École polytechnique
Promotion X2006
Jugé Vincent

Rapport de stage de recherche

Complexity of Decision Problems in Computational Logic

RAPPORT NON CONFIDENTIEL

Option :	Informatique
Champ de l'option :	Logique
Directeur de l'option :	Gilles Dowek
Directeur de stage :	Moshe Y. Vardi
Dates du stage :	8 avril 2009 – 17 juillet 2009
Adresse de l'organisme :	William Marsh Rice University Computer Science Department 3122 Duncan Hall, 6100 Main Street 77005 Houston, Texas États-Unis d'Amérique

Abstract

This report presents the study of algorithms whose goal is to determine whether two Datalog programs are equivalent. Datalog is a query language used over deductive databases. Datalog is a fragment of Prolog, and consists in Horn clauses without function symbols.

In general, it is undecidable whether two given Datalog programs are equivalent one to each other; however, this problem becomes decidable if we look at restricted classes of Datalog programs: *monadic* Datalog programs and *transitive* Datalog programs. Indeed, the algorithms designed in order to solve the equivalence problem of such Datalog programs involve similar techniques.

This report first presents Datalog programs in general, then the two particular classes of Datalog programs above mentioned. It continues with a study of the links between Datalog programs,a finite trees and finite automata. It eventually presents the algorithms designed during the internship, which solve a problem slightly more general than the equivalence problem: the inclusion of a Datalog program in a *monadic* Datalog program, then in a *transitive* Datalog program.

Résumé

Ce rapport présente l'étude d'algorithmes dont le but est d'évaluer l'équivalence entre des programmes Datalog. Datalog est un langage de programmation par requêtes, utilisé sur des bases de données déductives. Datalog constitue un fragment du langage Prolog, consistant en des clauses de Horn sans symboles de fonction.

Si l'équivalence entre deux programmes Datalog est, de manière générale, indécidable, il est cependant possible de résoudre ce problème dans le cadre d'instances particulières de programmes Datalog : les programmes Datalog *avec récursion monadique*, puis les programmes Datalog *avec clôture transitive*. En effet, il apparaît que les algorithmes développés pour résoudre le problème de l'équivalence de programmes, dans ces deux cadres particuliers, font en fait appel à des notions similaires.

Ce rapport présente, dans un premier temps, les programmes Datalog de manière générale, puis les instances particulières de programmes Datalog mentionnées ci-dessus. Il se poursuit par une étude sur les liens entre programmes Datalog, arbres finis, et automates finis. Enfin, il présente les algorithmes développés durant le stage, et permettant de résoudre un problème légèrement plus général que le problème d'équivalence entre deux programmes : l'inclusion d'un programme Datalog quelconque dans un programme Datalog *avec récursion monadique*, puis dans un programme Datalog *avec clôture transitive*.

Acknowledgements

It is with a great pleasure that I thank my advisor, Moshe Y. Vardi, for his numerous and invaluable pieces of advice, as well as for having let me discover how thrilling the research can be.

I also want to thank Deian Tabakov, who made me benefit of his own experience of what being a graduate student can be, and who, with his wife Linh, has been particularly nice to me during my stay in a country that I did not know.

In addition, I address my gratitude to the entire Computer Science Department for the warm welcome I received, for the people I met and who became friends of mine.

Finally, I want to thank in particular two people without the help of whom my internship in Rice University would not have been possible: Belia Martinez, who prevented me from being lost in the administrative journey that represented the application for a visa; and Gilles Dowek, who proposed me to work with Moshe Vardi and who supported me when I applied for the internship.

Remerciements

C'est avec un immense plaisir que je remercie mon tuteur, Moshe Y. Vardi, pour ses conseils nombreux et inestimables, ainsi que pour m'avoir fait découvir à quel point la recherche peut être passionnante.

Je tiens également à remercier Deian Tabakov, qui m'a fait bénéficier de sa propre expérience de la vie de *graduate student* et qui, avec sa femme Linh, s'est montré particulièrement sympathique avec moi, dans un pays qui m'était inconnu.

En outre, j'adresse ma gratitude au département d'informatique dans son ensemble, pour l'accueil chaleureux que j'ai reçu.

Enfin, je désire remercier deux personnes sans l'aide desquelles mon stage à Rice University n'aurait pu être possible : Belia Martinez, qui m'a permis de ne pas me perdre dans les méandres administratifs que représentait la demande de visa ; et Gilles Dowek, qui m'a invité à travailler avec Moshe Y. Vardi et qui m'a activement soutenu lors de ma demande de stage.

Contents

Chapter 1

Introduction

1.1 Goal of the Project

In database theory, first-order database query languages are lacking in expressive power. Since then, many higher-order query languages have been investigated, including DATALOG, the language of logic programs without function symbols.

Along with the evaluation of DATALOG programs comes naturally the question of the containment of DATALOG programs. Unfortunately, DATALOG program equivalence is non-decidable. Since the source of the difficulty in evaluating DATALOG programs is their recursive nature, it can be investigated whether equivalence of DATALOG programs with restricted recursion is decidable.

To that extent, a theorem of particular interest is a powerful general decidability result due to COURCELLE [8], which establishes the decidability of some decision problems involving DATALOG programs and monadic second-order queries. Unfortunately, while COURCELLE's result yields the decidability such problem, it provides only non-elementary time-bounds [7].

Therefore, my main goal was to specify some classes of DATALOG program to which COURCELLE's decidability result would apply. Then, my second task was to look for upper and lower bounds of the computational complexity of containment and equivalence, with the help of the automaton-theoretic approach advocated in [15].

1.2 Main Steps of the Project

My internship was divided in three main steps. First of all, my main objective was to discover and understand what a DATALOG program was, as well as the links that bind DATALOG programs to finite tree and finite automata. It is during that period that I managed to build my own intuition on finite trees, on automata, and on DATALOG programs. My two major sources of information were [5] and [9]: in particular, I chose to adopt, for this report, a structure quite similar to the one of [9], which has been the

reference I worked with during several months.

Then, Moshe Y. VARDI presented me the first class of DATALOG programs I would have to work on, the *monadic* DATALOG programs. Since he had already thought about this particular class of programs, what I had to do was mainly to understand the ideas he had had, then to design and algorithm based on these ideas, and that would decide the containment of a program in another one.

After that, he presented me the first class of DATALOG programs I would have to work on, the *transitive* DATALOG programs. This time, whereas it remained easy to see how to use COURCELLE's theorem, it was not clear whether COURCELLE's result gave a good upper bound for the computational complexity of the containment problem. After some unsuccessful attempts, the lecture of some articles [2, 3, 4] related to my problem gave me the ideas that eventually allowed me to design an algorithm solving the containment problem, and whose complexity was much better than the upper bound given by COURCELLE's general result.

1.3 Overview of the Report

This report first describes DATALOG programs, as well as the restricted classes of DATALOG programs on which I worked: the *monadic* DATALOG programs and the *transitive* DATALOG programs. Then, the report stresses the links that bind DATALOG programs, finite trees and finite automata, since automata techniques have been extensively while designing the two algorithms I found. Finally, the report focuses on the algorithms that were designed, as well as the main steps of the proof of the correctness of those algorithms.

However, if the general ideas of the proofs and of the tools introduced will be described in the main body of this report, the proofs used are often quite long and technical. Therefore, the reader eager to verify the proofs themselves can do so by going to Appendix A. Similarly, the use of COURCELLE's theorem, that assured me of the decidability of the problems I looked at, was not helpful at all while designing more efficient algorithms. This is why the part concerning this theorem is postponed in Appendix B.

Chapter 2

Datalog Programs

2.1 Conjunctive Queries

A *conjunctive query* is a positive existential conjunctive first-order formula: the only propositional connective allowed is $\wedge$ and the only quantifier allowed is $\exists$. Without loss of generality, we can assume that conjunctive queries are given as formulæ $\theta(x_1, \ldots, x_k)$ of the form $(\exists y_1, \ldots, y_m)(a_1 \wedge \ldots \wedge a_n)$ with free variables (also called *distinguished variables*) among $x_1, \ldots, x_k$, and where the a_i's are atomic formulæ of the form $p(z_1, \ldots, z_l)$ over the variables $x_1, \ldots, x_k, y_1, \ldots, y_m$.

For instance, $(\forall y)(E(x, y) \wedge E(y, z))$, $(\exists y)(E(x, y) \wedge \neg E(y, z))$ and $(\exists y)(E(x, y) \vee E(y, z))$ are not conjunctive queries, but $(\exists y)(E(x, y) \wedge E(y, z))$ is a conjunctive query. The latter query is satisfied by all pairs $\langle x, z \rangle$ such that there is a path of length 2 between x and z.

We need now to distinguish between variables and *occurrences* of variables in a conjunctive query, but we only consider occurrences of variables in the atomic formulæ of the query. For example, the variables x, y and z respectively have one, two and one occurrence in $(\exists y)(E(x, y) \wedge E(y, z))$.

Eventually, a *union of conjunctive queries* is a disjunction

$$\bigvee_{i=1}^{s} \theta_i(x_1, \ldots, x_k)$$

of conjunctive queries. A union of conjunctive queries $\Theta(x_1, \ldots, x_k)$ can be applied to a database D. The result

$$\Theta(D) = \{(a_1, \ldots, a_k) | D \vDash \Theta(a_1, \ldots, a_k)\}$$

is the set of k-ary tuples that satisfy Θ in D. If Θ has no distinguished variables, then it is viewed as a Boolean query: the result is either the empty relation (corresponding to **false**) or the relation containing the 0-ary tuple (corresponding to **true**).

2.2 Datalog and Containment

A DATALOG program consists of a set of HORN rules. A Horn rule consists of a single atom in the head of the rule and a conjunction of atoms in the body, where an atom is a formula of the form $p(z_1, \ldots, z_l)$ where p is a predicate symbol and $z_1, \ldots, z_l$ are variables. The predicates that occur in head of rules are called *intensional* (IDB) predicates. The other predicates are called *extensional* (EDB) predicates.

For instance, let $\mathbf{P}$ be the following DATALOG program:

$$\begin{array}{ll} \mathbf{odd}(Y) \wedge \mathbf{successor}(Y, X) & \Rightarrow \mathbf{even}(X) \\ \mathbf{even}(Y) \wedge \mathbf{successor}(Y, X) & \Rightarrow \mathbf{odd}(X) \\ \mathbf{zero}(X) & \Rightarrow \mathbf{even}(X) \end{array}$$

In the program $\mathbf{P}$, the IDB predicates are **even** and **odd**; the EDB predicates are **successor** and **zero**.

Let Π be a DATALOG program. Let $Q^i_\Pi(D)$ be the collection of facts about an IDB predicate Q that can be deduced from a database D by at most i applications of the rules in Π. Finally, let $Q^\infty_\Pi(D)$ be the collection of facts about Q that can be deduced from D by any number of applications of the rules in Π, that is,

$$Q^\infty_\Pi(D) = \bigcup_{i \geq 0} Q^i_\Pi(D)$$

Example 2.2.1.
Let us consider the deductive database

$$D = (\{0, 1, 2, 3\}, \{\mathbf{zero}(0), \mathbf{successor}(0, 1), \mathbf{successor}(1, 2), \mathbf{successor}(2, 3)\})$$

By applying the program $\mathbf{P}$ to D, we successively obtain the databases

- $(\{0, 1, 2, 3\}, \{\mathbf{zero}(0), \mathbf{successor}(0, 1), \mathbf{successor}(1, 2), \mathbf{successor}(2, 3), \mathbf{even}(0)\})$.
- $(\{0, 1, 2, 3\}, \{\mathbf{zero}(0), \mathbf{successor}(0, 1), \mathbf{successor}(1, 2), \mathbf{successor}(2, 3), \mathbf{even}(0), \mathbf{odd}(1)\})$.
- $(\{0, 1, 2, 3\}, \{\mathbf{zero}(0), \mathbf{successor}(0, 1), \mathbf{successor}(1, 2), \mathbf{successor}(2, 3), \mathbf{even}(0), \mathbf{odd}(1), \mathbf{even}(2)\})$.
- $(\{0, 1, 2, 3\}, \{\mathbf{zero}(0), \mathbf{successor}(0, 1), \mathbf{successor}(1, 2), \mathbf{successor}(2, 3), \mathbf{even}(0), \mathbf{odd}(1), \mathbf{even}(2), \mathbf{odd}(3)\})$.

Therefore, $\mathbf{even}^\infty_\mathbf{P}(D) = \{(0), (2)\}$. ∎

We say that the program Π with goal predicate Q is *contained* in a union of conjunctive queries Θ if $Q^\infty_\Pi(D) \subseteq \Theta(D)$ for each database D. Similarly, we say that the program Π with goal predicate Q is *contained* in another program $\overline{\Pi}$ with goal predicate $\overline{Q}$ if $Q^\infty_\Pi(D) \subseteq \overline{Q}^\infty_{\overline{\Pi}}(D)$ for each database D. Finally, we say that the program Π with goal predicate Q is *equivalent* to another program $\overline{\Pi}$ with goal predicate $\overline{Q}$ if $Q^\infty_\Pi(D) = \overline{Q}^\infty_{\overline{\Pi}}(D)$ for each database D.

For instance, let $\overline{\mathbf{P}}$ be the following DATALOG program:

$$\begin{array}{ll} \mathbf{integer}(Y) \wedge \mathbf{successor}(Y, X) & \Rightarrow \mathbf{integer}(X) \\ \mathbf{zero}(X) & \Rightarrow \mathbf{integer}(X) \end{array}$$

Then, the program $\mathbf{P}$ with goal predicate **even** is contained in the program $\overline{\mathbf{P}}$ with goal predicate **integer**, but these programs are not equivalent to each other.

2.3 Monadic Datalog Programs

A class of DATALOG programs of particular interest is the class of *monadic* DATALOG programs, defined below.

Definition 2.3.1.
Let Π be a DATALOG program, and $P_1, \ldots, P_k$ its predicates (i.e. the predicates occurring in any rule of Π). Π is a *monadic* DATALOG program if there exists a total mapping $\varphi : \{P_1, \ldots, P_k\} \to \mathbb{N}$ such that, for every rule $\mathcal{R} : I_1(\mathbf{x}^1) \wedge \ldots \wedge I_n(\mathbf{x}^n) \Rightarrow I_0(\mathbf{x}^0)$ of Π, then $\varphi(I_0) \geq \max\{\varphi(I_1), \ldots, \varphi(I_n)\}$, with strict inequality if I_0 is not a monadic predicate (i.e. a predicate of arity 1). ∎

Intuitively, in such a program, only a monadic IDB predicate can depend recursively of itself when being defined. That is why this program is said to be *monadic*.

2.4 Transitive Datalog Programs

Another class of particular interest is the class of *transitive* DATALOG programs, defined below.

Definition 2.4.1.
Let Π be a DATALOG program. Let us suppose that we can split its predicates $P_1, \ldots, P_k$ in two sorts: we call the predicates of the first sort *star predicates* (these predicates will often be denoted with asterisks), and we call the predicates of the second sort *star-free predicates*. Then, for each star predicate P^*, P^* must be binary and Π also contains a binary star-free predicate P. Furthermore, Π contains two rules $\top \Rightarrow P^*(X, X)$ and $P(X, Y) \wedge P^*(Y, Z) \Rightarrow P^*(X, Y)$; $P^a st$ may not appear in the head of any other rule of Π. Finally, we suppose that there is exists a total mapping $\varphi : \{P_1, \ldots, P_k\} \to \mathbb{N}$ such that, for every rule $\mathcal{R} : I_1(\mathbf{x}^1) \wedge \ldots \wedge I_n(\mathbf{x}^n) \Rightarrow I_0(\mathbf{x}^0)$ of Π, then $\varphi(I_0) \geq \max\{\varphi(I_1), \ldots, \varphi(I_n)\}$, with strict inequality if $\mathcal{R}$ is not a star rule $P(X, Y) \wedge P^*(Y, Z) \Rightarrow P^*(X, Y)$.

If a DATALOG program Π verifies all these properties, Π is said to be *transitive*.

Then, we call *baby-star rule* every rule $\top \Rightarrow P^*(X, X)$, *star rule* every rule $P(X, Y) \wedge P^*(Y, Z) \Rightarrow P^*(X, Y)$, and *star-free rule* every other rule (i.e. every rule whose head predicate is a star-free predicate). ∎

Intuitively, such a program features a logical formula implementing finite disjunctions and conjunctions of queries, existential and universal quantification, and recursion through *transitive closure* of binary predicates. That is why this program is said to be *transitive*.

2.5 Alternative Definitions

For more convenience, we will use slightly different definitions of the two classes of programs introduced above. If these definitions are equivalent to those already given, they are the definitions that will be used later in the report. In order to introduce these definitions, we define a well-ordering on the set $\mathbb{N}[X]$ of the polynomials with non-negative integer coefficients.

Proposition 2.5.1.
We denote by $\mathbb{N}[X]$ the set of polynomials with non-negative integer coefficients. Let $\leq$ be the ordering on $\mathbb{N}[X]$ defined as $P \leq Q$ if and only if $Q - P$ has a leading coefficient in $\mathbb{N}$. $\leq$ is a well-ordering. ∎

Proof. See Appendix A, Proof A.1.1 □

This proposition allows us to use a very efficient technique along the entire report: if S is a set and $\varphi : S \to \mathbb{N}[X]$ a total mapping, we use rewriting rules $r_1, \ldots, r_k$ over S such that, $\forall s \in S, \varphi(r_i(s)) < \varphi(s)$. Therefore, the process of rewriting must be finite.

It is such a technique that lets us give the following characterisation of monadic Datalog programs:

Proposition 2.5.2.
Let Π be a monadic Datalog *program. Π is equivalent to a monadic* Datalog *program Π' such that:*

- *the goal predicate of Π' does not appear in the body of any rule of Π'.*
- *every IDB predicate of Π' which is not the goal predicate is monadic.*

∎

Proof. See Appendix A, Proof A.1.2 □

Definition 2.5.3. In such a program, we call *internal IDB predicates* the IDB predicates that are not the goal predicate, *goal rules* the rules whose head predicate is the goal predicate, and *internal rules* the rules whose head predicate is an internal predicate. ∎

Likewise, we obtain the following characterisation of Datalog programs with transitive closure:

Proposition 2.5.4.
Let Π *be a transitive* DATALOG *program.* Π *is equivalent to a transitive* DATALOG *program* Π' *such that:*

- *the goal predicate of* Π' *does not appear in the body of any rule of* Π'*.*
- *every IDB predicate of* Π' *that appears in the body of some star-free rule of* Π' *must be a star predicate.*
- *every IDB predicate of* Π' *which is not the goal predicate is of arity* 2*.*

■

Proof. See Appendix A, Proof A.1.3 □

Chapter 3

Datalog, Finite Trees and Automata

The main part of this chapter is directly inspired from the article [9].

3.1 Containment of Conjunctive Queries

The notion of containment of conjunctive queries and of DATALOG programs are very similar. Therefore, it is useful to look first at theorems regarding the containment of conjunctive queries.

In particular, a very useful characterisation of the containment of conjunctive queries is related to the notion of *containment mapping* defined below:

Definition 3.1.1.
A *containment mapping* from a conjunctive query ψ to a conjunctive query θ is a renaming of variables subject to the following constraints:

- every distinguished variable must map to itself, and
- after renaming, every literal in ψ must be among the literals of θ.

■

Conjunctive-query containment can then be characterized in terms of containment mappings (cf. [14]).

Theorem 3.1.2.
A conjunctive query $\theta(x_1, \ldots, x_k)$ is contained in a conjunctive query $\psi(x_1, \ldots, x_k)$ if and only if there is a containment mapping from ψ to θ. ■

It may be convenient to view a containment mapping h from ψ to θ as a mapping from occurrences of variables in ψ to occurrences to variables in θ. Such a mapping has the property that v_1 and v_2 are occurrences of the same variable in ψ, then $h(\mathsf{v}_1)$ and $h(\mathsf{v}_2)$ are occurrences of the same variable in θ.

SAGIV and YANNAKAKIS [11] extended Theorem 3.1.2 to the case where queries are unions of conjunctive queries.

Theorem 3.1.3.
If $\Phi = \cup_i \phi_i$ and $\Psi = \cup_i \psi_i$ are (finite) unions of conjunctive queries, then Φ is contained in Ψ (i.e., $\Phi(D) \subseteq \Psi(D)$ for every database D) if and only if each ϕ_i is contained in some ψ_j , i.e., there is a containment mapping from ψ_j to ϕ_i. ∎

3.2 Expansion Trees

Expansions can be described in terms of *expansion trees*. The nodes of an expansion tree for a DATALOG program Π are labelled by pairs of the form (α, ρ), where α is an IDB atom and ρ is an instance of a rule r of Π such that the head of ρ is α. The atom labelling a node x is denoted α_x and the rule labelling a node x is denoted ρ_x. In an expansion tree for an IDB predicate Q, the root is labelled by a Q-atom. Consider a node x, where α_x is the atom $R(\mathbf{t})$, ρ_x is the rule

$$R_1(\mathbf{t}^1) \wedge \ldots \wedge R_m(\mathbf{t}^m) \Rightarrow R(\mathbf{t})$$

and the IDB atoms in the body of the rule are $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_l}(\mathbf{t}^{i_l})$. Then x has children $x1, \ldots, xl$ labelled with the atoms $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_l}(\mathbf{t}^{i_l})$. In particular, if all atoms in ρ_x are EDB atoms, then x must be a leaf.

The query corresponding to an expansion tree is the conjunction of all EDB atoms in ρ_x for all nodes x in the tree; the free variables of this query are the variables appearing in the head atom of the root of the tree, and the other variables are bound to an existential quantifier. Thus, we can view an expansion tree τ as a conjunctive query. Let $trees(Q, \Pi)$ denote the set of expansion trees for an IDB predicate Q in Π. (Note that $trees(Q, \Pi)$ is an infinite set.) Then for every database D, we have

$$Q^\infty_\Pi(D) = \bigcup_{\tau \in trees(Q,\Pi)} \tau(D)$$

It follows that Π is contained in a conjunctive query θ if there is a containment mapping from θ to each expansion tree τ in $trees(Q, \Pi)$, i.e. a mapping which maps distinguished variables to distinguished variables and the atoms of θ to atoms in the bodies of rules labelling nodes of τ.

The expansion trees that are obtained by "unfolding" the program Π are of particular interest.

Definition 3.2.1.
An expansion tree τ of a DATALOG program Π is an *unfolding expansion tree* if it satisfies the following conditions:

- the atom labelling the root is the head of a rule in Π.
- if a node x is labelled by (α_x, ρ_x), then the variables in the body of ρ_x either occur in α_x or do not occur in the label of any node above x.

