinite sets α of $\mathcal{P}_{cf}(A)$, $\# \in \alpha$ (it necessarily exists if there is at least one infinite set, which can be assumed wlog). We denote by active(A) the set

$$\text{active}(A) \;=\; \bigcup\{\alpha \mid \alpha \text{ is finite and } \alpha \in \mathcal{P}_{cf}(A)\} \;\cup\; \{\#\}$$

We now prove that there exist a tree t over Λ and a run r whose height is bounded by B such that $\mathbf{P}(t, r, F_{max}(t, r))$ holds iff there exist a tree t' over active(A) and a run r' whose height is bounded by B such that $\mathbf{P}(t', r', F_{max}(t', r'))$ holds.
The back direction is obvious as t' is also a tree over Λ. For the forth direction we let $r' = r$ and t' is obtained by changing some labels of t (in particular t and t' are edge-isomorphic). For all nodes $u \in \text{Dom}(t)$, we look at the rules which can be applied at node u in r. Thanks to property *, either rules with an infinite set or rules with a finite set can be applied at node u in r. If a rule with an infinite set can be applied, we substitute $\mathsf{lab}^t(u)$ by $\#$, otherwise we let it unchanged. By definition of t', if two node labels were equal in t they are still equal in t'.
The resulting tree t' and the run r' still satisfy the equality constraints. Suppose the contrary, *ie* there are two nodes $u, v \in \text{Dom}(t')$ such that $\mathsf{lab}^{r'}(u) =_A \mathsf{lab}^{r'}$ and $t'|_u \neq t'|_v$. Since t and r satisfy the equality constraints, $t|_u = t|_v$. Hence $t'|_u$ and $t'|_v$ are edge-isomorphic. Thus there are two isomorphic nodes u' and v' respectively (such that $\text{path}_{t|_u}(u, u')$ is edge-isomorphic to $\text{path}_{t|_v}(v, v')$) such that $\mathsf{lab}^{t'}(u') \neq \mathsf{lab}^{t'}(v')$. This is impossible by definition of t'. Finally, since $r = r'$, the conditions $2, 3, 4$ of $\mathbf{P}$ are still satisfied.

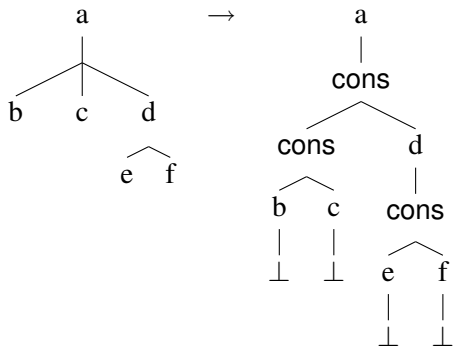
Consequently in order to decide emptiness of (A, k), it suffices to test whether there are a tree t over active(A) and a run r whose height is bounded by B and such that $\mathbf{P}(t, r, F_{max}(t, r))$ holds, which can be done in 2NEXPTIME in $\|A\|$ and k. □

5.7.2 Binary Encoding

Let Λ be an infinite alphabet, cons a fresh binary symbol such that cons $\notin \lambda$, and $\perp$ a constant symbol such that $\perp \notin \Lambda$. Intuitively, cons appends a tree at the end of a hedge. We view hedges over Λ as ranked trees over $\Lambda_r = \Lambda \cup \{\mathsf{cons}, \perp\}$, where every symbol of Λ is viewed as a unary function symbol. The encoding is given by the mapping $\Psi : \mathcal{H}(\Lambda) \to \mathcal{T}_{ran}(\Lambda_r)$, for all $t \in \mathcal{T}_{unr}(\Lambda)$, $h \in \mathcal{H}(\Lambda)$, $a \in \Lambda$:

$$\begin{array}{lcll} \Psi(\mathsf{0}) & = & \perp & \\ \Psi(h|t) & = & \mathsf{cons}(\Psi(h), \Psi(t)) & \text{where } h \neq \mathsf{0} \\ \Psi(a(h)) & = & a(\Psi(h)) & \end{array}$$

The following example illustrates this encoding:



Note that every subtree whose root is labeled in Λ in the encoding corresponds to a subtree in the original unranked tree.

Proposition 5.7.2 *For all unranked TAGEDs A over Λ, there is a TAGED A' such that $\mathcal{L}(A') = \{\Psi(t) \mid t \in \mathcal{L}(A)\}$.*
Moreover, if the horizontal languages of A are represented by finite word automata, then A' is computable in time $O(\|A\|)$ and $\|A'\| \in O(\|A\|)$.

Proof. Suppose that $A = (\Lambda, Q, F, \Delta, =_A, \neq_A)$. For each rule $r \in \Delta$, we suppose the horizontal language of r is represented by a finite word automaton $A_r = (Q, P_r, F_r, p_i^r, \delta_r)$, where Q is the alphabet, P_r is the finite set of states, F_r is the set of final states, p_i^r is the initial state, and $\delta_r \subseteq P_r \times Q \times P_r$ is the set of transitions. We also assume that there is no ϵ-transitions and that the sets P_r are pairwise disjoint, and disjoint from Q, for all $r \in \Delta$. We now construct $A' = (\Lambda, Q', F', \Delta', =_{A'}, \neq_{A'})$ as follows:

- $Q' = \{q_\perp\} \cup Q \cup \bigcup_{r \in \Delta} P_r$ for a fresh state $q_\perp$
- $F' = F$

- Δ' is defined by the following inference rules:

1. $\dfrac{}{\bot \to q_\bot \in \Delta'}$ 2. $\dfrac{\alpha(L) \to q \in \Delta \text{ and } \epsilon \in L}{(\alpha - \mathsf{cons})(q_\bot) \to q \in \Delta'}$

3. $\dfrac{(p_i^r, q, p_2) \in \delta_r \text{ and } (p_2, q', p_3) \in \delta_r}{\mathsf{cons}(q, q') \to p_3 \in \Delta'}$ 4. $\dfrac{(p_1, q, p_2) \in \delta_r}{\mathsf{cons}(p_1, q) \to p_2 \in \Delta'}$

5. $\dfrac{r = \alpha(L) \to q \in \Delta \text{ and } p_f \in F_r}{(\alpha - \mathsf{cons})(p_f) \to q \in \Delta'}$

- $=_{A'}$ is equal to $=_A$, and $\neq_{A'}$ is equal to $\neq_A$

Subtrees rooted by labels of Λ necessarily evaluate to states of Q, while subtrees rooted by cons evaluate to state of P_r, for some $r \in \Delta$. Below a node rooted by a symbol of Λ, there is either a node $\bot$ (encoding of the empty hedge), or a subtree whose root is labeled cons (encoding a non-empty hedge). These two cases are covered by rules 2 and 5. Horizontal transitions are simulated by rules 3 and 4. In particular, rule 3 simulates an horizontal transition for a word of state of the form qq', while rule 4 simulates an horizontal transition for a word of state of the form wq, where $|w| > 1$. □

Theorem 5.7.3 *Emptiness of vertically bounded unranked TAGEDs over an infinite alphabet is decidable in 2NEXPTIME.*

Proof. This is mainly because the encoding preserves the descendant relations. Hence if (A, k) is a vertically bounded unranked TAGED, and A' is its encoding (constructed as in Proposition 5.7.2), then we also have $\mathcal{L}(A', k) = \{\Psi(t) \mid t \in \mathcal{L}(A, k)\}$. The conclusion follows from Theorem 5.7.1. □

5.8 Conclusion

In this chapter, a new class of tree automata has been introduced. An expressive fragment has been proved to be decidable (wrt to the emptiness problem), and its MSO counterpart has been defined.
It would be interesting to consider the application of TAGEDs to the solvability of first-order disunification problems, with (regular) membership constraints. Dealing with membership constraints has been done in several papers. In (CD94), the authors prove solvability of first-order formulas whose atoms are either equations between terms or membership constraints $t \in L$ where L is a regular tree language and t a term (with possibly variables). In (KM06), the authors propose an algorithm to solve iterated matching of hedges against terms with flexible arity symbols, one-hole context and sequence variables constrained to range over a regular language. We could extend the logic of (CD94) with context variables (with arbitrarily many holes, and membership constraints) to allow arbitrary depth matching. Context unification is still an open problem, but motivated by XML tasks, we would not need to do full context unification. Imposing a strong linearity condition on context variables would be sufficient. Even with strong linearity restriction, solvability of first-order formulas should be undecidable. Considering an existential fragment with a natural embedding to vbTAGEDs would lead to decidability. A first attempt

has been done in (FTT08). It would be also interesting to use TAGEDs as a representation of the set of solutions, in order to decide finiteness of the set of solutions, or to enumerate them for instance. This can be done, for a system S with free variables $x_1, \ldots, x_n$, by defininining the set of hedges $\sigma(x_1) \ldots \sigma(x_n)$ such that σ is a solution of S. This raises the following questions:

- given a unary context C, and a TAGED-definable language L, is $\{t \mid C[t] \in L\}$ TAGED-definable?
- given a TAGED-definable hedge language L, is the following TAGED-definable?
$$\{t_1 \ldots t_{n-1} \mid \exists t_n,\ t_1 \ldots t_{n-1} t_n \in L\}$$

A long-standing open problem is to decide if the image of a regular tree language by a homomorphism is regular. This problem has been partially answered recently in (GMT08), where two interesting classes of homomorphisms are considered. Deciding whether the language recognized by a TAGED is regular would imply decidability of a larger fragment of the homomorphism problem. A first step would be to decide if the set of ground instances of a term with context variables (occuring in a linear manner) and term variables (which may occur many times), with regular membership constraints on context or term variables, is regular. It is non-trivial since for instance a TAGED may do several equality tests, but when taking its union with the tree automaton which recognizes all the trees, equality tests become useless, but are still present in the union.
Concerning the emptiness problem, complexity lower bounds are still missing for negative TAGEDs and vertically bounded TAGEDs. Deciding emptiness of full TAGEDs is still open.
It could be interesting to consider more general tests $=_R$, given by recognizable relations R on trees (as, in particular, tree (dis)equality is a binary recognizable relation). In particular, two subtrees t_1, t_2 which evaluate to states q_1, q_2 such that $q_1 =_R q_2$ would have to satisfy $(t_1, t_2) \in R$. Since it includes the emptiness problem of full TAGEDs, which is still open, in a first step, one could consider relations such that $=_R \subseteq id_Q$.
Finally, we would like to mention the on-going work on the modeling of cryptographic protocols by term rewriting systems and positive TAGEDs satisfying $=_A \subseteq id_Q$ (VJK08). Sequences of messages sent on a network are modeled by terms over symbols that represent cryptographic primitives. All the possible executions of a cryptographic protocol are then modeled by a term language definable by a TAGED. Testing emptiness of the closure of this language modulo some relevant equational theory allows to see whether the protocol is safe or not. This closure is not computable when the equational theory is given by an arbitrary term rewriting systems (TRS) and defining decidable relevant subclasses is still an ongoing but promising work.

TREE QUERY LOGIC

6

CONTENTS

THIS chapter introduces the Tree Query Logic (TQL) adapted from (CG04) to the context of ordered unranked trees. TQL formulas are built over variables that can be bound by trees, recursion via a least fixpoint operator, Boolean connectives, and the operations of the hedge algebra. Decidability of a powerful fragment with variables is proved by reduction to emptiness of vbTAGEDs.

6.1 INTRODUCTION

We consider the TQL logic defined in (CG04) and adapt it to the context of unranked ordered trees. This was mentioned as an open issue in (CG04, Gen06, Bon06).
TQL formulas are interpreted on hedges over an infinite alphabet. It allows for instance to take data values of XML documents into account. The logic integrates Boolean connectives, the operators of the hedge algebra (see Section 2.1.4), tree variables that are bound by trees, and a fixpoint operator for recursion. Variables have been introduced to define queries of arbitrary arities and allow one to test tree equalities. For instance, the formula $a[X|X]$ defines the set of all trees of the form $a(t,t)$. By the use of negation, disequality tests can also be performed. For instance, $a[X|\neg X]$ defines the set of all trees of the form $a(t,t')$, where $t \neq t'$.
We give a polynomial-time model-checking algorithm for TQL formulas. Then we investigate the satisfiability of TQL formulas in presence of tree variables. The full TQL logic is undecidable, as it can define the intersection of two context-free word grammars. A natural restriction is to guard recursion by hedge rooting, meaning that the recursion is applied on hedges whose heights are strictly smaller. Guarded TQL formulas without tree variables capture MSO over hedges. The presence of tree variables, when repeated in the formula, complicates the satisfiability problem, since one can express for instance that two non-empty paths starting from a node of a tree lead to two isomorphic subtrees, which goes beyond MSO. We define an expressive TQL fragments with variables, in which negations of tree variables can occur in a bounded manner only. This fragment is proved to be decidable by reduction to the emptiness test of vbTAGEDs. This reduction is non-standard and new, as it deals with tree variables. In particular the vbTAGED constructed from the TQL formula is non-deterministic. Decidability of the full guarded fragment is still open.

Related Work In (CG04), TQL allows hedge variables (variables that can be bound by hedges). The satisfiability problem in presence of hedge variables contains the very hard problem of word equations, first solved by Makanin in 1977. Here we prefer to focus on the ability of TQL to deal with trees, and therefore restrict TQL to tree variables, which is still in the spirit of (CG04). In addition, the TQL logic of (CG04) allows label variables and both kind of variables (for hedges and labels) can be existentially and universally quantified. We do not consider this extension here as it makes TQL undecidable.
The satisfiability problem for several fragments of TQL has already been considered in (Bon06, BTT05), but in the context of unordered trees and without variables. The techniques used in those papers are also different.
TQL for ordered trees can be seen as an extension of the (recursive) pattern-language of XDuce (HP03b). The main difference here is that we allow Boolean operators and drop the *linear* condition for variables of XDuce, meaning that the same variable can occur many times in a formula. The pattern-matching mechanism of CDuce (BCF03a) extends the one from XDuce with Boolean operations and weaker conditions on variables. However, no equality tests between terms can be performed making TQL more powerful.
Because we consider an infinite alphabet and we allow for equality tests between trees, we can, as a side effect, model data values of XML documents. However the expressive power of our TQL logic with respect to data value comparison allows

only to compare data values to a fixed data. For instance, we can express that for all subtrees whose root is labeled "book", there is one author named as "Church", but we cannot express that there is at least two authors with the same first name. FO fragments over unranked trees with the ability to compare data-values have been considered in (BDM*06, BMS*06). However with respect to this concern, TQL is weaker than these formalisms.
By the presence of a fixpoint operator, TQL seems to be related to the μ-calculus (Koz83), which is known to be decidable and as expressive as MSO over unranked trees, when it includes past transitions (BL05). There are major differences though: the presence, in TQL, of variables and the composition operator $.|.$ of the hedge algebra. The embedding of TQL formulas into TAGEDs is also different from the standard embedding of the μ-calculus into tree automata. In particular, the presence of variables makes the automaton non-deterministic.

Organization of the chapter Section 6.2 gives the syntax and the semantics of TQL. Examples are presented in Section 6.3. A model-checking algorithm for TQL is considered in Section 6.4. The decidable bounded fragment of TQL is defined in Section 6.5, and its embedding into vbTAGEDs is presented and proved in Section 6.6.

6.2 Syntax and Semantics

We define TQL formulas and their interpretation on hedges over an infinite alphabet.

6.2.1 Syntax

Let Λ be a countable alphabet, $\mathcal{X}_{\text{tree}}$ be a countable set of tree variables ranged over by X, Y and $\mathcal{X}_{\text{rec}}$ a countable set of recursion variables ranged over by ξ. Let α be a (co)finite set of labels from Λ. Formulas ϕ of TQL are defined by the following grammar:

$$
\begin{array}{rll}
\phi ::= & 0 & \text{empty hedge} \\
& \top & \text{truth} \\
& \alpha[\phi] & \text{extension} \\
& \phi|\phi & \text{composition} \\
& \neg\phi & \text{negation} \\
& \phi \vee \phi & \text{disjunction} \\
& X & \text{tree variable, } X \in \mathcal{X}_{\text{tree}} \\
& \xi & \text{recursion variables, } \xi \in \mathcal{X}_{\text{rec}} \\
& \mu\xi.\phi & \text{least fixpoint} \\
& \phi^* & \text{Kleene star}
\end{array}
$$

The following priorities apply to the operators: $\vee < \wedge < | < \neg$. For instance, $\phi_1 \vee \phi_2|\phi_3 = \phi_1 \vee (\phi_2|\phi_3)$, $\phi_1 \wedge \phi_2|\phi_3 = \phi_1 \wedge (\phi_2|\phi_3)$, and $\neg\phi_1|\phi_2 = (\neg\phi_1)|\phi_2$. We define the following abbreviations:

$$
\begin{array}{lcl}
\phi \wedge \phi' & \stackrel{\text{def}}{=} & \neg(\neg\phi \vee \neg\phi') \\
\bot & \stackrel{\text{def}}{=} & \neg\top
\end{array}
$$

The symbol μ is the binder for recursion variables and the notions of bound and free recursion variables $\text{Var}_{\text{rec}}(\phi)$ and $\text{FVar}_{\text{rec}}(\phi)$ respectively of a formula ϕ are defined as usual (like in FO with the binder $\exists$). Similarly, the set of tree variables occurring in ϕ is denoted by $\text{Var}(\phi)$. To ensure the existence of fixpoint, we will assume that in formulas $\mu\xi.\phi$, the recursion variable ξ occurs under an even number of negations. A formula ϕ is said to be *recursion-closed* if all the occurrences of its recursion variables are bound, *ie* $\text{FVar}_{\text{rec}}(\phi) = \varnothing$. A TQL *sentence* is a formula that does not contain tree variables and is recursion-closed. The set of subformulas of a formula ϕ is denoted $\text{Sub}(\phi)$, and the size of ϕ is the number of its subformulas, *ie* $\|\phi\| = |\text{Sub}(\phi)|$.
A TQL formula ϕ is *guarded* if for any of its subformula $\mu\xi.\phi'$, the variable ξ occurs in the scope of some extension operator $\alpha[\,]$ in ϕ'.
We assume from now on that recursion variables are bound only once in formulas. We may write $a[\phi]$ instead of $\{a\}[\phi]$.

Fisher-Ladner Closure Let ϕ, ϕ' be TQL formulas and ξ a recursion variable. The formula $\phi[\xi \mapsto \phi']$ is defined as the formula ϕ in which every occurrence of ξ has been substituted by ϕ'. The *Fisher-Ladner* closure $\text{FL}(\phi)$ of ϕ is the set of all subformulas of ϕ where recursion variables are unfolded once. Formally, it is the smallest set that contains ϕ and is closed by the following binary relation $\rightarrow_{fl}$:

$$\begin{array}{llll} \psi_1 \wedge \psi_2 & \rightarrow_{fl} & \psi_1 & \psi_1 \wedge \psi_2 \rightarrow_{fl} \psi_2 \\ \psi_1 \vee \psi_2 & \rightarrow_{fl} & \psi_1 & \psi_1 \vee \psi_2 \rightarrow_{fl} \psi_2 \\ \psi_1 | \psi_2 & \rightarrow_{fl} & \psi_1 & \psi_1 | \psi_2 \rightarrow_{fl} \psi_2 \\ \neg\psi & \rightarrow_{fl} & \psi & \alpha[\psi] \rightarrow_{fl} \psi \\ \mu\xi.\psi & \rightarrow_{fl} & \psi[\xi \mapsto \mu\xi.\psi] & \end{array}$$

Observe that $\text{FL}(\phi)$ is finite and $|\text{FL}(\phi)| = |\text{Sub}(\phi)| - |\text{Var}_{\text{rec}}(\phi) \backslash \text{FVar}_{\text{rec}}(\phi)|$, hence $|\text{FL}(\phi)| \leq \|\phi\|$. Note also that if ϕ is recursion-closed, then every formula of $\text{FL}(\phi)$ is recursion-closed too. For instance, if $\phi = \mu\xi.(a[\xi] \vee \mathsf{0})$, then $\text{FL}(\phi) = \{\phi,\ a[\phi] \vee \mathsf{0},\ a[\phi],\ \mathsf{0}\}$. Every formula of $\text{FL}(\phi)$ is called an *unfolded subformula*.

Partial order on hedges and formulas The *recursion-depth* $\text{rd}(\phi)$ of a TQL formula ψ is the maximal number of nested fixpoint binders $\mu\xi$ that are not in the scope of a tree extension $\alpha[.]$. It is inductively defined by:

$$\begin{array}{l} \text{rd}(\phi_1|\phi_2) = \text{rd}(\phi_1 \wedge \phi_2) = \text{rd}(\phi_1 \vee \phi_2) = max(\text{rd}(\phi_1), \text{rd}(\phi_2)) \\ \text{rd}(\mathsf{0}) = \text{rd}(\top) = \text{rd}(\xi) = \text{rd}(X) = 0 \\ \text{rd}(\mu\xi.\phi) = 1 + \text{rd}(\phi) \qquad \text{rd}(\alpha[\phi]) = 0 \end{array}$$

Given two hedges $h, h' \in \mathcal{H}(\Lambda)$, we let $h' \prec_{\mathcal{H}(\Lambda)} h$ if one of the following conditions holds:

- there exist $h_1, h_2 \in \mathcal{H}(\Lambda)$ such that $h_1 \neq \mathsf{0}$ or $h_2 \neq \mathsf{0}$, and $h = h_1|h'|h_2$;
- there exist $h_1, h_2 \in \mathcal{H}(\Lambda)$, $a \in \Lambda$, and $u \in \text{Dom}(h)$ such that $h|_u = a(h_1|h'|h_2)$.

Observe that $\prec_{\mathcal{H}(\Lambda)}$ is a well-founded partial order on hedges. Given two hedges $h, h' \in \mathcal{H}(\Lambda)$ and two TQL formulas ϕ, ϕ', we let $(h, \phi) \prec (h', \phi')$ if one of the following conditions holds:

- $h \prec_{\mathcal{H}(\Lambda)} h'$

- $h = h'$ and $\mathrm{rd}(\phi) < \mathrm{rd}(\phi')$
- $h = h'$ and $\mathrm{rd}(\phi) = \mathrm{rd}(\phi')$ and ϕ is a strict subformula of ϕ'

Proposition 6.2.1 *$\prec$ is a strict partial order on $\mathcal{H}(\Lambda) \times TQL$.*

Proof. This is because it is the lexicographic composition of three partial orders. □

In particular, for all guarded formulas ψ, and all hedges h, we have $(h, \psi[\xi \mapsto \mu\xi.\psi]) \prec (h, \mu\xi.\psi)$, since $\mathrm{rd}(\psi[\xi \mapsto \mu\xi.\psi]) = \mathrm{rd}(\mu\xi.\psi) - 1 < \mathrm{rd}(\mu\xi.\psi)$, as ψ is guarded. For instance, with $\phi = \mu\xi.\mu\xi'.(a[\xi|\xi'] \vee \mathsf{0})$, we have $\mathrm{rd}(\phi) = 2$, but $\mathrm{rd}(\mu\xi'.(a[\phi|\xi'] \vee \mathsf{0})) = 1$.