∎

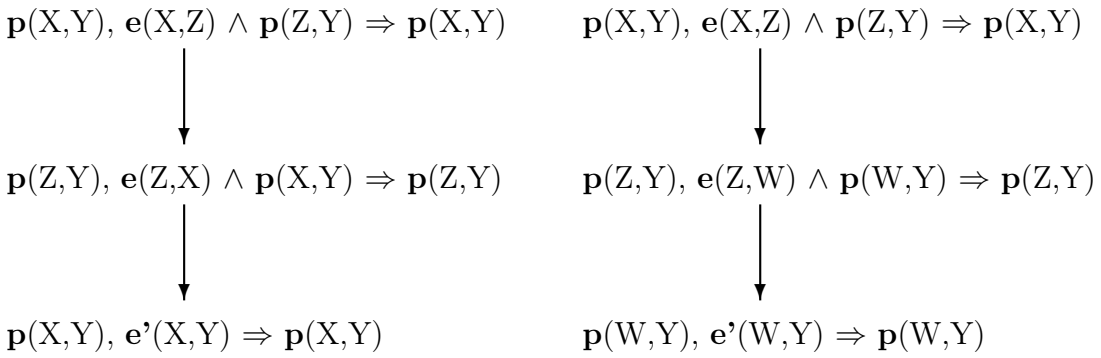
Intuitively, an unfolding expansion tree is obtained by starting with a head of a rule in Π as the atom labelling the root, and then creating children by unifying an atom labelling a node with a "fresh" copy of a rule in Π. Note that if a variable v occur in the atom labelling a node x but not in the atoms labelling the children of x, then v will not occur in the label of any descendant of x.

We denote the collection of unfolding expansion trees for an IDB predicate Q in a program Π by $u\_trees(Q, \Pi)$. It is easy to see that every expansion tree can be obtained by renaming variables in an unfolding expansion tree. Thus, every expansion tree, viewed as a conjunctive query, is contained in an unfolding expansion tree.

Example 3.2.2.
Figure 3.1 shows expansion trees for the IDB predicate p in the following transitive closure program.

$$\begin{array}{lcl} e(X,Z) \wedge p(Z,Y) & \Rightarrow & p(X,Y) \\ e'(X,Y) & \Rightarrow & p(X,Y) \end{array}$$

Note that the variable X is re-used in the child of the root of the expansion tree, while a new variable W is used instead of X in the child of the root of the unfolding expansion tree.



(a) (b)

Figure 3.1: (a) Expansion Tree (b) Unfolding Expansion Tree

■

The following proposition follows immediately.

Proposition 3.2.3.
Let Π be a program with a goal predicate Q. For every database D, we have

$$Q_\Pi^\infty(D) = \bigcup_{\tau \in u\_trees(Q,\Pi)} \tau(D)$$

■

3.3 Some Notations

Having introduced the concept of expansion trees, we will now introduce some notations that will be very useful below in this report. Indeed, since a tree can be seen as a set of nodes, and since we can also consider sets of predicates or of variables, the sooner we find a convenient notation for these objects, the better. Therefore, here are some notations that will be found throughout the entire report:

Definition 3.3.1.
Let Π be a DATALOG program. We denote by $program\_var(\Pi)$ twice the maximum number of variables occurring in any rule of Π, and by $var(\Pi)$ the set of variables $\{x_{\Pi,1}, \ldots, x_{\Pi,program\_var(\Pi)}\}$.

Let γ be a DATALOG program with goal predicate Γ such that $u\_trees(\Gamma, \gamma)$ be finite. We denote by $tree\_var(\gamma)$ twice the maximum number of variables occurring in any tree $\sigma \in u\_trees(\Gamma, \gamma)$, and by $var_\diamond(\gamma)$ the set of variables $\{x'_{\gamma,1}, \ldots, x'_{\gamma,tree\_var(\gamma)}\}$.

Let $\mathcal{A}$ be set of instances of atoms. We denote by $var(\mathcal{A})$ the set of variables occurring in any atom of $\mathcal{A}$.

Let $\mathbf{v}$ be tuple of variables. We denote by $var(\mathbf{v})$ the set of variables occurring in $\mathbf{v}$.

Let $\mathcal{R}$ be an instance of a rule of a DATALOG program Π. We denote by $var(\mathcal{R})$ the set of variables occurring in $\mathcal{R}$.

Let σ be an expansion tree of a DATALOG program Π. We denote by $var(\sigma)$ the set of variables occurring in σ.

Let σ be a tree. We denote by $\mathcal{N}_\sigma$ the set of nodes in σ.

Let σ be a tree, and $n \in \mathcal{N}_\sigma$ a node in σ. We denote by $\sigma_{(n)}$ the sub-tree of σ whose root is n. ∎

3.4 Proof Trees

If unfolding expansion trees are at the basis of the study of DATALOG programs, they have the disadvantage of involving an unbounded number of variables. Therefore, we introduce now the notion of *proof tree*: the basic idea behind proof trees is to describe expansion trees using a finite number of labels. We bound the number of labels by bounding the set of variables that can occur in labels of nodes in the tree, taking advantage of the notations introduced in the precedent section.

Definition 3.4.1.
Let Π be a DATALOG program. A *proof tree* for Π is an expansion tree for Π all of whose variables are from $var(\Pi)$. We denote the set of proof trees for a predicate Q of a program Π by $p\_trees(Q, \Pi)$. ∎

The intuition behind proof tree is that variables are re-used. In an unfolding expansion tree, when we "unfold" a node x we take a "fresh" copy of a rule r in Π. In a proof tree, we take instead an instance of r over $var(\Pi)$. Since the number of variables in $var(\Pi)$ is twice the number of variables in any rule of Π, we can instantiate the variables in the body of r by variables different from those in the goal α_x.

Example 3.4.2.
Figure 3.2 describes an unfolding expansion tree and a proof tree for the IDB predicate p in the transitive-closure program of Example 3.2.2.

Note that, in the proof tree, instead of using a new variable W, we re-use the variable X.

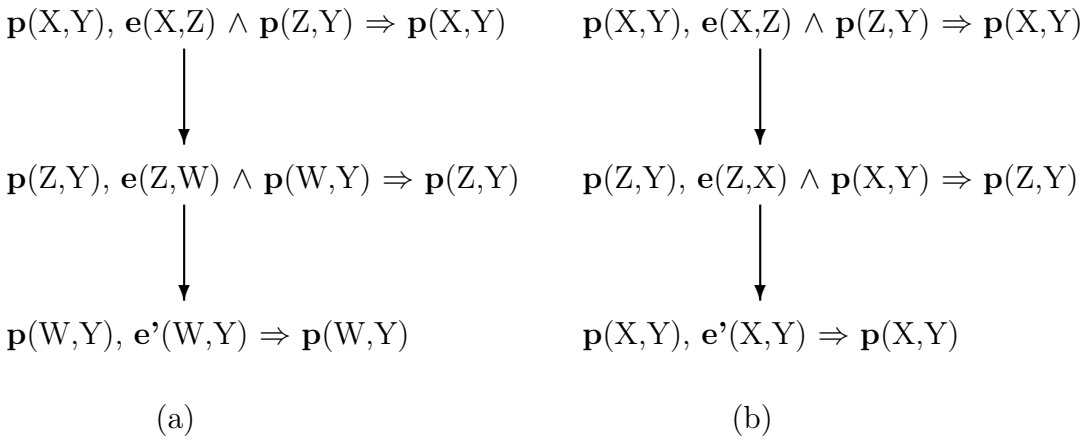


Figure 3.2: (a) Unfolding Expansion Tree (b) Proof Tree

■

A proof tree represents an expansion tree where variables are re-used. In other words, the same variable is used to represent a set of distinct variables in the expansion tree. Intuitively, to reconstruct an expansion tree for a given proof tree, we need to distinguish among occurrences of variables.

Definition 3.4.3.
Let n_1 and n_2 be nodes in a proof tree Σ, with a lowest common ancestor n, and let $\mathbf{v}_1$ and $\mathbf{v}_2$ be occurrences, in n_1 and n_2, respectively, of a variable v. $\mathbf{v}_1$ and $\mathbf{v}_2$ are said to be *connected* in Σ if the head of every node, except perhaps for n, on the simple path connecting n_1 and n_2, contains an occurrence of v. Connectedness is an equivalence relation and it partitions the occurrences of variables in the proof tree. We denote the equivalence class of an occurrence $\mathbf{v}$ of a variable v in a proof tree Σ by $[\mathbf{v}]_\Sigma$, and the set of such equivalence classes by $[\Sigma]$.

Similarly, if $(\mathbf{v}_1, \ldots, \mathbf{v}_k)$ is a tuple of occurrences of variables, the tuple $([\mathbf{v}_1]_\Sigma, \ldots, [\mathbf{v}_1]_\Sigma)$ is denoted by $[(\mathbf{v}_1, \ldots, \mathbf{v}_k)]_\Sigma$. And if ρ is an instance of a rule in Σ, we denote by $[var(\rho)]$ the set $\{[\mathbf{v}] | \rho$ contains the occurrence $\mathbf{v}$ of a variable $v\}$. ■

There is a canonical identification from proof trees to unfolding trees, which states the straightforward but essential following proposition:

Proposition 3.4.4.
Let Π *be a* DATALOG *program with goal predicate* Q*. For all* Σ *in* $p\_trees(Q, \Pi)$*, there exists an unique tree (up to renaming of its variables)* $\nu \in u\_trees(Q, \Pi)$ *such that some bijective mappings* $h : \mathcal{N}_\nu \to \mathcal{N}_\Sigma$ *and* $\theta : var(\nu) \to [\Sigma]$ *(which we can extend to a mapping* $\theta : var(\nu) \to var(\Pi)$ *in a natural way) have the following property:*

- $\forall n, n' \in \mathcal{N}_\nu$*,* $h(n)$ *is a child of* $h(n')$ *in* Σ *if and only if* n *is a child of* n' *in* ν*.*
- $\forall n \in \mathcal{N}_\nu$*, if* $n = (R(\boldsymbol{t}), \rho)$*, then* $h(n) = (\theta(R(\boldsymbol{t})), \theta(\rho))$*.*

We denote by $\mathcal{U}(\Sigma)$ *this tree* $\nu \in u\_trees(Q, \Pi)$*, and call* unfolding mapping *(resp.* unfolding node mapping*) of* Σ *the mapping* θ *(resp.* h*).* ∎

3.5 Automata on Trees

Let $\mathbb{N}_+$ denote the set of positive integers. The variables x and y denote elements of $\mathbb{N}_+^*$ (the set of finite sequences of positive integers). A *tree* τ is a finite subset of $\mathbb{N}_+^*$, such that if $xi \in \tau$, where $x \in \mathbb{N}_+^*$ and $i \in \mathbb{N}_+$, then also $x \in \tau$ and, if $i > 1$, then also $x(i-1) \in \tau$. The elements of τ are called *nodes*. If x and xi are nodes of τ, then x is the *parent* of xi and xi is the *child* of x. The node x is a *leaf* if it has no children. By definition, the empty sequence ε is a member of every tree; it is called the *root*.

A Σ*-labelled tree*, for a finite alphabet Σ, is a pair (τ, π), where τ is a tree and $\pi : \tau \to \Sigma$ assigns a label to every node. *Labelled trees* are often referred to as *trees*; the intention will be clear from the context. The set of Σ-labelled trees is denoted $trees(\Sigma)$.

A *tree automaton* A is a tuple $(\Sigma, S, S_0, \delta, F)$, where Σ is a finite alphabet, S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $F \subseteq S$ is a set of accepting states, and $\delta : S \times \Sigma \to 2^{S^*}$ is a transition function such that $\delta(s, a)$ is finite for all $s \in S$ and $a \in \Sigma$. A *run* $r : \tau \to S$ of A on a Σ-labelled tree (τ, π) is a labelling of τ by states of A, such that the root is labelled by an initial state and the transitions obey the transition function δ; that is, $r(\epsilon) \in S_0$, and if x is not a leaf and x has k children, then $\langle r(x1), \ldots, r(xk)\rangle \in \delta(r(x), \pi(x))$. If for every leaf x of τ there is a tuple $\langle s_1, \ldots, s_l\rangle \in \delta(r(x), \pi(x))$ such that $\{s_1, \ldots, s_l\} \subseteq F$, then r is *accepting*. A *accepts* (τ, π) if it has an accepting run on (τ, π). The *tree language* of A, denoted $T(A)$, is the set of trees accepted by A.

An important property of tree automata is their closure under Boolean operations.

Proposition 3.5.1. [6]
Let A_1, A_2 *be automata over an alphabet* Σ*. Then there are automata* A_3*,* A_4*, and* A_5 *such that* $L(A_3) = \Sigma^* - L(A_1)$*,* $L(A_4) = L(A_1) \cap L(A_2)$*, and* $L(A_5) = L(A_1) \cup L(A_2)$*.* ∎

As in word automata, the constructions for union and intersection involve only a polynomial blow-up in the size of the automata, while complementation may involve an exponential blow-up in the size of the automaton.

The *non-emptiness problem* for tree automata is to decide, given a tree automaton A, whether $T(A)$ is non-empty.

Proposition 3.5.2. [10, 13]
The non-emptiness problem for tree automata is decidable in polynomial time. ∎

Proof. See Appendix A, Proof A.2.1 □

We note that using techniques such as in [1], the non-emptiness problem for tree automata is decidable in linear time.

Another problem related to non-emptiness is the *containment problem*, which is to decide, given tree automata A_1 and A_2, whether $T(A_1) \subseteq T(A_2)$. As for word automata, the containment problem is reducible to the non-emptiness problem, though the reduction may be computationally expensive.

Proposition 3.5.3. [11]
The containment problem for tree automata is EXPTIME-complete. ∎

Chapter 4

Monadic Programs

4.1 Decorated Unfolding Trees

Unfolding expansion trees allow us to register sequences of rules that can be applied in order to answer positively to the goal predicate of a DATALOG program. In addition to that, we also may have to store information on the variables appearing during the process of answering this query.

Intuitively, the entire information that we can obtain is, for any variable appearing in an unfolding tree, the set of monadic predicates that hold on this variable. Therefore, we have to store this information with the tree itself, which is done by adding a *decoration* to the tree: we also indicate which monadic predicates hold on any variable appearing in the tree, then verify whether this indication is compatible with a given monadic DATALOG rule, or with a set of such rules.

To that extent, we introduce here the concept of decorated unfolding trees.

Definition 4.1.1.
A *decorated unfolding tree* of a DATALOG program Π with goal predicate Q is a triple $(\tau, \mathcal{L}, \varphi)$ where $\tau \in u\_trees(Q, \Pi)$, $\mathcal{L}$ is a finite set, $var(\tau)$ is the set of the variables appearing in τ, and $\varphi \in 2^{var(\tau)\times\mathcal{L}}$ can be seen as a total mapping from the variables in τ to the subsets of $\mathcal{L}$.

We denote by $u\_dec(Q, \Pi)$ the collection of decorated unfolding trees for program Π with goal predicate Q. Moreover, if $\mathcal{S}$ is a finite set, we denote by $u\_dec(\mathcal{S}, Q, \Pi)$ the set of trees $(\tau, \mathcal{L}, \varphi) \in u\_dec(Q, \Pi)$ such that $\mathcal{L} = \mathcal{S}$. ∎

DATALOG is a fragment of fixpoint logic. Therefore, it is natural, while dealing with DATALOG, to look for the existence of fixpoints under some applications. Intrinsically, the importance of the notion of fixpoint is highly related to monadic second-order (MSO) logic. And this notion allows us to specify whether a decorated unfolding tree gives information coherent with a monadic DATALOG rule or program. That we define now the notion of *fixpoint decorated unfolding tree*.

Definition 4.1.2.
Let $\mathcal{R}$ be a rule

$$R_1(\mathbf{t}^1) \wedge \ldots \wedge R_m(\mathbf{t}^m) \Rightarrow R(\mathbf{t})$$

where the IDB atom $R(\mathbf{t})$ in the head and the IDB atoms $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_l}(\mathbf{t}^{i_l})$ in the body are monadic, and where the EDB atoms in the body are $R_{j_1}(\mathbf{t}^{i_1}), \ldots, R_{j_{l'}}(\mathbf{t}^{j_{l'}})$. A decorated unfolding tree $\tau^{dec} = (\tau, \mathcal{L}, \varphi)$ is said to be a *fixpoint decorated unfolding tree* with respect to $\mathcal{R}$ if and only if the following holds:

- $\{R, R_{i_1}, \ldots, R_{i_l}\} \subseteq \mathcal{L}$.
- for each total mapping $h : var(\mathcal{R}) \rightarrow var(\tau)$ from the variables of $\mathcal{R}$ to the variables of τ, if $R_{j_1}(h(\mathbf{t}^{j_1})), \ldots, R_{j_{l'}}(h(\mathbf{t}^{j_{l'}}))$ hold and $\{(h(\mathbf{t}^{i_1}), R_{i_1}), \ldots, (h(\mathbf{t}^{i_l}), R_{i_l})\} \subseteq \varphi$, then $(h(\mathbf{t}), R) \in \varphi$.

If Π is a DATALOG program with goal predicate Q, we denote by $u\_dec^{fp}(\mathcal{R}, Q, \Pi)$ the set of fixpoint decorated unfolding trees τ of Π with respect to $\mathcal{R}$. And, if $\mathcal{S}$ is a finite set, we denote by $u\_dec^{fp}(\mathcal{S}, \mathcal{R}, Q, \Pi)$ the set of fixpoint decorated unfolding trees τ of Π with respect to $\mathcal{R}$ and such that $\mathcal{L}_\tau = \mathcal{S}$. ∎

A DATALOG program is a finite conjunction of Horn rules; therefore, after having defined the concept of fixpoint decorated unfolding tree with respect to a particular Horn rule, a natural generalisation is to define the concept of fixpoint decorated unfolding tree with respect to a DATALOG program.

Definition 4.1.3.
Let $\Pi_\mathcal{M}$ be a monadic DATALOG program with goal predicate $Q_\mathcal{M}$ and rules $\mathcal{R}_1, \ldots, \mathcal{R}_n$. Let $\mathcal{L}_{IDB}$ be the set of internal IDB predicates of $\Pi_\mathcal{M}$. A *fixpoint decorated unfolding tree* with respect to $\Pi_\mathcal{M}$ is a tree $\tau^{dec} = (\tau, \mathcal{L}, \varphi)$ such that $\mathcal{L} = \mathcal{L}_{IDB}$ and which is a fixpoint decorated expansion tree for every internal rule $\mathcal{R}_i$ in $\Pi_\mathcal{M}$.

If Π is a DATALOG program with goal predicate Q, we denote by $u\_dec^{fp}(\Pi_\mathcal{M}, Q, \Pi)$ the set of fixpoint decorated unfolding trees τ of Π with respect to $\Pi_\mathcal{M}$; ∎

Intuitively, in such a decorated unfolding tree, if some monadic predicate $R(X)$ holds for a variable X, then it follows that $(X, R) \in \varphi$. Moreover, it is important to see that we can always build fixpoint decorated unfolding trees, as states the immediate, following lemma:

Lemma 4.1.4.
Let be Π *a* DATALOG *program with goal predicate* Q *and* $\Pi_\mathcal{M}$ *a monadic* DATALOG *program.* $\forall \tau \in u\_trees(Q, \Pi), \exists \tau^{dec} = (\tau, \mathcal{L}, \varphi) \in u\_dec^{fp}(\Pi_\mathcal{M}, Q, \Pi)$. ∎

A very convenient way of proving properties over families of trees is to use proofs by induction. However, using such proofs may need to extract sub-trees from a given tree. And, while working on expansion trees, the leaves of a tree cannot be labelled with rules whose body contains any IBD predicate. Therefore, it is necessary to introduce "fresh" predicates, each one seen as formally equivalent to an IBD predicate. This is done while introducing the concept of *extensional closure* defined below:

Definition 4.1.5.
Let Π be a DATALOG program with IDB predicates $R_1, \ldots, R_n$. The *extensional closure* of Π, denoted $\Pi^\bullet$, is the DATALOG program obtained by adding "fresh" EDB predicates $R_1^\bullet, \ldots, R_n^\bullet$, and by setting new rules:

- if $R_1(\mathbf{v}^1) \wedge \ldots \wedge R_n(\mathbf{v}^n) \Rightarrow R_0(\mathbf{v}^0)$ is a rule $\mathcal{R}_i$ of Π, and $R_{i_1}, \ldots, R_{i_l}$ are some IDB predicates in the body of $\mathcal{R}_i$, then $R'_1(\mathbf{v}^1) \wedge \ldots \wedge R'_n(\mathbf{v}^n) \Rightarrow R_0(\mathbf{v}^0)$ is a rule of $\Pi^\bullet$, where $R'_k = R_k^\bullet$ if $k \in \{i_1, \ldots, i_l\}$, and $R'_k = R_k$ if $k \notin \{i_1, \ldots, i_l\}$; this new rule is said to be *derived* from the rule $\mathcal{R}_i$.
- if R is an IDB predicate of Π, then $R^\bullet(\mathbf{v}) \Rightarrow R(\mathbf{v})$ is a rule of $\Pi^\bullet$; the new predicate $R^\bullet$ is said to be *derived* from the IDB predicate R.

■

It is immediate that, if every EDB predicate of a DATALOG program Π is an EDB predicate of the monadic DATALOG program $\Pi_\mathcal{M}$, then Π is contained in $\Pi_\mathcal{M}$ if and only if is it contained in $\Pi_\mathcal{M}^\bullet$.

Now, as well as we have allowed the construction of unfolding expansion trees featuring sub-trees of a tree, it is necessary to find a way to extend the notion of containment mapping to these trees, preferably in a natural fashion. This is precisely what is done below:

Definition 4.1.6.
Let Π be DATALOG program with goal predicate Q, $\Pi_\mathcal{M}$ a monadic DATALOG program with goal predicate $Q_\mathcal{M}$, and $\Pi_\mathcal{M}^\bullet$ its extensional closure. Let be $\tau^{dec} = (\tau, \mathcal{L}, \varphi) \in u\_dec(Q, \Pi)$ and $\sigma^\bullet \in u\_trees(Q_\mathcal{M}, \Pi_\mathcal{M}^\bullet)$. A *decorating containment mapping* from $\sigma^\bullet$ to τ^{dec} is a total mapping $h : var(\sigma^\bullet) \to var(\tau)$ such that the following holds:

- h maps every distinguished variable in $\sigma^\bullet$ to itself.
- for every EDB atom $R^\bullet(\mathbf{v})$ derived from an IDB atom $R(\mathbf{v})$, $(h(\mathbf{v}), R) \in \varphi$.
- for every EDB atom $R(\mathbf{v})$ not treated above, $R(h(\mathbf{v}))$ is an EDB atom in τ.

■

Finally, it appears that the objects defined above respect the intuition I gave about capturing all the necessary pieces of information, and verifying that these pieces of information are coherent with a given set of rules. Indeed, here is the first main result that I obtained during the project, and that closely relates containment of a DATALOG program in a monadic DATALOG program to the existence of *decorating containment mappings*.

Theorem 4.1.7.
Let Π be a DATALOG *program with goal predicate Q, $\Pi_\mathcal{M}$ a monadic* DATALOG *program with goal predicate $Q_\mathcal{M}$. Π is contained in $\Pi_\mathcal{M}$ if and only if for all $\tau^{dec} = (\tau, \mathcal{L}, \varphi)$ in $u\_dec^{pf}(\Pi_\mathcal{M}, Q, \Pi)$, there exists a tree $\sigma^\bullet$ in $u\_trees(Q_\mathcal{M}, \Pi_\mathcal{M}^\bullet)$ whose root is a leaf, and a decorating containment mapping $\sigma^\bullet \xrightarrow{h^\bullet} \tau^{dec}$.* ■

Proof. See Appendix A, Section A.3.1. □

4.2 Decorated Proof Trees

In the precedent chapter, we decided to transform unfolding expansion trees into proof trees, in order to bound the number of variables used and to be able to work on a finite vocabulary. In the same way, we will now introduce *decorated proof trees*, that consist in a reduction of decorated expansion trees to trees with a number of variables bounded *a priori*.

However, the reduction from unfolding expansion trees to proof trees is only *locally* injective. This is why several "former" distinct variables whose image through φ were different may now appear with the same name. Therefore, all the information that was contained globally in the sets φ has now to be stored in each node of our new trees. Or, at least, since we cannot bound the quantity of information contained in φ by just knowing the two DATALOG programs that we are looking at (because a given DATALOG program may have expansion trees involving an arbitrary high number of variables), we want to store in each node of our trees the information that was contained in φ and that concerned the variables appearing in that node.

Definition 4.2.1.
A *pre-decorated proof tree* for a DATALOG program Π with goal predicate Q is a triple $(\tau, \mathcal{L}, \varphi)$ such that

- p is a proof tree: $\tau \in p\_trees(Q, \Pi)$.
- $\exists \varphi'$ such that $(\mathcal{U}(\tau), \mathcal{L}, \varphi') \in u\_dec(Q, \Pi)$.
- $\varphi \in 2^{\mathcal{N}_\tau \times var(\Pi) \times \mathcal{L}}$ verifies:
  - $\forall n \in \mathcal{N}_{\mathcal{U}(\tau)}, \forall v \in var(n), \forall l \in \mathcal{L}, v, l) \in \varphi' \Rightarrow (h(n), \theta(v), l) \in \varphi$.
  - $\forall n \in \mathcal{N}_\tau, \forall v \in var(\Pi), \forall l \in \mathcal{L}, v \notin var(n) \Rightarrow (n, v, l) \notin \varphi$.

■

Formally, and since it would be nice to have directly a structure of trees that can be treated by tree automata, we can identify pre-decorated proof trees in proof trees, which is done as follows:

Definition 4.2.2.
A *decorated proof tree* for a DATALOG program Π with goal predicate Q is a tree σ whose nodes n are quadruples $(\alpha_n, \rho_n, \mathcal{L}_n, \varphi_n)$, for which there exists a pre-decorated proof tree $(\tau, \mathcal{L}, \varphi)$ for Π and a bijection $i : \mathcal{N}_\sigma \to \mathcal{N}_\tau$ such that:

- $\forall n_1, n_2 \in \mathcal{N}_\sigma$, n_1 is the father of n_2 if and only if $i(n_1)$ is the father of $i(n_2)$.
- $\forall n \in \mathcal{N}_\sigma, \mathcal{L}_n = \mathcal{L}$ and $\varphi_n = \{(v, l) \in var(\Pi) \times \mathcal{L} | (n, v, l) \in \varphi\}$.

■

For more convenience, we will, whenever τ is a decorated proof tree, denote by $\mathcal{U}(\tau)$ the unfolding tree which is, in fact, the tree $\mathcal{U}(p)$ where p is the first component of the pre-decorated proof tree associated to τ. This mapping $\mathcal{U}$ stresses the close relationship that exists between decorated proof trees and decorated unfolding trees, this relationship being so close that it induces the concept of *fixpoint decorated proof trees*.

Definition 4.2.3.
Let Π be a DATALOG program with goal predicate Q, and $\Pi_{\mathcal{M}}$ a monadic DATALOG program whose set of internal IDB predicates is $\mathcal{L}$. A *fixpoint decorated proof tree* of Π with respect to $\Pi_{\mathcal{M}}$ is a tree $\tau \in p\_dec(\mathcal{L}, Q, \Pi)$ such that $\mathcal{U}(\tau) \in u\_dec^{fp}(\Pi_{\mathcal{M}}, Q, \Pi)$. We denote by $p\_dec^{fp}(\Pi_{\mathcal{M}}, Q, \Pi)$ the set of fixpoint decorated proof trees of Π with respect to $\Pi_{\mathcal{M}}$. ∎

This set of trees being defined, we now need a way to check whether some random tree $\tau \in p\_dec^{fp}(\Pi_{\mathcal{M}}, Q, \Pi)$ is fixpoint relatively to a monadic program. And, as well as a decorated unfolding tree is fixpoint with respect to a monadic program if and only if it is fixpoint with respect to each of its internal rules, we look now at a way of checking whether a decorated proof tree is fixpoint with respect to a monadic rule.

But, since it is often algorithmically easier to verify that a given set *is not* a fixpoint under some operations (since we only have to find an element that is sent outside the considered set) than to verify that this set *is* a fixpoint, we prefer look at *infixpoint certificates*, that certify that a given decorated proof tree *is not* a fixpoint decorated proof tree.

Definition 4.2.4.
Let τ be some decorated proof tree, and

$$\mathcal{R} = R_1(\mathbf{v}^1) \wedge \ldots \wedge R_m(\mathbf{v}^m) \Rightarrow R(\mathbf{v})$$

be a rule with monadic IDB atoms $R(\mathbf{v}), R_{i_1}(\mathbf{v}^{i_1}), \ldots, R_{i_l}(\mathbf{v}^{i_l})$ where $\{R, R_{i_1}, \ldots, R_{i_l}\} \subseteq \mathcal{L}_\tau$, and EDB atoms $R_{j_1}(\mathbf{v}^{j_1}), \ldots, R_{j'_l}(\mathbf{v}^{j'_l})$. An *infixpoint certificate* from $\mathcal{R}$ to τ is a mapping h from the occurrences of variables $v \in var(\mathcal{R})$ to occurrences of variables $v \in var(\tau)$ such that

- if v_1 and v_2 are two occurrences of a variable $v \in var(\mathcal{R})$, then the occurrences $h(\mathsf{v}_1), h(\mathsf{v}_2) \in var(\tau)$ are connected.
- if an EDB predicate $R_{i_l}(\mathbf{v}^{i_l})$ holds in $\mathcal{R}$, then $R_{i_l}(h(\mathbf{v}^{i_l}))$ holds in τ.
- if an IDB predicate $R_{j_l}(\mathbf{v}^{j_l})$ holds in the body of $\mathcal{R}$, then $(h(v^{j_l}), R_{j_l}) \in \varphi_n$, where n is the node of τ containing the occurrence $h(\mathbf{v}^{j_l})$.
- if n is the node of τ^{dec} containing the occurrence $h(\mathbf{v})$, then $(h(\mathbf{v}), R) \notin \varphi_n$.

If Π is a DATALOG program with goal predicate Q, we denote by $p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ the set of trees $\tau \in p\_dec(\mathcal{L}, Q, \Pi)$ such that an infixpoint certificate from $\mathcal{R}$ to τ exists. ∎

Indeed, this definition satisfies our will to discover certificates proving that a given decorated proof tree is not a fixpoint decorated proof tree, as states the following proposition:

Proposition 4.2.5.
Let Π be a DATALOG *program with goal predicate Q, $\mathcal{L}$ a finite set of IDB predicates, $\mathcal{R}$ a monadic rule whose IDB predicates are elements of $\mathcal{L}$, and $\tau \in p\_dec(\mathcal{L}, Q, \Pi)$. Then, $\tau \in p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ if and only if $\mathcal{U}(\tau) \notin u\_dec^{pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$.* ∎

Proof. See Appendix A, Proof A.3.10. □

Having seen how to check whether $\tau \in p\_dec^{fp}(\Pi_{\mathcal{M}}, Q, \Pi)$, we have to look at a criterion stating whether $\exists \sigma^\bullet \in u\_trees(Q_{\mathcal{M}}, \Pi^\bullet_{\mathcal{M}})$ whose root is a leaf and a decorating containment mapping $\sigma^\bullet \xrightarrow{h^\bullet} \tau$. But, this time, it appears that verifying that a decorating containment mapping exists is easier, since we only have to exhibit it.

Definition 4.2.6.
Let τ be some decorated proof tree, and

$$\mathcal{R} = R_1(\mathbf{v}^1) \wedge \ldots \wedge R_m(\mathbf{v}^m) \Rightarrow R(\mathbf{v})$$

be a rule whose body contains EDB atoms $R_{j_1}(\mathbf{v}^{j_1}), \ldots, R_{j'_l}(\mathbf{v}^{j'_l})$ and monadic IDB atoms $R_{i_1}(\mathbf{v}^{i_1}), \ldots, R_{i_l}(\mathbf{v}^{i_l})$, where $\{R_{i_1}, \ldots, R_{i_l}\} \subseteq \mathcal{L}_\tau$. A *reaching goal certificate* from $\mathcal{R}$ to τ is a mapping h from the occurrences of variables $v \in var(\mathcal{R})$ to occurrences of variables $v \in var(\tau)$ such that

- if v_1 and v_2 are two occurrences of a variable $v \in var(\mathcal{R})$, then the occurrences $h(\mathsf{v}_1), h(\mathsf{v}_2) \in var(\tau)$ are connected.
- if an EDB predicate $R_{i_l}(\mathbf{v}^{i_l})$ holds in $\mathcal{R}$, then $R_{i_l}(h(\mathbf{v}^{i_l}))$ holds in τ.
- if an IDB predicate $R_{j_l}(\mathbf{v}^{j_l})$ holds in the body of $\mathcal{R}$, then $(h(v^{j_l}), R_{j_l}) \in \varphi_n$, where n is the node of τ containing the occurrence $h(\mathbf{v}^{j_l})$.
- $R(h(\mathbf{v}))$ is the goal predicate of the the root r of τ.

If Π is a DATALOG program with goal predicate Q, we denote by $p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ the set of trees $\tau \in p\_dec(\mathcal{L}, Q, \Pi)$ such that a reaching goal certificate from $\mathcal{R}$ to τ exists. ∎

And, indeed, the *reaching goal certificate* defined above reaches our demands, which were to have a certificate proving that a decorating containment mapping exists as states the following proposition:

Proposition 4.2.7.
Let Π be a DATALOG *program with goal predicate Q, $\Pi_{\mathcal{M}}$ a monadic* DATALOG *program with goal predicate $Q_{\mathcal{M}}$, $\mathcal{L}$ the set of internal IDB predicates of $\Pi_{\mathcal{M}}$, and $\tau \in p\_dec(\mathcal{L}, Q, \Pi)$. Then, $\exists \sigma^\bullet \in u\_trees(Q_{\mathcal{M}}, \Pi^\bullet_{\mathcal{M}})$ whose root is a leaf and a decorating containment mapping $\sigma^\bullet \xrightarrow{h^\bullet} \mathcal{U}(\tau)$ if and only if $\tau \in p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ for some goal rule $\mathcal{R}$ of $\Pi_{\mathcal{M}}$.* ∎

Proof. See Appendix A, Proof A.3.11. □

The two above propositions, combined to Theorem 4.1.7, are sufficient to prove the following main theorem:

Theorem 4.2.8.
Let Π be a DATALOG *program with goal predicate Q, $\Pi_{\mathcal{M}}$ a monadic* DATALOG *program with goal predicate $Q_{\mathcal{M}}$, and let $\mathcal{L}$ be the set of the internal IDB predicates of $\Pi_{\mathcal{M}}$. Π is contained in $\Pi_{\mathcal{M}}$ if and only if every tree $\tau \in p\_dec(\mathcal{L}, Q, \Pi)$ satisfies one of the two following sentences:*

- $\tau \in p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ *for some internal rule* $\mathcal{R}$ *of* $\Pi_{\mathcal{M}}$.
- $\tau \in p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ *for some goal rule* $\mathcal{R}$ *of* $\Pi_{\mathcal{M}}$.

■

Proof. See Appendix A, Proof A.3.12. □

4.3 Tree Automata

The main feature of decorated proof trees, as opposed to decorated expansion trees, is the fact that the number of possible labels is finite. Because this set of labels is finite, the set of decorated proof trees $p\_dec(Q, \Pi)$, for an IDB predicate Q in a program Π, as well as the related sets of decorated proof trees $p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ and $p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$, can be described by a tree automaton.

In this section, after each proposition stating the existence of an automaton of bounded size that recognises a specified set of decorated proof trees, we indicate a construction of such an automaton. However, the correctness of these constructions will not be addressed in this report, not in its main body nor in the annex. Indeed, the proof of the correctness of these constructions is very similar to the proofs indicated in [9].

Proposition 4.3.1.
Let Π *be a* DATALOG *program with a monadic goal predicate* Q *and* $\mathcal{L}$ *a finite set. There is an automaton* $A^{p\_dec}_{\mathcal{L},Q,\Pi}$*, whose size is exponential in the size of* Π *and* $\mathcal{L}$*, such that* $T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) = p\_dec(\mathcal{L}, Q, \Pi)$. ■

Proof. We design an automaton that treats a tree τ and verifies that the triples labelling every node of τ are coherent with the fact that τ be a decorated proof tree. It follows that

$$T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) = \{\tau \in p\_dec(\mathcal{L}, Q, \Pi)\}$$

The construction of this automaton is detailed in Appendix A, Proof A.3.13. □

Proposition 4.3.2.
Let Π *be a* DATALOG *program with a monadic goal predicate* Q*,* $\mathcal{L}$ *be a finite set of IBD predicates, and* $\mathcal{R}$ *be a monadic rule whose IBD predicates are elements of* $\mathcal{L}$*. Then there is an automaton* $A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R},Q,\Pi}$*, whose size is exponential in the size of* Π*,* $\mathcal{L}$ *and* $\mathcal{R}$*, such that* $T(A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R},Q,\Pi}) \cap T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) = p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$. ■

Proof. We design a non-deterministic automaton that treats a tree τ, supposing that we have already certified that τ was a decorated proof tree, tries to guess an infixpoint certificate and accepts τ if an infixpoint certificate for the rule $\mathcal{R}$ has been found. It follows that

$$\{\tau \in p\_dec(\mathcal{L}, Q, \Pi) \cap T(A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R},Q,\Pi})\} = \{\tau \in p\_dec(\mathcal{L}, Q, \Pi) \cap p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)\}$$

The construction of this automaton is detailed in Appendix A, Proof A.3.14. □

Proposition 4.3.3.
Let Π *be a* Datalog *program with a monadic goal predicate* Q, $\mathcal{L}$ *be a finite set of IBD predicates, and* $\mathcal{R}$ *be a rule such that IBD predicates in its body are elements of* $\mathcal{L}$. *Then there is an automaton* $A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{R},Q,\Pi}$, *whose size is exponential in the size of* Π, $\mathcal{L}$ *and* $\mathcal{R}$, *such that* $T(A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{R},Q,\Pi}) \cap T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) = p\_dec^{goal}(\mathcal{L},\mathcal{R},Q,\Pi)$. ∎

Proof. We design a non-deterministic automaton that treats a tree τ, supposing that we have already certified that τ was a decorated proof tree, tries to guess a reaching-goal certificate and accepts τ if a reaching-goal certificate for the rule $\mathcal{R}$ has been found. It follows that

$$\{\tau \in p\_dec(\mathcal{L},Q,\Pi) \cap T(A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{R},Q,\Pi})\} = \{\tau \in p\_dec(\mathcal{L},Q,\Pi) \cap p\_dec^{goal}(\mathcal{L},\mathcal{R},Q,\Pi)\}$$

The construction of such an automaton is detailed in Appendix A, Proof A.3.15. □

Thanks to the existence of the automata involved in Propositions 4.3.1, 4.3.2 and 4.3.3, we can now reduce the containment problem for Datalog programs in unions of conjunctive queries to an automata-theoretic problem.

Theorem 4.3.4.
Let Π *be a program with monadic goal predicate* Q, $\Pi_{\mathcal{M}}$ *a monadic program with goal predicate* $Q_{\mathcal{M}}$. *Let* $\mathcal{L}$ *be the set of internal IDB predicates in* $\Pi_{\mathcal{M}}$, $\mathcal{R}_1,\ldots,\mathcal{R}_k$ *the internal rules of* $\Pi_{\mathcal{M}}$ *and* $\mathcal{Q}_1,\ldots,\mathcal{Q}_l$ *the goal rules of* $\Pi_{\mathcal{M}}$. *Then,* Π *is contained in* $\Pi_{\mathcal{M}}$ *if and only if*

$$T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) \subseteq \bigcup_{i=1}^{k} T(A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R}_i,Q,\Pi}) \cup \bigcup_{i=1}^{l} T(A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{Q}_i,Q,\Pi})$$

∎

Proof. See Appendix A, Proof A.3.16. □

Finally, Theorem 4.3.4 indicates us an algorithm checking whether a given Datalog program is contained in another given monadic Datalog program.

Theorem 4.3.5.
Containment of a Datalog *program in a monadic* Datalog *program is in 2EXPTIME.*
∎

Proof. See Appendix A, Proof A.3.17. □

Chapter 5

Transitive Programs

5.1 A Derived Non-Recursive Program

Similarly to what we did with monadic DATALOG programs, we solve the containment problem in transitive DATALOG programs by storing as much information as possible. However, similarly to what happened with decorated unfolding trees, which had been designed to solve the containment problem in monadic programs, information will have to be stored in the nodes of a tree, involving a bounded number of variables.

Therefore, an efficient idea was to reduce every transitive program to a non-recursive DATALOG program, so that we could bound *a priori* the number of variables appearing in each expansion tree of this new non-recursive program. However, the difficulty is that this new non-recursive program has to be rich enough to give us the information necessary to the resolution of the containment problem. It was while looking for such a suitable non-recursive DATALOG program that I invented the *diamond-reduction* of a transitive DATALOG program γ, which consists in the program defined below:

Definition 5.1.1.
Let γ be a transitive DATALOG program. The *diamond-reduction* of γ is the DATALOG program $\gamma^\diamond$ built hereafter:

Every predicate in γ is a predicate in $\gamma^\diamond$. Moreover, in addition to these predicates, $\gamma^\diamond$ contains a binary EDB predicate $G^\diamond$, called *diamond predicate*, for each star IDB predicate G^* in γ. The rules in $\gamma^\diamond$ are defined as follows:

- Each star-free rule

$$E_1(\mathbf{v}^1) \wedge \ldots \wedge E_k(\mathbf{v}^k) \wedge G_1^*(x_1^1, x_2^1) \wedge \ldots \wedge G_l^*(x_1^l, x_2^l) \Rightarrow G(\mathbf{t})$$

  in γ is also a rule in $\gamma^\diamond$.
- For each star IDB predicate G^* in γ, $\gamma^\diamond$ contains so-called *diamond-rules*, which are rules ρ having the following property:
  - $\exists n \in \{1, 2, 3, 4, 5\}$ such that ρ is a rule

$$E_1(x_0, x_1) \wedge E_2(x_1, x_2) \wedge \ldots \wedge E_n(x_{n-1}, x_n) \Rightarrow G^*(x_0, x_n)$$

– $\forall i \in \{1, 2, \ldots, n\}, E_i$ is either the predicate G or the predicate $G^\diamond$.
– $\forall i \in \{1, 2, \ldots, n-1\}$, if $E_i = G$, then $E_{i+1} = G^\diamond$.

■

As wanted initially, every diamond-reduction of every transitive program is a non-recursive program, and therefore has only finitely many unfolding trees:

Proposition 5.1.2.
Let γ be a transitive DATALOG *program with goal predicate Γ. Let $\gamma^\diamond$ be its diamond-reduction. $\gamma^\diamond$ is a non-recursive program, and $u\_trees(\Gamma, \gamma^\diamond)$ is finite.* ■

Proof. See Appendix A, Proof A.4.7. □

5.2 Labelled Proof Trees

The class of proof trees has the convenient property of requiring a limited number of variables. However, it only features unfolding expansion trees. Therefore, we decide to add some labellings to those proof trees, in order to have a richer structure.

Definition 5.2.1.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program with goal predicate Γ. For al τ in $u\_trees(\Gamma, \gamma^\diamond)$, we denote by $\mathcal{P}_\tau$ the set of occurrences of atoms appearing in τ.

We define $\mathcal{L}(\gamma, \Pi)$ to be the set of tuples (τ, S, r, V, φ) where $\tau \in u\_trees(\Gamma, \gamma^\diamond)$ has its variables among $var_\diamond(\gamma^\diamond)$, S is a subset of $\mathcal{P}_\tau$, r is an instance of a rule of Π with variables among $var(\Pi)$, $V \subseteq var(r)$ and $\varphi \in (V \cup var_\diamond(\gamma^\diamond))^{var_\diamond(\gamma^\diamond)}$ is a total mapping from $var_\diamond(\gamma^\diamond)$ to $V \cup var_\diamond(\gamma^\diamond)$.

Let $\Sigma \in p\_trees(Q, \Pi)$ be a tree whose nodes are $N_1, \ldots, N_{|\Sigma|}$. A *labelling* of Σ is a tree σ whose nodes are $n_1, \ldots, n_{|\Sigma|}$, each n_i being labelled by a couple (N_i, L_i) such that

- $L_i \subseteq \mathcal{L}(\gamma, \Pi)$.
- if $N_i = (R_i(\mathbf{t}_i), \rho_i)$, then $\forall l = (\tau, S, r, V, \varphi) \in L_i$, $r = \rho_i$.
- $\forall i, j, n_i$ is a child of n_j if and only if N_i is a child of N_j.

Conversely, we also say that Σ is the *de-labelling* of σ, which we denote $\Sigma = d(\sigma)$.