6.2.2 Semantics

TQL formulas ϕ are interpreted as sets of hedges $[\![\phi]\!]^{\rho,\delta} \subseteq \mathcal{H}(\Lambda)$, modulo some assignment (also called valuation) $\rho : \mathrm{Var}(\phi) \to \mathcal{T}_{unr}(\Lambda)$ of tree variables into trees and some assignment $\delta : \mathrm{FVar}_{\mathrm{rec}}(\phi) \to 2^{\mathcal{H}(\Lambda)}$ of the free recursion variables into sets of hedges. The semantics is given in Figure 6.1. When ϕ is recursion-closed, we omit the subscript δ, and when it is a sentence, we omit the subscript ρ as well. The semantics of the fixpoint operator $[\![\mu\xi.\phi]\!]^{\rho,\delta}$ is the least fixpoint of the functional

$$\begin{array}{rcccc} F_\phi & : & 2^{\mathcal{H}(\Lambda)} & \to & 2^{\mathcal{H}(\Lambda)} \\ & & S & \mapsto & [\![\phi]\!]^{\rho,\delta[\xi \mapsto S]} \end{array}$$

Since ξ does not occur under an even number of negations, F is monotonic for inclusion, which ensures the existence of a least fixpoint for F. See (EF05) for more details on fixpoints.

$$\begin{array}{ll} [\![\mathsf{0}]\!]^{\rho,\delta} & = \{\mathsf{0}\} \\ [\![\top]\!]^{\rho,\delta} & = \mathcal{H}(\Lambda) \\ [\![\alpha[\phi]]\!]^{\rho,\delta} & = \{a(h) \mid h \in [\![\phi]\!]^{\rho,\delta},\ a \in \alpha\} \\ [\![\phi|\phi']\!]^{\rho,\delta} & = \{h|h' \mid h \in [\![\phi]\!]^{\rho,\delta}, h' \in [\![\phi']\!]^{\rho,\delta}\} \\ [\![\neg\phi]\!]^{\rho,\delta} & = \mathcal{H}(\Lambda) \backslash [\![\phi]\!]^{\rho,\delta} \\ [\![\phi \vee \phi']\!]^{\rho,\delta} & = [\![\phi]\!]^{\rho,\delta} \cup [\![\phi']\!]^{\rho,\delta} \\ [\![X]\!]^{\rho,\delta} & = \{\rho(X)\} \\ [\![\xi]\!]^{\rho,\delta} & = \delta(\xi) \\ [\![\mu\xi.\phi]\!]^{\rho,\delta} & = \bigcap\{S \subseteq \mathcal{H}(\Lambda) \mid [\![\phi]\!]^{\rho,\delta[\xi \mapsto S]} \subseteq S\} \\ [\![\phi^*]\!]^{\rho,\delta} & = \{\mathsf{0}\} \cup \bigcup_{i>0} \underbrace{[\![\phi]\!]^{\rho,\delta}|\ldots|[\![\phi]\!]^{\rho,\delta}}_{i\ times} \end{array}$$

Figure 6.1: Semantics of TQL formulas

The *satisfaction relation* $\models$ is defined by $h, \rho, \delta \models \phi$ if $h \in [\![\phi]\!]^{\rho,\delta}$, for all hedges $h \in \mathcal{H}(\Lambda)$ and assignments ρ, δ. Another characterization of the least fixpoint is given by the following lemma:

Lemma 6.2.2

$$h, \rho, \delta \models \mu\xi.\phi \ \textit{iff}\ h, \rho, \delta \models \phi[\xi \mapsto \mu\xi.\phi]$$

in which $\phi[\xi \mapsto \mu\xi.\phi]$ is the formula ϕ where each occurrence of ξ has been replaced by $\mu\xi.\phi$.

Proof. We start by the following fact: if ϕ_1, ϕ_2 are TQL formulas, then $[\![\phi_1]\!]^{\rho,\delta[\xi\mapsto[\![\phi_2]\!]^{\rho,\delta}]} = [\![\phi_1[\xi \mapsto \phi_2]]\!]^{\rho,\delta}$. It follows from the semantics of TQL formulas.
Now, since $[\![\mu\xi.\phi]\!]^{\rho,\delta}$ is the least fixpoint of the functional F_ϕ, we have

$$F_\phi([\![\mu\xi.\phi]\!]^{\rho,\delta}) \;=\; [\![\mu\xi.\phi]\!]^{\rho,\delta},\ \textit{ie}\quad [\![\phi]\!]^{\rho,\delta[\xi\mapsto[\![\mu\xi.\phi]\!]^{\rho,\delta}]} \;=\; [\![\mu\xi.\phi]\!]^{\rho,\delta}$$

By the previous fact, we get: $[\![\mu\xi.\phi]\!]^{\rho,\delta} = [\![\phi[\xi \mapsto \mu\xi.\phi]]\!]^{\rho,\delta}$. The conclusion follows from the definition of the satisfaction relation. □

From Lemma 6.2.2 and direct applications of the semantics of TQL formulas, several properties of the satisfaction relation can be proved. They are summarized in Figure 6.2.

$$\begin{array}{lll}
h,\rho,\delta \models \mathsf{0} & \text{iff} & h = \mathsf{0} \\
h,\rho,\delta \models \alpha[\phi] & \text{iff} & \exists h' \in \mathcal{H}(\Lambda), \exists a \in \alpha,\ h = a(h') \text{ and } h',\rho,\delta \models \phi \\
h,\rho,\delta \models \phi|\phi' & \text{iff} & \exists h_1, h_2 \in \mathcal{H}(\Lambda),\ h = h_1|h_2 \text{ and } h_1,\rho,\delta \models \phi_1 \text{ and } h_2,\rho,\delta \models \phi_2 \\
h,\rho,\delta \models \neg\phi & \text{iff} & h,\rho,\delta \not\models \phi \\
h,\rho,\delta \models \phi \vee \phi' & \text{iff} & h,\rho,\delta \models \phi \text{ or } h,\rho,\delta \models \phi' \\
h,\rho,\delta \models X & \text{iff} & h = \rho(X) \\
h,\rho,\delta \models \xi & \text{iff} & h \in \delta(\xi) \\
h,\rho,\delta \models \mu\xi.\phi & \text{iff} & h,\rho,\delta \models \phi[\xi \mapsto \mu\xi.\phi] \\
h,\rho,\delta \models \phi^* & \text{iff} & \exists n \in \mathbb{N}, \exists h_1,\ldots,h_n \in \mathcal{H}(\Lambda), \left\{ \begin{array}{l} h = h_1|\ldots|h_n \\ \forall i \in \{1,\ldots,n\},\ h_i,\rho,\delta \models \phi \end{array} \right.
\end{array}$$

Figure 6.2: Properties of the Satisfaction Relation

TQL can be viewed as a query language, thanks to tree variables. However, TQL queries output tuples of trees instead of tuple of nodes, and those trees may not be subtrees of the input. For instance, the formula $a[\neg X]$ is satisfied by any tree of the form $a(h)$ under some valuation ρ such that $\rho(X) \neq h$. Viewed as a unary query, the set of answers of this query on $a(h)$ is infinite. However it can be finitely represented as the complement of $\{h\}$. Formally, a recursion-closed TQL formula $\phi(X_1,\ldots,X_n)$ with n free variables $X_1,\ldots,X_n$ defines the n-ary query

$$\begin{array}{rcccl}
\mathcal{Q}(\phi) & : & \mathcal{H}(\Lambda) & \to & 2^{\mathcal{T}_{unr}(\Lambda)^n} \\
 & & h & \mapsto & \{(\rho(X_1),\ldots,\rho(X_n)) \mid h,\rho \models \phi\}
\end{array}$$

6.3 EXAMPLES

In this section, we give a series of examples of Boolean and n-ary queries. In particular, the last example shows how to translate an extended DTD into a TQL formula, making TQL as expressive as MSO over hedges.

1. The formula $\Lambda[\top]$ is interpreted as the set of trees over Λ.

2. The following formula ϕ is interpreted as the set of hedges of length at least 1, such that the root of the first tree is labeled a:

$$\phi = a[\top]|\top$$

3. The formula ϕ_s is interpreted as the set of hedges $\{a[h_1]|\ldots|a[h_n] \mid h_i \in \mathcal{H}(\Lambda), n \geq 0\}$. Note that it is equivalent to $(a[\top])^*$:

$$\phi_s = \mu\xi.(a[\top]|\xi \ \vee \ 0)$$

4. The formula ϕ_{dtd} is interpreted as the set of hedges defining employees by their name, the department they work in, and their manager whereas the formula ϕ_{dd} expresses that an employee X occurs twice in the database:

$$\begin{array}{rcl} \phi_{dtd} & = & (employee[name[\Lambda[0]] \,|\, dpt[\Lambda[0]] \,|\, manager[\Lambda[0]]])^* \\ \phi_{dd} & = & \phi_{dtd} \wedge \top \,|\, employee[X] \,|\, \top \,|employee[X] \,|\, \top \end{array}$$

5. The models of the formula $\phi_{path(a),0}$ are hedges with a path labeled by a's from one of the roots to some leaf (*ie* the empty hedge 0):

$$\phi_{path(a),0} = \mu\xi.(\top|a[\xi]|\top \ \vee \ 0)$$

6. The formula ϕ_{odd} is interpreted as the set of hedges having an odd number of nodes:

$$\begin{array}{rrcl} & \phi_{odd} & = & \mu\xi_o.(\Lambda[\phi_{even}(\xi_o)]|\phi_{even}(\xi_o) \ \vee \ \Lambda[\xi_o]|\xi_o) \\ \text{where} & \phi_{even}(\xi_o) & = & \mu\xi_e.(0 \ \vee \ (\Lambda[\xi_o]|\xi_e \ \vee \ \Lambda[\xi_e]|\xi_o)) \end{array}$$

7. Let us denote $\phi_{path(L),\psi} = \mu\xi.(\top|L[\xi]|\top \ \vee \ \psi)$ the formula whose models are hedges containing a path labeled by elements from L from one of the roots to a hedge satisfying ψ. The models of the following formula are the hedges having two paths labeled respectively by as and by bs from two of the top-level nodes, those two paths leading to some identical subtrees:

$$a[\phi_{path(a),X}]|b[\phi_{path(b),X}] \ \vee \ b[\phi_{path(b),X}]|a[\phi_{path(a),X}]$$

8. The formula $\phi_{id\_not\_key}$ is interpreted as the set of hedges for which two nodes labeled id have identical subtrees (roughly speaking "the (data) value of the element *id* cannot be used as a key in this XML document"):

$$\phi_{id\_not\_key} = \top|\Lambda[\phi_{path(\Lambda),id[X]}]|\top|\Lambda[\phi_{path(\Lambda),id[X]}]|\top$$

9. The following formula states that two trees $employee$ have identical subtrees rooted by dpt but different subtrees rooted by $manager$:

$$\begin{array}{ll} \phi_{dtd} \wedge \top & |\, employee[\top\,|dpt[X]|manager[Y]] \,|\, \top \\ & |\, employee[\top\,|dpt[X]|manager[\neg Y]] \,|\, \top \end{array}$$

A hedge satisfying this formula may be considered as ill-formed assuming the existence of some functional dependency stating that $department$ has only one $manager$.

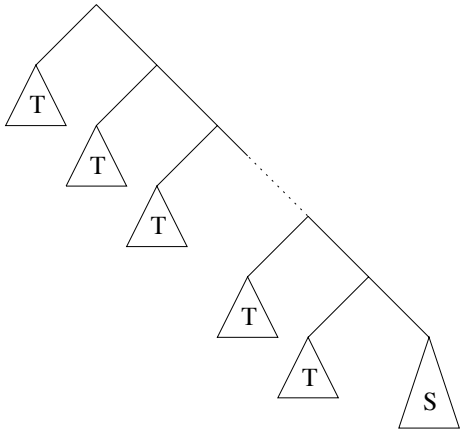


Figure 6.3: A tree with $T \neq S$

10. Models of the formula ϕ_{branch} are the trees whose shapes are described on Figure 6.3:

$$\phi_{branch} = \Lambda[X | \mu\xi.(\Lambda[X \mid \xi] \vee \neg X \wedge \Lambda[\top])]$$

ϕ_s and ϕ_{odd} are the only two formulas that are not guarded; however ϕ_s is equivalent to $a[\top]^*$, which is guarded and ϕ_{odd} to the following guarded formula

$$\begin{array}{rrcl} & \phi_{odd} & = & \mu\xi_o.\phi_{even}(\xi_o) \mid \Lambda[\phi_{even}(\xi_o)] \mid \phi_{even}(\xi_o) \\ \text{where} & \phi_{even}(\xi_o) & = & \mu\xi_e.\Lambda[\xi_o]^* \mid (\Lambda[\xi_e] \mid \Lambda[\xi_o]^* \mid \Lambda[\xi_e] \mid \Lambda[\xi_o]^*)^* \end{array}$$

The idea is that an odd hedge can be decomposed as the sequence of an even hedge, an odd tree, and an even hedge, while an even hedge always contains an even number of odd trees.
However the formula $\mu\xi.(a[0]|\xi|b[0] \vee 0)$ is neither guarded nor equivalent to any guarded formula.

Extended DTD Embedding Let $d' = (d, T)$ be an extended DTD over a finite alphabet Σ (where T is a finite set and $d = ((s, \sigma), \delta)$ is a DTD over $\Sigma \times T$ with starting symbol $(s, \sigma) \in \Sigma \times T$, see Section 2.7.2 for a formal definition of extended DTDs). The extended DTD d' defines the same tree language as the guarded TQL formula $(\!|(s, \sigma)|\!)_\varnothing$ defined below, over recursion variables $\xi_{a,\sigma}$, $(a, \sigma) \in \Sigma \times T$. For any set of symbols $L \subseteq \Lambda \times T$, and any regular expression e over $\Sigma \times T$, $(\!|e|\!)_L$ is inductively defined by:

$$\begin{array}{lcl} (\!|\epsilon|\!)_L & = & 0 \\ (\!|e_1 + e_2|\!)_L & = & (\!|e_1|\!)_L \vee (\!|e_2|\!)_L \\ (\!|e_1.e_2|\!)_L & = & (\!|e_1|\!)_L | (\!|e_2|\!)_L \\ (\!|e^*|\!)_L & = & (\!|e|\!)_L^* \\ (\!|(a, \sigma)|\!)_L & = & \begin{cases} a[\ \xi_{a,\sigma}\ ] & \text{if } (a, \sigma) \in L \\ a[\ \mu\xi_{a,\sigma}.(\!|\delta(a, \sigma)|\!)_{L \cup \{(a,\sigma)\}}\ ] & \text{if } (a, \sigma) \notin L, \end{cases} \end{array}$$

Informally, the recursion of the DTD is simulated by recursion variables. The set L is intended to memorize which symbols (a, σ) have already been met. If it is the first time, a new recursion variable $\xi_{a,\sigma}$ is introduced via the binder $\mu\xi_{a,\sigma}$,

otherwise we are in the scope of $\mu\xi_{a,\sigma}$, so that the current formula can be translated to $\xi_{a,\sigma}$.
We illustrate the construction for a simple DTD (and omit the PCDATA terminals):

$$\begin{array}{lcllcl} books & \rightarrow & book^* & title & \rightarrow & \epsilon \\ book & \rightarrow & title.author & name & \rightarrow & \epsilon \\ author & \rightarrow & name.books & & & \end{array}$$

It is translated into the formula

$$\begin{array}{l} books[\ \ \mu\xi_{books}.(\ \ book[\ \ \mu\xi_{book}.(\ \ title[\mu\xi_{title}.\mathsf{0}] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad |\ author[\mu\xi_{author}.(\ \ name[\mu\xi_{name}.\mathsf{0}] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad |books[\xi_{books}])] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad)^* \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad] \end{array}$$

which is equivalent to:

$$books[\ \mu\xi_{books}.(\ book[\ title[\mathsf{0}]\ |\ author[\ name[\mathsf{0}]|books[\xi_{books}]\ ]\ ]\ )^*]$$

Proposition 6.3.1 *An extended DTD $d' = (d, T)$ over a finite alphabet Σ, with a starting symbol $(s, \sigma) \in \Sigma \times T$ is equivalent to the guarded TQL formula $(\!|s, \sigma|\!)_\varnothing$, ie:*

$$\mathcal{L}(d') \ = \ [\![(\!|s, \sigma|\!)_\varnothing]\!]$$

Since extended DTDs captures hedge automata (Theorem 2.7.1), we get the following corollary:

Corollary 6.3.2 *Guarded TQL formulas can define unranked tree languages over a finite alphabet definable by hedge automata. The translation is effectively computable in linear time if the horizontal languages of the hedge automaton are defined by regular expressions.*

In fact, only guarded recursion, disjunction, composition .|., extension $\alpha[.]$ and Kleene star are needed to capture the expressiveness of hedge automata.

6.4 Model-Checking Algorithm

In this section, we present a model-checking algorithm for recursion-closed TQL formulas (not necessarily guarded). The algorithm is very similar to the one of (Bon06), adapted for the ordered case.
We suppose that no recursion variable is bound twice. The algorithm processes the formula recursively with memoization, with cycle detection to avoid infinite loops. The only difficulty is to deal with fixpoints. The algorithm uses the fact that $h \models \mu\xi.\phi$ iff $h \models \phi[\xi \mapsto \mu\xi.\phi]$. Suppose that the algorithm checks the satisfiability of some formula $\mu\xi.\phi$ against some hedge h, and that after several computational steps, it has to check satisfiability of $\mu\xi.\phi$ against h again, then it means it is in a cycle. Hence it has to return false by definition of the least fixpoint. For instance, the formula $\mu\xi.\xi|\mathsf{0}$ can generate a cycle, as $h \models \mu\xi.\xi|\mathsf{0}$ iff $h \models (\mu\xi.\xi|\mathsf{0})|\mathsf{0}$ iff $h \models \mu\xi.\xi|\mathsf{0}$. So the algorithm has to detect cycles. It can be done by associating

with each recursion variable ξ the current hedge when the binder $\mu\xi$ was met for the last time in the model-checking process. An association table from recursion variables to hedges stores this information.
Let ϕ be a recursion-closed variable, and Δ_0 an empty association table. On a hedge h, modulo some assignement ρ, CHECK$(\phi, h, \rho, \Delta_0)$ outputs true iff $h \in [\![\phi]\!]^\rho$.

CHECK(ϕ, h, ρ, Δ)

switch
- **case** $\phi = \top$:
  - **return** true
- **case** $\phi = 0$:
  - **if** $h = 0$
    - **then return** true
    - **else return** false
- **case** $\phi = X$:
  - **if** $h = \rho(X)$
    - **then return** true
    - **else return** false
- **case** $\phi = \alpha[\phi']$:
  - **if** $\exists a \in \alpha,\ h = a(h')$
    - **then return** CHECK(ϕ', h', Δ)
    - **else return** false
- **case** $\phi = \neg\phi'$:
  - **return** not CHECK(ϕ', h, Δ)
- **case** $\phi = \phi_1|\phi_2$:
  - **return** $\bigvee_{h_1|h_2=h}$ CHECK$(\phi_1, h_1, \Delta) \wedge$ CHECK(ϕ_2, h_2, Δ)
- **case** $\phi = (\phi')^*$:
  - **if** $h = 0$
    - **then return** true
    - **else return** $\bigvee_{h_1|h_2=h, h_1\neq 0}$ CHECK$(\phi', h_1, \Delta) \wedge$ CHECK(ϕ, h_2, Δ)
- **case** $\phi = \phi_1 \vee \phi_2$:
  - **return** CHECK$(\phi_1, h, \Delta) \vee$ CHECK(ϕ_2, h, Δ)
- **case** $\phi = \mu\xi.\phi'$:
  - **if** $\Delta(\xi)$ is undefined
    - **then** CHECK$(\phi[\xi \mapsto \mu\xi.\phi'], h, \Delta[\xi \mapsto h])$
    - **else if** $h = \Delta(\xi)$
      - **then return** false
      - **else** CHECK$(\phi[\xi \mapsto \mu\xi.\phi'], h, \Delta[\xi \mapsto h])$

Theorem 6.4.1 *Model-checking of TQL formulas can be done in time* $O(\|\phi\|^2\|h\|^5)$.

Proof. Correctness of CHECK has already been proved in (Bon06) in the context of unordered trees, which is very similar to ordered trees, wrt to the model-checking problem. For the time complexity, if one uses memoization, then there are at most $F.N.D$ recursive calls, where F is the number of subformulas ($F = \|\phi\|$), N is the number of subhedges of h ($N = O(\|h\|^2)$), and D is the number of tables ($D = O(F.N) = O(\|\phi\|\|h\|^2)$). Each recursive call costs at most $O(\|h\|)$ instructions. The overall complexity is then $O(\|\phi\|^2\|h\|^5)$. □

In practice, we suggest the following optimizations:

- modulo a linear time preprocessing, equality tests between hedges (at lines 9 and 29) can be done in a time bounded by the length of the hedge. It suffices to associate identifiers with nodes, such that two nodes have the same identifier if their respective subtrees are equal. This can be done by a bottom-up pass on the hedge. For each subtree of the form $f(t_1, t_2)$, if the identifiers of the roots of t_1 and t_2 have been computed (we call them n_1 and n_2), then we can compute the identifier of the root of $f(t_1, t_2)$. We introduce a new identifier depending on whether $f(t_1, t_2)$ have already been met somewhere else in the hedge or not. This can be verified by using a global hash table which associates with triples (f, n_1, n_2) their identifier n. Therefore, if the binding $(f, n_1, n_2) \mapsto n$ is already in the hash table, then the current subtree $f(t_1, t_2)$ is identified by n, otherwise we add the new binding $(f, n_1, n_2) \mapsto n$ to the hashtable, for some fresh n, and identify $f(t_1, t_2)$ by n. Finally, to check equality of two hedges, it suffices to check the equality of their respective sequences of root identifiers.

- given a formula $\phi|\phi'$ and a hedge h, instead of testing each pair (h', h'') such that $h'|h'' = h$, it would be interesting to lower the number of pairs (h', h'') that have to be tested. This can be done for example by a preprocessing detecting that ϕ is always satisfied by hedges of constant length. For example, given a formula ϕ, it is possible to compute in linear time, for each subformulas ϕ', an interval $inter(\phi') = [n, m]$, where $n, m \in \mathbb{N} \cup \{+\infty\}$, meaning that models of ϕ' are hedges of length at least n and at most m. For example, if $inter(\phi_1) = [n, m]$, $inter(\phi_2) = [n', m']$, then $inter(\phi_1 \vee \phi_2) = [min(n, n'), max(m, m')]$.

Query evaluation The model-checking algorithm can be extended to a query evaluation algorithm: it suffices to process the formula recursively, and to map any subformula into a term of a relational algebra which will be evaluated in a second step with particular optimizations. This idea has been developed in (CFG02), for unordered trees and the full TQL logic including universal quantification over trees and labels. Several optimizations for the evaluation of the algebra are proposed. This algorithm can easily be adapted to ordered trees. The main difference is the interpretation of $|$, which in unordered trees corresponds to union of sets of children. The complexity however is not polynomial in the size of the output, since there is no restriction on the use of variables and negations. It might possible to restrict the use of tree variables, similarly to the composition language (see the first part of this thesis, in particular Section 3.2.2) to get a polynomial-time query evaluation algorithm. However, it is not our goal here and we next focus on the satisfiability problem.

6.5 TQL Fragments and Satisfiability

The satisfiability problem for TQL formulas is: given a recursion-closed formula ϕ, is there an assignment ρ of the tree variables of ϕ such that $[\![\phi]\!]^\rho \neq \varnothing$? This section goes towards a fragment of TQL with tree variables for which satisfiability is decidable. In particular, a decidable expressive fragment is defined, where tree variables which are below an odd number of negations have to occur in a bounded manner. Decidability is proved by embedding formulas into vertically bounded

unranked TAGEDs. In the rest of the Chapter, when we speak about TAGEDs, it is always assumed to be in their unranked settings over the infinite alphabet Λ.

6.5.1 Undecidable Fragments

It is easy to prove that TQL sentences can describe sets of hedges that are not MSO-definable; for instance, the TQL sentence $\mu\xi.(a[\mathsf{0}]|\xi|b[\mathsf{0}] \vee \mathsf{0})$ describes a "flat" hedge of the form $(a(\mathsf{0})^n b(\mathsf{0})^n)$ for $n \in \mathbb{N}$. This super expressive power has a cost:

Theorem 6.5.1 *Satisfiability for TQL sentences is undecidable.*

Proof. In Section 6.3, we saw how to translate an extended DTD into a TQL formula. Very similarly, one can translate a context-free grammar into an equivalent TQL formula. Using conjunction, it is therefore possible to reduce the emptiness problem of the intersection of two context-free grammars, which is known to be undecidable (see, for example, (SIP96)). □

Adding quantification As in (CG04), one could also consider quantification over tree variables $\exists X$ and $\forall X$ with the following semantics:

$$[\![\exists X.\phi]\!]^{\rho,\delta} = \bigcup_{t \in \mathcal{T}_{unr}(\Lambda)} [\![\phi]\!]^{\rho[X \to t],\delta} \qquad [\![\forall X.\phi]\!]^{\rho,\delta} = \bigcap_{t \in \mathcal{T}_{unr}(\Lambda)} [\![\phi]\!]^{\rho[X \to t],\delta}$$

The satisfiability problem for formulas with free variables $X_1, \ldots, X_n$ is equivalent to the one of closed formulas of the form $\exists X_1 \ldots \exists X_n \phi$ where ϕ is a recursion-closed TQL formula.
For more complicated alternation of quantifiers, one can easily adapt the proof from (CT01) about the undecidability of the fragment of TQL without Kleene star, recursion and tree variable but with quantification over labels to prove that

Theorem 6.5.2 *Satisfiability for recursion-closed TQL formulas with quantification is undecidable (this holds even for recursion-free formulas).*

6.5.2 The Bounded Fragment

We consider a strict but powerful fragment of guarded TQL formulas which allow one to use tree variables. Informally, in *bounded TQL formulas*, variables can occur in recursion only in a restricted manner: intuitively a formula is bounded if there exists a bound on the number of disequality tests performed on a root-to-leaf path to (non-deterministically) verify that a hedge is a model of the formula. Before defining bounded TQL formulas formally, let us give some intuitive examples. The formula:

$$\phi_1 \;=\; a[X] \mid \mu\xi.(a[\neg X \wedge \xi] \vee \mathsf{0})$$

is not bounded since in order to check that a hedge h belongs to $[\![\phi_1]\!]^\rho$, for some assignment $\rho : X \mapsto t$, one has to check that h is of the form $a(t)|a(a(a(\ldots a(\mathsf{0}))))$, and that t is different from each subtree of the right tree of h. This latter verification requires as many disequality tests as the height of $a(a(a(\ldots a(\mathsf{0}))))$.
However, the following formula is bounded:

$$\phi_2 \;=\; a[X] \mid \mu\xi.(a[\neg X] \mid \xi \;\vee\; \mathsf{0})$$

since at most one disequality test per root-to-leaf path has to be done.
The notion of disequality test is related to the notion of negative occurrences of tree variables. We say that a tree variable occurs negatively (resp. positively) in a formula ϕ if it occurs under an odd number (resp. even number) of negations. Similarly, we can define positive or negative occurrences of connectives $\wedge, \vee, |$. Let ϕ be a recursion-closed TQL formula. The number of *vertical variable occurrences* of $\psi \in \text{Sub}(\phi)$, denoted $\text{Vert}_\phi(\psi)$, is intuitively the maximal number of times a negative occurrence of variables can occur along the same root-to-leaf path, when checking (non-deterministically) that a hedge is a model of ϕ. Formally, consider the system S_ϕ of recursive equations over the semiring $(\mathbb{N} \cup \{+\infty\}, max, +, 0, 1)$ and variables Z_ψ, for all $\psi \in \text{FL}(\phi)$:

$$Z_0 = 0 \qquad Z_\top = 0 \qquad Z_{\alpha[\psi]} = Z_\psi \qquad Z_{\psi^*} = Z_\psi \qquad Z_{\neg\psi} = Z_\psi$$
$$Z_{\mu\xi.\psi} = Z_{\psi[\xi \mapsto \mu\xi.\psi]} \qquad Z_{\psi_1|\psi_2} = max\,(Z_{\psi_1}, Z_{\psi_2})$$
$$Z_X = \begin{cases} 1 & \text{if } X \text{ occurs negatively in } \phi \\ 0 & \text{otherwise} \end{cases}$$
$$Z_{\psi_1 \vee \psi_2} = \begin{cases} max\,(Z_{\psi_1}, Z_{\psi_2}) & \text{if } \vee \text{ occurs positively in } \phi \\ Z_{\psi_1} + Z_{\psi_2} & \text{otherwise} \end{cases}$$
$$Z_{\psi_1 \wedge \psi_2} = \begin{cases} max\,(Z_{\psi_1}, Z_{\psi_2}) & \text{if } \wedge \text{ occurs negatively in } \phi \\ Z_{\psi_1} + Z_{\psi_2} & \text{otherwise} \end{cases}$$

Since $\text{FL}(\phi)$ is finite, S_ϕ is also finite, and there are at most $\|\phi\|$ equations. The mapping $\text{Vert}_\phi : \text{FL}(\phi) \to \mathbb{N} \cup \{+\infty\}$ is defined as the least solution of S_ϕ. It exists since the operations are monotonic over the complete lattice $(\mathbb{N} \cup \{+\infty\}, max, +, 0, 1)$. We let $\text{Vert}(\phi) = \text{Vert}_\phi(\phi)$. For instance, $\text{Vert}_{\phi_1}(\phi_1) = +\infty$, $\text{Vert}_{\phi_1}(a[X]) = 0$, and $\text{Vert}_{\phi_2}(\phi_2) = 1$.

Definition 2 (Bounded Formulas)**.** *Let $k \in \mathbb{N}$. A TQL formula ϕ is k-bounded if all the following conditions hold:*

1. *it is guarded*
2. *$|$ and the Kleene star do not occur under an odd number of negations*
3. *$\text{Vert}(\phi) \leq k$.*

It is bounded if it is k-bounded for some $k \in \mathbb{N}$. The set of bounded TQL formulas is denoted TQL_b.

Proposition 6.5.3 *For all TQL formulas ϕ, it can be decided in $O(\|\phi\|)$ if ϕ is bounded. Moreover, if it is bounded, then it is $2^{\|\phi\|+1}$-bounded.*

Proof. The first two conditions of the definition of bounded formulas can be checked in $O(\|\phi\|)$. For the third condition, it is known that a system S of equations on the semiring $(\mathbb{N} \cup \{+\infty\}, max, +, 0, 1)$ can be solved in $O(\|S\|)$, where $\|S\|$ is the sum of the sizes of the equations (Sei96). The size of an equation is the size of the term representing the rhs of the equation plus 1 (for the lhs). In our case, there are $|\text{FL}(\phi)|$ equations of size 4 at most. We can conclude, since $|\text{FL}(\phi)| \leq \|\phi\|$.
The upper bound is obtained by associating with the system S_ϕ a *tree automaton with costs* (FTAC). Every transition of an FTAC is associated with a cost function, which typically is a polynomial over some semiring. The variables

of the polynomial are intended to be substituted by the costs of some computations on the respective subtrees of the current node. More precisely, if the FTAC runs on binary trees, then each rule δ is associated with a polynomial $P_\delta(x_1, x_2)$ with two variables. The cost $C(q(r_1, r_2))$ of a run $q(r_1, r_2)$ such that root^{r_1} is labeled q_1 and root^{r_2} is labeled q_2 for some state q_1, q_2 is defined by $max_{f(q_1,q_2)\in\Delta} P_{f(q_1,q_2)}(C(r_1), C(r_2))$. The cost of a tree automaton is the maximal cost of a run. Seidl gives tight upper bounds for the cost of a tree automaton (when finite) for several semiring (Sei92), like for instance the arctical semiring $(\mathbb{N} \cup \{-\infty\}, max, +, 0, 1)$. It is technical but not difficult to associate with the system S_ϕ a tree automaton with $|\mathrm{FL}(\phi)|$ states, whose cost is lesser than $\mathrm{Vert}(\phi)$. Finally the upper-bound given in Theorem 3 of (Sei92) applies. □

Section 6.6 is devoted to the proof of the following theorem:

Theorem 6.5.4 *For all bounded recursion-closed TQL formulas ϕ, there is a vbTAGED A_ϕ such that $\|A_\phi\| = 2^{O(\|\phi\|^2)}$ and $\mathcal{L}(A_\phi, Vert(\phi)) \neq \varnothing$ iff ϕ is satisfiable. Moreover, A_ϕ is computable in time $2^{O(\|\phi\|^2)}$.*

As a corollary of Theorem 6.5.4, Theorem 5.7.3 and Proposition 6.5.3:

Theorem 6.5.5 *Satisfiability of bounded TQL formulas is decidable in 3NEXPTIME. It is in EXPTIME for bounded sentences.*

The EXPTIME upper bound when considering bounded sentences comes from the fact that the TAGED equivalent to the formula does not perform any test. In particular, it is a classical tree automaton (whose size is exponential) for which emptiness is decidable in linear time. The positive fragment of TQL_b is the set of TQL_b formulas in which there are no negative occurrences of tree variables. The bound does not really matter here as such formulas are always bounded by 0.

Theorem 6.5.6 *Satisfiability of bounded positive TQL formulas is decidable in 2EXPTIME.*

Proof. This is because the construction of the proof of Theorem 6.5.4 produces a positive TAGED of exponential size, whose emptiness is known to be decidable in EXPTIME by Theorem 5.4.1. □

6.5.3 Discussion on Expressiveness

The converse of Theorem 6.5.4 is not true in general. For instance, the set of trees of the form $f(t_1, t_2, \ldots, t_n, t'_1, \ldots, t'_m)$, $n, m \in \mathbb{N}$, where for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, $t_i \neq t'_j$ is recognizable by a vbTAGED but not definable by a TQL formula. This is because TQL allows to test disequalities between trees by the use of a variable X and its negated form $\neg X$. Hence a TQL formula can test disequalities only between a tree and all trees of a set of subtrees, while a vbTAGED can test whether two sets of subtrees contain pairwise different subtrees. We say that a vbTAGED A is *weak* if for all states $q, q' \in Q$, if $q \neq_A q'$, then either $q \in \mathrm{dom}(=_A)$ or $q' \in \mathrm{dom}(=_A)$. Actually, existentially quantified bounded TQL formulas are equally expressive as weak vbTAGEDs.
We already saw (Proposition 6.3.1) that guarded TQL sentences that use only disjunction, composition, recursion, and Kleene star can define recognizable hedge languages over a finite alphabet. These formulas form a strict fragment of bounded sentences. Moreover, the construction of Section 6.6 produces a TAGED without

tests if the input formula is a sentence. This makes bounded sentences equally expressive as hedge automata. Actually, we prove in (FTT07) that guarded TQL sentences are equally expressive as hedge automata[1]. This means that the condition 2 of the definition of bounded formulas can be removed for sentences, without extra expressiveness. However, there is a price to pay, as satisfiability becomes non-elementary in time complexity. This is because the problem of testing whether two regular expressions e and e' built over negation, Kleene star, intersection and the letter a and b, denote different word languages is non-elementary in time complexity (SM73b). This problem can easily be encoded as the satisfiability problem of a guarded TQL sentence, because negation, intersection, and Kleene star are native in TQL. However, we let as a future work the problem to know whether the second condition of the definition of bounded formulas can be removed without extra expressive power for arbitrary guarded formulas (with variables). We conjecture that it is true, but for the same reasons as before, the price to pay would be a non-elementary gap in time complexity, wrt the satisfiability problem.

6.6 Bounded TQL Formulas to vbTAGEDs

This section shows how to construct, from a bounded formula ϕ, a vertically bounded (unranked) TAGED A_Φ such that $L(A_\Phi) = \varnothing$ iff ϕ is satisfiable. The two main difficulties are to manage variables, which induce non-determinism in the rules of the automaton, and the concatenation operator $|$. Usually, constructing a tree automaton equivalent to a μ-calculus formula is done by defining an automaton whose runs mimic a recursive model-checking algorithm: states are sets of subformulas, and any tree t evaluates to the set of subformulas satisfied by t (BL05). The final states are naturally the states which contain the input formula. The information carried by the states is somehow maximal: it is known which subformulas are or not satisfied by the tree. Hence the behavior of the automaton is deterministic, and managing negations present in the logic becomes easy: for all states q and all subformulas $\neg\phi$, either ϕ or $\neg\phi$ is in q (but not both). With TQL formulas however, the use of variables induces non-determinism in the automaton. For instance, consider the formula $\mu\xi.(a[\xi] \vee X)$. The automaton has to guess a position where a subtree is matched by X. The information carried by states cannot be maximal, since it depends on the choice made by the automaton. Dealing with negations is therefore non easy, but we use the syntactic restrictions imposed by the fragment in order to eliminate negations (by pushing them down to the leaves). This is done thanks to a new operator $\widehat{X}$, which negates tree variables in the tree domain: $\widehat{X} = \neg X \wedge \Lambda[\top]$. States of the TAGED are subsets of subformulas of the form $\alpha[\phi], X$ or $\widehat{X}$. Going to a state containing X or $\widehat{X}$ is done non-deterministically, and intuitively means that the current subtree is matched by X, or has to be disequal to the subtree matched by X elsewhere in the hedge. A subtree which evaluates to a state containing $\alpha[\phi]$ has to satisfy $\alpha[\phi]$, under some valuation of its tree and recursion variables. Consider now a singleton state $\{\alpha[\phi]\}$. This state should be reachable thanks to a transition $\alpha(L_\phi) \to \{\alpha[\phi]\}$, where L_ϕ is an horizontal language of states, such that any hedge which evaluates to a word of L_ϕ satisfies ϕ. Defining L_ϕ is done by interpreting ϕ on regular word languages of states. For instance, $L_{a[0]|\top|a[0]}$ is denoted by the regular expression

[1] This result is not proved in this thesis to avoid redundancy with the construction given in Section 6.6. Moreover, this thesis is more concerned with the ability of a logic to define n-ary queries, and therefore with fragments with variables.

$\{a[0]\}.(\{\Lambda[\top]\})^*.\{a[0]\}$, while $L_{(\alpha[0]\wedge\alpha'[0])^*}$ is denoted by the regular expression $(\{\alpha[0],\alpha'[0]\})^*$. The next subsections formally define all these notions and some full examples are given. The last subsection proves correctness of the construction.

6.6.1 Elimination of Negation

In a first step, we eliminate negations from bounded formulas by pushing them down to the leaves of the formula, and by using the new operator $\widehat{X}$. It is the negation of X among the set of trees: $\widehat{X} = \Lambda[\top] \wedge \neg X$. Modulo some valuation ρ, $[\![\widehat{X}]\!]^\rho = \mathcal{T}_{unr}(\Lambda) - \rho(X)$. Note that $\neg X$ is equivalent to $\widehat{X} \vee \Lambda[\top]|\Lambda[\top]|\top \vee 0$, where the second member of the disjunction defines the set of hedges of length at least 2. We denote by $\widehat{\text{TQL}}$ the set of TQL formulas extended with atoms $\widehat{X}$. The notion of vertical variable occurrence can be extended naturally to formulas with atoms $\widehat{X}$. Hence we can define $\text{TQL}_b^{\not\downarrow}$ the set of negation-free bounded $\widehat{\text{TQL}}$-formulas. Formally, a TQL formula ϕ is a $\text{TQL}_b^{\not\downarrow}$ formula if it is bounded and is an element of the language generated by the following grammar:

$$\phi ::= 0 \mid \top \mid \alpha[\phi] \mid \phi|\phi \mid X \mid \widehat{X} \mid \xi \mid \mu\xi.\phi \mid \phi^* \mid \phi \vee \phi \mid \phi \wedge \phi$$

Proposition 6.6.1 *TQL_b and $\text{TQL}_b^{\not\downarrow}$ are equally expressive, modulo linear-time translations.*

Proof. The back inclusion is obvious. It suffices to replace each occurrence of $\widehat{X}$ by $\Lambda[\top] \wedge \neg X$.
The forth translation is done by applying the following rewriting rules on subformulas:

$$\begin{array}{lcl}
\neg\neg\phi & \rightarrow & \phi \\
\neg(\phi_1 \vee \phi_2) & \rightarrow & \neg\phi_1 \wedge \neg\phi_2 \\
\neg X & \rightarrow & \widehat{X} \vee \Lambda[\top]|\Lambda[\top]|\top \vee 0 \\
\neg 0 & \rightarrow & \Lambda[\top]|\top \\
\neg\top & \rightarrow & \varnothing[\top] \\
\neg\mu\xi.\phi & \rightarrow & \mu\xi.\neg\phi[\xi \mapsto \neg\xi] \\
\neg\alpha[\phi] & \rightarrow & \Lambda[\top]|\Lambda[\top]|\top \vee \overline{\alpha}[\top] \vee \alpha[\neg\phi]
\end{array}$$

The translation is correct since it preserves the semantics and composition as well as Kleene star occurs positively. In particular, the rule $\neg\mu\xi.\phi \rightarrow \mu\xi.\neg\phi[\xi \mapsto \neg\xi]$ is correct since formulas are guarded. Proving correctness of these rules is standard and already done in (Bon06). □

6.6.2 Horizontal Languages

In this section, we define the sets of hedges generated by word languages over sets of subformulas. This will be used to define the transitions of the vbTAGED.
Let S be a set of recursion-closed TQL formulas. Any word w over 2^S together with a valuation ρ of tree variables defines a set of hedges $\mathsf{hedges}(w,\rho) \subseteq \mathcal{H}(\Lambda)$. It is inductively defined as follows, for all $s \subseteq S$:

$$\begin{array}{lcl}
\mathsf{hedges}(\epsilon,\ \rho) & = & \{0\} \\
\mathsf{hedges}(s.w,\ \rho) & = & \{t|h \mid t \in \bigcap_{\phi\in s}[\![\phi]\!]^\rho \cap \mathcal{T}_{unr}(\Lambda),\ h \in \mathsf{hedges}(w,\rho)\}
\end{array}$$

More generally, a language W of words over 2^S defines the set of hedges $\mathsf{hedges}(W,\rho) = \bigcup_{w\in W} \mathsf{hedges}(w,\rho)$.
Given two words $w = s_1 \dots s_n$ and $w' = s'_1 \dots s'_n$ over 2^S, where $n \in \mathbb{N}$, $w \oplus w'$ denotes the word $(s_1 \cup s'_1) \dots (s_n \cup s'_n)$. This operation is naturally extended to word languages over 2^S (where only words of equal lengths are combined).
Let ϕ be a recursion-closed $\mathrm{TQL}_b^{\nearrow}$ formula. We next show that the formula ϕ defines the same language as a word over 2^S, for some suitable set S. Let $\mathrm{Sub}_{\mathrm{ext}}(\phi) \subseteq \mathrm{FL}(\phi)$ be the set of unfolded subformulas of ϕ of the form $\alpha[\psi]$, X or $\widehat{X}$. For all $\psi \in \mathrm{FL}(\phi)$, the *horizontal language* H_ψ of ψ is a word language over $2^{\mathrm{Sub}_{\mathrm{ext}}(\phi)\cup\{\Lambda[\top]\}}$ inductively defined by:

$$
\begin{array}{lcllcl}
H_{\psi|\psi'} & = & \{w.w' \mid w \in H_\psi,\ w' \in H_{\psi'}\} & H_{\psi\vee\psi'} & = & H_\psi \cup H_{\psi'} \\
H_{\psi\wedge\psi'} & = & H_\psi \oplus H_{\psi'} & H_{\alpha[\psi]} & = & \{\{\alpha[\psi]\}\} \\
H_X & = & \{\{\Lambda[\top], X\}\} & H_{\widehat{X}} & = & \{\{\Lambda[\top], \widehat{X}\}\} \\
H_{\mathsf{0}} & = & \{\epsilon\} & H_\top & = & (\{\Lambda[\top]\})^* \\
H_{\mu\xi.\psi} & = & H_{\psi[\xi\mapsto\mu\xi.\psi]} & H_{\psi^*} & = & H_\psi^*
\end{array}
$$

The subformula $\Lambda[\top]$ in H_X and $H_{\widehat{X}}$ is not necessary here but it is needed in the next subsection to guarantee the existence of a run.
Since ϕ is recursion-closed, every formula of $\mathrm{Sub}_{\mathrm{ext}}(\phi)$ is also recursion-closed. The formula ϕ and the language H_ϕ are equivalent in the following sense:

Proposition 6.6.2 *For all recursion-closed $\mathrm{TQL}_b^{\nearrow}$ formulas ϕ, all formulas $\psi \in \mathrm{FL}(\phi)$, and all valuations ρ of the tree variables of ϕ,*

$$[\![\psi]\!]^\rho = \mathsf{hedges}(H_\psi, \rho).$$

Proof. The proof is done by induction on unfolded subformulas, via the order $\prec$.