We call *labelled proof trees* of Π the labelling of the proof trees $\Sigma \in p\_trees(Q, \Pi)$. If $\Sigma \in p\_trees(Q, \Pi)$ is a proof tree, we denote by $p\_label(\Sigma, \gamma, Q, \Pi)$ the set of all labellings of Σ. We also define $p\_label(\gamma, Q, \Pi)$ to be the set of all the labelled proof trees of Π:

$$p\_label(\gamma, Q, \Pi) = \bigcup_{\Sigma \in p\_trees(Q, \Pi)} p\_label(\Sigma, \gamma, Q, \Pi)$$

■

We develop again the similarity to the path followed while designing the algorithm of decision of the containment in monadic programs: we had used the concept of proof trees fixpoint relatively to a monadic program; more precisely, it consisted in saying that the statements that have been approved as being true should be closed under application of any internal rule of the monadic program we looked at.

Here, let us consider a transitive program γ. We say that a labelled proof tree τ is fixpoint relatively to a transitive program if, supposing that the EDB atoms of γ present in τ and that the rules in γ are true, the labels of τ contain as many true statements as possible, in the sense of their *interpretation* defined in Appendix A, Definitions A.4.4 and A.4.10.

Definition 5.2.2.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. We identify every tree $\sigma \in p\_label(\gamma, Q, \Pi)$ to a couple $(\Sigma, \mathcal{E})$, where $\Sigma = d(\sigma)$ and $\mathcal{E}$ is the set $\mathcal{E} = \{(N, l) | \exists L \subseteq \mathcal{L}(\gamma, \Pi)$ such that $l \in L$ and (N, L) is a node of $\sigma\} \subseteq \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$.

Now, $\forall E \subseteq \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$, we define $\Psi_\Sigma(E) \subseteq \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$ to be the set of couples $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$ verifying one of the following conditions:

1. $(N, l) \in E$.
2. N is the father, in the tree Σ, of some node $N' = (R'(\mathbf{t}'), \rho') \in \mathcal{N}_\Sigma$, $V \subseteq var(\mathbf{t}')$ and $(N', (\tau, S, \rho', V, \varphi)) \in E$.
3. N is a child, in the tree Σ, of some node $N' = (R'(\mathbf{t}'), \rho') \in \mathcal{N}_\Sigma$, $V \subseteq var(\mathbf{t})$ and $(N', (\tau, S, \rho', V, \varphi)) \in E$.
4. $\exists S' \supseteq S$ such that $(N, (\tau, S', \rho, V, \varphi)) \in E$.
5. $\exists S', S'' \subseteq S$ such that $S = S' \cup S''$, $\varphi(var(S')) \cap \varphi(var(S'')) \subseteq V$, and such that both $(N, (\tau, S', \rho, V, \varphi)) \in E$ and $(N, (\tau, S'', \rho, V, \varphi)) \in E$.
6. $\exists \tau' \in u\_trees(\Gamma, \gamma^\diamond)$ such that $(N, (\tau', S, \rho, V, \varphi)) \in E$.
7. $\exists V' \subseteq var(\rho)$ such that $(N, (\tau, S, \rho, V', \varphi)) \in E$.
8. $\exists h : var_\diamond(\gamma^\diamond) \cup V \to var_\diamond(\gamma^\diamond) \cup V$ such that $h_{|V} = Id_V$ and $(N, (\tau, S, \rho, V, h \circ \varphi)) \in E$.
9. $\exists \varphi' \in (var_\diamond(\gamma^\diamond) \cup V)^{var_\diamond(\gamma^\diamond)}$ such that $\varphi'_{|var(S)} = \varphi_{|var(S)}$ and $(N, (\tau, S, \rho, V, \varphi')) \in E$.
10. $\exists n' = (R'(\mathbf{t}'), \rho') \in \mathcal{N}_\tau$ and $\exists S' \subseteq S$ such that $S = S' \cup \{R'(\mathbf{t}')\}$, S' contains all atoms in the body of ρ' and $(N, (\tau, S', \rho, V, \varphi)) \in E$.
11. $\exists \tau' \in u\_trees(\Gamma, \gamma^\diamond)$ and an atom $G^\diamond(x, y) \in S$ such that $(N, (\tau', S', \rho, V, \varphi)) \in E$, where $S' = S \backslash \{G^\diamond(x, y)\} \cup \{G(x, y)\}$.
12. $\exists \tau' \in u\_trees(\Gamma, \gamma^\diamond)$ an atom $G^\diamond(x, y) \in S$ and a variable $z \in var_\diamond(\gamma^\diamond)$ such that $(N, (\tau', S', \rho, V, \varphi)) \in E$, where $S' = S \backslash \{G^\diamond(x, y)\} \cup \{G^\diamond(x, z), G^\diamond(z, y)\}$.
13. S is a singleton $\{G^\diamond(x, y)\}$ such that $\varphi(x) = \varphi(y)$.
14. S is a singleton $\{P(\mathbf{v})\}$ where P is an EDB predicate and φ maps $P(\mathbf{v})$ to an atom $P(\mathbf{x})$ appearing in ρ.
15. $S = \emptyset$.

A *fixpoint labelled proof tree* is a tree $\sigma \in p\_label(\gamma, Q, \Pi)$ such that $i(\sigma) = (\Sigma, \mathcal{E})$ verifies $\mathcal{E} = \Psi_\Sigma(\mathcal{E})$. We denote by $p\_label^{fp}(\Sigma, \gamma, Q, \Pi)$ the set of fixpoint labelled proof

trees $\sigma \in p\_label(\Sigma, \gamma, Q, \Pi)$, and by $p\_label^{fp}(\gamma, Q, \Pi)$ the set of fixpoint labelled proof trees $\sigma \in p\_label(\gamma, Q, \Pi)$. ∎

It seems that both notions of labelled proof trees and of fixpoint labelled proof trees are dual. Indeed, since finding an adapted definition for labelled proof trees was obviously a necessary step, choosing which of those would be fixpoints was also critical. And, in fact, I got the ideas of these two notions at the same time.

After having found the algorithm detailed in the Chapter 4 and read many other articles dealing with automata on trees, practical uses of DATALOG, or various containment problems of DATALOG programs, I acquired a good intuition of how tree automata could work on proof trees, and how that would be used in the context of containment problems.

The idea behind those objects is the following: in any node of a proof tree, we can identify only the variables present in this proof tree, the other variables of the proof tree being not defined in this node. Therefore, every label of any labelled proof tree represents a logical formula which is guaranteed to be true, and where the free variables are all variables present in the node. More precise information of the detail of these formulaæ is available in Appendix A, Definitions A.4.4 and A.4.10.

Now, here are the arguments that led me to design the notion of diamond-reduction and the associated fixpoint labelled proof trees: a general method to show that a program A is contained in a program B is to exhibit, for any unfolding tree τ_A of A, a containment mapping from an unfolding tree τ_B of B to τ_A. Similarly, here, if a program Π is contained in a transitive program γ, the main idea is to find, for every unfolding tree τ of Π, a containment mapping from some unfolding tree τ' of γ to τ. And, in fact, finding such a mapping is somehow equivalent to finding a proof, given the rules of γ, the variables present in τ, and the predicates indicated in the nodes τ, that a certain predicate over the variables of τ is true: if $Q(\mathbf{t})$ is the atom in the head of τ and if Γ is the goal predicate of γ, we want to prove that $\Gamma(\mathbf{t})$ holds.

How is it possible to find such a proof? An idea is that it is possible to prove every EDB atom that is true, since such an EDB atom must be directly present among the EDB atoms of τ. Now, we proceed inductively: if it is possible to prove $\phi_1(\mathbf{t}^1)$ or to prove $\phi_2(\mathbf{t}^2)$, it must be possible to prove $\phi_1(\mathbf{t}^1) \vee \phi_2(\mathbf{t}^2)$; and if it is possible to prove $\phi_1(\mathbf{t}^1)$ and to prove $\phi_2(\mathbf{t}^2)$, it must be possible to prove $\phi_1(\mathbf{t}^1) \wedge \phi_2(\mathbf{t}^2)$. Therefore, the entire difficulty is to find how to prove $\phi^*(x, y)$, where ϕ^* is the transitive closure of some binary relation ϕ, if we can prove enough formulæ involving ϕ. Indeed, the difficulty appears here since it is here the appears the recursion, and therefore that we cannot bound *a priori* the number of formulæ $\phi(a, b)$ that we will have to prove, and the number of variables that may be involved in such a proof.

However, there is a trick that allows us to skip this potential infinity: our aim is to prove a statement of the form

$$\Lambda(x, y, \mathbf{v}) \wedge \phi^*(x, y)$$

where Λ is some logical formula, $\mathbf{v}$ is the tuple formed of the variables present in the current node, and x, y are two more variables. It is sufficient to prove some stronger

statement of the form

$$(\exists a, b)\,(\Lambda(\mathbf{v}) \wedge \phi^*(x, a) \wedge \phi(a, b) \wedge \phi^*(b, y))$$

Again, it is sufficient to prove separately statements of the form

$$(\exists a)\,(\Lambda_1(\mathbf{v}) \wedge \phi^*(x, a) \wedge \phi_1(a, \mathbf{v}))$$

$$(\exists b)\,(\Lambda_2(\mathbf{v}) \wedge \phi^*(b, y) \wedge \phi_2(b, \mathbf{v}))$$

if $\Lambda(\mathbf{v}) \equiv \Lambda_1(\mathbf{v}) \wedge \Lambda_2(\mathbf{v})$ and if we can show $(\forall a, b, \mathbf{w})\,(\phi_1(a, \mathbf{w}) \wedge \phi_2(b, \mathbf{w}) \Rightarrow \phi(a, b))$.

The question that shall be asked now is how to find suitable a and b. Since the idea above implies splitting the predicate $\phi^*(x, y)$ in two *smaller* predicates, here is the method I found: when looking at some formula in a node of the unfolding tree, the only variables that may be identified are those appearing in the body of the rule labelling the node. By removing this node, we split the tree in several sub-trees, and can look at which variables are defined in which sub-trees. If x and y are defined both in one same sub-tree, we may delay the treatment of the predicate $\phi^*(x, y)$. If they are not, there must be variables a and b, one being defined only in the sub-tree of x and the other not, such that a and b be two consecutive steps in a shorter path labelled by ϕ and linking x to y: we choose these variables and then, recursively, we are able to apply the trick mentioned in the paragraph just above.

Finally, the structure of labelled proof tree associated to the notion of fixpoint labelled proof trees defined just above are suitable for the design of an algorithm deciding the containment of a program in a transitive program. Indeed, the following theorem establishes the equivalence of the containment of a DATALOG program in a transitive DATALOG program and the existence of a common characteristic to all fixpoint labelled proof trees, existence that is very easy to check thanks to tree automata:

Theorem 5.2.3.
Let Π *be a* DATALOG *program with goal predicate* Q*,* γ *a transitive* DATALOG *program with goal predicate* Γ*.*

Π *is contained in* γ *if and only if, for every tree* $\sigma \in p\_label^{fp}(\gamma, Q, \Pi)$*, the root* $(N, L) = ((R(\boldsymbol{t}), \rho), L)$ *of* σ *is such that, for some tuple of variables* $\boldsymbol{v}$ *and some label* $l = (\tau, \{\Gamma(\boldsymbol{v})\}, \rho, V, \varphi) \in L$*,* $\varphi(\boldsymbol{v}) = \boldsymbol{t}$*.* ∎

Proof. See Appendix A, Sections A.4.1, A.4.2, A.4.3 and A.4.4. □

5.3 Tree Automata

We define here another non-decreasing application, forgetting only the two criteria that link the labels associated to neighbour nodes of the tree. Indeed, these two criteria are the only two that refer to the labels associated to different nodes, while the thirteen other refer only to the labels of a node itself.

Definition 5.3.1.
Now, $\forall E \subseteq \mathcal{L}(\gamma, \Pi)$, let $\Phi(E) \subseteq \mathcal{L}(\gamma, \Pi)$ be the set of 5-uples $l = (\tau, S, \rho, V, \varphi) \in \mathcal{L}(\gamma, \Pi)$ verifying one of the following conditions:

1. $l \in E$.
2. $\exists S' \supseteq S$ such that $(\tau, S', \rho, V, \varphi) \in E$.
3. $\exists S', S'' \subseteq S$ such that $S = S' \cup S''$, $\varphi(var(S')) \cap \varphi(var(S'')) \subseteq V$, $(\tau, S', \rho, V, \varphi) \in E$ and $(\tau, S'', \rho, V, \varphi) \in E$.
4. $\exists \tau' \in u\_trees(\Gamma, \gamma^\diamond)$ such that $(\tau', S, \rho, V, \varphi) \in E$.
5. $\exists V' \subseteq var(\rho)$ such that $(\tau, S, \rho, V', \varphi) \in E$.
6. Some total mapping $h : var_\diamond(\gamma^\diamond) \cup V \to var_\diamond(\gamma^\diamond) \cup V$ verifies $h_{|V} = Id_V$ and $(\tau, S, \rho, V, h \circ \varphi) \in E$.
7. $\exists \varphi' \in (var_\diamond(\gamma^\diamond) \cup V)^{var_\diamond(\gamma^\diamond)}$ such that $\varphi'_{|var(S)} = \varphi_{|var(S)}$ and $(\tau, S, \rho, V, \varphi') \in E$.
8. $\exists n' = (R'(\mathbf{t}'), \rho') \in \mathcal{N}_\tau$ and $\exists S' \subseteq S$ such that $S = S' \cup \{R'(\mathbf{t}')\}$, S' contains all atoms in the body of ρ' and $(\tau, S', \rho, V, \varphi) \in E$.
9. $\exists \tau' \in u\_trees(\Gamma, \gamma^\diamond)$ and an atom $G^\diamond(x, y) \in S$ such that $(\tau', S', \rho, V, \varphi) \in E$, where $S' = S \backslash \{G^\diamond(x, y)\} \cup \{G(x, y)\}$.
10. $\exists \tau' \in u\_trees(\Gamma, \gamma^\diamond)$, an atom $G^\diamond(x, y) \in S$ and a variable $z \in var_\diamond(\gamma^\diamond)$ such that $(\tau', S', \rho, V, \varphi) \in E$, where $S' = S \backslash \{G^\diamond(x, y)\} \cup \{G^\diamond(x, z), G^\diamond(z, y)\}$.
11. S is a singleton $\{G^\diamond(x, y)\}$ such that $\varphi(x) = \varphi(y)$.
12. S is a singleton $\{P(\mathbf{v})\}$ where P is an EDB predicate and φ maps $P(\mathbf{v})$ to an atom $P(\mathbf{x})$ appearing in ρ.
13. $S = \emptyset$.

∎

This non-decreasing application facilitates the description of an automaton that will recognise only the trees in $p\_label^{fp}(\gamma, Q, \Pi)$, therefore allowing us to determine an algorithm that decides the containment of a Datalog program in a transitive Datalog program.

Theorem 5.3.2.
Let Π *be a* Datalog *program with goal predicate* Q*, and* γ *be a transitive* Datalog *program with goal predicate* Γ*. There is an automaton* $\mathcal{A}^{Q,\Pi}_{\Gamma,\gamma}$*, whose size is doubly exponential in the size of* Π *and triply exponential in the size of* γ*, such that* $T(\mathcal{A}^{Q,\Pi}_{\Gamma,\gamma}) = \emptyset$ *if and only if* Π *is contained in* Γ*.* ∎

Proof. We design a non-deterministic automaton that treats a tree τ, checks whether τ is a labelled proof tree, whether the labels (N, L) of its nodes verify $\Phi(L) = L$, and whether the criteria #2 and #3 of the Definition 5.2.2 are verified throughout the tree: the three points above are verified if and only if $\tau \in p\_label^{fp}(\gamma, Q, \Pi)$. Then, the automaton accepts τ if the three points above are verified, and if the root $(N, L) = ((R(\mathbf{t}), \rho), L)$ of τ is such that no tuple of variables $\mathbf{v}$ and no $l = (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi) \in L$ satisfy the equality $\varphi(\mathbf{v}) = \mathbf{t}$.

It follows that $T(\mathcal{A}^{Q,\Pi}_{\Gamma,\gamma}) = \emptyset$ if and only if Π is contained in Γ. The construction of such an automaton is detailed in Appendix A, Proof A.4.38. □

Finally, Theorem 5.3.2 indicates us an algorithm checking whether a given DATALOG program is contained in another given transitive DATALOG program.

Theorem 5.3.3.
Containment of a DATALOG *program in a transitive* DATALOG *program is in 3EXPTIME.*
■

Proof. See Appendix A, Proof A.4.39. □

Chapter 6

Conclusion

6.1 Further Work

If the containment or the equivalence of DATALOG programs is not decidable in the general case, the particular classes that are monadic and transitive DATALOG programs are particular instances for which these two problems are decidable. Indeed, both monadic and transitive DATALOG programs are expressible in monadic second-order logic, which allows us to use COURCELLE's theorem, and therefore to prove the decidability of the containment problem for these instances.

However, COURCELLE's theorem provides us a much too inefficient algorithm, since its complexity was non-elementary, while it was possible to design algorithms with elementary complexity. Therefore, we have now new upper bounds for the containment and the equivalence problems in the case of monadic or transitive DATALOG programs.

In both cases, the use of automata-theoretic techniques, as advocated in [9], happened to be quite successful, since it allowed me to find and prove the two algorithms presented in this report. It is remarkable that these two attempts were successful, since it was highly not self-evident that such efficient algorithms would exist.

But two points remain unexplored: first, we have not studied the existence of lower bounds for these problems, and it remains possible that better algorithms exist. Furthermore, if equivalence is reducible to containment, it is not obvious that, in the particular cases that interest us, these problems be of the same complexity class. In particular, the most uncertain point is the existence of more efficient algorithms that would not be based on automata techniques, but would involve totally different ideas. Indeed, since the dramatic improvement that was obtained during this project was due to the use of techniques that were not involved in the algorithm provided by COURCELLE's result, we cannot assure now that that automata techniques allow us to design the most efficient algorithms.

6.2 Personal Reflexion

This research internship was, for me, an exceptional opportunity to discover what the world of academic research can be. In that perspective, I am particularly glad of the experience I lived.

Indeed, since I started my upper education, mathematics and theoretical computer science have always interested me. That is why doing research appeared as the eventual conclusion of my formation. However, six months ago, even while imagining that I *would* want to do research later, I still did not know what research could be. And I have to acknowledge that I enjoyed it very much.

I enjoyed it mostly because of the thrill of the discovery. Previously, while at school, I could not discover anything. Trying to guess the demonstration of a theorem before the teacher tells you what it is provided me an edulcorated version of this thrill: I could have discovered a theorem, even if someone did it before me. But, otherwise, what I did was discovering objects, learning what other people had already found, and eventually solving problems that had been especially designed to be solved with a prepared set of tools.

During these four months, I had, first of all, to learn what people had previously found, like what I had done before. However, there was already a little improvement: if I could, and very often did, ask questions to my advisor, Moshe Y. Vardi, I could also take the initiative of looking by myself for sources of explications that would be adapted to my level of comprehension.

Then arrived the fear: the fear before the unknown, since I would investigate problems for which no solution had, to my knowledge, ever been found before. I could not be assured to succeed, since it was even possible that the problems I wanted to solve be intrinsically unsolvable. Hopefully, Moshe Y. Vardi had the excellent idea of letting me work first on a problem — about the containment in monadic Datalog programs — he had *almost* solved himself: he had not spent enough time on it to find a good, closed solution, but he had already thought of all the elements of the solution.

That is why, while investigating solutions on the second problem — about the containment in transitive Datalog programs — I was confident that I would find a solution. And it is when finding such a solution that I had this feeling of thrill. I had experienced fear again, while looking desperately for a simple proof, and thinking that I might have spent one month to prove a theorem that would eventually be false. But, when I first thought of the solution presented in this report, as well as when, one month later, I proved it, I had the terrific feeling of having discovered something, which is the feeling that made research so attractive to me when I was younger.

Finally, I want to thank again all the people I met during this internship. Because these people allowed me to live a great experience, an experience that assures me now that, even if I eventually choose not to do research later, it will be for something that I will love even more, and therefore that I will enjoy very much.

Bibliography

[1] C. Beeri: On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. on Database Systems*, 5(1980), pp. 241–259.

[2] D. Calvanese, G. de Giacomo, M.Y. Vardi: Decidable containment of recursive queries. *Theoretical Computer Science* 336(2005), pp. 33–56.

[3] D. Calvanese, G. de Giacomo, M. Lenzerini, M.Y. Vardi: Containment of conjunctive regular path queries with inverse. *Symposium on Principles of Database Systems* 19(2000), pp. 58–66.

[4] D. Calvanese, G. de Giacomo, M. Lenzerini, M.Y. Vardi: An automata-theoretic approach to regular XPath. *Symposium on Database Programming Languages* 12(2009).

[5] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi: *Tree Automata Techniques and Applications*, 2007.

[6] O.L. Costich: A Medvedev characterization of sets recognized by generalized finite automata. *Math. System Theory* 6(1972), pp. 263–267.

[7] B. Courcelle: The monadic second-order theory of graphs I – Recognizable sets of finite graphs. *Information and Computation* 85(1990), pp. 12–75.

[8] B. Courcelle: Recursive queries and context-free graph grammars. *Theoretical Computer Science* 78(1991), pp. 217–244.

[9] S. Chaudhuri, M.Y. Vardi: On the equivalence of recursive and nonrecursive Datalog programs. *J. Computer and System Sciences* 54(1994), pp. 61–78.

[10] J.E. Doner: Tree acceptors and some of their applications. *J. Computer and System Sciences* 4(1971), pp. 406–451.

[11] Y. Sagiv, M. Yannakakis: Equivalences among Relational Expressions with the union and difference operators. *JACM*, 27:4, pp 633-655.

[11] H. Seidl: Deciding equivalence of finite tree automata. *SIAM J. Computing* 19(1990), pp. 424–437.

[13] J.W. Thatcher, J.B. Wright: Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical System Theory* 2(1968), pp. 57–81.

[14] J.D. Ullman: *Principles of Database and Knowledge Base Systems*, Vol. 2, Computer Science Press, 1989.

[15] M.Y. Vardi: Automata theory for database theoreticians. In *Theoretical Studies in Computer Science* (J.D. Ullman, ed.), Academic Press, 1992, pp. 153–180.

Appendix A

Detailed Proofs

A.1 Datalog Programs

Proposition A.1.1. [Proposition 2.5.1]
We denote by $\mathbb{N}[X]$ the set of polynomials with non-negative integer coefficients. Let $\leq$ be the ordering on $\mathbb{N}[X]$ defined as $P \leq Q$ if and only if $Q - P$ has a leading coefficient in $\mathbb{N}$. $\leq$ is a well-ordering. ∎

Proof. $\leq$ is obviously a total ordering. Let us assume that some set $S \neq \emptyset$ has no minimal element: we can build a decreasing sequence $(P_i)_{i\geq 0}$ of elements of $S \subseteq \mathbb{N}[X]$.

Therefore, the set $\mathcal{D} = \{(P_i)_{i\geq 0} | \forall i \in \mathbb{N}, P_i \in \mathbb{N}[X] \text{ and } P_{i+1} < P_i\} \neq \emptyset$. Let $\delta = \min\{\deg(P_0) | (P_i) \in \mathcal{D}\}$ be the minimal degree of all first terms of such sequences, and $\Delta = \{(P_i) \in \mathcal{D} | \deg(P_0) = \delta\} \neq \emptyset$. We then set $\gamma = \min\{(P_0)_\delta | (P_i) \in \Delta\}$ to be the minimal leading coefficient of all first terms of sequences of Δ, and $\Gamma = \{(P_i) \in \Delta | (P_0)_\delta = \gamma\} \neq \emptyset$.

Let be $(P_i)_{i\geq 0} \in \Gamma$. $\forall n \in \mathbb{N}$, the sequence $(P_{i+n})_{i\geq 0} \in \mathcal{D}$ and $P_{0+n} \leq P_0$, so that $(P_{i+n})_{i\geq 0} \in \Gamma$. Therefore, we also know that $(P_i - \gamma X^\delta)$ is a decreasing sequence of polynomials in $\mathbb{N}[X]$. Then, $(P_i - \gamma X^\delta) \in \mathcal{D}$ and $\deg(P_0 - \gamma X^\delta) < \delta$, which is impossible since δ is minimal.

This proves that our initial assumption was false, and therefore that the Proposition 2.5.1 is true. □

Proposition A.1.2. [Proposition 2.5.2]
Let Π be a monadic Datalog *program. Π is equivalent to a monadic* Datalog *program Π' such that:*

- *the goal predicate of Π' does not appear in the body of any rule of Π'.*
- *every IDB predicate of Π' which is not the goal predicate is monadic.*

∎

Proof. Let $P_1, \ldots, P_k$ be the predicates in Π. If P is a predicate in Π and $\mathcal{R}$ is a DATALOG rule with k occurrences of P in its body, we associate the couple $(P, \mathcal{R})$ to the polynomial $\mathbf{N}_{P,\mathcal{R}}(X) = X^k$. Then, if D is a DATALOG program with predicates among $P_1, \ldots, P_k$ and whose rules are $\mathcal{R}_1, \ldots, \mathcal{R}_d$, we associate the couple (P, D) to the polynomial $\mathbf{N}_{P,D}(X) = \sum_{i=0}^{d} \mathbf{N}_{P,\mathcal{R}_i}(X)$.

Now, we rewrite the program Π in order to obtain a program of the form described above. Let be Q the goal predicate of Π. We create a new predicate Q_{goal} and a new rule $\mathcal{R}_{goal} : Q(\mathbf{x}) \Rightarrow Q_{goal}(\mathbf{x})$, to build a new DATALOG program Π_1 which, if its goal predicate is Q_{goal}, is equivalent to Π.

If Π_1 contains a non-monadic IDB predicate which is not Q_{goal}, we choose such a predicate P. Let be $\mathcal{R}_1, \ldots, \mathcal{R}_m$ the rules of Π_1 whose heads contain an occurrence of P: by renaming their variables, we suppose that each $\mathcal{R}_i$ is a rule $I_1^i(\mathbf{x}_1^i) \wedge \ldots \wedge I_{r_i}^i(\mathbf{x}_{r_i}^i) \Rightarrow P(\mathbf{x})$. If $\mathcal{R}$ is a rule $P(\mathbf{x}^1) \wedge I_2(\mathbf{x}^2) \wedge \ldots \wedge I_l(\mathbf{x}^l) \Rightarrow I_0(\mathbf{x}^0)$ whose body contains an occurrence of P, by renaming its variables, we suppose that $\mathbf{x}^1 = \mathbf{x}$ and that every variable appearing in $\mathcal{R}$ and in a rule $\mathcal{R}_i$ appears in the tuple $\mathbf{x}$. Now, we create m rules $\mathcal{R}'_1, \ldots, \mathcal{R}'_m$, where $\mathcal{R}'_i$ is the rule $I_1^i(\mathbf{x}_1^i) \wedge \ldots \wedge I_{r_i}^i(\mathbf{x}_{r_i}^i) \wedge I_2(\mathbf{x}^2) \wedge I_l(\mathbf{x}^l) \Rightarrow I_0(\mathbf{x}^0)$.

If we replace the rule $\mathcal{R}$ by the m rules $\mathcal{R}'_1, \ldots, \mathcal{R}'_m$, we transform the program Π_1 in an equivalent program Π_2 such that $\mathbf{N}_{P,\Pi_2}(X) < \mathbf{N}_{P,\Pi_1}(X)$. This process must be finite; therefore, by repeating it, we calculate a monadic program Π_3, equivalent to Π, such that P does not appear in the body of any rule of Π_3. Therefore, if we remove every rule of Π_3 whose head predicate is P, we obtain an equivalent monadic program Π_4, with predicates among $P_1, \ldots, P_n$, but whose set of predicates does not contain P.

By repeating this rewriting process for every non-monadic IDB predicate different from Q_{goal}, we successively obtain programs with at most $n, n-1, \ldots$ predicates. This process must be finite, and we eventually compute a monadic program Π' which is equivalent to Π, and that does not contain any non-monadic IDB predicate different from Q_{goal}. Since Q_{goal} cannot appear in the body of any rule of Π', Π' is the program whose existence is equivalent to the correctness of the Proposition 2.5.2. □

Proposition A.1.3. [Proposition 2.5.4]
Let Π *be a transitive* DATALOG *program.* Π *is equivalent to a transitive* DATALOG *program* Π' *such that:*

- *the goal predicate of* Π' *does not appear in the body of any rule of* Π'.
- *every IDB predicate of* Π' *that appears in the body of some star-free rule of* Π' *must be a star predicate.*
- *every IDB predicate of* Π' *which is not the goal predicate is of arity* 2.

■

Proof. Let $P_1, \ldots, P_k$ be the predicates in Π. If P is a predicate in Π and $\mathcal{R}$ is a DATALOG rule with k occurrences of P in its body, we associate the couple $(P, \mathcal{R})$ to the polynomial $\mathbf{N}_{P,\mathcal{R}}(X) = X^k$. Then, if D is a DATALOG program with predicates among $P_1, \ldots, P_k$ and whose rules are $\mathcal{R}_1, \ldots, \mathcal{R}_d$, we associate the couple (P, D) to the polynomial $\mathbf{N}_{P,D}(X) = \sum_{i=0}^{d} \mathbf{N}_{P,\mathcal{R}_i}(X)$.

Now, we rewrite the program Π in order to obtain a program of the form described above. Let be Q the goal predicate of Π. We create a new predicate Q_{goal} and a new rule $\mathcal{R}_{goal} : Q(\mathbf{x}) \Rightarrow Q_{goal}(\mathbf{x})$, to build a new DATALOG program Π_1 which, if its goal predicate is Q_{goal}, is equivalent to Π.

If the body of some star-free rule of Π_1 contains an IDB predicate which is not a star predicate, we choose such a predicate P. Let be $\mathcal{R}_1, \ldots, \mathcal{R}_m$ the rules of Π_1 whose heads contain an occurrence of P: by renaming their variables, we suppose that each $\mathcal{R}_i$ is a rule $I_1^i(\mathbf{x}_1^i) \wedge \ldots \wedge I_{r_i}^i(\mathbf{x}_{r_i}^i) \Rightarrow P(\mathbf{x})$. If $\mathcal{R}$ is a star-free rule $P(\mathbf{x}^1) \wedge I_2(\mathbf{x}^2) \wedge \ldots \wedge I_l(\mathbf{x}^l) \Rightarrow I_0(\mathbf{x}^0)$ whose body contains an occurrence of P, by renaming its variables, we suppose that $\mathbf{x}^1 = \mathbf{x}$ and that every variable appearing in $\mathcal{R}$ and in a rule $\mathcal{R}_i$ appears in the tuple $\mathbf{x}$. Now, we create m rules $\mathcal{R}'_1, \ldots, \mathcal{R}'_m$, where $\mathcal{R}'_i$ is the rule $I_1^i(\mathbf{x}_1^i) \wedge \ldots \wedge I_{r_i}^i(\mathbf{x}_{r_i}^i) \wedge I_2(\mathbf{x}^2) \wedge I_l(\mathbf{x}^l) \Rightarrow I_0(\mathbf{x}^0)$.

If we replace the rule $\mathcal{R}$ by the m rules $\mathcal{R}'_1, \ldots, \mathcal{R}'_m$, we transform the program Π_1 in an equivalent program Π_2 such that $\mathbf{N}_{P,\Pi_2}(X) < \mathbf{N}_{P,\Pi_1}(X)$. This process must be finite; therefore, by repeating it, we calculate a transitive program Π_3 equivalent to Π, such that P does not appear in the body of any star-free rule of Π_3. The set of predicates appearing in the body of any star-free rule of Π_3 is smaller than the set of predicates appearing in the body of any star-free rule of Π_1

By repeating this rewriting process for every IDB predicate which is not a star predicate, we successively obtain programs with at most $n, n-1, \ldots$ predicates in the bodies of their star-free rules. This process must be finite, and we eventually calculate a transitive program Π' equivalent to Π, whose star-free rules do not contain any star-free IDB predicate in their body.

Now, no IDB predicate Q of arity different from 2 can appear in the body of any star-free rule of Π' (since Q cannot be a star predicate). Moreover, Q cannot appear in the body of any baby-star or star rule of Π'. Therefore, if we erase every rule whose head predicate is not of arity 2, we obtain a DATALOG program Π'' equivalent to Π', and therefore to Π. Since Q_{goal} cannot appear in the body of any rule of Π'', Π'' is the program whose existence is equivalent to the correctness of the Proposition 2.5.4. □

A.2 Datalog, Finite Trees and Automata

Proposition A.2.1. [Proposition 2.5.4] [10, 13]
The non-emptiness problem for tree automata is decidable in polynomial time. ∎

Proof. Let $A = (\Sigma, S, S_0, \delta, F)$ be the given tree automaton. Let $accept(A)$ be the minimal set of states in S such that

- $F \subseteq accept(A)$, and
- if s is a state such that there are a letter $a \in \Sigma$ and a transition $\langle s_1, \ldots, s_k \rangle \in \delta(s, a) \cap accept(A)^*$, then $s \in accept(A)$.

It is easy to see that $T(A)$ is non-empty if and only if $S_0 \cap accept(A) \neq \emptyset$. Intuitively, $accept(A)$ is the set of all states that label the roots of accepting runs. Thus, $T(A)$ is

non-empty precisely when some initial state is in $accept(A)$. The claim follows, since $accept(A)$ can be computed bottom-up in polynomial time. □

A.3 Monadic Programs

A.3.1 Proof of Theorem 4.1.7

First of all, as said in the Proposition 2.5.1, we define a total mapping that maps every finite tree to a polynomial $P \in \mathbb{N}[X]$, so that the well-ordering $\leq$ defined on $\mathbb{N}[X]$ induces a well-ordering on equivalence classes of finite trees, which will ultimately allow us to run a proof by induction.

Definition A.3.1.
A finite tree τ being given, we define inductively the *order* ω of its nodes by setting

- $\omega(\rho) = 0$, where ρ is the root of τ.
- if n_1 is a child of a node n_2 in τ, then $\omega(n_1) = \omega(n_2) + 1$.

We then define the *weight* of the tree τ relatively to a finite set $\mathcal{S}$ of predicates as being the polynomial with non-negative integer coefficients

$$W_{\mathcal{S}}(\tau)(X) = \sum_{node\ n \in \tau} W_{\mathcal{S}}(n) X^{2\omega(n)} + W_{\overline{\mathcal{S}}}(n) X^{2\omega(n)+1}$$

where $W_{\mathcal{S}}(n)$ (resp. $W_{\overline{\mathcal{S}}}(n)$) is the number of occurrences of predicates that appear in the body the rule ρ of n and that are (resp. are not) elements of $\mathcal{S}$.

Eventually, we denote $\tau_1 >_{\mathcal{S}} \tau_2$ the inequality $W_{\mathcal{S}}(\tau_1) > W_{\mathcal{S}}(\tau_2)$. Then, every set of trees $E \neq \emptyset$ has a $\mathcal{S}$-minimal element, that is to say an element $\tau \in E$ whose weight is minimal in $W_{\mathcal{S}}(E)$. ∎

Indeed, this well-ordering is sufficient to let us prove the proposition below, which consists in the first implication of the equivalence stated by Theorem 4.1.7:

Proposition A.3.2.
Let be Π *a* Datalog *program with goal predicate* Q *and* $\Pi_{\mathcal{M}}$ *a monadic* Datalog *program with goal predicate* $Q_{\mathcal{M}}$. $\forall \tau^{dec} = (\tau, \mathcal{L}, \varphi) \in u\_dec^{fp}(\Pi_{\mathcal{M}}, Q, \Pi)$*, if some mapping* $\sigma \xrightarrow{h} \tau$ *from a tree* $\sigma \in u\_trees(Q_{\mathcal{M}}, \Pi_{\mathcal{M}})$ *to* τ *exists, then* $\exists \sigma^* \in u\_trees(Q_{\mathcal{M}}, \Pi^*_{\mathcal{M}})$ *whose root is a leaf and such that some decorating containment mapping* $\sigma^* \xrightarrow{h^*} \tau^{dec}$ *from* σ^* *to* τ^{dec} *exists.* ∎

Proof. Let T be the set of unfolding expansion trees ν of $\Pi^*_{\mathcal{M}}$ such that some decorating containment mapping $\nu \xrightarrow{\theta} \tau^{dec}$ exists. The above defined $\sigma \in T$, which therefore is not empty. Indeed, σ also is an unfolding expansion tree of $\Pi^*_{\mathcal{M}}$, and every (traditional) containment mapping h from σ to τ can be extended to a decorating containment mapping

h^* from σ to τ^{dec}, for instance by binding all variables of σ that do not belong to the domain of h to some variable X of τ.

Let now $\mathcal{S} = \{R^* | R \text{ is an internal IDB predicate of } \Pi_\mathcal{M}\}$ be the set of the derived EDB predicates in $\Pi^*_\mathcal{M}$, i.e. appearing in $\Pi^*_\mathcal{M}$ but not in $\Pi_\mathcal{M}$ itself. Let be $\mu \in T$, and a decorating containment mapping $h : \mu \to \tau^{dec}$. Let us look at a leaf n of μ : its rule ρ has an IBD predicate $R(\mathbf{t})$ in its head, and only EDB predicates in its body.

If n has a father $\tilde{n}$ with rule $\tilde{\rho}$, then ρ must be an internal rule of $\Pi^*_\mathcal{M}$. $\tau^{dec} \in u\_dec^{fp}(\Pi_1, Q, \Pi)$, so that $(h(\mathbf{t}), R) \in \varphi$. Therefore, by erasing n and replacing the occurrence $R(\mathbf{t})$ in $\tilde{\rho}$ by the EDB predicate $R^*(\mathbf{t})$, we get a tree $\tilde{\mu} <_\mathcal{S} \mu$ such that h is also a decorative containment mapping from $\tilde{\mu}$ do τ^{dec}: $\tilde{\mu} \in T$, and μ is not a $\mathcal{S}$-minimal element of T.

That is why every $\mathcal{S}$-minimal element of T has its root as only node. Such an element exists, which completes our proof. □

We have to prove now that, conversely, the existence of decorative containment mappings implies the existence of containment mappings. And it appears that this converse property is true on *least fixpoints*, defined as follows:

Definition A.3.3.
Let Π be a DATALOG program with goal predicate Q, and $\Pi_\mathcal{M}$ a monadic DATALOG program. Let now be $\tau^{dec} = (\tau, \mathcal{L}, \varphi) \in u\_dec^{fp}(\Pi_\mathcal{M}, Q, \Pi)$. We define inductively sets Ω_i of couples $(X, l) \in Var(\tau^{dec}) \times \mathcal{L}$ such that $l \in \varphi(X)$ by setting

- $\Omega_0 = \emptyset$
- $\forall i \geq 0$, $(X, l) \in \Omega_{i+1}$ if and only if either
  - $(X, l) \in \Omega_i$
  - for some internal rule $\mathcal{R}$ in $\Pi_\mathcal{M}$:

$$R_1(\mathbf{t}^1) \wedge \ldots \wedge R_m(\mathbf{t}^m) \Rightarrow R(\mathbf{t})$$

with IDB atoms $R(\mathbf{t}), R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_l}(\mathbf{t}^{i_l})$, EDB atoms $R_{j_1}(\mathbf{t}^{i_1}), \ldots, R_{j_{l'}}(\mathbf{t}^{j_{l'}})$, there exists a total mapping $h : var(\mathcal{R}) \to var(\tau)$ such that $R_{j_1}(h(\mathbf{t}^{j_1})), \ldots, R_{j_{l'}}(h(\mathbf{t}^{j_{l'}}))$ hold and that $\{(h(\mathbf{t}_1), R_{i_1}), \ldots, (h(\mathbf{t}_{i_l}), R_{i_l})\} \subseteq \Omega_i$.

If $\varphi^{min} = \cup_{i \geq 0} \Omega_i$, we define the *least fixpoint decorated unfolding tree* relative to τ^{dec} to be $\tau^{min} = (\tau, \mathcal{L}, \varphi^{min})$. We also define the *degree* d° of a couple $(X, l) \in \mathcal{L}^{min}$ to be $d^\circ(X, l) = \min\{i \geq 0 | (X, l) \in \Omega_i\}$. Finally, we denote by $u\_dec^{lpf}(\Pi_\mathcal{M}, Q, \Pi)$ the set of all least fixpoint decorated unfolding trees relative to any tree $\tau^{dec} \in u\_dec^{fp}(\Pi_\mathcal{M}, Q, \Pi)$.
■

Thanks to the algorithm of construction of least fixpoint decorated unfolding trees this definition consists in, the following three propositions are now straightforward:

Proposition A.3.4.
Let be Π *a* DATALOG *program with goal predicate* Q, $\Pi_\mathcal{M}$ *a monadic* DATALOG *program and* $\tau \in u\_trees(Q, \Pi)$ *an unfolding tree.*

If $\tau^{dec} = (\tau, \mathcal{L}, \varphi) \in u\_dec^{fp}(\Pi_{\mathcal{M}}, Q, \Pi)$ is a fixpoint decorated unfolding tree, the least fixpoint decorated unfolding tree $\tau^{min} = (\tau, \mathcal{L}, \varphi^{min})$ relative to τ^{dec} is also a fixpoint decorated unfolding tree, and it does not depend from the choice of φ. ∎

Corollary A.3.5. *Let be Π a* Datalog *program with goal predicate Q, $\Pi_{\mathcal{M}}$ a monadic* Datalog *program and $\tau \in u\_trees(Q, \Pi)$ an unfolding tree.*

If $\tau^{dec} = (\tau, \mathcal{L}, \varphi) \in u\_dec^{fp}(\Pi_{\mathcal{M}}, Q, \Pi)$ is a fixpoint decorated unfolding tree and $\tau^{min} = (\tau, \mathcal{L}^{min}, \varphi^{min})$ is a least fixpoint decorated unfolding tree, then $\varphi^{min} \subseteq \varphi$. ∎

Lemma A.3.6.
Let be Π a Datalog *program with goal predicate Q and $\Pi_{\mathcal{M}}$ a monadic* Datalog *program. $\forall \tau \in u\_trees(Q, \Pi), \exists \tau^{dec} = (\tau, \mathcal{L}, \varphi) \in u\_dec^{lfp}(\Pi_{\mathcal{M}}, Q, \Pi)$.* ∎

Moreover, in addition to the concept of least fixpoint decorated unfolding tree, in Definition A.3.3, we also defined the *degree* of the couples $(X, l) \in \varphi^{min}$. This notion of degree allows us to introduce a specific comparison relation on trees.

Definition A.3.7.
Let Π be a Datalog program with goal predicate Q, $\Pi_{\mathcal{M}}$ a monadic Datalog program $\tau^{dec} = (\tau, \mathcal{L}, \varphi) \in u\_dec^{lpf}(\Pi_{\mathcal{M}}, Q, \Pi)$. We denote by $\mathcal{C}_{\tau^{dec}}$ the set of couples (σ, h) where $\sigma \in u\_trees(Q_{\mathcal{M}}, \Pi_{\mathcal{M}})$ and $h : \sigma \to \tau^{dec}$ is a decorating containment mapping. We define the *depth* of a couple $(\sigma, h) \in \mathcal{C}_{\tau^{dec}}$ to be the polynomial in $\mathbb{N}[X]$

$$\mathcal{D}_{\tau^{dec}}(\sigma, h)(X) = \sum_{(V,l) \in \Omega_\infty} \mathcal{D}(V, l) X^{d^\circ(V,l)}$$

where $\mathcal{D}(V, l)$ is the number of occurrences of derived predicates $l^*(\mathbf{t})$ that appear in σ and such that $h(\mathbf{t}) = V$.

We finally denote $(\sigma_1, h_1) >_{\tau^{dec}} (\sigma_2, h_2)$ the inequality $\mathcal{D}_{\tau^{dec}}(\sigma_1, h_1) > \mathcal{D}_{\tau^{dec}}(\sigma_2, h_2)$. Every non-empty subset $E \subseteq \mathcal{C}_{\tau^{dec}}$ has a τ^{dec}-minimal element, that is to say an element $(\sigma, h) \in E$ whose depth is minimal in $\mathcal{D}_{\tau^{dec}}(E)$. ∎

With the help of this new induced well-ordering, we can prove by induction the following proposition, which is complementary to the Proposition A.3.2:

Proposition A.3.8.
Let be a Datalog *program Π with goal predicate Q and a monadic* Datalog *program $\Pi_{\mathcal{M}}$ with goal predicate $Q_{\mathcal{M}}$. For every least fixpoint decorated tree $\forall \tau^{dec} = (\tau, \mathcal{L}, \varphi) \in u\_dec^{lpf}(\Pi_{\mathcal{M}}, Q, \Pi)$ and $\forall \sigma^* \in u\_trees(Q_{\mathcal{M}}, \Pi^*_{\mathcal{M}})$ whose root is a leaf, if some decorating containment mapping $\sigma^* \xrightarrow{h^*} \tau^{dec}$ exists, then $\exists \sigma \in u\_trees(Q_{\mathcal{M}}, \Pi_{\mathcal{M}})$ such that a containment mapping $\sigma \xrightarrow{h} \tau$ exists.* ∎

Proof. Let be $\mathcal{C}_{\tau^{dec}}$ be the set of couples (σ, h) where $\sigma \in u\_trees(Q_{\mathcal{M}}, \Pi^*_{\mathcal{M}})$ and $h : \sigma \to \tau^{dec}$ is a decorating containment mapping. By hypothesis, $\mathcal{C}_{\tau^{dec}} \neq \emptyset$. Let $(\sigma, h) \in \mathcal{C}_{\tau^{dec}}$,

and $d_{max} = \deg(\mathcal{D}_{\tau^{dec}}(\sigma, h))$. If $d_{max} \neq 0$, let n be a node in σ such that some derived EDB predicate $R^*(\mathbf{t})$ in the rule ρ of n verifies $d^\circ(h(\mathbf{t}), R) = d_{max}$; let $\mathcal{R}$ be a rule in $\Pi_\mathcal{M}$:

$$R_1(\mathbf{v}^1) \wedge \ldots \wedge R_m(\mathbf{v}^m) \Rightarrow R(\mathbf{v})$$

with IDB atoms $R(\mathbf{v}), R_{i_1}(\mathbf{v}^{i_1}), \ldots, R_{i_l}(\mathbf{v}^{i_l})$ and EDB atoms $R_{j_1}(\mathbf{v}^{j_1}), \ldots, R_{j_{l'}}(\mathbf{v}^{j_{l'}})$, and a total mapping $g : var(\mathcal{R}) \to var(\tau)$ such that $\{(g(\mathbf{v}^{i_1}), R_{i_1}), \ldots, (g(\mathbf{v}^{i_l}), R_{i_l})\} \subseteq \Omega_{d_{max}-1}$ and $R_{j_1}(g(\mathbf{v}^{j_1})), \ldots, R_{j_{l'}}(g(\mathbf{v}^{j_{l'}}))$ hold.