- $h \in [\![\psi_1|\psi_2]\!]^\rho$. Equivalently, there are h_1, h_2 such that $h = h_1|h_2$, $h_1 \in [\![\psi_1]\!]^\rho$ and $h_2 \in [\![\psi_2]\!]^\rho$. Since $h_1 \preceq_{\mathcal{H}(\Lambda)} h$, $\mathrm{rd}(\psi_1) \leq \mathrm{rd}(\psi_1|\psi_2)$, and ψ_1 is a strict subformula of $\psi_1|\psi_2$, we have $(h_1,\psi_1) \prec (h,\psi_1|\psi_2)$. Similarly, $(h_2,\psi_2) \prec (h,\psi_1|\psi_2)$. Consequently, by induction hypothesis, $h \in [\![\psi_1|\psi_2]\!]^\rho$ is equivalent to $h_1 \in \mathsf{hedges}(H_{\psi_1},\rho)$ and $h_2 \in \mathsf{hedges}(H_{\psi_2},\rho)$. By definition of $H_{\psi_1|\psi_2}$, it is finally equivalent to $h \in \mathsf{hedges}(H_{\psi_1|\psi_2},\rho)$;

- $h \in [\![\alpha[\psi]]\!]^\rho$. By definition of $\mathsf{hedges}(.)$, since $H_{\alpha[\psi]} = \{\alpha[\psi]\}$, we have $h \in \mathsf{hedges}(H_{\alpha[\psi]},\rho,\rho)$. The converse holds also by a direct application of the definition of $\mathsf{hedges}()$ and $H_{\alpha[\psi]}$;

- $h \in [\![\mu\xi.\psi]\!]^\rho$ iff $h \in [\![\psi[\xi \mapsto \mu\xi.\psi]]\!]^\rho$. Since ψ is guarded, $\mathrm{rd}(\psi[\xi \mapsto \mu\xi.\psi]) < \mathrm{rd}(\mu\xi.\psi)$, therefore $(h,\psi[\xi \mapsto \mu\xi.\psi]) \prec (h,\mu\xi.\psi)$. Consequently, by induction hypothesis, $h \in [\![\mu\xi.\psi]\!]^\rho$ is equivalent to $h \in \mathsf{hedges}(H_{\psi[\xi\mapsto\mu\xi.\psi]},\rho)$. By definition of $H_{\psi[\xi\mapsto\mu\xi.\psi]}$, this is again equivalent to $h \in \mathsf{hedges}(H_{\mu\xi.\psi},\rho)$;

- $h \in [\![\psi_1 \wedge \psi_2]\!]^\rho$ iff $h \in [\![\psi_1]\!]^\rho$ and $h \in [\![\psi_2]\!]^\rho$ iff (by induction) $h \in \mathsf{hedges}(H_{\psi_1},\rho)$ and $h \in \mathsf{hedges}(H_{\psi_2},\rho)$ iff (by definition of $\mathsf{hedges}()$ and $\oplus$) iff $h \in \mathsf{hedges}(H_{\psi_1} \oplus H_{\psi_2},\rho)$ iff $h \in \mathsf{hedges}(H_{\psi_1\wedge\psi_2},\rho)$;

- disjunction is proved similarly as conjunction;

- $h \in [\![X]\!]^\rho$ iff $h = \rho(X)$ iff (by definition of $\mathsf{hedges}()$) $h \in \mathsf{hedges}(\{\Lambda[\top], X\}, \rho)$, i.e. $h \in \mathsf{hedges}(H_X, \rho)$;
- other cases are obvious.

□

Moreover, the horizontal languages are definable by finite word automata of exponential sizes:

Proposition 6.6.3 *For all recursion-closed TQL$_b^{\not\downarrow}$ formulas ϕ, all $\psi \in FL(\phi)$, H_ψ is recognizable by a finite word automata A over $2^{Sub_{ext}(\phi)\cup\{\Lambda[\top]\}}$ such that: $\|A\| = 2^{O(\|\psi\|)}$, and A is computable in time $2^{O(\|\psi\|)}$.*

Proof. Word automata A_{H_ψ} are computed recursively with memoization to avoid redundant operations. For instance, $A_{H_{\psi_1\vee\psi_2}} = A_{H_{\psi_1}} \cup A_{H_{\psi_2}}$, where $A_{H_{\psi_1}} \cup A_{H_{\psi_2}}$ is a word automaton recognizing the union of the recognized languages of $A_{H_{\psi_1}}$ and $A_{H_{\psi_2}}$ respectively. All the needed operations on word automata are standard, except for $\oplus$, next described.
Let Σ be a finite alphabet. Let $A_1 = (2^\Sigma, Q_1, F_1, q_{i_1}, \delta_1)$ and $A_2 = (2^\Sigma, Q_2, F_2, q_{i_2}, \delta_2)$ be two finite word automata over 2^Σ. Remind that for all $j \in \{1, 2\}$, Q_j is the set of states, F_j the set of final states, q_{i_j} the initial state, and $\delta_j \subseteq Q_j \times 2^\Sigma \times Q_j$ the transition function. We define an automaton $A_1 \oplus A_2$ such that $L(A_1 \oplus A_2) = L(A_1) \oplus L(A_2)$. We let $A_1 \oplus A_2 = (2^\Sigma, Q_1 \times Q_2, F_1 \times F_2, (q_{i_1}, q_{i_2}), \delta)$, where for all $\sigma_1, \sigma_2 \subseteq \Sigma$, all $q_1, p_1 \in Q_1$ and all $q_2, p_2 \in Q_2$:

$$\delta = \{((q_1, q_2), \sigma_1 \cup \sigma_2, (p_1, p_2)) \mid (q_1, \sigma_1, p_1) \in \delta_1 \text{ and } (q_2, \sigma_2, p_2) \in \delta_2\}$$

The automaton $A_1 \oplus A_2$ has $|Q_1|.|Q_2|$ states, and for each pair of transitions of A_1 and A_2 respectively, we can create a transition of $A_1 \oplus A_2$. Consequently, $A_1 \oplus A_2$ has at most $|\Delta_1|.|\Delta_2|$ transitions. Finally, $\|A_1 \oplus A_2\| = O(\|A_1\|.\|A_2\|)$. Moreover, $A_1 \oplus A_2$ is computable in time $O(\|A_1\|.\|A_2\|)$. □

6.6.3 Construction of the vbTAGED

Let ϕ be a recursion-closed TQL$_b^{\not\downarrow}$-formula with tree variables $X_1, \ldots, X_n$, which is supposed to be fixed from now on. We now give the construction of a TAGED $A_\phi = (\Lambda, Q_\phi, F_\phi, \Delta_\phi, =_{A_\phi}, \neq_{A_\phi})$ such that $L(A_\phi) = [\![\exists X_1 \ldots \exists X_n \phi]\!]$. Although vbTAGEDs are tree acceptors, and ϕ defines hedges, we can assume that the models of ϕ are trees, otherwise it suffices to add a virtual root $\# \in \Lambda$, as $\#[\phi]$ is satisfiable iff ϕ is. We also assume that if ϕ is satisfiable on a tree t modulo some assignment ρ, then $\rho(X_i)$ is a subtree of t, for all $i \in \{1, \ldots, n\}$. This can be done by replacing ϕ by $\phi' = \#[\phi|X_1|\ldots|X_n]$. Indeed, ϕ is satisfiable iff ϕ' is satisfiable, but if there is t, ρ such that $t \in [\![\phi']\!]^\rho$, then $\rho(X_i)$ is necessarily a subtree of t, for all $i \in \{1, \ldots, n\}$.
The TAGED A_ϕ is defined as follows:

$$\begin{array}{lcl}
Q_\phi & = & 2^{\mathrm{Sub_{ext}}(\phi)\cup\{\Lambda[\top]\}} \\
F_\phi & = & \{w \in H_\phi \mid \text{the length of } w \text{ is } 1\} \text{ (necessarily } w \in Q_\phi) \\
\Delta_\phi & = & \{\cap_{\alpha[\psi]\in s}\alpha\ (\bigoplus_{\alpha[\psi]\in s} H_\psi) \rightarrow\ s \mid s \in Q_\phi \text{ and } \exists\alpha\exists\psi,\ \alpha[\psi] \in s\} \\
=_{A_\phi} & = & \{(s_1, s_2) \in Q_\phi \times Q_\phi \mid \exists X,\ X \in s_1 \cap s_2\} \\
\neq_{A_\phi} & = & \text{symmetric closure of } \{(s_1, s_2) \in Q_\phi \times Q_\phi \mid \exists X,\ X \in s_1 \text{ and } \widehat{X} \in s_2\}
\end{array}$$

6.6.4 Examples

- Example with a variable: $\phi_1 = X$.

$$\begin{array}{rcl}
\text{Sub}_{\text{ext}}(\phi) & = & \{X\} \\
Q_{\phi_1} & = & \{\ \{X\}, \{\Lambda[\top], X\}, \{\Lambda[\top]\}\} \\
F_{\phi_1} & = & \{\{X\}\} \\
H_\top & = & (\{\Lambda[\top]\})^* \\
\Delta_{\phi_1} & = & \{\Lambda(H_\top) \to \{\Lambda[\top]\}, \\
 & & \Lambda(H_\top) \to \{\Lambda[\top],\ X\}\ \} \\
=_A & = & (Q_{\phi_1} - \{\Lambda[\top]\})^2
\end{array}$$

  This example shows that adding the formula $\Lambda[\top]$ to the set of states is needed to guarantee the existence of at least one run per tree. Without it the produced automaton would be empty.

- Example with recursion: $\phi_2 = \mu\xi.(a[\xi]\ \vee \mathsf{0})$.

$$\begin{array}{rcl}
\text{Sub}_{\text{ext}}(\phi) & = & \{a[\phi_2]\} \\
Q_{\phi_2} & = & \{\ \{a[\phi_2]\}, \{\Lambda[\top], a[\phi_2]\},\ \{\Lambda[\top]\}\ \} \\
F_{\phi_2} & = & \{\{a[\phi_2]\}\} \\
 & & \\
H_{\phi_2} & = & \{\{a[\phi_2]\}, \epsilon\} \\
H_\top & = & (\{\Lambda[\top]\})^* \\
H_\top \oplus H_{\phi_2} & = & \{\{a[\phi_2], \Lambda[\top]\},\ \epsilon\} \\
 & & \\
\Delta_{\phi_2} & = & \{a(H_{\phi_2}) \to \{a[\phi_2]\}, \\
 & & \Lambda(H_\top) \to \{\Lambda[\top]\}, \\
 & & a(H_\top \oplus H_{\phi_2}) \to \{\Lambda[\top], a[\phi_2]\}
\end{array}$$

  It is equivalent to the tree automaton $(\Lambda, \{q\}, \{q\}, \{a(q + \epsilon) \to q\})$.

- Example with intersection: $\phi_3 = \Lambda[\mathsf{0}] \wedge \phi_2$.

$$\begin{array}{rcl}
\text{Sub}_{\text{ext}}(\phi) & = & \{a[\phi_2],\ \Lambda[\mathsf{0}]\} \\
Q_{\phi_3} & = & 2^{\{a[\phi_2],\ \Lambda[\mathsf{0}],\ \Lambda[\top]\}} \\
 & & \\
H_{\Lambda[\mathsf{0}]} & = & \{\{\Lambda[\mathsf{0}]\}\} \\
H_{\mathsf{0}} & = & H_{\mathsf{0}} \oplus H_{\phi_2} \ = \ H_\top \oplus H_{\mathsf{0}} \oplus H_{\phi_2} \ = \ H_{\mathsf{0}} \oplus H_\top \ = \ \{\epsilon\} \\
H_{\phi_3} & = & H_{\Lambda[\mathsf{0}]} \oplus H_{\phi_2} \ = \ \{\{\Lambda[\mathsf{0}], a[\phi_2]\}\} \\
 & & \\
\Delta_{\phi_3} & = & \Delta_{\phi_2} \cup \{ \\
 & & \Lambda(H_{\mathsf{0}}) \to \{\Lambda[\mathsf{0}]\} \\
 & & \Lambda(H_{\mathsf{0}} \oplus H_\top) \to \{\Lambda[\mathsf{0}], \Lambda[\top]\} \\
 & & a(H_\top \oplus H_{\mathsf{0}} \oplus H_{\phi_2}) \to \{\Lambda[\top], \Lambda[\mathsf{0}], a[\phi_2]\} \\
 & & a(H_{\mathsf{0}} \oplus H_{\phi_2}) \to \{\Lambda[\mathsf{0}], a[\phi_2]\} \\
 & & \}
\end{array}$$

  It is equivalent to the tree automaton $(\Lambda, \{q\}, \{q\}, \{a(\epsilon) \to q\})$.

6.6.5 Proof of Correctness

In this section we prove correctness of the construction of the TAGED A_ϕ associated with ϕ.
Let ρ be a valuation of tree variables, $\overline{p}$ a word over Q_ϕ, and h a hedge whose length is equal to the length of $\overline{p}$. A run r of A_ϕ on h is a $(\overline{p}, \rho)$-*run* if the word formed by the labels of the roots (in order) of r is $\overline{p}$, and, for all nodes $u \in \mathrm{Dom}(h)$, and all tree variables X, if $X \in \mathsf{lab}^r(u)$, then $\rho(X) = h|_u$, and if $\widehat{X} \in \mathsf{lab}^r(u)$, then $\rho(X) \neq h|_u$.
A *run is bounded* by some $k \in \mathbb{N}$ if there are no more than k nodes ordered by $\prec^r_{ch^+}$ whose labels contain a negated variable of the form $\widehat{X}$.
Finally, we extend the operation $\oplus$ to edge-isomorphic runs. In particular, $r \oplus r'$ is obtained by taking the union of the respective labels of r and r' at isomorphic positions, where r and r' are edge-isomorphic runs. We first prove a result which states that the overlapping of two runs is still a run.

Lemma 6.6.4 *Let $h \in \mathcal{H}(\Lambda)$, $\overline{p}, \overline{p}'$ two words over Q_ϕ, and ρ a valuation. Let r be a $(\overline{p}, \rho)$-run of A_ϕ on h, and r' a $(\overline{p}', \rho)$-run of A_ϕ on h.*
Then $r \oplus r'$ is a $(\overline{p} \oplus \overline{p}', \rho)$-run of A_ϕ on h. Moreover, if r and r' are bounded by k and k' respectively, then $r \oplus r'$ is bounded by $k + k'$.

Proof. Let $p, q \in Q_\phi$ be two states. They define the transitions $\alpha_p(L_p) \to p \in \Delta_\phi$ and $\alpha_q(L_q) \to q \in \Delta_\phi$, where:

$$\begin{array}{rclcrcl} \alpha_p & = & \bigcap_{\alpha[\psi] \in p} \alpha & \qquad & \alpha_q & = & \bigcap_{\beta[\psi] \in q} \beta \\ L_p & = & \bigoplus_{\alpha[\psi] \in p} H_\psi & & L_p & = & \bigoplus_{\beta[\psi] \in q} H_\psi \end{array}$$

The result follows since $\alpha_p \cap \alpha_q(L_p \oplus L_q) \to p \cup q$ is also a transition of Δ_ϕ. Indeed, $p \cup q \in Q_\phi$ defines the transition $\alpha_{p \cup q}(L_{p \cup q}) \to p \cup q \in \Delta_\phi$ where:

$$\alpha_{p \cup q} \;=\; \bigcap_{\gamma[\psi] \in p \cup q} \gamma \qquad\qquad L_{p \cup q} \;=\; \bigoplus_{\gamma[\psi] \in p \cup q} H_\psi$$

Moreover $\alpha_{p \cup q} = \alpha_p \cap \alpha_q$, and $L_{p \cup q} = L_p \oplus L_q$.
Finally, the bound is $k + k'$ because every union of a state from $\mathrm{dom}(\neq_{A_\phi})$ with any other state is still a state of $\mathrm{dom}(\neq_{A_\phi})$, by definition of $\neq_{A_\phi}$. □

Lemma 6.6.5 *Let $k \in \mathbb{N}$, $\psi \in FL(\phi)$ a k-bounded formula, $h \in \mathcal{H}(\Lambda)$, and ρ a valuation of the tree variables of ϕ.*
If $h \in [\![\psi]\!]^\rho$, there exists a $\mathrm{Vert}_\phi(\psi)$-bounded $(\overline{p}, \rho)$-run r of A_ϕ on h, for some word of states $\overline{p} \in H_\psi$.

Proof. The proof is by induction on pairs of hedges and formulas, ordered by $\prec$:

- $\psi = \psi_1 \wedge \psi_2$. By induction hypothesis, there are a $(\overline{p}_1, \rho)$-run r_1 on h bounded by $\mathrm{Vert}_\phi(\psi_1)$ and a $(\overline{p}_2, \rho)$-run r_2 on h bounded by $\mathrm{Vert}_\phi(\psi_2)$, for some $\overline{p}_1 \in H_{\psi_1}$ and $\overline{p}_2 \in H_{\psi_2}$. By Lemma 6.6.4, $r_1 \oplus r_2$ is a $(\overline{p}_1 \oplus \overline{p}_2, \rho)$-run on h, bounded by $\mathrm{Vert}_\phi(\psi_1) + \mathrm{Vert}_\phi(\psi_2)$, *ie* $\mathrm{Vert}_\phi(\psi)$;
- $\psi = \psi_1 \vee \psi_2$. Suppose that $h_1 \in [\![\psi_1]\!]^\rho$. By induction hypothesis, there is a $(\overline{p}_1, \rho)$-run r_1 on h bounded by $\mathrm{Vert}_\phi(\psi_1)$. We can conclude since by definition of H_ψ, $\overline{p}_1 \in H_\psi$, and $\mathrm{Vert}_\phi(\psi_1) \leq \mathrm{Vert}_\phi(\psi)$;

- $\psi = \alpha[\psi']$. Necessarily, $h = a(h')$, for some $a \in \alpha$ and $h' \in \mathcal{H}(\Lambda)$. By induction hypothesis, there is a $(\overline{p}', \rho)$-run r' on h' bounded by $\text{Vert}_\phi(\psi')$, for some $\overline{p}' \in H_{\psi'}$. Since $\alpha(H_{\psi'}) \to \{\alpha[\psi']\} \in \Delta_{\psi'}$, we can extend r' into a (p, ρ)-run on h, bounded by $\text{Vert}_\phi(\psi) = \text{Vert}_\phi(\psi')$, where $p = \{\alpha[\psi']\}$. Moreover, $p \in H_\psi$ since $H_\psi = \{p\}$;

- $\psi = \mu\xi.\psi'$. Hence $h \in [\![\psi'[\xi \mapsto \psi]]\!]^\rho$. By induction hypothesis, there is a $(\overline{p}, \rho)$-run r on h bounded by $\text{Vert}_\phi(\psi'[\xi \mapsto \psi])$, for some $\overline{p} \in H_{\psi'[\xi \mapsto \psi]}$. Since $H_{\psi'[\xi \mapsto \psi]} = H_\psi$ and $\text{Vert}_\phi(\mu\xi.\psi') = \text{Vert}_\phi(\psi'[\xi \mapsto \psi])$, r is also a $(\overline{p}, \rho)$-run on h bounded by $\text{Vert}_\phi(\psi)$ such that $\overline{p} \in H_\psi$;

- $\psi = X$. Necessarily h is a tree and has a root root^h. Since $\Lambda(H_\top) \to \{\Lambda[\top]\} \in \Delta$ and $\Lambda(H_\top) \to \{\Lambda[\top], X\} \in \Delta$, there exists a run r on h such that its root is labeled $\{\Lambda[\top], X\}$, and all other nodes are labeled $\{\Lambda[\top]\}$. Moreover, $\{\Lambda[\top], X\} = H_X$ and r is bounded by $0 = \text{Vert}_\phi(X)$;

- $\psi = \widehat{X}$. This case is similar to the previous case except that the run r is bounded by 1, *ie* $\text{Vert}_\phi(\widehat{X})$;

- $\psi = \psi_1|\psi_2$. There are two hedges h_1, h_2 such that $h = h_1|h_2$, $h_1 \in [\![\psi_1]\!]^\rho$, and $h_2 \in [\![\psi_2]\!]^\rho$. By induction hypothesis, there are a $(\overline{p}_1, \rho)$-run r_1 on h_1 bounded by $\text{Vert}_\phi(\psi_1)$ and a $(\overline{p}_2, \rho)$-run r_2 on h_2 bounded by $\text{Vert}_\phi(\psi_2)$, for some $\overline{p}_1 \in H_{\psi_1}$, and $\overline{p}_2 \in H_{\psi_2}$. Therefore, $r_1|r_2$ is a $(\overline{p}_1.\overline{p}_2, \rho)$-run on h, bounded by $max(\text{Vert}_\phi(\psi_1), \text{Vert}_\phi(\psi_2))$, *ie* $\text{Vert}_\phi(\psi)$. Moreover, by definition of H_ψ, $\overline{p}_1.\overline{p}_2 \in H_\psi$;

- $\psi = \mathsf{0}$. In this case $h = \mathsf{0}$ and the empty run is a 0-bounded (ϵ, ρ)-run. Moreover, $\epsilon \in H_\psi$;

- $\psi = \top$. This case is obvious since there always exists a run whose nodes are all labeled by $\{\Lambda[\top]\}$.

□

The converse also holds, as next shown. To deal with negated variables $\widehat{X}$, we introduce a new symbol † which satisfies, for all trees $t \in \mathcal{T}_{unr}(\Lambda)$, $t \neq \dagger$. A valuation ρ is said to be *extended* if it goes from tree variables to $\mathcal{T}_{unr}(\Lambda) \cup \{\dagger\}$. Let r be a $\overline{p}$-run of A_ϕ on some hedge h which satisfies the constraints. We can associate with r an extended valuation ρ_r as follows. It maps tree variables of $\{X \mid \exists u \in \text{Dom}(r),\ X \in \mathsf{lab}^r(u) \text{ or } \widehat{X} \in \mathsf{lab}^r(u)\}$ to $\mathcal{T}_{unr}(\Lambda) \cup \{\dagger\}$. For all $X \in \mathcal{X}_{\text{tree}}$, and all nodes $u \in \text{Dom}(r)$, if $X \in \mathsf{lab}^r(u)$, we let $\rho_r(X) = h|_u$. If $\widehat{X}$ occurs in the label of some node of $\text{Dom}(r)$, and X does not occur in any label of r, we let $\rho_r(X) = \dagger$. The valuation ρ_r is well-defined since r satisfies the equality constraints.

Lemma 6.6.6 *Let $\overline{p} \in Q_\phi^*$ be a word of states, and $h \in \mathcal{H}(\Lambda)$.*
If there is a $\overline{p}$-run of A_ϕ on h which satisfies the constraints, then $h \in \mathsf{hedges}(\overline{p}, \rho_r)$.

Proof. The proof goes by induction on h.

- if $h = \mathsf{0}$, then necessarily $\overline{p}$ is empty, and $h \in \mathsf{hedges}(\epsilon, \rho_r)$;

- if $h = t|h'$, for some tree t and non-empty hedge h'. There are $q, \overline{p}'$ and r_1, r' such that $r = r_1|r'$, $\overline{p} = q.\overline{p}'$, r_1 is a q-run on t, r' is a $\overline{p}'$-run on h', and both runs satisfy the constraints. By induction hypothesis, $t \in \mathsf{hedges}(q, \rho_{r_1})$ and $h' \in \mathsf{hedges}(\overline{p}', \rho_{r'})$.

  We also have $h' \in \mathsf{hedges}(\overline{p}', \rho_r)$. This is because r satisfies the disequality constraints, and, for all X that occurs in the label of some node of r', $\rho_r(X) = \rho_{r'}(X)$. Similarly, we can prove that $t \in \mathsf{hedges}(q, \rho_r)$. Hence $t|h' \in \mathsf{hedges}(q.\overline{p}', \rho_r)$, *ie* $h \in \mathsf{hedges}(\overline{p}, \rho_r)$;

- if $h = a(h')$, $\overline{p}$ is equal to some state $q \in Q_\phi$. Hence $r = q(r')$, for some r'. Let $\overline{s}$ be the word of states formed by the sequence of roots of r'. By definition of Δ_ϕ, $\overline{s} \in \bigoplus_{\alpha[\psi]\in q} H_\psi$, and $a \in \bigcap_{\alpha[\psi]\in q} \alpha$. By induction hypothesis, $h' \in \mathsf{hedges}(\overline{s}, \rho_{r'})$. Hence $h' \in \mathsf{hedges}(\bigoplus_{\alpha[\psi]\in q} H_\psi, \rho_{r'})$, which is equivalent to $h' \in \bigcap_{\alpha[\psi]\in q} \mathcal{H}(H_\psi, \rho_{r'})$, and by Proposition 6.6.2, this is again equivalent to $h' \in \bigcap_{\alpha[\psi]\in q} [\![\psi]\!]^{\rho_{r'}}$. As r satisfies the disequality constraints, and for all X that occurs in the label of some node of r', $\rho_r(X) = \rho_{r'}(X)$, we have $[\![\psi]\!]^{\rho_{r'}} = [\![\psi]\!]^{\rho_r}$. Therefore $h' \in \bigcap_{\alpha[\psi]\in q} [\![\psi]\!]^{\rho_r}$.

  For all X, if $X \in p$, then $\rho_r(X) = h$. If $\widehat{X} \in p$, then $\rho_r(X) \neq h$. Indeed, if X does not occur in any label of r, $\delta_r(X) = \dagger$, and $\dagger \neq h$ by definition. If X occurs in the label of some node u of r, then $\rho_r(X) = h|_u$. Since r satisfies the disequality constraints, necessarily $h|_u \neq h$. Consequently, we have:

$$a(h') \in \bigcap_{\alpha[\psi]\in q} [\![\alpha[\psi]]\!]^{\rho_r} \cap \bigcap_{X\in q} [\![X]\!]^{\rho_r} \cap \bigcap_{\widehat{X}\in q} [\![\widehat{X}]\!]^{\rho_r}$$

  By definition of $\mathsf{hedges}(q, \rho_r)$, this is equivalent to $h \in \mathsf{hedges}(q, \rho_r)$.

□

These two lemmata allow one to conclude that the construction is correct.

Theorem 6.6.7 (Correctness) *Let $X_1, \ldots, X_n$ be the tree variables of ϕ,*

$$[\![\exists X_1 \ldots \exists X_n \phi]\!] = L(A_\phi, \mathit{Vert}(\phi))$$

Proof. Let $t \in [\![\exists X_1 \ldots \exists X_n \phi]\!]$. There is a valuation ρ such that $t \in [\![\phi]\!]^\rho$. By Lemma 6.6.5, there are a state $q \in H_\phi$ and a $\mathrm{Vert}_\phi(\phi)$-bounded (q, ρ)-run r of A_ϕ on t. Since the run r is constrained to match the valuation ρ, r satisfies the constraints. Moreover, by definition of F_ϕ, $q \in F_\phi$. Hence $t \in A_\phi, \mathrm{Vert}(\phi))$.
Conversely, let $t \in L(A_\phi, \mathrm{Vert}(\phi))$. Since $L(A_\phi, \mathrm{Vert}(\phi)) \subseteq L(A_\phi)$, $t \in L(A_\phi)$. Hence there exist $q \in F_\phi$ and a q-run of A_ϕ on t. By Lemma 6.6.6, $t \in \mathsf{hedges}(q, \rho_r)$. Since $q \in H_\phi$, $t \in \mathsf{hedges}(H_\phi, \rho_r)$, and by Proposition 6.6.2, $t \in [\![\phi]\!]^{\rho_r}$. It remains to remove $\dagger$ from the range of ρ_r. This is done by replacing it by a tree which is not a subtree of t. We get a valuation ρ such that $t \in [\![\phi]\!]^\rho$, and the range of ρ is a subset of $\mathcal{T}_{unr}(\Lambda)$. Finally, we have $t \in [\![\exists X_1 \ldots \exists X_n \phi]\!]$. □

Complexity The number of states of A_ϕ is $2^{|\mathrm{FL}(\phi)|+1}$, that is less than $2^{\|\phi\|+1}$. Thanks to Proposition 6.6.3, the $\|\phi\|$ horizontal languages can be computed in time $\|\phi\|.2^{O(\|\phi\|)}$, *ie* $2^{O(\|\phi\|)}$. Consider now a transition $\cap_{\alpha[\psi]\in s}\alpha\ (\bigoplus_{\alpha[\psi]\in s} H_\psi) \rightarrow$

s of Δ_ϕ. A word automaton recognizing $\bigoplus_{\alpha[\psi]\in s} H_\psi$ can be computed in time $2^{O(|s|.\|\phi\|)}$, and therefore in time $2^{O(\|\phi\|^2)}$, since $|s| \leq \|\phi\|$. Moreover, its size is also in $2^{O(\|\phi\|^2)}$. Finally, there are $|Q_\phi|$ such transitions. The size of A_ϕ is therefore $|Q_\phi| + |Q_\phi|(2^{O(\|\phi\|^2)} + 2)$, *ie* $2^{O(\|\phi\|^2)}$.

6.7 Conclusion

In this chapter, we have adapted the TQL logic of (CG04) to the context of unranked ordered trees, which was mentioned as an open issue in (CG04, Gen06, Bon06). The main features of TQL are tree variables, operations from the hedge algebra (composition .|. and extension $\alpha[.]$), recursion, and Boolean operations. The presence of tree variables and the composition operator .|. makes TQL very different from the μ-calculus. And indeed, translating bounded TQL formulas to tree automata requires a novel automata construction. In particular, the resulting automata is non-deterministic. To the best of our knowledge, similar constructions that deal with temporal logics with variables do not exist in the literature.

Several questions remain however. As already mentioned in Section 6.5.3, we conjecture that positive occurrences of .|. and Kleene star in bounded formulas is not a needed condition and can be dropped without any extra expressive power. More generally, decidability of the full guarded TQL fragment is still unknown. This question is related to the emptiness of TAGEDs, which is still open.

It would be interesting to consider the (non)-inclusion problem of two TQL queries. A TQL-query $\phi(X_1, \ldots, X_n)$ over variables $X_1, \ldots, X_n$ is not included in a TQL-query $\phi'(X_1, \ldots, X_n)$ if $\phi(X_1, \ldots, X_n) \wedge \neg\phi'(X_1, \ldots, X_n)$ is unsatisfiable. Unfortunately the decidable TQL fragments considered in this thesis are not closed by negation. It would interesting to relax the use of negation to get fragments closed by negation, or at least to consider fragments where satisfiability of formulas of the form $\phi \wedge \neg\phi'$ is decidable. This would also allow one to decide if a query is *safe*, meaning that it selects only trees which are subtrees of the input tree.

Finally, TQL formulas can also be seen as a way to define languages of tree tuples. For instance, a formula ϕ with n tree variables $X_1, \ldots, X_n$ defines the tree tuple language $\{(\rho(X_1), \ldots, \rho(X_n)) \mid \exists h \exists \rho, h \in [\![\phi]\!]^\rho\}$. In this setting, it would be interesting to compare TQL to first-order logic on the domain of trees, where atomic predicates are taken from some tree algebra (BL02).

CONCLUSION 7

7.1 MAIN RESULTS

In this dissertation we studied two n-ary query languages in finite ordered unranked trees through the questions of expressiveness, query evaluation and satisfiability. The first part of the thesis was devoted to a mechanism to combine fixed arity queries in order to define arbitrary arity queries. Polynomial-time query evaluation as well as a reasonable expressiveness were the two pursued objectives. In Chapter 3, we introduced a composition language $\mathcal{C}(L)$ that allows one to combine binary queries of a language L into queries of arbitrary arities. In a tree, the binary queries define binary relations of nodes, which are used to navigate in the tree, and can be composed thanks to the usual composition operation for binary relations. Variables are used along the navigation to select node components of the output tuples. The composition language is also closed by disjunction and conjunction. In particular, conjunction is used to define a kind of "branching" in the navigation, *ie* the navigation is split into several branches by which to select independent parts of the output tuples. The branching structure of the query often mimics the branching structure of the document. We defined a simple syntactic fragment $\mathcal{C}^{\text{nvs}}(L)$ of the composition language, which can be evaluated in polynomial-time, both in the size of the tree, the query and the set of answers. This fragment yields a navigational variant of conjunctive queries over binary predicates with disjunction and a simple acyclicity notion. We proved that $\mathcal{C}^{\text{nvs}}(L)$ is as expressive as FO (resp. MSO) with respect to n-queries, as soon as L is as expressive as FO (resp. MSO) with respect to binary queries. In Chapter 4 we gave two instances of L that lead to two FO-complete n-ary query languages: Conditional XPath with variables, and a fragment of *CoreXPath2.0*. Both languages admit a polynomial-time query evaluation algorithm.

The second part of the thesis was concerned by the satisfiability of another n-ary query language, the TQL logic, adapted from (CG04) to the ordered tree context (Chapter 6). While variables of $\mathcal{C}(L)$ select nodes, variables of TQL are used to select subtrees of the input tree. By repeating a variable, or by using negations of variables, they can also be used to test subtree equalities, or disequalities. This ability makes TQL strictly more expressive than MSO, and indeed, TQL is undecidable. We defined a powerful decidable fragment of TQL, called the bounded fragment. The restriction is that only a bounded number of tree disequalities can be done along a root-to-leave paths. There is no restriction on tree equalities. Considering the satisfiability problem drives us to the definition of a new class of tree automata, called TAGEDs (Chapter 5). TAGEDs allow one to test tree equalities or disequalities between subtrees that might be arbitrarily faraway, in contrast to the classical models of automata with constraints. Testing emptiness of a powerful

TAGED subclass was proved to be decidable. In this class, only a bounded number of tree disequalities along a root-to-leaves path are allowed, but there is no restriction on tree equalities. We gave an exponential time reduction of the satisfiability of the bounded TQL fragment into emptiness of bounded TAGEDs, making the bounded TQL fragment decidable.

7.2 PERSPECTIVES

Several specific perspectives have already been given along the dissertation: in Section 3.6 for the composition language, in Section 5.8 for TAGEDs, and in Section 6.7 for TQL. We recall some of the main perspectives and address some other problems.

Efficient query evaluation algorithms for fragments of the composition language still have to be investigated. Indeed, the query evaluation of $\mathcal{C}^{\text{nvs}}(L)$ is quadratic in the size of the input tree. On large trees it may be intractable. A linear complexity in the size of the tree would be more reasonable. To tackle this problem, one should consider tree patterns, or equivalently, composition formulas over descendant and child axis, label tests, and satisfying the non-variable sharing restriction. Those queries can be represented as trees where edges are labeled by descendant or child axis, and node are labeled by tags and/or variables. Even though tree patterns are not as expressive as FO, their expressiveness is sufficient in many practical cases. There are many existing methods to answer tree patterns (BKS02, JWLY03, Che06, ZXM07), but they are either not optimal or they assume that every node of the tree patterns are part of the output, *ie* are labeled by some variable. To the best of our knowledge, there is no existing method for tree patterns whose time complexity is linear both in the size of the tree and the size of the output (the set of answers), and polynomial in the size of the query. It would be interesting to know if a query evaluation algorithm matching this complexity exists, otherwise a negative result should be proved.
Gottlob, Koch and Schulz (GKS04) establish a tractability frontier for the model-checking problem of conjunctive queries over XPath axis. It would be interesting to see whether the same frontier is still valid for the query evaluation problem.
Many applications are distributed and XML data exchange is usually done via a network. Moreover, XML documents can be very large so that only a fragment of them should be kept in memory. This strengthens the need to have query evaluation algorithms that work with XML data streams. The input XML document is therefore given by a data stream, and the output should also be returned as a data stream. Each answer to a query should be produced as soon as possible. Streaming evaluation algorithms of XPath fragments (for unary queries) are investigated in (BJ07, Olt07). More general n-ary queries by tree automata are considered in (GCNT08, BS04b). However, nothing is known about the streaming evaluation of logical formalisms for n-ary queries. Therefore, evaluation of fragments of the composition language, such as tree patterns, should also be investigated in the streaming perspective.
Answer enumeration algorithms for n-ary queries produce an output stream containing all the answers. The whole input tree however is considered to be in memory. Answer enumeration algorithms consists of two phases, the *preprocessing* and the *enumeration*. Obviously, any query evaluation algorithm can be transformed into an answer enumeration algorithm, it suffices to compute all the

answers and to output them one by one. Therefore, constraints are imposed on the time complexity of the preprocessing, as well as on the delay between the enumeration of two consecutive answers. Different classes of time complexity constraints yield a wide range of enumeration algorithms. For instance, it is known that n-ary queries by a tree automata A on a tree t can be enumerated with a preprocessing in time $O(\|A\|^3\|t\|)$ and a delay $O(n)$ between two consecutive answers (Bag06). Answer enumeration algorithms for fragments of acyclic conjunctive queries are also given in (BDG07). Fragments of the composition language should also be investigated under the answer enumeration perspective.

Several fundamental questions arise from the composition method used in Section 3.5 to prove expressiveness results on the composition language. In particular, given a signature σ, and an operation f on σ-structures, is it possible to give sufficient or necessary conditions on f for a composition lemma to hold. In other words, in which conditions (not necessarily decidable) the FO or MSO-type of a σ-structure $f(M_1, M_2, \ldots, M_n)$ is determined by the FO and MSO-types of the σ-structures M_i? This question seems to be related to Courcelle's MSO-transductions (Cou94), since they are used to logically define decompositions of graphs of bounded clique or tree-width.

Concerning TAGEDs, we already mentioned that deciding whether the language recognized by a TAGED is regular would imply decidability of a non-trivial fragment of the homomorphism problem[1]. More generally, which class of transducers preserves TAGED-recognizability? Connections between TAGEDs and unification problems should also be investigated because TAGEDs are well-suited to represent ground instances of (non-linear) terms with regular membership constraints. A first attempt has already been done in (FTT08), but TAGEDs are only used to decide existence of a solution of a unification problem. They could be used to finitely represent the set of solutions.

XML access control policies (ACPs) aim at defining rules to assign privileges to users at some node of an XML document (DdVPS00, DdVPS02, DFGM08). Several ACP models have been proposed. In (DdVPS00) for instance, the following rule (direction, /child::job/child::candidate/child::decision, write, allow) gives write privileges to the user group "direction" for subparts that can be accessed by the XPath expression /child::job/child::candidate/child::decision. There are two main approaches to XML queries: either generate a view of the document thanks to the ACP and query the view, or rewrite the query so that it conforms to the ACP. Although only unary queries have been considered, we think that n-ary queries are an interesting issue in this context.
Still about ACPs, the typechecking problem in presence of ACPs is of practical relevance. Consider a document transformation T, from a class of documents constrained by an ACP P_{in} into a class of documents constrained by an ACP P_{out}. It might be the case that the transformation of a document d into $T(d)$ may cause an information which was private in d (according to P_{in}) to be public in $T(d)$ (according to P_{out}). One may want to verify that such a situation cannot happen. This asks two questions: how to express access constraints between the input and the output, and how to decide them? Since it may be useful to check equalities of whole subtrees between the input tree and the output tree, one way to tackle

[1] Given a homomorphism h and a regular tree language L, is $h(L)$ also regular?

the problem is to consider TAGED-definable transformations. A transformation T is defined by a TAGED A if the language of trees of the form $\#(t, T(t))$ for a fresh symbol $\#$ is equal to $L(A)$. The policy, the transformation, and the access constraints should be compiled into the TAGED-definable transformation. In this context, it might be interesting to consider extensions of TAGEDs with more general constraints instead of just tree equality or disequality, for instance recognizable binary tree relations (CDG*07).

Several papers consider first-order logic over the infinite structure whose domain is the set of all trees and predicates are operations on a tree algebra (BLSS01, BLN07). In would be interesting to study variants of TQL where tree operations are chosen in those papers and to see what would be the complexity of the satisfiability problem, as well as the expressive power of these variants.

8 RÉSUMÉ

8.1 MOTIVATIONS ET OBJECTIFS

Les langages de requêtes pour les bases de données ont été très étudiés par la communauté base de données. Dans le contexte du Web cependant, les données ont tendance à être structurées de manière arborescente, comme en témoigne la popularité grandissante des langages à balises comme HTML et XML (BPSM*06). Les systèmes de base de données relationnelles ne sont pas bien adaptés aux données arborescentes. Par conséquent, avec le développement des applications du Web, le besoin d'avoir des outils spécifiques aux données arborescentes s'est considérablement accru. En particulier, la capacité de faire des requêtes dans les données XML est devenu crucial depuis qu'XML s'est imposé comme un standard pour l'échange de données et le stockage d'information. Cela a donc renforcé le besoin d'avoir des langages de requêtes conçus spécifiquement pour les données XML. Les documents XML sont naturellement modélisés par des arbres, une des structures les plus utilisées et les plus étudiées en science informatique: l'algorithmique, la logique, et la théorie des langages constituent quelques exemples de grandes disciplines de l'informatique ayant abordé les structures d'arbres. Néanmoins, les applications liées à XML posent de nouvelles questions qui nécessitent un va-et-vient constant entre théorie et pratique.

Les documents XML sont *semi-structurés*: la structure et les données ne sont pas séparées. Cela fait d'XML un format très flexible qui facilite l'échange de données provenant de différentes applications ou systèmes de bases de données. La Figure 8.1 donne deux exemples de documents XML. Le premier représente un bon de commande, avec l'adresse de livraison et l'adresse de facturation, ainsi que la liste des produits commandés. Le second document représente l'ordre de livraison lié au bon de commande. Les documents XML sont structurés par des *balises ouvrantes et fermantes*, qui doivent être bien balancées, par exemple la balise ouvrante `<ville>` et la fermante `</ville>`. Entre une balise ouvrante et sa balise fermante respective, il est possible d'inclure des données non-structurées (texte brut), ou récursivement d'autres sous-parties XML. Les balises ouvrantes peuvent contenir des *attributs*, c'est-à-dire des paires (nom,valeur), comme représenté sur la figure.

Les documents XML peuvent naturellement être représentés par des arbres. Les représentations arborescentes respectives du bon de commande et de l'ordre de livraison sont données par la Figure 8.2. Chaque noeud est étiqueté par un symbole et peut avoir un nombre arbitraire d'enfants, naturellement ordonnés par l'ordre séquentiel de leur apparition dans les documents XML. Les attributs sont représentés par des branches additionnelles dont les étiquettes de noeuds sont précédées du symbole `@`. Le modèle d'arbre sous-jacent aux documents XML est plus connu

sous le nom *d'arbres ordonnés d'arité arbitraire et noeuds étiquetés*, à la différence des *arbres d'arité fixée* qui ne peuvent avoir qu'un nombre fixé d'enfants par noeud.

```
<bonCommande date="2008-06-30">
  <Livraison pays="FR">
    <nom>Pierre Jouet</nom>
    <rue>132, rue Lecomte</rue>
    <ville>Lille</ville>
    <cp>59000</cp>
  </Livraison>
  <ListeProduits>
    <Produit ref="1548732">
      <NomProduit>OCaml reference manual</NomProduit>
      <Quantité>1</Quantité>
      <Prix>31.50</Prix>
    </Produit>
    <Produit ref="3213575">
      <NomProduit>MSO for all</NomProduit>
      <Quantité>1</Quantité>
      <Prix>23.5</Prix>
    </Produit>
  </ListeProduits>
  <Facture pays="BE">
    <nom>Guy Hecke</nom>
    <rue>10, rue Haute</rue>
    <ville>Brussels</ville>
    <cp>1000</cp>
  </Facture>
</bonCommande>

<ordreLivraison>
    <adresse>
      <nom>Pierre Jouet</nom>
      <rue>132, rue Lecomte</rue>
      <ville>Lille</ville>
      <cp>59000</cp>
      <pays>FR</pays>
    </adresse>
    <références>
       <ref>1548732</ref>
       <ref>3213575</ref>
    </références>
</ordreLivraison>
```

Figure 8.1: Représentations XML d'un bon de commande et de son ordre de livraison.