Without loss of generality, we can assume that $var(\mathcal{R}) \cap var(\sigma) = \emptyset$. In order to merge the predicate $R(\mathbf{v})$ in $\mathcal{R}$ with the predicate $R^*(\mathbf{t})$ in ρ, we introduce a total mapping

$$\begin{array}{rlcl} \psi : & var(\mathcal{R}) & \to & var(\mathcal{R}) \cup \{\mathbf{t}\} \\ & \mathbf{v} & \to & \mathbf{t} \\ & x & \to & x \text{ if } x \neq \mathbf{v} \end{array}$$

We can now build a new tree $\tilde{\sigma} \in u\_trees(Q_\mathcal{M}, \Pi^*_\mathcal{M})$ by replacing every occurrence of $R^*(\mathbf{t})$ in the rule ρ of n by an occurrence of $R(\mathbf{t})$, and by adding a child to n, this child being a leaf $\tilde{n}$ labelled by $(R(\mathbf{t}), R_1(\psi(\mathbf{v}^1)) \wedge \ldots \wedge R_m(\psi(\mathbf{v}^m)) \Rightarrow R(\mathbf{t}))$. Then, we also build the mapping

$$\begin{array}{rlcl} \tilde{h} : & var(\tilde{\sigma}) & \to & var(\tau) \\ & x & \to & h(x) \text{ if } x \in var(\sigma) \\ & x & \to & g(x) \text{ if } x \notin var(\sigma) \end{array}$$

This gives us a new couple $(\tilde{\sigma}, \tilde{h}) \in \mathcal{C}_{\tau^{dec}}$, such that $(\tilde{\sigma}, \tilde{h}) <_{\tau^{dec}} (\sigma, h)$.

Let now $(\sigma, h) \in \mathcal{C}_{\tau^{dec}}$ be a τ^{dec}-minimal element of $\mathcal{C}_{\tau^{dec}}$. The above analysis tells us that $\mathcal{D}_{\tau^{dec}}(\sigma, h)$ is constant, so that every occurrence of a derived predicate $R^*(\mathbf{t})$ in σ verifies $(h(\mathbf{t}), R) \in \Omega_0 = \emptyset$. Therefore, σ contains no derived predicate, and also is an unfolded expansion tree of $\Pi_\mathcal{M}$. In addition, h induces a (normal) containment mapping $\sigma \xrightarrow{h} \tau$, which ends the proof. □

Propositions A.3.2 and A.3.8 are now sufficient to prove Theorem 4.1.7:

Theorem A.3.9. [Theorem 4.1.7]
Let Π *be a* Datalog *program with goal predicate* Q, $\Pi_\mathcal{M}$ *a monadic* Datalog *program with goal predicate* $Q_\mathcal{M}$. Π *is contained in* $\Pi_\mathcal{M}$ *if and only if for all* $\tau^{dec} = (\tau, \mathcal{L}, \varphi)$ *in* $u\_dec^{pf}(\Pi_\mathcal{M}, Q, \Pi)$, *there exists a tree* $\sigma^\bullet$ *in* $u\_trees(Q_\mathcal{M}, \Pi^\bullet_\mathcal{M})$ *whose root is a leaf, and a decorating containment mapping* $\sigma^\bullet \xrightarrow{h^\bullet} \tau^{dec}$. ∎

Proof. If Π is contained in $\Pi_\mathcal{M}$, let be $\tau^{dec} = (\tau, \mathcal{L}, \varphi) \in u\_dec^{fp}(\Pi_\mathcal{M}, Q, \Pi)$. By Proposition 3.1.3, $\exists \sigma \in u\_trees(Q_\mathcal{M}, \Pi_\mathcal{M})$ and a containment mapping $h : \sigma \to \tau$. Then, by Proposition A.3.2, $\exists \sigma^* \in u\_trees(Q_\mathcal{M}, \Pi^*_\mathcal{M})$ whose root is a leaf and a decorating containment mapping $\sigma^* \xrightarrow{h^*} \tau^{dec}$.

Conversely, let τ be some tree in $u\_trees(Q, \Pi)$. By Lemma 4.1.4, $\exists \tau^{dec} = (\tau, \mathcal{L}, \varphi) \in u\_dec^{lfp}(\Pi_{\mathcal{M}}, Q, \Pi)$. Then, $\exists \sigma^* \in u\_trees(Q_{\mathcal{M}}, \Pi^*_{\mathcal{M}})$ whose root is a leaf and a decorating containment mapping $\sigma^* \xrightarrow{h^*} \tau^{dec}$. By Proposition A.3.8, $\exists \sigma \in u\_trees(Q_{\mathcal{M}}, \Pi_{\mathcal{M}})$ and a containment mapping $h : \sigma \to \tau$. This is true $\forall \tau \in u\_trees(Q, \Pi)$; therefore, by Proposition 3.1.3, Π must be contained in $\Pi_{\mathcal{M}}$. □

A.3.2 From Unfolding to Proof Trees

Proposition A.3.10. [Proposition 4.2.5]
Let Π be a Datalog *program with goal predicate Q, $\mathcal{L}$ a finite set of IDB predicates, $\mathcal{R}$ a monadic rule whose IDB predicates are elements of $\mathcal{L}$, and $\tau \in p\_dec(\mathcal{L}, Q, \Pi)$. Then, $\tau \in p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ if and only if $\mathcal{U}(\tau) \notin u\_dec^{pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$.* ∎

Proof. If $\tau \in p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$, let $(\sigma, \mathcal{L}, \varphi) = \mathcal{U}(\tau) \in u\_dec(\mathcal{L}, Q, \Pi)$. The above defined infixpoint certificate from $\mathcal{R}$ to τ induces a total mapping $h : var(\mathcal{R}) \to var(\sigma)$ such that $R_{j_1}(h(\mathbf{t}^{j_1})), \ldots, R_{j_{l'}}(h(\mathbf{t}^{j_{l'}}))$ hold and $\{(h(\mathbf{t}^{i_1}), R_{i_1}), \ldots, (h(\mathbf{t}^{i_l}), R_{i_l})\} \subseteq \varphi$, but $(h(\mathbf{t}), R) \notin \varphi$. Therefore, $\mathcal{U}(\tau) \notin u\_dec^{pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$.

Conversely, if $\mathcal{U}(\tau) \notin u\_dec^{pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$, since $\mathcal{L} \supseteq \{R_{i_1}, \ldots, R_{i_l}\}$, there exists a mapping $h : var(\mathcal{R}) \to var(\sigma)$ such that $R_{j_1}(h(\mathbf{t}^{j_1})), \ldots, R_{j_{l'}}(h(\mathbf{t}^{j_{l'}}))$ hold and that $\{(h(\mathbf{t}^{i_1}), R_{i_1}), \ldots, (h(\mathbf{t}^{i_l}), R_{i_l})\} \subseteq \varphi$, but $(h(\mathbf{t}), R) \notin \varphi$. h induces an infixpoint certificate from $\mathcal{R}$ to τ, which proves that $\tau \in p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$. □

Proposition A.3.11. [Proposition 4.2.7]
Let Π be a Datalog *program with goal predicate Q, $\Pi_{\mathcal{M}}$ a monadic* Datalog *program with goal predicate $Q_{\mathcal{M}}$, $\mathcal{L}$ the set of internal IDB predicates of $\Pi_{\mathcal{M}}$, and $\tau \in p\_dec(\mathcal{L}, Q, \Pi)$. Then, $\exists \sigma^* \in u\_trees(Q_{\mathcal{M}}, \Pi^*_{\mathcal{M}})$ whose root is a leaf and a decorating containment mapping $\sigma^* \xrightarrow{h^*} \mathcal{U}(\tau)$ if and only if $\tau \in p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ for some goal rule $\mathcal{R}$ of $\Pi_{\mathcal{M}}$.* ∎

Proof. If $\exists \sigma^* \in u\_trees(Q_{\mathcal{M}}, \Pi^*_{\mathcal{M}})$ whose root is a leaf and a decorating containment mapping $\sigma^* \xrightarrow{h^*} \mathcal{U}(\tau)$, let $\mathcal{R}^*$ be the rule of $\Pi^*_{\mathcal{M}}$ in its body, and let $\mathcal{R}$ be the rule of $\Pi_{\mathcal{M}}$ from which $\mathcal{R}^*$ is derived. $\mathcal{R}$ is a goal rule of $\Pi_{\mathcal{M}}$ and the decorating containment mapping $\sigma^* \xrightarrow{h^*} \mathcal{U}(\tau)$ induces a reaching goal certificate from $\mathcal{R}$ to τ. Therefore, $\tau \in p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$.

If $\tau \in p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ for some goal rule $\mathcal{R}$ of $\Pi_{\mathcal{M}}$, let $\mathcal{R}^*$ be the rule of $\Pi^*_{\mathcal{M}}$ that is derived from $\mathcal{R}$ and that has no IDB predicate in its body. Finally, let $\nu \in u\_trees(Q_{\mathcal{M}}, \Pi^*_{\mathcal{M}})$ be the tree whose root is the leaf $r = (Q_{\mathcal{M}}(\mathbf{v}), \mathcal{R}^*)$. Then, the above defined reaching goal certificate from $\mathcal{R}$ to τ induces a decorating containment mapping from ν to $\mathcal{U}(\tau)$. □

Theorem A.3.12. [Theorem 4.2.8]
Let Π be a Datalog *program with goal predicate Q, $\Pi_{\mathcal{M}}$ a monadic* Datalog *program with goal predicate $Q_{\mathcal{M}}$, and let $\mathcal{L}$ be the set of the internal IDB predicates of $\Pi_{\mathcal{M}}$. Π is contained in $\Pi_{\mathcal{M}}$ if and only if every tree $\tau \in p\_dec(\mathcal{L}, Q, \Pi)$ satisfies one of the two following sentences:*

- $\tau \in p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ *for some internal rule* $\mathcal{R}$ *of* $\Pi_{\mathcal{M}}$.
- $\tau \in p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ *for some goal rule* $\mathcal{R}$ *of* $\Pi_{\mathcal{M}}$.

■

Proof. If Π is contained in $\Pi_{\mathcal{M}}$, let be $\tau \in p\_dec(\mathcal{L}, Q, \Pi)$. If, for every internal rule $\mathcal{R}$ of $\Pi_{\mathcal{M}}$, $\tau \notin p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$, then $\mathcal{U}(\tau) \in u\_dec^{pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ for every such rule, and $\mathcal{U}(\tau) \in u\_dec^{pf}(\Pi_{\mathcal{M}}, Q, \Pi)$. Π is contained in $\Pi_{\mathcal{M}}$, so that $\exists \sigma^* \in u\_trees(Q_{\mathcal{M}}, \Pi^*_{\mathcal{M}})$ whose root is a leaf and a decorating containment mapping $\sigma^* \xrightarrow{h^*} \mathcal{U}(\tau)$, and then $\tau \in p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ for some goal rule $\mathcal{R}$ of $\Pi_{\mathcal{M}}$.

Let us now assume that every tree $\tau \in p\_dec(\mathcal{L}, Q, \Pi)$ satisfies one of the two following sentences:

- $\tau \in p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ for some internal rule $\mathcal{R}$ of $\Pi_{\mathcal{M}}$.
- $\tau \in p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ for some goal rule $\mathcal{R}$ of $\Pi_{\mathcal{M}}$.

$\forall (\sigma, \mathcal{L}, \varphi) \in u\_dec^{pf}(\Pi_{\mathcal{M}}, Q, \Pi)$, $\exists \tau \in p\_dec(\mathcal{L}, \Pi_{\mathcal{M}}, Q, \Pi)$ such that $\mathcal{U}(\tau) = (\sigma, \mathcal{L}, \varphi)$. Then, for every internal rule $\mathcal{R}$ of $\Pi_{\mathcal{M}}$, $\mathcal{U}(\tau) \in u\_dec^{pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$, which shows that $\tau \notin p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$. That is why $\tau \in p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$ for some goal rule $\mathcal{R}$ of $\Pi_{\mathcal{M}}$. Then, $\exists \sigma^* \in u\_trees(Q_{\mathcal{M}}, \Pi^*_{\mathcal{M}})$ whose root is a leaf and a decorating containment mapping $\sigma^* \xrightarrow{h^*} \mathcal{U}(\tau)$. So, by Theorem 4.1.7, Π is contained in $\Pi_{\mathcal{M}}$. □

A.3.3 Tree Automata

Proposition A.3.13. [Proposition 4.3.1]
Let Π *be a* DATALOG *program with a monadic goal predicate* Q *and* $\mathcal{L}$ *a finite set. There is an automaton* $A^{p\_dec}_{\mathcal{L},Q,\Pi}$*, whose size is exponential in the size of* Π *and* $\mathcal{L}$*, such that* $T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) = p\_dec(\mathcal{L}, Q, \Pi)$. ■

Proof. We describe the construction of the automaton

$$A^{p\_dec}_{\mathcal{L},Q,\Pi} = (\Sigma, \mathcal{S} \cup \{accept\}, \mathcal{S}_Q, \delta, \{accept\})$$

The state set $\mathcal{S}$ and the alphabet Σ both are the set of quadruples $(R(\mathbf{t}), \rho, \mathcal{L}, \varphi)$ where $R(\mathbf{t})$ is an IDB atom with variables among $var(\Pi)$, ρ is an instance of a rule of Π whose set of variables $var(\rho) \subseteq var(\Pi)$ and whose head atom is $R(\mathbf{t})$, and $\varphi \in 2^{\mathcal{L} \times var(\Pi)}$ verifies $\varphi(var(\Pi) \backslash var(\rho)) = \{\emptyset\}$. The start-state set $\mathcal{S}_Q$ is the set of all triples $(Q(\mathbf{t}), \rho, \mathcal{L}, \varphi) \in \mathcal{S}$. The transition function δ is constructed as follows:

- Let ρ be a rule instance

$$R_1(\mathbf{t}^1) \wedge \ldots \wedge R_m(\mathbf{t}^m) \Rightarrow R(\mathbf{t})$$

in Π, with IDB atoms $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_l}(\mathbf{t}^{i_l})$ in its body. Let be $s = (R(\mathbf{t}), \rho, \mathcal{L}, \varphi)$, $s_1 = (R_{i_1}(\mathbf{t}_{i_1}), \rho_1, \mathcal{L}, \varphi_1), \ldots, s_l = (R_{i_l}(\mathbf{t}_{i_l}), \rho_l, \mathcal{L}, \varphi_l) \in \mathcal{S}$. If $\varphi(\mathbf{t}_{i_1}) = \varphi_1(\mathbf{t}_{i_1}), \ldots$ and $\varphi(\mathbf{t}_{i_l}) = \varphi_l(\mathbf{t}_{i_l})$, then $\langle s_1, \ldots, s_l \rangle \in \delta(s, s)$.

- Let ρ be a rule instance

$$R_1(\mathbf{t}^1) \wedge \ldots \wedge R_m(\mathbf{t}^m) \Rightarrow R(\mathbf{t})$$

with only EDB atoms in its body. Let be $s = (R(\mathbf{t}), \rho, \mathcal{L}, \varphi) \in \mathcal{S}$. Then $\langle accept \rangle \in \delta(s, s)$.

By using technical arguments similar to those developed in [9], it follows that

$$T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) = \{\tau \in p\_dec(\mathcal{L}, Q, \Pi)\}$$

It is easy to see that the number of states and transitions in the automaton is exponential in the size of Π and $\mathcal{L}$. □

Proposition A.3.14. [Proposition 4.3.2]
Let Π be a DATALOG *program with a monadic goal predicate Q, $\mathcal{L}$ be a finite set of IBD predicates, and $\mathcal{R}$ be a monadic rule whose IBD predicates are elements of $\mathcal{L}$. Then there is an automaton $A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R},Q,\Pi}$, whose size is exponential in the size of Π, $\mathcal{L}$ and $\mathcal{R}$, such that $T(A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R},Q,\Pi}) \cap T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) = p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)$.* ∎

Proof. We describe the construction of the automaton $A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R},Q,\Pi}$. If $\mathcal{R}$ is a DATALOG rule $R_1(\mathbf{t}^1) \wedge \ldots \wedge R_m(\mathbf{t}^m) \Rightarrow R_0(\mathbf{t}^0)$ whose body contains IDB atoms $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_l}(\mathbf{t}^{i_l})$ and EDB atoms $R_{j_1}(\mathbf{t}^{j_1}), \ldots, R_{j_{l'}}(\mathbf{t}^{j_{l'}})$, we introduce the (possibly empty) sets $\theta_E = \{R_{j_1}(\mathbf{t}^{j_1}), \ldots, R_{j_{l'}}(\mathbf{t}^{j_{l'}})\}$, $\theta_I = \{R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_l}(\mathbf{t}^{i_l})\}$, $\theta_H = \{R_0(\mathbf{t}^0)\}$.

The automaton $A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R},Q,\Pi}$ is $(\Sigma, \mathcal{S} \cup \{accept\}, \mathcal{S}_Q, \delta, \{accept\})$. The the alphabet Σ is as in the proof of Proposition 4.3.1. The state set $\mathcal{S}$ is the set $\Sigma \times 2^{\theta_E} \times 2^{\theta_I} \times 2^{\theta_H} \times (var(\Pi) \cup \{\emptyset\})^{var(\mathcal{R})}$. The second (resp. third, fourth) component in $\mathcal{S}$ represents the collection of subsets of θ_E (resp. θ_I, θ_H) and the final component is the set of total mappings from $var(\mathcal{R})$ to $var(\Pi) \cup \{\emptyset\}$, which also can be seen as the set of partial mappings from $var(\mathcal{R})$ to $var(\Pi)$. The start-state set $\mathcal{S}_Q$ is the set of all tuples $((Q(\mathbf{t}), \rho, \mathcal{L}, \varphi), \theta_E, \theta_I, \theta_H, M_\emptyset)$ where $((Q(\mathbf{t}), \rho, \mathcal{L}, \varphi) \in \Sigma$ and $M_\emptyset(var(\mathcal{R})) = \{\emptyset\}$. The transition function is constructed as follows:

- Let $\mathcal{P}$ be a rule instance

$$P_1(\mathbf{t}^1) \wedge \ldots \wedge P_m(\mathbf{t}^m) \Rightarrow P(\mathbf{t})$$

in Π, with IDB atoms $P_{i_1}(\mathbf{t}^{i_1}), \ldots, P_{i_l}(\mathbf{t}^{i_l})$ in its body. Let be $s = ((P(\mathbf{t}), \rho, \mathcal{L}, \varphi), \beta_E, \beta_I, \beta_H, M), s_1 = ((P_{i_1}(\mathbf{t}_{i_1}), \rho_1, \mathcal{L}, \varphi_1), \beta^1_E, \beta^1_I, \beta^1_H, M'), \ldots, s_l = ((P_{i_l}(\mathbf{t}_{i_l}), \rho_l, \mathcal{L}, \varphi_l), \beta^l_E, \beta^l_I, \beta^l_H, M') \in \mathcal{S}$. Then,

$$\langle s_{i_1}, \ldots, s_{i_l} \rangle \in \delta(s, (P(\mathbf{t}), \rho, \mathcal{L}, \varphi))$$

if some sets $\beta'_E \in 2^{\theta_E}, \beta'_I \in 2^{\theta_I}, \beta'_H \in 2^{\theta_H}$ satisfy the following:

1. $\forall a \in \{E, I, H\}, \beta'_a, \beta^1_a, \ldots, \beta^l_a$ is a partition of β_a.
2. $\forall v \in var(\mathcal{R}), M(v) \neq \emptyset \Rightarrow M'(v) = M(v)$.
3. If two sets β^j_a and β^k_b (where $a, b \in \{E, I, H\}$) share a variable v, $M'(v) \neq \emptyset$.

4. If a variable v occurs in some β_a^j (where $a \in \{E, I, H\}$) and $M'(v) \neq \emptyset$, $M'(v)$ is in $\mathbf{t}^{i_j}$.
5. For every instance of EDB atom $R(\mathbf{x}_1, \ldots, \mathbf{x}_k) \in \beta'_E$, $\emptyset \notin M'(\{\mathbf{x}_1, \ldots, \mathbf{x}_k\})$ and $R(M'(\mathbf{x}_1), \ldots, M'(\mathbf{x}_k))$ is an EDB atom of $\mathcal{P}$.
6. For every instance of IDB atom $R(\mathbf{t}) \in \beta'_I$, $M'(\mathbf{t}) \neq \emptyset$ and $R \in \varphi(M'(\mathbf{t}))$.
7. For every instance of IDB atom $R(\mathbf{t}) \in \beta'_H$, $M'(\mathbf{t}) \neq \emptyset$ and $R \notin \varphi(M'(\mathbf{t}))$.