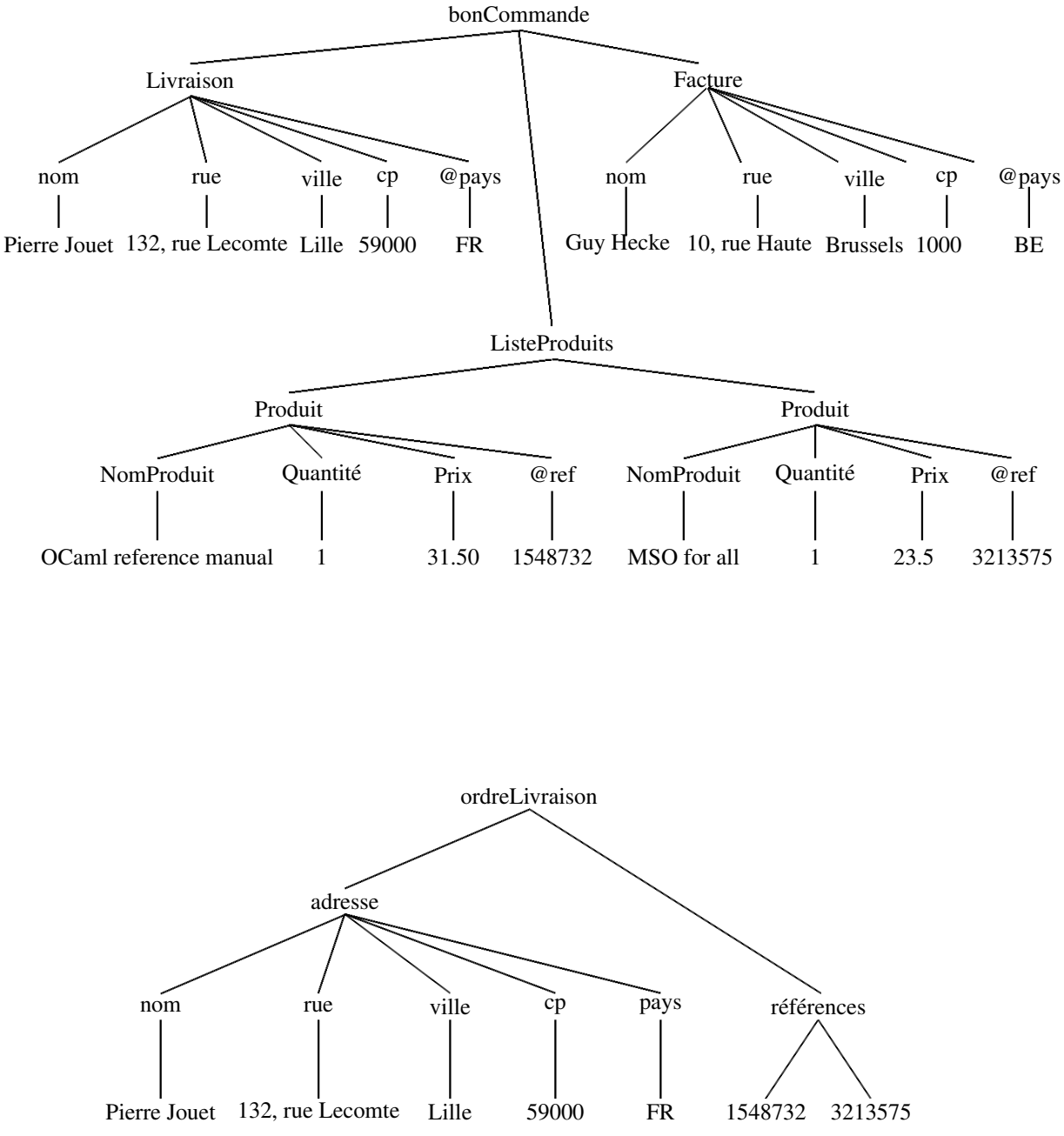


Figure 8.2: Représentations arborescentes des documents XML de la figure 8.1

La structure et la syntaxe des documents XML peuvent être contraintes par des *langages de schémas*, l'exemple le plus connue étant les *Document Type Definition* (DTD). Par exemple, une DTD pourrait contraindre un ordre de livraison XML à toujours commencer par une balise `ordreLivraison`, et à toujours contenir une adresse apparaissant avant les références de produits, comme sur la Figure 8.1. Maintenant, on suppose que le vendeur (celui qui a reçu le bon de commande) doit ordonner à un livreur indépendant la livraison des produits listés sur le bon de commande de la Figure 8.1. En supposant que l'application du livreur n'accepte que des ordres de livraison d'un certain format XML, l'application du vendeur doit donc d'abord transformer le bon de commande en un ordre de livraison compatible avec celui du livreur. Cela peut être fait par exemple avec des *transformations XML* basées sur des *requêtes XML*.
Faire une requête dans un arbre consiste à en extraire (ou *sélectionner*) des parties pertinentes. Ces parties sélectionnées sont généralement des sous-arbres, souvent représentés par leur noeud racine. Ainsi plus formellement, faire une requête dans un arbre consiste à sélectionner des noeuds de cet arbre. Ce type de requêtes est appelé *requêtes à sélection de noeuds*. Elles sont au coeur de beaucoup d'applications XML, comme les transformations de documents (MBPS05), l'extraction d'information dans le web (GK04), la programmation XML (HP03b, BCF03a), et l'échange de données (AL08). Les *requêtes unaires* sélectionnent des ensembles de noeuds, alors que les *requêtes n-aires* sélectionnent des ensembles de n-uplets de noeuds. Considérons encore le bon de commande et l'ordre de livraison de la Figure 8.2. La première étape d'une transformation du bon de commande en ordre de livraison pourrait être d'extraire l'adresse ainsi que les références de produits. Extraire l'adresse peut être fait avec la requête 5-aire qui sélectionne le nom, la rue, la ville, le code postal et le pays. Les références de produits sont sélectionnées avec une requête unaire. En deuxième étape, les sous-parties de documents représentées par les (uplets de) noeuds sélectionnés peuvent être recombinées pour reformer un document XML correspondant à un ordre de livraison compatible avec le schéma du livreur.
Les formalismes d'arbres ont été étudiés depuis longtemps par les communautés de la théorie des langages et de la théorie des modèles finis. La plupart d'entre eux ont été introduits pour les arbres d'arité fixée, mais depuis le succès grandissant d'XML, plusieurs formalismes ont été revisités ou proposés dans le contexte des langages de requêtes pour les arbres d'arité arbitraire. Ces formalismes se divisent principalement en deux catégories, les formalismes déclaratifs (comme les logiques d'arbres) et les formalismes procéduraux (comme les automates d'arbres). Il existe des liens très forts entre ces deux catégories, les formalismes procéduraux étant souvent utilisés comme modèles de calcul pour les formalismes déclaratifs. Un des liens les plus connus est la correspondance entre la logique monadique du second ordre (MSO) et les automates finis d'arbres (TW68). En particulier, toute requête définie par une formule de MSO peut-être définie par un automate d'arbres et réciproquement.
La plupart des formalismes de requêtes existants sont des formalismes pour requêtes unaires et binaires (Lib06). A l'exception de quelques formalismes procéduraux pour requêtes n-aires (NV02, NPTT05), les formalismes logiques pour requêtes n-aires n'ont été que très peu étudiés, alors qu'ils l'ont été pour les cas unaire et binaire. De plus, ils constituent un sujet de recherche important pour la communauté XML, comme mentionné dans (GKS04, Lib06, ABL07, Sch07). Le principal objectif de cette thèse est de proposer et d'étudier des formalismes

logiques pour requêtes n-aires. Nous proposons en particulier deux langages de requêtes n-aires. Le premier est un langage de composition de requêtes qui permet d'étendre tout langage de requêtes unaires ou binaires en un langage de requêtes n-aires. Concernant ce langage, le sous-objectif était d'obtenir un langage de composition avec une bonne expressivité, et la possibilité de répondre aux requêtes en temps polynomial (dans le nombre de réponses et la taille de la requête). Le second langage proposé est une adaptation aux arbres ordonnés de la logique TQL (*Tree Query Logic*), introduite par Cardelli et Ghelli (CG04) pour les arbres non ordonnés. L'objectif principal était de définir un fragment décidable expressif de TQL permettant de définir des requêtes n-aires, car la logique TQL complète est indécidable (pour le problème de satisfiabilité). Les deux langages proposés possèdent des variables qui permettent de sélectionner les composantes des uplets réponses à la requête.

8.2 Description de la Thèse

La thèse se divisent en deux parties indépendantes portant sur les deux langages de requêtes n-aires proposés.

8.2.1 Langage de Composition

Le standard du W3C *XPath* (BBC*07) est un langage *navigationnel* pour XML qui permet de sélectionner des ensembles noeuds d'un arbre en se déplaçant dans cet arbre grâce à des expressions de chemins. Par exemple, l'expression de chemin

$$/\text{child} :: \text{Facture}/\text{child} :: \text{nom}/\text{child} :: *$$

sélectionne dans le premier arbre de la Figure 8.2 le noeud étiqueté par le nom de la personne à qui doit être envoyée la facture. Le symbole '/' est un opérateur de composition d'expression de chemins. Cette requête XPath est interprétée comme suit: on part de la racine de l'arbre, on descend à un noeud enfant étiqueté par Facture, et finallement on accède à l'enfant étiqueté par le nom de la personne (l'étoile signifie que n'importe quelle étiquette est acceptée). XPath est utilisé comme langage de sélection de noeuds dans plusieurs autres standards du W3C dédié au traitement de documents XML, comme par exemple le langage de requêtes très haut niveau *XQuery* (BCF*07), le langage de transformation *XSLT* (Cla99), le langage de schémas *XML Schema* (FW04), ou le langage d'adressage *XPointer* (DMJ01). L'essentiel du coeur navigationnel de XPath (sans sucre syntaxique) est formellement défini au Chapitre 2. Les expressions de chemins sont généralement vues comme des requêtes unaires, puisqu'elles sont évaluées à partir de la racine. Elles peuvent aussi être vues comme des requêtes binaires dès lors qu'on les évalue depuis n'importe quel noeud de l'arbre. En particulier, elles relient un noeud de départ de la navigation à un noeud d'arrivée: une paire de noeuds (u, v) est sélectionnée par une expression de chemin p si le noeud v peut-être atteint en partant du noeud u et en suivant l'expression de chemin p.
Le langage de composition généralise le paradigme navigationnel *à la* XPath aux requêtes n-aires: des variables de noeuds sont utilisées pour sélectionner les composantes des uplets réponses au cours de la navigation. En toute généralité, tout formalisme de requêtes binaires peut être utilisé pour la partie navigationnel du langage de composition. En effet, une requête binaire q définit sur un arbre t

une relation binaire entre noeuds qui peut permettre de se déplacer d'un noeud à l'autre. L'idée du langage de composition est alors d'étendre tout formalisme de requêtes binaires, permettant de naviguer dans l'arbre, avec des variables de noeuds, permettant de sélectionner les réponses. Le langage est ensuite clos par union et intersection pour ajouter plus d'expressivité. L'opérateur principal du langage de composition est l'opérateur de composition $\circ$. Par exemple, sélectionner le triplet (NomProduit, Prix, Réference) dans le premier arbre de la Figure 8.2 peut être fait comme suit: on part de la racine pour atteindre le nom du produit, on le sélectionne avec une variable x, ensuite on part du nom de produit pour aller au prix, qu'on capture avec une variable y, et finalement on va du prix à la référence, capturée par une variable z. Les parties navigationnelles de cette requête peuvent être exprimées dans n'importe quel formalisme binaire, XPath par exemple. Plus formellement, cette requête s'écrit comme suit:

$$p_{prod} \circ x \circ p_{price} \circ y \circ p_{ref} \circ z$$

où p_{prod}, p_{price} et p_{ref} sont des requêtes binaires définies par exemple par les expressions de chemins XPath suivantes [1]:

$$\begin{array}{lcl} p_{prod} & = & /\text{child} :: \text{ListeProduits}/\text{child} :: \text{Produit}/\text{child} :: \text{NomProduit}/\text{child} :: * \\ p_{price} & = & /\text{parent} :: */\text{next-sibling} :: \text{Quantité}/\text{next-sibling} :: \text{Prix}/\text{child} :: * \\ p_{ref} & = & /\text{parent} :: */\text{next-sibling} :: @\text{ref}/\text{child} :: * \end{array}$$

Bien sûr il y a beaucoup de façons de définir une telle requête, et en général, la définition d'une requête dépend de ce qui est connu *a priori* sur la forme des documents sur lesquels elle est appliquée. L'ensemble des requêtes de composition définies sur un ensemble de requêtes binaires L est dénoté $\mathcal{C}(L)$. Nous décrivons maintenant les résultats obtenus pour le langage de composition, en terme d'expressivité et de complexité de l'évaluation des requêtes.
Évaluer une requête sur un arbre signifie retourner toutes les solutions de celle-ci. Ce problème prend donc en entrée un entier $n \in \mathbb{N}$, une requête n-aire et un arbre, et retourne en sortie l'ensemble des n-uplets solutions de cette requête. Son problème de décision associé est le problème du *model-checking*. Il prend un entier $n \in \mathbb{N}$, une requête n-aire, un arbre, et un n-uplet de noeuds en entrée, et retourne vrai si et seulement si le n-uplet est solution de la requête sur cet arbre. Tout algorithme de model-checking peut être transformé en un algorithme d'évaluation. Il suffit de générer tous les n-uplets de noeuds et de tester s'ils sont solution. Bien que cette approche reste raisonnable pour des petits arbres et des requêtes de faible arité, elle est impraticable en général. En effet, les arbres peuvent être très grands (plusieurs gigaoctets (Wik)), et l'arité des requêtes élevée. Par exemple, le bon de commande XML de la Figure 8.1 pourrait faire partie d'un grand document XML regroupant plusieurs centaines de bons de commande, et pourrait aussi contenir d'autres informations qu'il serait intéressant de sélectionner au sein d'un long n-uplet, comme par exemple des commentaires de l'acheteur, le mode de livraison, l'identifiant de l'acheteur, etc... L'évaluation d'une requête doit donc se faire dans un temps proportionnel au nombre de solutions, et non proportionnel au nombre de n-uplets de noeuds (qui est exponentiel). Nous définissons un fragment simple et expressif de $\mathcal{C}(L)$, dénoté $\mathcal{C}^{\text{nvs}}(L)$. L'idée est d'interdire l'utilisation

[1] L'expression XPath "parent" (resp. "next-sibling") relativise un noeud à son noeud parent (resp. à son voisin de droite)

d'une même variable de chaque côté d'une composition, ce qui créerait des cycles et forcerait à tester des égalités entre noeuds. Cette notion est assez proche de la notion d'acyclicité dans les requêtes conjonctives sur des relations binaires. Cependant, elle est plus simple et définie même en présence de disjonction (ce qui n'est pas le cas pour les requêtes conjonctives). Les requêtes exprimées dans le fragment $\mathcal{C}^{\text{nvs}}(L)$ peuvent être évaluées en temps polynomial dans l'arité de la requête, la taille de la requête, la taille de l'arbre, mais aussi la taille de la sortie, à condition que l'évaluation des requêtes binaires exprimées avec L soient elles aussi évaluables en temps polynomial. Ce n'est en revanche plus possible si la restriction syntaxique est levée.
La *logique du premier ordre* (FO) et la *logique monadique du second ordre* (MSO) sont deux logiques généralement acceptées comme des standards auxquels les autres formalismes de requêtes sont souvent comparés (Lib06). Ces logiques peuvent exprimées des requêtes n-aires, grâce à leurs variables, mais leur évaluation est dure, lorsqu'on considère l'arbre et la requête comme faisant partie de l'entrée (complexité combinée). Cependant, lorsque la requête est considérée fixée (ne faisant pas partie de l'entrée), l'évaluation devient linéaire dans la taille de l'arbre et le nombre de solutions (Bag06, Cou07), mais implique de grandes constantes que l'on veut ici éviter. Nous prouvons que le langage $\mathcal{C}^{\text{nvs}}(L)$ permet de définir toutes les requêtes n-aires exprimables en FO et MSO, dès lors que L permet d'exprimer toutes les requêtes binaires définissables en FO et MSO respectivement. La preuve est assez standard et s'appuie sur des résultats connus de la théorie des modèles finis, basée sur la méthode de composition de Shelah (She), souvent utilisée dans la littérature, comme par exemple dans (Tho84, Sch00, Mar05a). En se basant sur cette preuve, nous proposons une manière de composer des requête unaires d'un langage L, tout en gardant le même pouvoir d'expressivité que le langage de composition de requêtes binaires. Cela est possible en restreignant à chaque étape de composition le domaine d'interprétation de la requête unaire.
A la différence de la première version de XPath, XPath 1.0 (CD99), sa seconde version XPath 2.0 (BBC*07) possède des variables de noeuds. Alors qu'XPath 2.0 est considéré comme un langage de requêtes unaires, nous montrons qu'il peut être utilisé dans sa définition actuelle pour exprimer des requêtes n-aires. En se basant sur le langage de composition, nous définissons un fragment de XPath 2.0 qui permet d'exprimer toutes les requêtes n-aires FO-définissables, tout en admettant un algorithme d'évaluation en temps polynomial.

8.2.2 La Logique TQL

La logique *TQL* (*Tree Query Logic*) est une logique spatiale introduite par Cardelli et Ghelli (CG04) pour faire des requêtes dans des documents représentés par des arbres non ordonnés. Cette logique permet de définir des requêtes n-aires grâce à ses variables. Le problème de l'évaluation des requêtes pour TQL a été étudié dans (CFG02). La satisfiabilité et l'expressivité de fragments de TQL sans variables ont été étudiés dans (BTT05, Bon06). Cependant, rien n'est connu sur la satisfiabilité de TQL en tant que langage de requêtes n-aires, *ie* sur des fragments avec variables. Nous étudions dans cette thèse le problème de satisfiabilité pour les arbres ordonnés. L'étude de TQL dans les arbres ordonnés était mentionnée comme une perspective intéressante dans plusieurs travaux (CG04, Gen06, Bon06). TQL est un langage de requêtes qui généralise les langages de filtrage des langages de programmation XML XDuce et CDuce (HP03b, BCF03a).

La mise en correspondance de la formule TQL suivante

$$\text{bonCommande}[\text{Livraison}[\top] \wedge X \mid \top]$$

avec le premier arbre de la Figure 8.2 produit une fonction qui associe à X le sous-arbre dont la racine est étiquetée par "Livraison". En revanche, avec le deuxième arbre de la Figure 8.2, la mise en correspondance échoue. Chaque étiquette de noeud a est vue comme un constructeur d'arbre qui prend une séquence d'arbres et l'enracine par un noeud étiqueté par a. Le symbole $\mid$ est vu aussi comme un constructeur qui prend deux séquences d'arbres et les concatènent. La formule $\top$ est satisfaite par n'importe quelle séquence d'arbres. Comme le montre l'exemple précédent, les variables dénotent des arbres, à la différence des variables du langage de composition. En répétant une même variable plusieurs fois dans une formule, il est possible d'exprimer des égalités entre sous-arbres. Par exemple, la formule

$$\text{bonCommande}[\top \mid \text{ListeProduits}[\top|X|\top|X|\top] \mid \top]$$

est satisfaite par un arbre qui représente un bon de commande si la liste des produits achetés contient au moins un doublon. En utilisant la négation, il est aussi possible de tester des différences entre sous-arbres:

$$\text{bonCommande}[\top \mid \text{ListeProduits}[\top|X|\top|\neg X|\top] \mid \top]$$

est satisfaite si la liste des produits contient au moins deux produits différents. Les formules de TQL peuvent contenir aussi des opérateurs de point fixe permettant la récursion. Par exemple, la formule

$$\text{bonCommande}[\top \mid \text{ListeProduits}[\mu\xi.(X|\xi \vee X)] \mid \top]$$

est satisfaite par un bon de commande si la liste des produits ne contient que des produits identiques. La récursion permet aussi de se déplacer à des profondeurs arbitraires dans un arbre. Par exemple, la formule

$$\mu\xi.(a[\xi] \vee b[\top])$$

est satisfaite par un arbre s'il existe un chemin depuis la racine étiqueté par des as seulement puis un b.

La satisfiabilité de TQL est indécidable (même sans variables). La première restriction est de contraindre les les récursions à être *gardées*, *ie* chaque variable de récursion ξ doit apparaître sous un constructeur de la forme $a[.]$, lui même dans la portée de $\mu\xi$. Cela implique que les récursions s'appliquent sur des arbres strictement moins hauts. Ensuite, la principale difficulté pour décider la satisfiabilité des formules de TQL vient de leur capacité à tester des égalités ou des différences entre sous-arbres. Nous définissons alors un fragment appelé *fragment borné*. Dans le fragment borné, les récursions sont gardées et le nombre des tests de différences pouvant être faits le long d'un chemin allant de la racine à une feuille est borné par une constante indépendante de l'arbre.

La problème de satisfiabilité du fragment borné se réduit au test du vide pour une nouvelle classe d'automates d'arbres avec contraintes d'égalités et de différences, appelés TAGEDs. Les TAGEDs permettent de tester des égalités ou différences entre sous-arbres qui peuvent être arbitrairement éloignés dans l'arbre, à la différence des automates à contraintes classiques, qui ne testent que des égalités entre enfants

ou cousins (CDG*07). Les TAGEDs restent néanmoins incomparables avec les automates à contraintes classiques. En effet, dans les TAGEDs, les contraintes sont testées globalement avec les états, alors que pour les automates à contraintes classiques, les contraintes sont testées localement au niveau de chaque transition de l'automate. Nous prouvons que les langages définissables par TAGEDs sont effectivement clos par union et intersection. Cependant, ils ne sont pas clos par complément. De plus, les TAGEDs ne sont pas déterminisables et le test d'universalité est indécidable. Nous prouvons la décision du test du vide pour plusieurs sous-classes des TAGEDs: les *TAGEDs positifs*, qui ne font que des tests d'égalités, et les *TAGEDs négatifs*, qui ne font que des tests de différences. Nous prouvons aussi la décision du vide pour une sous-classe, appelé *TAGEDs bornés*, qui autorise les deux types de tests, mais seulement un nombre borné de tests de différences le long de tout chemin de la racine aux feuilles, comme pour le fragment borné de TQL. Nous donnons une correspondance entre les TAGEDs bornés et le fragment borné de TQL. Cette correspondance introduit une nouvelle construction qui doit prendre en compte les variables de la logique, ce qui implique un comportement non-déterministe de l'automate.
Finalement, nous donnons aussi une correspondance naturelle et effective entre les TAGEDs bornés et une extension de MSO avec tests d'égalités. Cette extension de MSO est par conséquent décidable (pour la satisfiabilité).

INDEX

BIBLOGRAPHY

[ABD*05] AFANASIEV L., BLACKBURN P., DIMITRIOU I., GAIFFE B., GORIS E., MARX M., RIJKE M. D.: Pdl for ordered trees. (Cited page 41.)

[ABL07] ARENAS M., BARCELO P., LIBKIN L.: Combining temporal logics for querying XML documents. In *International Conference on Database Theory* (2007), pp. 359–373. (Cited pages 4, 43, 44, 49, 50, and 161.)

[ABS00] ABITEBOUL S., BUNEMAN P., SUCIU D.: *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann, 2000. (Cited page 39.)

[ADdR03] ALECHINA N., DEMRI S., DE RIJKE M.: A modal perspective on path constraints, 2003. Journal of Logic and Computation. (Cited page 79.)

[AHV95] ABITEBOUL S., HULL R., VIANU V.: *Foundations of Databases*. 1995. (Cited page 42.)

[AKW95] AIKEN A., KOZEN D., WIMMERS E. L.: Decidability of systems of set constraints with negative constraints. *Information and Computation 122*, 1 (1995), 30–44. (Cited page 105.)

[AL08] ARENAS M., LIBKIN L.: XML data exchange: Consistency and query answering. *J. ACM 55*, 2 (2008). (Cited pages 4 and 161.)

[AM04] ALUR R., MADHUSUDAN P.: Visibly pushdown languages. In *36th ACM Symposium on Theory of Computing* (2004), ACM-Press, pp. 202–211. (Cited page 23.)

[ANR05] ANANTHARAMAN S., NARENDRAN P., RUSINOWITCH M.: Closure properties and decision problems of dag automata. *Inf. Process. Lett. 94*, 5 (2005), 231–240. (Cited page 90.)

[AU69] AHO A. V., ULLMAN J. D.: Translations on a context free grammar. In *STOC '69: Proceedings of the first annual ACM symposium on Theory of computing* (1969), pp. 93–112. (Cited page 22.)

[Bag06] BAGAN G.: Mso queries on tree decomposable structures are computable with linear delay. In *Computer Science Logic* (2006), vol. 4646 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 208–222. (Cited pages 6, 30, 31, 155, and 164.)

[BBC*07] BERGLUND A., BOAG S., CHAMBERLIN D., FERNÀNDEZ M. F., KAY M., ROBIE J., SIMÉON J.: XML path language (XPath) 2.0 w3c recommendation, 2007. http://www.w3.org/TR/2007/REC-xpath20-20070123/. (Cited pages 5, 6, 34, 35, 162, and 164.)

[BC05] BOJAŃCZYK M., COLCOMBET T.: Tree-walking automata do not recognize all regular languages. In *37th Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2005), ACM-Press, pp. 234–243. (Cited page 22.)

[BC06] BOJAŃCZYK M., COLCOMBET T.: Tree-walking automata cannot be determinized. *Theor. Comput. Sci. 350*, 2 (2006), 164–173. (Cited page 22.)

[BCF03a] BENZAKEN V., CASTAGNA G., FRISCH A.: CDuce: an XML-centric general-purpose language. In *8th ACM International Conf. on Functional Programming* (2003), pp. 51–63. (Cited pages 4, 7, 131, 161, and 164.)

[BCF03b] BENZAKEN V., CASTAGNA G., FRISCH A.: CDuce: an XML-centric general-purpose language. *ACM SIGPLAN Notices 38*, 9 (2003), 51–63. (Cited page 44.)

[BCF*07] BOAG S., CHAMBERLIN D., FERNÀNDEZ M. F., FLORESCU D., ROBIE J., SIMÉON J.: Xquery 1.0: An xml query language, w3c recommendation, 2007. http://www.w3.org/TR/2007/REC-xquery-20070123/. (Cited pages 5, 34, 43, and 162.)

[BDG07] BAGAN G., DURAND A., GRANDJEAN E.: On acyclic conjunctive queries and constant delay enumeration. In *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL* (2007), vol. 4646, pp. 208–222. (Cited pages 42 and 155.)

[BDM*06] BOJANCZYK M., DAVID C., MUSCHOLL A., SCHWENTICK T., SEGOUFIN L.: Two-variable logic on data trees and xml reasoning. In *ACM 25th Symp. on Principles of database systems* (2006), pp. 10–19. (Cited pages 26, 39, and 132.)

[Ber06] BERLEA A.: Online evaluation of regular tree queries. *Nordic Journal of Computing 13*, 4 (2006), 1–26. (Cited page 32.)

[BJ07] BENEDIKT M., JEFFREY A.: Efficient and expressive tree filters. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)* (2007), vol. 4855 of *Lecture Notes in Computer Science*, Springer, pp. 461–472. (Cited pages 32 and 154.)

[BK07] BENEDIKT M., KOCH C.: XPath leashed. *ACM computing surveys* (2007). to appear. (Cited page 37.)

[BKS02] BRUNO N., KOUDAS N., SRIVASTAVA D.: Holistic twig joins: optimal XML pattern matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2002), ACM Press, pp. 310–321. (Cited pages 42 and 154.)

[BKW98] BRUGGEMANN-KLEIN A., WOOD D.: One-unambiguous regular languages. *Information and Computation 140*, 2 (1998), 229–253. (Cited page 33.)

[BKW00] BRÜGGEMANN-KLEIN A., WOOD D.: Caterpillars, context, tree automata and tree pattern matching. In *Developments in Language Theory, Foundations, Applications, and Perspectives (1999)* (2000), World Scientific, pp. 270–285. (Cited pages 39 and 40.)

[BKWM01] BRÜGGEMANN-KLEIN A., WOOD D., MURATA M.: Regular tree and regular hedge languages over unranked alphabets: Version 1, Apr. 07 2001. (Cited pages 19 and 22.)

[BL02] BENEDIKT M., LIBKIN L.: Tree extension algebras: Logics, automata, and query languages. In *LICS* (2002), IEEE Computer Society, p. 203. (Cited page 152.)

[BL05] BARCELO P., LIBKIN L.: Temporal logics over unranked trees. In *20th Annual IEEE Symposium on Logic in Computer Science* (2005), IEEE Comp. Soc. Press, pp. 31–40. (Cited pages 41, 43, 71, 132, and 144.)

[BLN07] BENEDIKT, LIBKIN, NEVEN: Logical definability and query languages over ranked and unranked trees. *ACMTCL: ACM Transactions on Computational Logic 8* (2007). (Cited page 156.)

[BLSS01] BENEDIKT M., LIBKIN L., SCHWENTICK T., SEGOUFIN L.: A model-theoretic approach to regular string relations. In *LICS* (2001), p. 431. (Cited page 156.)

[BMS*06] BOJANCZYK M., MUSCHOLL A., SCHWENTICK T., SEGOUFIN L., DAVID C.: Two-variable logic on words with data. In *21th IEEE Symp. on Logic in Computer Science* (2006), pp. 7–16. (Cited pages 26 and 132.)

[Boj04] BOJAŃCZYK M.: *Decidable Properties of Tree Languages*. PhD thesis, Warsaw University, 2004. (Cited page 26.)

[Bon06] BONEVA I.: Logics for unranked and unordered trees and their use for querying semistructured data. PhD thesis. Université des Sciences et Technologies de Lille - Lille 1, 2006. (Cited pages 7, 43, 131, 138, 139, 145, 152, and 164.)

[BPSM*06] BRAY T., PAOLI J., SPERBERG-MCQUEEN C., MALER E., YERGEAU F.: Extensible markup language (xml) 1.0 (fourth edition), w3c recommendation. http://www.w3.org/TR/REC-xml/, 2006. (Cited pages 1, 33, and 157.)

[BS04a] BERLEA A., SEIDL H.: Binary queries for document trees. *Nordic Journal of Computing 11*, 1 (2004), 41–71. (Cited page 32.)

[BS04b] BERLEA A., SEIDL H.: Binary queries for document trees. *Nord. J. Comput 11*, 1 (2004), 41–71. (Cited page 154.)

[BS05a] BENEDIKT, SEGOUFIN: Towards a characterization of order-invariant queries over tame structures. In *CSL: 19th Workshop on Computer Science Logic* (2005), LNCS, Springer-Verlag. (Cited page 43.)

[BS05b] BENEDIKT M., SEGOUFIN L.: Regular tree languages definable in fo and fomod. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS)* (2005). (Cited page 26.)

[BSSS06a] BOJAŃCZYK M., SAMUELIDES M., SCHWENTICK T., SEGOUFIN L.: Expressive power of pebbles automata. In *International Colloquium on Automata Languages and Programming (ICALP'06)* (2006), Lecture Notes in Computer Science, Springer Verlag, pp. 157–168. (Cited page 22.)

[BSSS06b] BOJAŃCZYK M., SAMUELIDES M., SCHWENTICK T., SEGOUFIN L.: Expressive power of pebbles automata. In *International Colloquium on Automata Languages and Programming (ICALP'06)* (2006), Lecture Notes in Computer Science, Springer Verlag, pp. 157–168. (Cited page 30.)

[BT92] BOGAERT B., TISON S.: Equality and disequality constraints on direct subterms in tree automata. In *9th Annual Symposium on Theoretical Aspects of Computer Science* (1992), vol. 577 of *LNCS*, pp. 161–171. (Cited pages 89 and 90.)

[BT05] BONEVA I., TALBOT J.-M.: Automata and logics for unranked and unordered trees. In *20th International Conference on Rewriting Techniques and Applications* (2005), Lecture Notes in Computer Science, Springer Verlag. (Cited page 43.)

[BTT05] BONEVA I., TALBOT J., TISON S.: Expressiveness of a spatial logic for trees. In *20th IEEE Symposium on Logic in Computer Science* (2005), pp. 280–289. (Cited pages 7, 43, 131, and 164.)

[Büc60] BÜCHI J.: On a decision method in a restricted second order arithmetic. In *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science* (1960), Press. S. U., (Ed.), pp. 1–11. (Cited page 27.)

[CC05] COMON H., CORTIER V.: Tree automata with one memory, set constraints and cryptographic protocols. *TCS 331*, 1 (2005), 143–214. (Cited page 90.)

[CD94] COMON H., DELOR C.: Equational formulae with membership constraints. *Information and Computation 112*, 2 (1994), 167–216. (Cited page 126.)

[CD99] CLARD J., DEROSE S.: Xml path language (xpath): W3c recommendation, 1999. (Cited pages 6, 35, and 164.)

[CDG*07] COMON H., DAUCHET M., GILLERON R., LÖDING C., JACQUEMARD F., LUGIEZ D., TISON S., TOMMASI M.: Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2007. (Cited pages 7, 18, 19, 20, 21, 22, 23, 27, 32, 91, 102, 103, 120, 121, 122, 156, and 166.)

[CE82] CLARKE E. M., EMERSON E. A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop* (London, UK, 1982), Springer-Verlag, pp. 52–71. (Cited page 41.)

[CE95] COURCELLE B., ENGELFRIET J.: A logical characterization of the sets of hypergraphs defined by hyperedge replacement. *Mathematical Systems Theory 28*, 6 (1995), 515–552. (Cited page 29.)

[CFG02] CONFORTI G., FERRARA O., GHELLI G.: TQL algebra and its implementation. In *Proc. of IFIP TCS* (2002), Kluwer Academic Publishers, pp. 422–434. (Cited pages 7, 140, and 164.)

[CG04] CARDELLI L., GHELLI G.: TQL: A Query Language for Semistructured Data Based on the Ambient Logic. *Mathematical Structures in Computer Science 14* (2004), 285–327. (Cited pages 4, 7, 43, 129, 131, 141, 152, 153, 162, and 164.)

[CGG02] CARDELLI L., GARDNER P., GHELLI G.: A spatial logic for querying graphs. In *29th International Colloquium on Automata, Languages and Programming* (2002), vol. 2380 of *Lecture Notes in Computer Science*, Springer, pp. 597–610. (Cited page 43.)

[CGLN08] CHAMPAVÈRE J., GILLERON R., LEMAY A., NIEHREN J.: Efficient inclusion checking for deterministic tree automata and DTDs. In *2nd International Conference on Language and Automata Theory and Applications* (Apr. 2008), Lecture Notes in Computer Science, Springer Verlag. to appear. A long version is available at http.www.grappa.univ-lille3.fr/ niehren/inclusion-june-08.pdf. (Cited page 21.)

[Cha99] CHARATONIK W.: *Automata on DAG Representations of Finite Trees*. Tech. rep., 1999. (Cited page 90.)

[Che06] CHE D.: Mytwigstack: A holistic twig join algorithm with effective path merging support. In *7th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* (2006), IEEE Computer Society, pp. 184–189. (Cited pages 42 and 154.)

[Chu36] CHURCH A.: A note on the entscheidungsproblem. *Journal of Symbolic Logic 1*, 1 (1936), 40–41. (Cited page 26.)

[Cla99] CLARK J.: Xsl transformations (xslt) version 1.0, w3c recommendation, 1999. http://www.w3.org/TR/1999/REC-xslt-19991116. (Cited pages 5, 34, and 162.)

[CM01] CLARK J., MURATA M.: Relax ng specification. http://www.oasis-open.org/committees/relax-ng/spec-20011203.html, 2001. (Cited page 34.)

[CNT04] CARME J., NIEHREN J., TOMMASI M.: Querying unranked trees with stepwise tree automata. In *19th International Conference on Rewriting Techniques and Applications* (2004), vol. 3091 of *Lecture*

Notes in Computer Science, Springer Verlag, pp. 105 – 118. (Cited page 23.)

[CO07] COURCELLE B., OUM S.: Vertex-minors, monadic second-order logic, and a conjecture by seese. *J. Comb. Theory Ser. B 97*, 1 (2007), 91–126. (Cited page 29.)

[Cou90a] COURCELLE B.: Graph rewriting: and algebraic and logic approach. In *Handbook of Theoretical Computer Science*, vol. B. Elsevier, 1990, pp. 193–242. (Cited page 29.)

[Cou90b] COURCELLE B.: The monadic second-order logic of graphs I. recognizable sets of finite graphs. *Information and Computation 85* (1990), 12–75. (Cited page 43.)

[Cou91] COURCELLE B.: The monadic second-order logic of graphs v: On closing the gap between definability and recognizability. *Theor. Comput. Sci. 80* (1991), 153–202. (Cited page 43.)

[Cou94] COURCELLE B.: Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science 126*, 1 (1994), 53–75. (Cited pages 29 and 155.)

[Cou97] COURCELLE B.: Handbook of graph grammars and computing by graph transformations, volume 1: Foundations. In *Handbook of Graph Grammars* (1997), Rozenberg G., (Ed.). (Cited page 29.)

[Cou07] COURCELLE B.: Linear delay enumeration and monadic second-order logic. To Appear in Discrete Applied Mathematics. (Cited pages 6, 30, and 164.)

[CP94] CHARATONIK W., PACHOLSKI L.: Set constraints with projections are in NEXPTIME. In *IEEE Symposium on Foundations of Computer Science* (1994). (Cited page 105.)

[CT01] CHARATONIK W., TALBOT J.: The Decidability of Model Checking Mobile Ambients. In *15th Annual Conference of the European Association for Computer Science Logic* (2001), vol. 2142 of *Lecture Notes in Computer Science*, Springer, pp. 339–354. (Cited page 141.)

[CW87] COPPERSMITH D., WINOGRAD S.: Matrix multiplication via arithmetic progressions. In *ACM conference on Theory of computing* (1987). (Cited page 81.)

[DCC95] DAUCHET M., CARON A.-C., COQUIDÉ J.-L.: Reduction properties and automata with constraints. *JSC 20* (1995), 215–233. (Cited pages 89 and 90.)

[DdVPS00] DAMIANI E., DI VIMERCATI S. D. C., PARABOSCHI S., SAMARATI P.: Securing xml documents. In *EDBT '00: Proceedings of the 7th International Conference on Extending Database Technology* (London, UK, 2000), Springer-Verlag, pp. 121–135. (Cited page 155.)

[DdVPS02] DAMIANI E., DI VIMERCATI S. D. C., PARABOSCHI S., SAMARATI P.: A fine-grained access control system for xml documents. *ACM Trans. Inf. Syst. Secur. 5*, 2 (2002), 169–202. (Cited page 155.)

[DFGM08] DAMIANI E., FANSI M., GABILLON A., MARRARA S.: A general approach to securely querying xml. *Comput. Stand. Interfaces 30*, 6 (2008), 379–389. (Cited page 155.)

[DG06] DURAND A., GRANDJEAN E.: The complexity of acyclic conjunctive queries revisited. *CoRR abs/cs/0605008* (2006). informal publication. (Cited page 42.)

[DGG04] DAWAR A., GARDNER P., GHELLI G.: *Expressiveness and Complexity of Graph Logic*. Tech. rep., Imperial College, 2004. (Cited page 43.)

[dM60] DE MORGAN A.: On the syllogism, no. iv, and on the logic of relations. *Trans. Cambridge Phi. Soc. 10* (1860), 331–358. (Cited page 40.)

[DMJ01] DEROSE S., MALER E., JR. R. D.: Xml pointer language (xpointer) version 1.0, 2001. http://www.w3.org/TR/2001/WD-xptr-20010108/. (Cited pages 5, 35, and 162.)

[Don70] DONER J. E.: Tree acceptors and some of their applications. *Journal of Computer and System Science 4* (1970), 406–451. (Cited pages 18, 26, 27, and 28.)

[EF05] EBBINGHAUS H., FLUM J.: *Finite Model Theory*. Springer Verlag, Berlin, 2005. (Cited pages 25, 41, 63, 67, and 134.)

[EH99] ENGELFRIET J., HOOGEBOOM H. J.: Tree-walking pebble automata. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa* (1999), Springer-Verlag, pp. 72–83. (Cited page 22.)

[EH07] ENGELFRIET J., HOOGEBOOM H. J.: Automata with nested pebbles capture first-order logic with transitive closure. *LMCS 3* (2007), 3. (Cited page 22.)

[EHB99] ENGELFRIET J., HOOGEBOOM H. J., BEST J.-P. V.: Trips on trees. *Acta Cybern. 14*, 1 (1999), 51–64. (Cited page 22.)

[EJ91] EMERSON E. A., JUTLA C. S.: Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd annual symposium on Foundations of computer science* (1991), pp. 368–377. (Cited page 41.)

[EvO97] ENGELFRIET J., VAN OOSTROM V.: Logical description of context-free graph languages. *J. Comput. Syst. Sci. 55*, 3 (1997), 489–503. (Cited page 29.)

[FG02] FRICK M., GROHE M.: The complexity of first-order and monadic second-order logic revisited. In *LICS '02: Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science* (Washington, DC, USA, 2002), pp. 215–224. (Cited pages 28 and 29.)

[FGK03] FRICK M., GROHE M., KOCH C.: Query evaluation on compressed trees. In *18th IEEE Symposium on Logic in Computer Science* (2003), pp. 188–197. (Cited page 31.)

[FNTT07] FILIOT E., NIEHREN J., TALBOT J.-M., TISON S.: Polynomial time fragments of xpath with variables. In *ACM Symposium on Principles of Database Systems* (2007). (Cited page 38.)

[FTT07] FILIOT E., TALBOT J.-M., TISON S.: Satisfiability of a spatial logic with tree variables. In *Computer Science Logic* (2007), pp. 130–145. (Cited page 144.)

[FTT08] FILIOT E., TALBOT J.-M., TISON S.: Tree automata with global constraints. In *12th International Conference on Developments in Language Theory (DLT)* (2008), Lecture Notes in Computer Science, Springer Verlag. To appear. (Cited pages 127 and 155.)

[FW04] FALLSIDE D. C., WALMSLEY P.: Xml schema part 0: Primer second edition, October 2004. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/. (Cited pages 5, 34, and 162.)

[GCNT08] GAUWIN O., CARON A.-C., NIEHREN J., TISON S.: Complexity of earliest query answering with streaming tree automata. In *ACM SIGPLAN Workshop on Programming Language Techniques for XML (PLAN-X)* (Jan. 2008). PLAN-X Workshop of ACM POPL. (Cited pages 32 and 154.)

[Gen06] GENEVÈS P.: *Logics for XML*. PhD thesis, Institut National Polytechnique de Grenoble, December 2006. (Cited pages 7, 18, 38, 41, 43, 120, 131, 152, and 164.)

[Giv06] GIVANT S.: The calculus of relations as a foundation for mathematics. *J. Autom. Reason. 37*, 4 (2006), 277–322. (Cited page 40.)

[GK04] GOTTLOB G., KOCH C.: Monadic datalog and the expressive power of languages for web information extraction. *Journal of the ACM 51*, 1 (2004), 74–113. (Cited pages 4, 41, 42, and 161.)

[GKB*04] GOTTLOB G., KOCH C., BAUMGARTNER R., HERZOG M., FLESCA S.: The Lixto data extraction project - back and forth between theory and practice. In *23rd ACM SIGPLAN-SIGACT Symposium on Principles of Database Systems* (2004), ACM-Press, pp. 1–12. (Cited page 42.)

[GKP03a] GOTTLOB G., KOCH C., PICHLER R.: The complexity of xpath query evaluation. In *22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (2003), pp. 179–190. (Cited page 37.)

[GKP03b] GOTTLOB G., KOCH C., PICHLER R.: Xpath processing in a nutshell. *SIGMOD Rec. 32*, 2 (2003), 21–27. (Cited page 37.)

[GKP05] GOTTLOB G., KOCH C., PICHLER R.: Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems 30*, 2 (2005), 444–491. (Cited pages 35, 37, and 81.)

[GKPS05] GOTTLOB G., KOCH C., PICHLER R., SEGOUFIN L.: The complexity of xpath query evaluation and xml typing. *J. ACM 52*, 2 (2005), 284–335. (Cited page 38.)

[GKS04] GOTTLOB G., KOCH C., SCHULZ K. U.: Conjunctive queries over trees. In *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (New York, NY, USA, 2004), ACM-Press, pp. 189–200. (Cited pages 4, 42, 43, 70, 154, and 161.)

[GKV97] GRÄDEL E., KOLAITIS P. G., VARDI M. Y.: On the decision problem for two-variable first-order logic. *The Bulletin of Symbolic Logic 3*, 1 (1997), 53–69. (Cited page 26.)

[GL06] GENEVÈS P., LAYAÏDA N.: A system for the static analysis of XPath. *ACM Trans. Inf. Syst. 24*, 4 (2006), 475–502. (Cited page 38.)

[GLS07] GENEVÈS P., LAYAÏDA N., SCHMITT A.: Efficient static analysis of XML paths and types. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation* (2007), pp. 342–351. (Cited page 41.)

[GM05] GORIS E., MARX M.: Looping caterpillars. In *Proceedings of the Twentieth Annual IEEE Symp. on Logic in Computer Science, LICS 2005* (June 2005), Panangaden P., (Ed.), IEEE Comp. Soc. Press, pp. 51–60. (Cited page 40.)

[GMT08] GODOY G., MANETH S., TISON S.: Classes of tree homomorphisms with decidable preservation of regularity. In *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS* (2008), vol. 4962 of *Lecture Notes in Computer Science*, Springer, pp. 127–141. (Cited page 127.)

[Grz53] GRZEGORCZYK A.: Some classes of recursive functions. *Rozprawy Matematyczne 4* (1953), 1–45. (Cited page 28.)

[GS00] GROHE, SCHWENTICK: Locality of order-invariant first-order formulas. *ACMTCL: ACM Transactions on Computational Logic 1* (2000). (Cited page 43.)

[GTT94] GILLERON R., TISON S., TOMMASI M.: Some new decidability results on positive and negative set constraints. In *CCL '94: Proceedings of the First International Conference on Constraints in Computational Logics* (1994), pp. 336–351. (Cited page 105.)

[GTW02] GRÄDEL E., THOMAS W., WILKE T. (Eds.): *Automata logics, and infinite games: a guide to current research*. Springer-Verlag New York, Inc., New York, NY, USA, 2002. (Cited page 41.)

[Hid03] HIDDERS J.: Satisfiability of XPath expressions. In *The 9th International Workshop on Data Base Programming Languages* (2003), pp. 21–36. (Cited page 39.)

[HP03a] HOSOYA H., PIERCE B.: Regular expression pattern matching for XML. *Journal of Functional Programming 6*, 13 (2003), 961–1004. (Cited page 44.)

[HP03b] HOSOYA H., PIERCE B. C.: XDuce: A statically typed xml processing language. *ACM Trans. Internet Techn. 3*, 2 (2003), 117–148. (Cited pages 4, 7, 131, 161, and 164.)

[HT87] HAFER T., THOMAS W.: Computation tree logic ctl* and path quantifiers in the monadic theory of the binary tree. In *14th International Colloquium on Automata, languages and programming* (London, UK, 1987), Springer-Verlag, pp. 269–279. (Cited page 41.)