- Let $\mathcal{P}$ be a rule instance

$$P_1(\mathbf{t}^1) \wedge \ldots \wedge P_m(\mathbf{t}^m) \Rightarrow P(\mathbf{t})$$

in Π, where all atoms in the body of the rule are EDB atoms. Let $s = ((P(\mathbf{t}), \rho, \mathcal{L}, \varphi), \beta_E, \beta_I, \beta_H, M) \in \mathcal{S}$. Then, $\langle accept \rangle \in \delta(s, (P(\mathbf{t}), \rho, \mathcal{L}, \varphi))$ if the following holds:

1. For every instance of EDB atom $R(\mathbf{x}_1, \ldots, \mathbf{x}_k) \in \beta_E$, $\emptyset \notin M(\{\mathbf{x}_1, \ldots, \mathbf{x}_k\})$ and $R(M(\mathbf{x}_1), \ldots, M(\mathbf{x}_k))$ is an EDB atom of $\mathcal{P}$.
2. For every instance of IDB atom $R(\mathbf{t}) \in \beta_I$, $M(\mathbf{t}) \neq \emptyset$ and $R \in \varphi(M(\mathbf{t}))$.
3. For every instance of IDB atom $R(\mathbf{t}) \in \beta_H$, $M(\mathbf{t}) \neq \emptyset$ and $R \notin \varphi(M(\mathbf{t}))$.

By using technical arguments similar to those developed in [9], it follows that

$$\{\tau \in p\_dec(\mathcal{L}, Q, \Pi) \cap T(A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R},Q,\Pi})\} = \{\tau \in \tau \in p\_dec(\mathcal{L}, Q, \Pi) \cap p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}, Q, \Pi)\}$$

It is easy to see that the number of states and transition in the automaton is exponential in the size of Π, $\mathcal{L}$ and $\mathcal{R}$. □

Proposition A.3.15. [Proposition 4.3.3]
Let Π be a DATALOG *program with a monadic goal predicate Q, $\mathcal{L}$ be a finite set of IBD predicates, and $\mathcal{R}$ be a rule such that are IBD predicates in its body are elements of $\mathcal{L}$. Then there is an automaton $A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{R},Q,\Pi}$, whose size is exponential in the size of Π, $\mathcal{L}$ and $\mathcal{R}$, such that $T(A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{R},Q,\Pi}) \cap T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) = p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)$.* ∎

Proof. We describe the construction of the automaton $A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{R},Q,\Pi}$. If $\mathcal{R}$ is a DATALOG rule $R_1(\mathbf{t}^1) \wedge \ldots \wedge R_m(\mathbf{t}^m) \Rightarrow R_0(\mathbf{t}^0)$ whose body contains IDB atoms $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_l}(\mathbf{t}^{i_l})$ and EDB atoms $R_{j_1}(\mathbf{t}^{j_1}), \ldots, R_{j_{l'}}(\mathbf{t}^{j_{l'}})$, we introduce the (possibly empty) sets $\theta_E = \{R_{j_1}(\mathbf{t}^{j_1}), \ldots, R_{j_{l'}}(\mathbf{t}^{j_{l'}})\}$, $\theta_I = \{R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_l}(\mathbf{t}^{i_l})\}$.

The automaton $A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{R},Q,\Pi}$ is $(\Sigma, \mathcal{S} \cup \{accept\}, \mathcal{S}_Q, \delta, \{accept\})$. The the alphabet Σ is as in the proof of Proposition 4.3.1. The state set $\mathcal{S}$ is the set $\Sigma \times 2^{\theta_E} \times 2^{\theta_I} \times (var(\Pi) \cup \{\emptyset\})^{var(\mathcal{R})}$. The second (resp. third) component in $\mathcal{S}$ represents the collection of subsets of θ_E (resp. θ_I) and the final component is the set of total mappings from $var(\mathcal{R})$ to $var(\Pi) \cup \{\emptyset\}$, which also can be seen as the set of partial mappings from $var(\mathcal{R})$ to $var(\Pi)$. The start-state set $\mathcal{S}_Q$ is the set of all tuples $((Q(\mathbf{u}), \rho, \mathcal{L}, \varphi), \theta_E, \theta_I, M)$ where $((Q(\mathbf{u}), \rho, \mathcal{L}, \varphi) \in \Sigma$ and every variable v in $\mathbf{t}^0$ is such that $M(v)$ is a variable in $\mathbf{u}$. The transition function is constructed as follows:

- Let $\mathcal{P}$ be a rule instance

$$P_1(\mathbf{t}^1) \wedge \ldots \wedge P_m(\mathbf{t}^m) \Rightarrow P(\mathbf{t})$$

in Π, with IDB atoms $P_{i_1}(\mathbf{t}^{i_1}), \ldots, P_{i_l}(\mathbf{t}^{i_l})$ in its body. Let $s_1 = ((P_{i_1}(\mathbf{t}_{i_1}), \rho_1, \mathcal{L}, \varphi_1), \beta_E^1, \beta_I^1, M'), \ldots, s_l = ((P_{i_l}(\mathbf{t}_{i_l}), \rho_l, \mathcal{L}, \varphi_l), \beta_E^l, \beta_I^l, M')$ and $s = ((P(\mathbf{t}), \rho, \mathcal{L}, \varphi), \beta_E, \beta_I, M) \in \mathcal{S}$. Then,

$$\langle s_{i_1}, \ldots, s_{i_l} \rangle \in \delta(s, (P(\mathbf{t}), \rho, \mathcal{L}, \varphi))$$

if some sets $\beta'_E \in 2^{\theta_E}, \beta'_I \in 2^{\theta_I}$ satisfy the following:

1. $\forall a \in \{E, I\}, \beta'_a, \beta_a^1, \ldots, \beta_a^l$ is a partition of β_a.
2. $\forall v \in var(\mathcal{R}), M(v) \neq \emptyset \Rightarrow M'(v) = M(v)$.
3. If two sets β_a^j and β_b^k (where $a, b \in \{E, I\}$) share a variable v, $M'(v) \neq \emptyset$.
4. If a variable v occurs in some β_a^j (where $a \in \{E, I\}$) and $M'(v) \neq \emptyset$, $M'(v)$ is in $\mathbf{t}^{i_j}$.
5. For every instance of EDB atom $R(\mathbf{x}_1, \ldots, \mathbf{x}_k) \in \beta'_E$, $\emptyset \notin M'(\{\mathbf{x}_1, \ldots, \mathbf{x}_k\})$ and $R(M'(\mathbf{x}_1), \ldots, M'(\mathbf{x}_k))$ is an EDB atom of $\mathcal{P}$.
6. For every instance of IDB atom $R(\mathbf{t}) \in \beta'_I$, $M'(\mathbf{t}) \neq \emptyset$ and $R \in \varphi(M'(\mathbf{t}))$.

- Let $\mathcal{P}$ be a rule instance

$$P_1(\mathbf{t}^1) \wedge \ldots \wedge P_m(\mathbf{t}^m) \Rightarrow P(\mathbf{t})$$

in Π, where all atoms in the body of the rule are EDB atoms. Let $s = ((P(\mathbf{t}), \rho, \mathcal{L}, \varphi), \beta_E, \beta_I, M) \in \mathcal{S}$. Then, $\langle accept \rangle \in \delta(s, (P(\mathbf{t}), \rho, \mathcal{L}, \varphi))$ if the following holds:

1. For every instance of EDB atom $R(\mathbf{x}_1, \ldots, \mathbf{x}_k) \in \beta_E$, $\emptyset \notin M(\{\mathbf{x}_1, \ldots, \mathbf{x}_k\})$ and $R(M(\mathbf{x}_1), \ldots, M(\mathbf{x}_k))$ is an EDB atom of $\mathcal{P}$.
2. For every instance of IDB atom $R(\mathbf{t}) \in \beta_I$, $M(\mathbf{t}) \neq \emptyset$ and $R \in \varphi(M(\mathbf{t}))$.

By using technical arguments similar to those developed in [9], it follows that

$$\{\tau \in p\_dec(\mathcal{L}, Q, \Pi) \cap T(A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{R},Q,\Pi})\} = \{\tau \in \tau \in p\_dec(\mathcal{L}, Q, \Pi) \cap p\_dec^{goal}(\mathcal{L}, \mathcal{R}, Q, \Pi)\}$$

It is easy to see that the number of states and transition in the automaton is exponential in the size of Π, $\mathcal{L}$ and $\mathcal{R}$. □

Theorem A.3.16. [Theorem 4.3.4]
Let Π be a program with monadic goal predicate Q, $\Pi_{\mathcal{M}}$ a monadic program with goal predicate $Q_{\mathcal{M}}$. Let $\mathcal{L}$ be the set of internal IDB predicates in $\Pi_{\mathcal{M}}$, $\mathcal{R}_1, \ldots, \mathcal{R}_k$ the internal rules of $\Pi_{\mathcal{M}}$ and $\mathcal{Q}_1, \ldots, \mathcal{Q}_l$ the goal rules of $\Pi_{\mathcal{M}}$ Then Π is contained in $\Pi_{\mathcal{M}}$ if and only if

$$T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) \subseteq \bigcup_{i=1}^{k} T(A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R}_i,Q,\Pi}) \cup \bigcup_{i=1}^{l} T(A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{Q}_i,Q,\Pi})$$

■

Proof. By Theorem 4.2.8, Π is contained in $\Pi_{\mathcal{M}}$ if and only if

$$p\_dec(\mathcal{L}, Q, \Pi) \subseteq \bigcup_{i=1}^{k} p\_dec^{\neg pf}(\mathcal{L}, \mathcal{R}_i, Q, \Pi) \cup \bigcup_{i=1}^{l} p\_dec^{goal}(\mathcal{L}, \mathcal{Q}_i, Q, \Pi)$$

By Propositions 4.3.1, 4.3.2 and 4.3.3, this means that

$$T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) \subseteq \bigcup_{i=1}^{k} \left(T(A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R},Q,\Pi}) \cap T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) \right) \cup \bigcup_{i=1}^{l} \left(T(A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{R},Q,\Pi}) \cap T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) \right)$$

i.e.

$$T(A^{p\_dec}_{\mathcal{L},Q,\Pi}) \subseteq \bigcup_{i=1}^{k} T(A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R}_i,Q,\Pi}) \cup \bigcup_{i=1}^{l} T(A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{Q}_i,Q,\Pi})$$

□

Theorem A.3.17. [Theorem 4.3.5]
Containment of a recursive Datalog *program in a union of conjunctive queries is in 2EXPTIME.* ∎

Proof. For more convenience, we suppose that the monadic Datalog program that we consider is as in Proposition A.1.2. Then, by Proposition 3.5.1, we obtain an automaton $A^{Q_{\mathcal{M}},\Pi_{\mathcal{M}}}_{Q,\Pi}$, whose size is exponential in the size of $\Pi_{\mathcal{M}}$ and Π, such that

$$T(A^{Q_{\mathcal{M}},\Pi_{\mathcal{M}}}_{Q,\Pi}) = \bigcup_{i=1}^{k} T(A^{p\_dec^{\neg pf}}_{\mathcal{L},\mathcal{R}_i,Q,\Pi}) \cup \bigcup_{i=1}^{l} T(A^{p\_dec^{goal}}_{\mathcal{L},\mathcal{Q}_i,Q,\Pi})$$

Thus, by Theorem 4.3.4, containment in a monadic programs can be reduced to containment of tree automata whose size of exponential size. By Proposition 3.5.3, containment of tree automata can be decided in exponential time. Therefore, the result follows. □

A.4 Transitive Programs

A.4.1 Proof of Theorem 5.2.3 : First Steps

First of all, we transform expanding trees into a new type of tree, called *transitive tree*. Indeed, in transitive programs, the recursion is essentially restricted to star rules which, while being derived along an unfolding expansion tree, let us see locally a structure of comb. Such a structure implies an arbitrary height of the tree, and could be nicely re-organized by involving only one node whose number of children would be arbitrary high: be reducing the height of the tree, we would sacrifice the boundedness of the number of children of its nodes. It is this re-organisation that has been performed in *transitive trees*, which are not more expanding trees *stricto sensu*, but remain very close to expanding trees, and appear to be much more adapted to the study of transitive Datalog programs.

Definition A.4.1.
A *transitive tree* for a transitive Datalog program γ with goal predicate Γ is a tree whose nodes are labelled by couples (α, ρ), where α is an IDB atom and ρ is an instance of a Datalog rule whose head is α. The atom (resp. the rule) labelling a node x is denoted α_x (resp. ρ_x). The root of the tree must be labelled by a Γ-atom.

Consider a node x:

- if α_x is an atom $P(\mathbf{t})$ where P is a star-free predicate of γ, then ρ_x must be a star-free rule $\mathcal{R}$ of γ:

$$I_1^*(x_1, y_1) \wedge \ldots \wedge I_k^*(x_1, y_1) \wedge P_1(\mathbf{x}^1) \wedge \ldots \wedge P_l(\mathbf{x}^l) \Rightarrow P(\mathbf{t})$$

  where $I_1, \ldots, I_k$ are star predicates of γ, $P_1, \ldots, P_l$ are EDB predicates, and x has k children $x1, \ldots, xk$ such that $\alpha_{xi} = I_i^*(x_i, y_i)$ for each $i \in \{1, \ldots, k\}$. In particular, if the body of ρ_x does not contain any star predicate, then x must be a leaf.
- if α_x is an atom $I^*(x, y)$ where I is a star predicate of γ, then ρ_x must be either
  - a DATALOG rule $\top \Rightarrow I^*(x, y)$, where $x = y$.
  - a DATALOG rule $\bigwedge_{i=0}^{n} I(x_i, x_{i+1}) \Rightarrow I^*(x_0, x_{n+1})$, where $n \in \mathbb{N}$, $x = x_0$ and $y = x_{n+1}$ (this case is possible even if $x = y$).

■

The query corresponding to an expansion tree is the conjunction of all EDB atoms in ρ_x for all nodes x in the tree, with the variables in the root atom as the free variables. It is immediate that the queries corresponding to any expansion tree of a transitive DATALOG program γ with goal predicate Γ are the queries corresponding to any transitive tree of γ. Therefore, if we denote by $trans(\Gamma, \gamma)$ the set of transitive trees, we have, for every database D,

$$\Gamma_\gamma^\infty(D) = \bigcup_{\tau \in trees(\Gamma,\gamma)} \tau(D) = \bigcup_{\tau \in trans(\Gamma,\gamma)} \tau(D)$$

Once again, of particular interest are transitive trees that are obtained by "unfolding" the program Γ.

Definition A.4.2.
A transitive tree τ of a transitive DATALOG program γ is an *unfolding transitive tree* if it satisfies the following conditions:

- the atom labelling the root is the head of a rule in γ.
- if a node x is labelled by (α_x, ρ_x), then the variables in the body of ρ_x either occur in α_x or they do not occur in the label of any node above x.

■

We denote by $u\_trans(\Gamma, \gamma)$ the set of unfolding transitive trees of Γ for the predicate γ. The following proposition follows immediately.

Proposition A.4.3.
Let Γ be a transitive program with a goal predicate Γ. For every database D, we have

$$\Gamma_\gamma^\infty(D) = \bigcup_{\tau \in u\_trans(\Gamma,\gamma)} \tau(D)$$

■

Now, as we already said, it may be convenient to extract from a tree as many pieces of information as possible. Therefore, a natural idea is to look at how a set of predicates, with the help of such rules, can imply another set of predicates. This is why we introduce now the concept of *interpretations* of a transitive DATALOG program, as well as the concept of *positive instances*.

Definition A.4.4.
Let γ be a transitive DATALOG program, $\{P_1, \ldots, P_n\}$ the set of its predicates, and $\|\gamma\|$ the number of its rules. Each rule $\mathcal{R}_i$ of γ is associated to a formula $\mathcal{F}_i$ in Second-Order Logic:

- if $\mathcal{R}_i$ is a star-free rule:

  $$P_{s_1}(\mathbf{v}_1) \wedge \ldots \wedge P_{s_k}(\mathbf{v}_k) \Rightarrow P_{s_0}(\mathbf{v}_0)$$

  whose set of variables is $\{v_1, \ldots, v_m\}$, it is associated to the formula

  $$\mathcal{F}_i(\mathbf{P}_1, \ldots, \mathbf{P}_n) \equiv (\forall v_1, v_2, \ldots, v_m) \left(\left(\bigwedge_{i=1}^{k} \mathbf{P}_{s_i}(\mathbf{v}_i) \right) \Rightarrow \mathbf{P}_{s_0}(\mathbf{v}_0) \right)$$

  with the free predicates among $\mathbf{P}_1, \ldots, \mathbf{P}_n$ and no free variable.
- if $\mathcal{R}_i$ is a star rule:

  $$P_{s_1}(x, z) \wedge P_{s_0}(z, y) \Rightarrow P_{s_0}(x, y)$$

  it is associated to the formula

  $$\mathcal{F}_i(\mathbf{P}_1, \ldots, \mathbf{P}_n) \equiv (\forall x, y)\, (\exists n \in \mathbb{N})\, (\exists v_0, \ldots, v_n)$$
  $$\left(\left((x = v_0) \wedge (y = v_n) \wedge \bigwedge_{i=1}^{n} \mathbf{P}_{s_1}(v_{i-1}, v_i) \right) \Rightarrow \mathbf{P}_{s_0}(x, y) \right)$$

  with the free predicates among $\mathbf{P}_1, \ldots, \mathbf{P}_n$ and no free variable.
- if $\mathcal{R}_i$ is a baby star rule:

  $$\top \Rightarrow P_{s_0}(x, x)$$

  it is associated to the formula

  $$\mathcal{F}_i(\mathbf{P}_1, \ldots, \mathbf{P}_n) \equiv \top$$

  with the free predicates among $\mathbf{P}_1, \ldots, \mathbf{P}_n$ and no free variable.

Now, let $\mathcal{V} = \{v_1, \ldots, v_m\}$ and $\mathcal{V}' = \{v'_1, \ldots, v'_{m'}\}$ be finite sets of variables, then $\mathcal{A} = \{P_{s_1}(\mathbf{v}_1), \ldots, P_{s_a}(\mathbf{v}_a)\}$ a finite set of atoms with variables among $\mathcal{V}$ and $\mathcal{A}' = \{P_{s'_1}(\mathbf{v}'_1), \ldots, P_{s'_{a'}}(\mathbf{v}'_{a'})\}$ a finite set of atoms with variables among $\mathcal{V} \cup \mathcal{V}'$. The *interpretation* of γ on the 4-uple $(\mathcal{V}, \mathcal{V}', \mathcal{A}, \mathcal{A}')$ is the formula

$$\mathcal{I}_\gamma(\mathcal{V}, \mathcal{V}', \mathcal{A}, \mathcal{A}') \equiv (\forall \mathbf{P}_1, \ldots, \mathbf{P}_n)\, (\forall v_1, \ldots, v_m)\, (\exists v'_1, \ldots, v'_{m'})$$
$$\left(\left(\bigwedge_{i=1}^{\|\gamma\|} \mathcal{F}_i(\mathbf{P}_1, \ldots, \mathbf{P}_n) \wedge \bigwedge_{i=1}^{a} \mathbf{P}_{s_i}(\mathbf{v}_i) \right) \Rightarrow \bigwedge_{i=1}^{a'} \mathbf{P}_{s'_i}(\mathbf{v}'_i) \right)$$

with no free predicate and no free variable.

If $\mathcal{I}_\gamma(\mathcal{V}, \mathcal{V}', \mathcal{A}, \mathcal{A}') \equiv \top$, a *positive instance* of γ on the 4-uple $(\mathcal{V}, \mathcal{V}', \mathcal{A}, \mathcal{A}')$ is a total mapping $\phi : \mathcal{V} \cup \mathcal{V}' \to \mathcal{V}$ such that $\phi_{|\mathcal{V}} = Id_{\mathcal{V}}$ and that

$$\top \equiv (\forall \mathbf{P}_1, \ldots, \mathbf{P}_n)\,(\forall v_1, \ldots, v_m)$$
$$\left(\left(\bigwedge_{i=1}^{\|\gamma\|} \mathcal{F}_i(\mathbf{P}_1, \ldots, \mathbf{P}_n) \wedge \bigwedge_{i=1}^{a} \mathbf{P}_{s_i}(\mathbf{v}_i) \right) \Rightarrow \bigwedge_{i=1}^{a'} \mathbf{P}_{s'_i}(\phi(\mathbf{v}'_i)) \right)$$

■

Now, similarly to the Definition 5.1.1 that we have given, we define another program, called *triangle-reduction* of a transitive program, and which consists in a fragment of the diamond-reduction of the transitive program. In fact, the difference between triangle-reduction and diamond-reduction lies in the length of the diamond-rules that are allowed: the diamond-rules in the diamond-reduction are designed to allow a "fusion" of the diamond-rules present in the triangle-reduction.

Definition A.4.5.
Let γ be a transitive DATALOG program. The *triangle-reduction* of γ is the DATALOG program γ^∇ built hereafter:

Every predicate in γ is a predicate in $\gamma^\diamond$. Moreover, in addition to these predicates, γ^∇ contains a binary EDB predicate $G^\diamond$, called *diamond predicate*, for each star IDB predicate G^* in γ. The rules in γ^∇ are defined as follows:

- Each star-free rule

$$E_1(\mathbf{v}^1) \wedge \ldots \wedge E_k(\mathbf{v}^k) \wedge G_1^*(x_1^1, x_2^1) \wedge \ldots \wedge G_l^*(x_1^l, x_2^l \Rightarrow G(\mathbf{t}))$$

  in γ is also a rule in γ^∇.
- For each star IDB predicate G^* in γ, γ^∇ contains the diamond-rules involving at most 3 atoms, which are rules ρ verifying:
  - $\exists n \in \{1, 2, 3\}$ such that ρ is a rule

$$E_1(x_0, x_1) \wedge E_2(x_1, x_2) \wedge \ldots \wedge E_n(x_{n-1}, x_n) \Rightarrow G^*(x_0, x_n)$$

  - $\forall i \in \{1, 2, \ldots, n\}, E_i$ is either the predicate G or the predicate $G^\diamond$.
  - $\forall i \in \{1, 2, \ldots, n-1\}$, if $E_i = G$, then $E_{i+1} = G^\diamond$.

■

Now, we derive from the existence of the mapping φ described in the Definition 2.4.1 the notion of *degree* of a predicate of the diamond-reduction $\gamma^\diamond$ of a transitive DATALOG program γ.

Definition A.4.6.
Let P be a predicate in $\gamma^\diamond$ and φ the mapping described in the Definition 2.4.1 The *degree* of P is the integer $d^\circ P$ defined as follows:

- If P is an EDB predicate of γ, then $d^\circ P = 2 + 3\varphi(P)$.

- If P is an star-free IBD predicate of γ, then $d^\circ P = 3\varphi(P)$.
- If P is a predicate $G^\diamond$, then $d^\circ P = 3\varphi(P) + 1$.
- If P is an star IBD predicate of γ, then $d^\circ P = 3\varphi(P) + 2$.

■

This notion of degree of a predicate has the immediate consequence of proving that both $\gamma^\diamond$ and γ^∇ must be non-recursive programs, and therefore that both $u\_trees(\Gamma, \gamma^\diamond)$ and $u\_trees(\Gamma, \gamma^\nabla)$ must be finite.