[IK89] IMMERMAN N., KOZEN D.: Definability with bounded number of bound variables. *Information and Computation 83* (1989), 121–139. (Cited page 26.)

[Imm87] IMMERMAN N.: Languages that capture complexity classes. *SIAM J. Comput. 16*, 4 (1987), 760–778. (Cited page 30.)

[JRV06] JACQUEMARD F., RUSINOWITCH M., VIGNERON L.: *Tree automata with equality constraints modulo equational theories*. Research Report LSV-06-07, LSV, ENS Cachan, France, 2006. (Cited pages 89 and 90.)

[JWLY03] JIANG H., WANG W., LU H., YU J. X.: Holistic twig joins on indexed XML documents. In *Proceeding of VLDB* (2003), pp. 273–284. (Cited pages 42 and 154.)

[Kay08] KAY M.: *XSLT 2.0 and XPath 2.0 Programmer's Reference*. Wrox, 4th edition, 2008. (Cited page 35.)

[KL06] KARIANTO W., LÖDING C.: *Unranked Tree Automata with Sibling Equalities and Disequalities*. Research Report 0935-3232, RWTH Aachen, Germany, Oct. 2006. 34 pages. (Cited pages 89 and 90.)

[KM06] KUTSIA T., MARIN M.: Solving regular constraints for hedges and contexts. In *Proceedings of the 20th International Workshop on Unification (UNIF'06)* (2006), pp. 89–107. (Cited page 126.)

[Koc05] KOCH C.: On the complexity of nonrecursive XQuery and functional query languages on complex values. In *24th SIGMOD-SIGACT-SIGART Symposium on Principles of Database systems* (2005), ACM-Press, pp. 84–97. (Cited page 43.)

[Koz83] KOZEN D.: Results on the propositional μ-calculus. *Theoretical Computer Science 27*, 1 (1983), 333–354. (Cited pages 41 and 132.)

[Lew80] LEWIS H. R.: Complexity results for classes of quantificational formulas. *J. Comput. Syst. Sci. 21*, 3 (1980), 317–353. (Cited page 26.)

[Lib04a] LIBKIN L.: *Elements of Finite Model Theory*. Springer Verlag, 2004. (Cited page 25.)

[Lib04b] LIBKIN L.: *Elements Of Finite Model Theory*. SpringerVerlag, 2004. (Cited pages 43 and 63.)

[Lib06] LIBKIN L.: Logics over unranked trees: an overview. *Logical Methods in Computer Science 3*, 2 (2006), 1–31. (Cited pages 4, 6, 19, 27, 41, 43, 161, and 164.)

[Loh01] LOHREY M.: On the parallel complexity of tree automata. In *International Conference on Rewriting Techniques and Applications (RTA)* (2001), pp. 201–215. (Cited page 21.)

[Mar04] MARX M.: XPath with conditional axis relations. In *9th International Conference on Extending Database Technology* (2004), vol. 2992, Springer Verlag, pp. 477–494. (Cited page 39.)

[Mar05a] MARX M.: Conditional XPath. *ACM Transactions on Database Systems 30*, 4 (2005), 929–959. (Cited pages 6, 38, 39, 42, 49, 50, 63, 73, 75, 76, and 164.)

[Mar05b] MARX M.: First order paths in ordered trees. In *International Conference on Database Theory* (2005), pp. 114–128. (Cited pages 26, 38, 78, and 80.)

[Mar05c] MARX M.: Xml navigation and tarski's relation algebras. In *Computer Science Logic* (2005). (Cited page 40.)

[Mat02] MATEESCU R.: Local model-checking of modal mu-calculus on acyclic labeled transition systems. In *TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2002), Springer-Verlag, pp. 281–295. (Cited page 41.)

[MBPS05] MANETH S., BERLEA A., PERST T., SEIDL H.: XML type checking with macro tree transducers. In *Proceedings of the Twenty-Fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)* (2005), ACM Press, pp. 283–294. (Cited pages 4 and 161.)

[MdR05] MARX M., DE RIJKE M.: Semantic characterizations of navigational xpath. *SIGMOD Rec. 34*, 2 (2005), 41–46. (Cited page 38.)

[Mey73] MEYER A. R.: *WEAK MONADIC SECOND ORDER THEORY OF SUCCESSOR IS NOT ELEMENTARY-RECURSIVE*. Tech. rep., Massachusetts Institute of Technology, 1973. (Cited page 28.)

[MLM01] MURATA M., LEE D., MANI M.: Taxonomy of XML schema languages using formal language theory. In *Extreme Markup Languages* (Montreal, Canada, 2001). (Cited pages 33 and 34.)

[MNS04] MARTENS W., NEVEN F., SCHWENTICK T.: Complexity of decision problems for simple regular expressions. In *Mathematical Foundations of Computer Science 2004, 29th International Symposium* (2004), pp. 889–900. (Cited page 33.)

[MNSB06] MARTENS W., NEVEN F., SCHWENTICK T., BEX G. J.: Expressiveness and complexity of xml schema. *ACM Trans. Database Syst. 31*, 3 (2006), 770–813. (Cited pages 27 and 34.)

[Mon81] MONGY J.: *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, Université de Lille, 1981. (Cited pages 90 and 95.)

[Mor75] MORTIMER M.: On languages with two variables. *Z. Math. Logik Grundlagen Math 70*, 4 (1975), 569–572. (Cited page 26.)

[Mor94] MORIYA E.: On two-way tree automata. *Inf. Process. Lett. 50*, 3 (1994), 117–121. (Cited page 32.)

[MP71] MCNAUGHTON R., PAPERT S. A.: *Counter-Free Automata (M.I.T. research monograph no. 65)*. The MIT Press, 1971. (Cited pages 24 and 26.)

[MR03] MOLLER F., RABINOVICH A.: Counting on CTL: on the expressive power of monadic path logic. *Information and Computation 184*, 1 (2003), 147–159. (Cited pages 49, 50, and 63.)

[MS04] MIKLAU G., SUCIU D.: Containment and equivalence for a fragment of XPath. *Journal of the ACM 51*, 1 (2004), 2–45. (Cited page 38.)

[Nev00] NEVEN F.: Extensions of attribute grammars for structured document queries. In *DBPL '99: Revised Papers from the 7th International Workshop on Database Programming Languages* (2000), Springer-Verlag, pp. 99–116. (Cited page 32.)

[Nev02] NEVEN F.: Automata, logic, and XML. In *Computer Science Logic* (2002), Lecture Notes in Computer Science, Springer Verlag, pp. 2–26. (Cited pages 11, 19, and 27.)

[Ng84] NG K. C.: *Relation Algebras with Transitive Closure*. PhD thesis, University of California, Berkeley, 1984. (Cited page 40.)

[Niw88] NIWINSKI D.: Fixed points vs. infinite generation. In *Proceedings of the Proceedings of the Third Annual Symposium on Logic in Computer Science* (1988), pp. 402–409. (Cited page 41.)

[NPTT05] NIEHREN J., PLANQUE L., TALBOT J.-M., TISON S.: N-ary queries by tree automata. In *10th International Symposium on Database Programming Languages* (2005), vol. 3774 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 217–231. (Cited pages 4, 27, 31, 43, 122, and 161.)

[NS00] NEVEN F., SCHWENTICK T.: Expressive and efficient pattern languages for tree-structured data. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (2000), ACM, pp. 145–156. (Cited pages 27, 29, and 43.)

[NS02a] NEVEN F., SCHWENTICK T.: Query automata over finite trees. *Theoretical Computer Science 275*, 1-2 (2002), 633–674. (Cited pages 27, 31, 32, and 65.)

[NS02b] NEVEN F., SCHWENTICK T.: Xpath containment in the presence of disjunction, dtds, and variables. In *ICDT '03: Proceedings of the 9th International Conference on Database Theory* (London, UK, 2002), Springer-Verlag, pp. 315–329. (Cited pages 38 and 89.)

[NT77] NG K. C., TARSKI A.: Relation algebras with transitive closure. *Notices of the American Mathematical Society 24* (1977), A29–A30. (Cited page 40.)

[NV02] NEVEN F., VAN DEN BUSSCHE J.: Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM 49*, 1 (2002), 56–100. (Cited pages 4, 27, 32, 43, and 161.)

[Olt07] OLTEANU D.: SPEX: Streamed and progressive evaluation of XPath. *IEEE Trans. Knowl. Data Eng 19*, 7 (2007), 934–949. (Cited page 154.)

[Pap94] PAPADIMITRIOU C.: *Computational Complexity*. Addison Wesley, 1994. (Cited page 21.)

[PQ68] PAIR C., QUERE A.: Définition et étude des bilangages réguliers. *Information and Control 13*, 6 (1968), 565–593. (Cited pages 19 and 22.)

[Pra92] PRATT V. R.: Origins of the calculus of binary relations. In *Logic in Computer Science* (1992), pp. 248–254. (Cited page 40.)

[PV00] PAPAKONSTANTINOU Y., VIANU V.: DTD Inference for Views of XML Data. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (2000), pp. 35–46. (Cited page 34.)

[Rey02] REYNOLDS J. C.: Separation logic: A logic for shared mutable data structures. In *17th IEEE Symp. on Logic in Computer Science* (2002), IEEE, pp. 55–74. (Cited page 43.)

[Sch65] SCHÜTZENBERGER M. P.: On finite monoids having only trivial subgroups. *Information and Control 8* (1965), 190–194. (Cited page 26.)

[Sch00] SCHWENTICK T.: On diving in trees. In *25th International Symposium on Mathematical Foundations of Computer Science* (2000), pp. 660–669. (Cited pages 6, 43, 44, 49, 50, 63, and 164.)

[Sch04] SCHWENTICK T.: Trees, automata and xml. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2004), ACM, pp. 222–222. (Cited page 19.)

[Sch07] SCHWENTICK T.: Automata for xml – a survey. *J. Comput. Syst. Sci. 73*, 3 (2007), 289–315. (Cited pages 4, 19, 22, 32, 89, and 161.)

[See91] SEESE: The structure of the models of decidable monadic theories of graphs. *Annals of Pure and Applied Logic 53* (1991). (Cited page 29.)

[See96] SEESE D.: Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science 6*, 6 (1996), 505–526. (Cited page 29.)

[Sei92] SEIDL H.: Finite tree automata with cost functions. In *CAAP '92: Proceedings of the 17th Colloquium on Trees in Algebra and Programming* (1992), Springer-Verlag, pp. 279–299. (Cited page 143.)

[Sei96] SEIDL: Least and greatest solutions of equations over N. In *Nordic Journal of Computing*, vol. 3. 1996. (Cited page 142.)

[She] SHELAH S.: The monadic theory of order. *Annals of Mathematics*, 102, 379–419. (Cited pages 6, 49, 50, 63, and 164.)

[Shi08] SHIPMAN J. W.: *XSLT Reference*. 2008. (Cited pages 36 and 190.)

[SIP96] SIPSER M.: *Introduction to the Theory of Computation*. PWS, Boston, MA, 1996. (Cited page 141.)

[SM73a] STOCKMEYER L. J., MEYER A. R.: Word problems requiring exponential time. In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing* (1973), pp. 1–9. (Cited pages 28 and 123.)

[SM73b] STOCKMEYER L. J., MEYER A. R.: Word problems requiring exponential time: Preliminary report. In *STOC* (1973), pp. 1–9. (Cited page 144.)

[Ste94] STEFANSSON K.: Systems of set constraints with negative constraints are nexptime-complete. In *IEEE Symposium on Logic in Computer Science* (1994). (Cited page 105.)

[Sto74] STOCKMEYER L. J.: *The Complexity of Decision Problems in Automata Theory*. PhD thesis, Department of Electrical Engineering, MIT, 1974. (Cited pages 25, 28, 29, and 77.)

[SV01] SATTLER, VARDI: The hybrid mu-calculus. In *IJCAR: International Joint Conference on Automated Reasoning* (2001), LNCS. (Cited page 71.)

[Tak75] TAKAHASHI M.: Generalizations of regular sets and their application to a study of context-free languages. *Information and Control 27*, 1 (1975), 1–36. (Cited pages 19 and 22.)

[Tar41] TARSKI A.: On the calculus of relations. *Journal of Symbolic Logic 6*, 3 (1941), 73–89. (Cited page 40.)

[tC06] TEN CATE B.: The expressiveness of XPath with transitive closure. In *25st ACM SIGMOD-SIGACT Symposium on Principles of Database Systems* (2006), ACM-Press. (Cited page 39.)

[tCL07] TEN CATE B., LUTZ C.: The complexity of query containment in expressive fragments of xpath 2.0. In *PODS '07: Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2007), ACM, pp. 73–82. (Cited page 38.)

[tCM07] TEN CATE B., MARX M.: Axiomatizing the logical core of XPath 2.0. In *International Conference on Database Theory* (2007). (Cited pages 35 and 38.)

[tCS08] TEN CATE B., SEGOUFIN L.: Xpath, transitive closure logic, and nested tree walking automata. In *PODS '08: Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2008). (Cited pages 22, 30, and 39.)

[Tho84] THOMAS W.: Logical aspects in the study of tree languages. In *Proceedings of the 9th International Colloquium on Trees in Algebra and Programming, CAAP '84* (1984), pp. 31 – 50. (Cited pages 6, 49, 50, 63, and 164.)

[Tho97] THOMAS W.: Languages, automata, and logic. *Handbook of formal languages, vol. 3: beyond words* (1997), 389–455. (Cited pages 18, 27, and 28.)

[Tra75] TRAKHTENBROT B.: The impossibility of an algorithm for the decidability problem on finite classes. *Doklady AN SSR 21* (1975), 135–140. (Cited page 26.)

[Tur37] TURING A. M.: Computability and lambda-definability. *Journal of Symbolic Logic 2*, 4 (1937), 153–163. (Cited page 26.)

[TW68] THATCHER J. W., WRIGHT J. B.: Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory 2* (1968), 57–82. (Cited pages 4, 18, 26, 27, 28, 122, and 161.)

[Var82] VARDI M. Y.: The complexity of relational query languages (extended abstract). In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing* (1982), ACM, pp. 137–146. (Cited pages 25 and 29.)

[Var95] VARDI M. Y.: On the complexity of bounded-variable queries. In *Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (1995), pp. 266–276. (Cited page 25.)

[Var98] VARDI M. Y.: Reasoning about the past with two-way automata. In *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming* (1998), pp. 628–641. (Cited page 41.)

[VJK08] VACHER C., JACQUEMARD F., KLAY F.:, 2008. Mostrare Seminar, INRIA Lille-Nord Europe. (Cited page 127.)

[Wik] XML Wikipedia Database. http://en.wikipedia.org/wiki/Wikipedia:Database_download. (Cited pages 70 and 163.)

[Woo03] WOOD P. T.: Containment for xpath fragments under dtd constraints. In *ICDT '03: Proceedings of the 9th International Conference on Database Theory* (London, UK, 2003), Springer-Verlag, pp. 300–314. (Cited page 38.)

[Yan81] YANNAKAKIS M.: Algorithms for acyclic database schemes. In *Proceeding of VLDB* (1981), IEEE Computer Society, pp. 82–94. (Cited pages 42, 57, 58, and 61.)

[Zei94] ZEITMAN R. S.: *The composition method*. PhD thesis, 1994. Adviser-Yuri Gurevich and Adviser-Alexis Manaster Ramer. (Cited page 63.)

[ZXM07] ZHOU J., XIE M., MENG X.: Twigstack+: Holistic twig join pruning using extended solution extension. 855–860. (Cited pages 42 and 154.)

NOTATIONS

Notation	Description	Section
	Generic Notations	
$\stackrel{\text{def}}{=}$	definition	
$\mathbb{N}$	set of natural numbers	
$\circ$	composition operator	
$\|A\|$	cardinality of the set A	
$\|\|A\|\|$	size of the representation of A	
2^S	powerset of set S	
π_i	i-th projection	
$[\![.]\!]$	semantics	
$(\!\|.\|\!)$	translation	
A	tree automaton	
$\mathcal{L}(A)$	language recognized by A	
$\|\|A\|\|$	size of the tree automaton A	2.3.1
$\mathfrak{q}$	query expression	2.2.2
$\mathbb{Q}$	set of query expressions	2.2.2
$\mathcal{Q}(.)$	interpretation of query expressions as queries	2.2.2
	Trees and Hedges	
$\mathcal{P}_{cf}(\Lambda)$	cofinite (and finite) subsets of Λ	2.1.7
Σ	finite alphabet (ranked or unranked)	2.1.3
Λ	infinite alphabet (ranked or unranked)	2.1.7
$\sigma_{unr}(\Sigma)$	signature of unranked trees over Σ	2.1.3
$\|\|t\|\|$	size of the tree t	2.1.3
$\text{Dom}(t)$	domain of t	2.1.2
$\prec_{fc}^t$	first-child relation in t	2.1.3
$\prec_{ns}^t$	next-sibling relation in t	2.1.3
$\prec_{ns^+}^t$	transitive closure of $\prec_{ns}^t$	**??**
$\prec_{ns^*}^t$	transitive and reflexive closure of $\prec_{ns}^t$	2.1.3
$\prec_{ch}^t$	child relation in t	2.1.3
$\prec_{ch^+}^t$	transitive closure of $\prec_{ch}^t$	2.1.3
$\prec_{ch^+}^t$	transitive and reflexive closure of $\prec_{ch}^t$	2.1.3
root^t	root of t	2.1.3
$\mathsf{lab}^t(u)$	label of u in t	2.1.3
lab_a^t	set of nodes of t labeled a	2.1.3
$t\|_u$	subtree of t at nodes u	2.1.3
$h\|_u^{sh}$	subhedge of h at nodes u	3.5.2
$\mathcal{T}_{unr}(\Sigma)$	set of unranked trees over Σ	2.1.3
$\mathcal{H}(\Sigma)$	set of hedges over Σ	2.1.4

Notation	Description	Section
$\mathcal{T}_{ran}(\Sigma)$	set of ranked trees over Σ	2.1.5
$\mathcal{T}_i(\Sigma)$	set of ranked trees over Σ of arity at most i	2.1.5
$\mathcal{T}_{wran}(\Sigma)$	set of weakly ranked trees over Σ	2.1.5
$\mathsf{ar}(\Sigma)$	arity of alphabet Σ	
$\mathsf{node}^t(w)$	node of t at position w	
$\mathsf{0}$	empty hedge	2.1.4
$.\|.$	hedge concatenation	2.1.4
	FO, MSO and the Composition Language	
$\|\phi\|$	size of the FO or MSO formula ϕ	2.4.1
$\models$	satisfiability relation	2.4.1
ρ	valuation	2.4.1
$\mathrm{Var}(\phi)$	variables of ϕ	2.4.1
$\mathrm{FVar}(\phi)$	free variables of ϕ	2.4.1
$\mathrm{type}_n^{\mathcal{L}}(M)$	set of $\mathcal{L}$-formulas of quantifier depth less than n satisfied in the structure M	3.5.1
$\equiv_n^{\mathcal{L}}$	equivalence relation on structures	3.5.1
FO^k	k-variable fragment of FO	
FO_{bin}	FO formulas with two free variables	
$\circ$	composition operator	3.2.1
$\|\phi\|$	size of the composition formula ϕ	3.2.1
$\mathcal{C}(L)$	composition formulas over L	3.2.1
$\mathcal{C}^{\mathrm{nvs}}(L)$	composition formulas over L with non-variable sharing	3.2.2
$\mathrm{NVS}(\circ)$	non-variable sharing restriction	3.2.2
$\mathrm{llca}(u,v)$	least left common ancestor	**??**
Axis	set of XPath axis	2.8
$x \sim y$	tree equality predicate	5.6
$\mathrm{MSO}[\sim]$	MSO formulas over $\prec_{ch_1}, \prec_{ch_2}$ and $\sim$	5.6
$\mathsf{diff}_k X, Y$	k-bounded difference predicate	5.6
$\mathrm{MSO}_{=}^{\exists}$	existential MSO fragment with equality tests	5.6
	TAGED	
$=_A$	equality relation on states	5.2
$\neq_A$	disequality relation on states	5.2
$\mathrm{dom}(=_A)$	domain of $=_A$	5.2
$\mathrm{dom}(\neq_A)$	domain of $\neq_A$	5.2
$\|A\|$	size of the TAGED A	5.2
id_Q	identity relation on Q	
$\mathcal{L}(A)$	language recognized by A	5.2
$\mathrm{path}_t(u,v)$	shortest path from u to v in t	5.3.3
$\sim_{t,r}$	equivalence relation on nodes	5.3.3
$\leftrightarrow_{t,r}$	pre-equivalence relation on nodes	5.3.3
(A,k)	vertically bounded TAGED	5.5
$\mathcal{C}r$	roots of elementary contexts of r	5.5.1
$\mathrm{cxt}_r(u)$	elementary context rooted at node u	5.5.1
$\prec_{\sim_{t,r}}$	partial order on $\sim_{t,r}$-equivalence classes	5.5.1

Notation	Description	Section
$[u]_{t,r}$	$\sim_{t,r}$-equivalence class of u	5.5.1
F	frontier	5.5.1
$F_{max}(t,r)$	maximal frontier of t and r	5.5.1
$\mathbf{P}$	predicate	5.5.1
Rep	repair predicate	5.5.1
$\uparrow(U)$	set of ancestors of the nodes of U	5.5.3
	TQL	
ξ	recursion variables	6.2.1
$\mu\xi.\phi$	least fixpoint constructor	6.2.1
FL(ϕ)	Fisher-Ladner closure of ϕ	6.2.1
rd(ϕ)	recursion-depth of ϕ	6.2.1
$\prec_{\mathcal{H}(\Lambda)}$	partial order on hedges over Λ	6.2.1
$\prec$	partial order on hedges and TQL formulas	6.2.1
ρ	valuation of tree variables	6.2.2
δ	valuation of recursion variables	6.2.2
$[\![.]\!]^{\rho,\delta}$	semantics of TQL formulas under the valuations ρ and δ	6.2.2
$\phi[\xi \mapsto \phi']$	substitution of ξ by ϕ' in ϕ	6.2.1
qd(ϕ)	quantifier depth of ϕ	2.4.1
$\mathsf{0}$	empty hedge	2.1.4
$.\vert.$	hedge concatenation	2.1.4
$\widehat{X}$	negation of X in the tree domain ($\widehat{X} = \Lambda[\top] \wedge \neg X$)	6.6
A_ϕ	TAGED associated with ϕ	6.6.3
Vert(ϕ)	number of vertical variable occurences of ϕ	6.5.2
$\text{Sub}_{\text{ext}}(\phi)$	formulas of FL(ϕ) of the form $\alpha[\psi]$, X, or $\widehat{X}$	6.6.2
$w \oplus w'$	word combination	6.6.2
$\mathsf{hedges}(w,\rho)$	set of hedges defined by w under the valuation ρ	6.6.2
H_ψ	horizontal language defined by ψ	6.6.2
ρ_r	valuation associated with the run r	6.6.5

LIST OF FIGURES