Proposition A.4.7.
Let γ be a transitive DATALOG *program with goal predicate Γ. Let $\gamma^\diamond$ be its diamond-reduction. $\gamma^\diamond$ is a non-recursive program, and $u\_trees(\Gamma, \gamma^\diamond)$ is finite.* ■

Proof. In $\gamma^\diamond$, the definition of a predicate P may not depend on a predicate Q such that $d^\circ P \leq d^\circ Q$. Therefore, no predicate can depend recursively on itself when being defined. That is why $\gamma^\diamond$ is a *non-recursive* program, which immediately implies that $u\_trees(\Gamma, \gamma^\diamond)$ is finite. □

Since every rule of γ^∇ is a rule of $\gamma^\diamond$, the following corollary is now straightforward:

Corollary A.4.8.
Let γ be a transitive DATALOG *program with goal predicate Γ. Let γ^∇ be its triangle-reduction. γ^∇ is a non-recursive program, and $u\_trees(\Gamma, \gamma^\nabla)$ is finite.* ■

After that, and now that we know what is a fixpoint labelling, two special such fix points are very susceptible of being of a great interest: the bigger one and the least one. $\mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$ being the bigger fix point, we therefore have too look at the least fix point.

Definition A.4.9.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. $\forall \Sigma \in p\_trees(Q, \Pi)$, it is straightforward that Ψ_Σ is a non-decreasing function, such that $\forall E \subseteq \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi), E \subseteq \Psi_\Sigma(E)$. We denote by $\Psi^\omega_\Sigma(E)$ the set $\bigcup_{n\in\mathbb{N}} \Psi^n_\Sigma(E)$. It is straightforward that $\Psi^\omega_\Sigma(E)$ is the smallest set F such that $E \subseteq F = \Psi_\Sigma(F)$.

We say that $i^{-1}(\Sigma, \Psi^\omega_\Sigma(\emptyset))$ is the *minimal labelling* of Σ, and we denote this tree by $l_{min}(\Sigma)$. By the above property, $l_{min}(\Sigma) \in p\_label^{fp}(\Sigma, \gamma, Q, \Pi)$. ■

Definition A.4.10.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. Let $\Sigma \in p\_trees(Q, \Pi)$ be a proof tree of Π, and (N, l) be a couple $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$. If $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q, \Pi)$, let $\mathcal{A}$ be the set of all atoms $E(\mathbf{v})$ in κ where E is an EDB predicate in both Π and γ; and let θ be the unfolding mapping of Σ.

We call *κ-mapping* the total mapping $\Lambda_{\varphi,\kappa} : var_\diamond(\gamma^\diamond) \cup var(\rho) \to var_\diamond(\gamma^\diamond) \cup var(\kappa)$ such that

- $\Lambda_{\varphi,\kappa}(v) = \varphi(v)$ if $\varphi(v) \in var_\diamond(\gamma^\diamond)$.
- $\Lambda_{\varphi,\kappa}(v) = \theta^{-1}([\varphi(v)]_\Sigma)$ if $\varphi(v) \in var(\rho)$.

If $S = \{P_{s_i}(\mathbf{v}_i) | 1 \leq i \leq |S|\}$, we set $S' = \{P'_{s_i}(\Lambda_{\varphi,\kappa}(\mathbf{v}_i)) | 1 \leq i \leq |S|\}$, where, $\forall i \in \{1, \ldots, |S|\}$,

- if P_{s_i} is a predicate $G^\diamond$, then P'_{s_i} is the predicate G^*.
- otherwise, $P'_{s_i} = P_{s_i}$.

The *interpretation* of (N, l) is the formula $\mathcal{I}_N(l) \equiv \mathcal{I}_\gamma(var(\kappa), var_\diamond(\gamma^\diamond), \mathcal{A}, S')$.

If $\mathcal{I}_N(l) \equiv \top$, then a *positive instance* of (N, l) is a triple (N, l, ϕ) where ϕ is a positive instance ϕ of γ on $(var(\kappa), var_\diamond(\gamma^\diamond), \mathcal{A}, S')$. ∎

Then comes a very important theorem, that links the interpretation of predicates to the existence of containment mappings:

Proposition A.4.11.
Let Π *be a* DATALOG *program with goal predicate* Q, γ *a transitive* DATALOG *program with goal predicate* Γ, $\Sigma \in p\_trees(Q, \Pi)$ *a proof tree,* $N = (R(\boldsymbol{t}), \rho)$ *the root of* Σ, $\sigma \in p\_label(\Sigma, \gamma, Q, \Pi)$ *a* γ*-labelled prof tree and* $\Gamma(\boldsymbol{v})$ *an instance of* Γ *with variables among* $var_\diamond(\gamma^\diamond)$.

There is a containment mapping from an unfolding tree $\nu \in u\_trees(\Gamma, \gamma)$ *to* $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q, \Pi)$ *if and only if* $\exists l = (\tau, \{\Gamma(\boldsymbol{v})\}, \rho, V, \varphi) \in \mathcal{L}(\gamma, \Pi)$ *such that* $\varphi(\boldsymbol{v}) = \boldsymbol{t}$ *and* $\mathcal{I}_N(l) \equiv \top$. ∎

Proof. Let us first assume that there exists a containment mapping $\mathbf{M}$ from ν to $\kappa = \mathcal{U}(\Sigma)$. By a direct induction, we obtain that, if S' is the set whose elements are atoms $P(\mathbf{M}(\mathbf{t}))$, where $P(\mathbf{t})$ is an atom in ν, then $\top \equiv \mathcal{I}_\gamma(var(\kappa), var_\diamond(\gamma^\diamond), \mathcal{A}, S')$. Therefore, if $(\gamma(\mathbf{v}), r)$ is the root of ν, we must obtain that $\top \equiv \mathcal{I}_\gamma(var(\kappa), var_\diamond(\gamma^\diamond), \mathcal{A}, \{\Gamma(\mathbf{M}(\mathbf{v}))\})$. We point out that, if $(R(\mathbf{t}'), \rho')$ is the root of κ, then $\mathbf{M}$ maps $\mathbf{v}$ to $\mathbf{t}'$, and therefore that $\mathbf{v}, \mathbf{t}'$ and $\mathbf{t}$ must be of the same size. Then, $\exists l = (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi) \in \mathcal{L}(\gamma, \Pi)$, such that $\varphi(\mathbf{v}) = \mathbf{t}$. That is why

$$\mathcal{I}_N(l) \equiv \mathcal{I}_\gamma(var(\kappa), var_\diamond(\gamma^\diamond), \mathcal{A}, \{\Gamma(\mathbf{t}')\}) \equiv \mathcal{I}_\gamma(var(\kappa), var_\diamond(\gamma^\diamond), \mathcal{A}, \{\Gamma(\mathbf{M}(\mathbf{v}))\}) \equiv \top$$

Now, let us assume that $\exists l = (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi) \in \mathcal{L}(\gamma, \Pi)$, such that $\varphi(\mathbf{v}) = \mathbf{t}$ and $\mathcal{I}_N(l) \equiv \top$. We can select a proof of $\mathcal{I}_N(l) \equiv \top$. Then, by introducing the extensional closure of γ, as defined in Definition 4.1.5, we easily prove by induction that there exists a transitive tree $\nu' \in u\_trans(\Gamma, \gamma)$ such that some containment mapping $\mathbf{M}'$ from ν' to κ exists. Then, we can transform ν' in an unfolding tree $\nu \in u\_tree(\Gamma, \gamma)$ with the same variables, and such that each EDB atom in ν' be an EDB predicate in ν. Therefore, $\mathbf{M}'$ induces a containment mapping $\mathbf{M}' : \nu \to \kappa$, which closes the proof. □

A.4.2 Proof of Theorem 5.2.3 : A Sufficient Condition

Intuitively, the application introduced in the Definition 5.2.2 conserves the truthfulness of a set of atoms. This intuition is confirmed by the fact that, starting from the knowledge of no atom of any sort, this application can reach only sets of atoms whose interpretation is true:

Lemma A.4.12.
Let Π *be a* DATALOG *Program with goal predicate* Q *and* γ *a transitive* DATALOG *program with goal predicate* Γ. $\forall \Sigma \in p\_trees(Q, \Pi), \forall (N, l) \in \Psi^{\omega}_{\Sigma}(\emptyset), \mathcal{I}_N(l) \equiv \top$. ∎

Proof. We prove the above statement by the induction below.

If $\Sigma \in p\_trees(Q, \Pi)$ and $n \in \mathbb{N}$, let H_n be the property : "$\forall (N, l) \in \Psi^n_{\Sigma}(\emptyset), \mathcal{I}_N(l) \equiv \top$".

First of all, since $\Psi^0_{\Sigma}(\emptyset) = \emptyset$, it is straightforward that H_0 and H_1 are true.

Now, if $n \geq 1$ and H_n is true, let be some $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \Psi^{n+1}_{\Sigma}(\emptyset) = \Psi_{\Sigma}(E)$, where $E = \Psi^n_{\Sigma}(\emptyset)$. We are in (at least) one of the following 15 cases:

1. $(N, l) \in E$.
   Therefore, $\mathcal{I}_N(l) \equiv \top$.
2. N is the father, in the tree Σ, of some node $N' = (R'(\mathbf{t}'), \rho') \in \mathcal{N}_{\Sigma}$, $V \subseteq var(\mathbf{t}')$ and $(N', (\tau, S, \rho', V, \varphi)) \in E$.
   $\top \equiv \mathcal{I}_{N'}((\tau, S, \rho', V, \varphi))$ and $\mathcal{I}_{N'}((\tau, S, \rho', V, \varphi)) \equiv \mathcal{I}_N(l)$. Therefore, $\mathcal{I}_N(l) \equiv \top$.
3. N is a child, in the tree Σ, of some node $N' = (R'(\mathbf{t}'), \rho') \in \mathcal{N}_{\Sigma}$, $V \subseteq var(\mathbf{t})$ and $(N', (\tau, S, \rho', V, \varphi)) \in E$.
   $\top \equiv \mathcal{I}_{N'}((\tau, S, \rho', V, \varphi))$ and $\mathcal{I}_{N'}((\tau, S, \rho', V, \varphi)) \equiv \mathcal{I}_N(l)$. Therefore, $\mathcal{I}_N(l) \equiv \top$.
4. $\exists S' \supseteq S$ such that $(N, (\tau, S', \rho, V, \varphi)) \in E$.
   $\top \equiv \mathcal{I}_N((\tau, S', \rho, V, \varphi))$ and $\mathcal{I}_N((\tau, S', \rho, V, \varphi)) \Rightarrow \mathcal{I}_N(l)$. Therefore, $\mathcal{I}_N(l) \equiv \top$.
5. $\exists S', S'' \subseteq S$ such that $S = S' \cup S''$, $\varphi(var(S')) \cap \varphi(var(S'')) \subseteq V$, and such that both $(N, (\tau, S', \rho, V, \varphi)) \in E$ and $(N, (\tau, S'', \rho, V, \varphi)) \in E$.
   $\top \equiv \mathcal{I}_N((\tau, S', \rho, V, \varphi))$ and $\top \equiv \mathcal{I}_N((\tau, S'', \rho, V, \varphi))$. Moreover,
   $$\mathcal{I}_N((\tau, S', \rho, V, \varphi)) \wedge \mathcal{I}_N((\tau, S'', \rho, V, \varphi)) \equiv \mathcal{I}_N(l)$$
   Therefore, $\mathcal{I}_N(l) \equiv \top$.
6. $\exists \tau' \in u\_trees(\Gamma, \gamma^{\diamond})$ such that $(N, (\tau', S, \rho, V, \varphi)) \in E$.
   $\top \equiv \mathcal{I}_N((\tau', S, \rho, V, \varphi))$ and $\mathcal{I}_N((\tau', S, \rho, V, \varphi)) \equiv \mathcal{I}_N(l)$. Therefore, $\mathcal{I}_N(l) \equiv \top$.
7. $\exists V' \subseteq var(\rho)$ such that $(N, (\tau, S, \rho, V', \varphi)) \in E$.
   $\top \equiv \mathcal{I}_N((\tau, S, \rho, V', \varphi))$ and $\mathcal{I}_N((\tau, S, \rho, V', \varphi)) \equiv \mathcal{I}_N(l)$. Therefore, $\mathcal{I}_N(l) \equiv \top$.
8. Some total mapping $h : var_{\diamond}(\gamma^{\diamond}) \cup V \to var_{\diamond}(\gamma^{\diamond}) \cup V$ verifies $h_{|V} = Id_V$ and $(N, (\tau, S, \rho, V, h \circ \varphi)) \in E$.
   $\mathcal{I}_N((\tau, S, \rho, V, h \circ \varphi)) \Rightarrow \mathcal{I}_N(l)$ and $\mathcal{I}_N((\tau, S, \rho, V, h \circ \varphi)) \equiv \top$. Therefore, $\mathcal{I}_N(l) \equiv \top$.
9. $\exists \varphi' \in (var_{\diamond}(\gamma^{\diamond}) \cup V)^{var_{\diamond}(\gamma^{\diamond})}$ such that $\varphi'_{|var(S)} = \varphi_{|var(S)}$ and $(N, (\tau, S, \rho, V, \varphi')) \in E$.
   $\mathcal{I}_N((\tau, S, \rho, V, \varphi')) \equiv \mathcal{I}_N(l)$ and $\mathcal{I}_N((\tau, S, \rho, V, \varphi')) \equiv \top$. Therefore, $\mathcal{I}_N(l) \equiv \top$.
10. $\exists n' = (R'(\mathbf{t}'), \rho') \in \mathcal{N}_{\tau}$ and $\exists S' \subseteq S$ such that $S = S' \cup \{R'(\mathbf{t}')\}$, S' contains all atoms in the body of ρ' and $(N, (\tau, S', \rho, V, \varphi)) \in E$.

$\mathcal{I}_N((\tau, S', \rho, V, \varphi)) \Rightarrow \mathcal{I}_N(l)$ and $\mathcal{I}_N((\tau, S', \rho, V, \varphi)) \equiv \top$. Therefore, $\mathcal{I}_N(l) \equiv \top$.

11. $\exists \tau' \in u\_trees(\Gamma, \gamma^\diamond)$ and an atom $G^\diamond(x, y) \in S$ such that $(N, (\tau', S', \rho, V, \varphi)) \in E$, where $S' = S \backslash \{G^\diamond(x, y)\} \cup \{G(x, y)\}$.
$\mathcal{I}_N((\tau', S', \rho, V, \varphi)) \Rightarrow \mathcal{I}_N(l)$ and $\mathcal{I}_N((\tau', S', \rho, V, \varphi)) \equiv \top$. Therefore, $\mathcal{I}_N(l) \equiv \top$.
12. $\exists \tau' \in u\_trees(\Gamma, \gamma^\diamond)$, an atom $G^\diamond(x, y) \in S$ and a variable $z \in var_\diamond(\gamma^\diamond)$ such that $(N, (\tau', S', \rho, V, \varphi)) \in E$, where $S' = S \backslash \{G^\diamond(x, y)\} \cup \{G^\diamond(x, z), G^\diamond(z, y)\}$.
$\mathcal{I}_N((\tau', S', \rho, V, \varphi)) \Rightarrow \mathcal{I}_N(l)$ and $\mathcal{I}_N((\tau', S', \rho, V, \varphi)) \equiv \top$. Therefore, $\mathcal{I}_N(l) \equiv \top$.
13. S is a singleton $\{G^\diamond(x, y)\}$ such that $\varphi(x) = \varphi(y)$.
$(N, l) \in \Psi_\Sigma(\emptyset) \subseteq E$. Therefore, $\mathcal{I}_N(l) \equiv \top$.
14. S is a singleton $\{P(\mathbf{v})\}$ where P is an EDB predicate and φ maps $P(\mathbf{v})$ to an atom $P(\mathbf{x})$ appearing in ρ.
$(N, l) \in \Psi_\Sigma(\emptyset) \subseteq E$. Therefore, $\mathcal{I}_N(l) \equiv \top$.
15. $S = \emptyset$.
$(N, l) \in \Psi_\Sigma(\emptyset) \subseteq E$. Therefore, $\mathcal{I}_N(l) \equiv \top$.

This proves that $H_n \Rightarrow H_{n+1}$ and, therefore, that H_ω is true, which is the statement of our lemma. □

This truthfulness directly implies the following theorem, which expresses one of the two implications Theorem 5.2.3 consists in:

Theorem A.4.13.
Let Π *be a* DATALOG *program with goal predicate* Q, γ *a transitive* DATALOG *program with goal predicate* Γ. *Let be* $\Sigma \in p\_trees(Q, \Pi), \sigma = l_{min}(\Sigma) \in p\_label^{fp}(\Sigma, \gamma, Q, \Pi)$, $r = (N, L)$ *the root of* σ. *Let* $\Gamma(\boldsymbol{v})$ *be an instance of* Γ *with variables among* $var_\diamond(\gamma^\diamond)$. *If* $N = (R(\boldsymbol{t}), \rho)$ *is such that* $\varphi(\boldsymbol{v}) = \boldsymbol{t}$ *and some* $l = (\tau, \{\Gamma(\boldsymbol{v})\}, \rho, V, \varphi) \in L$, *then there exists a containment mapping from a tree* $\nu \in u\_trees(\Gamma, \gamma)$ *to* $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q, \Pi)$. ∎

Proof. With the above notations, $(N, l) = ((R(\mathbf{t}), \rho), (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi)) \in \Psi^\omega_\Sigma(\emptyset)$. Thus, $\mathcal{I}_N(l) = \mathcal{I}_\gamma(var(\kappa), var_\diamond(\gamma^\diamond), \mathcal{A}, \{\Gamma(\mathbf{t})\}) \equiv \top$, which means that there exists a containment mapping from a tree $\nu \in u\_trees(\Gamma, \gamma)$ to $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q, \Pi)$. □

A.4.3 Proof of Theorem 5.2.3 : A Necessary Condition

Now, we have to demonstrate a result which is much less intuitive than the precedent one, and which states that the pieces of information caught through the concepts of γ-labelled proof trees and obtained with the application defined in the Definition 5.2.2 are exhaustive enough to contain the pieces of information we are looking for (i.e. the existence of containment mappings). It is this result that we will now prove. However, the proof of this result involves the definition of many objects, whose definition may sometimes appear complicated since some of them were designed in the only purpose of avoiding nasty cases.

First of all, we introduce the concept of *descendant-free sets of atoms*, and some concepts related to this one. This concept is quite natural: in order to reduce the number of

proof trees we will have to look at, we want to quantify the existence of redundancy in the labellings of a proof tree: indeed, if we can remove redundant atoms from a given set, we will obtain a "smaller" set, but will not change anything to its interpretation. This quantification is made possible through the invention of the concept of *descendant-free sets of atoms*, which correspond to labellings without redundancy.

Definition A.4.14.
Let Π be a DATALOG Program with goal predicate Q, $\tau \in u\_trees(Q, \Pi)$, $\mathcal{P}_\tau$ the set of atoms in τ and $N, N' \in \mathcal{N}_\tau$.

If $\exists k \geq 0, \exists N_0, \ldots, N_k \in \mathcal{N}_\tau$ such that $N = N_0, N' = N_k$ and $\forall i \in \{0, \ldots, k-1\}$, N_i is the father of N_{i+1}, then N' is a *descendant* of N (we also say that N is an *ascendant* of N').

Let $S \subseteq \mathcal{P}_\tau$ be a non-empty set of atoms, and $\mathbf{P}(\mathbf{t})$ an occurrence of an atom in $P(\mathbf{t}) \in S$. If the node $N \in \mathcal{N}_\tau$ whose rule ρ contains $\mathbf{P}(\mathbf{t})$ has an ascendant N' such that S contains the head predicate $Q(\mathbf{u})$ of its rule ρ', then $\mathbf{P}(\mathbf{t})$ is said to be *overwhelmed* by S in τ.

If every atom $P(\mathbf{t}) \in S$ has an occurrence $\mathbf{P}(\mathbf{t})$ that is not overwhelmed by S in τ, we say that S is *descendant-free* in τ. ∎

We define now the concept of *natural partition* and of *natural label*, that will both be useful in order to formalize the splitting operation I briefly introduced in the page 29. Intuitively, the natural partition is the most efficient partition of a set of atoms that can be found to divide the atoms and allow us to prove separately the conjunctions of atoms present in each of the sets the partition consists in.

Definition A.4.15.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. Let $\Sigma \in p\_trees(Q, \Pi)$ be a proof tree of Π and $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$.

We define the symmetric relationship $\approx_{(N,l)}$ between variables $a, b \in \varphi^{-1}(var_\diamond(\gamma^\diamond))$ by: $a \approx_{(N,l)} b \Leftrightarrow \exists P(\mathbf{v}) \in S, \exists c, d \in var(\mathbf{v})$ such that $\varphi(a) = \varphi(c)$ and $\varphi(b) = \varphi(d)$. Then, we define the equivalence relationship $\equiv_{(N,l)}$ between variables $a, b \in \varphi^{-1}(var_\diamond(\gamma^\diamond))$ as the reflexive transitive closure of $\approx_{(N,l)}$. This lets us define classes of equivalence $\mathcal{C}_1, \ldots, \mathcal{C}_k \subseteq \varphi^{-1}(var_\diamond(\gamma^\diamond))$ such that $\forall i \in \{1, \ldots, k\}, \forall a \in \mathcal{C}_i, \forall b \in \varphi^{-1}(var_\diamond(\gamma^\diamond)), b \in \mathcal{C}_i \Leftrightarrow a \equiv_{(N,l)} b$.

We can now split S into sets $S_0, S_1, \ldots, S_k \subseteq S$ such that $S_0 = \{P(\mathbf{v}) \in S_i | var(\mathbf{v}) \subseteq V\}$ and $\forall i \in \{1, \ldots, k\}, S_i = \{P(\mathbf{v}) \in S_i | var(\mathbf{v}) \cap \mathcal{C}_i \neq \emptyset\}$. This partition $(S_0, \ldots, S_k)$ of S is called *natural partition* of (N, l). ∎

Definition A.4.16.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. Let $\Sigma \in p\_trees(Q, \Pi)$ be a proof tree of Π and $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$.

We say that (N, l) is *natural* if, whenever τ contains a node n whose rule is

$$\rho = E_1(x_0, x_1) \wedge E_2(x_1, x_2) \wedge \ldots \wedge E_n(x_{n-1}, x_n) \Rightarrow G^*(x_0, x_n)$$

- if $n \geq 2$;
- if $E_i = G^\diamond$ for some $i \in \{1, \ldots, n\}$;
- if, for some $j \in \{1, \ldots, i\}, v = x_{j-1}$ or $v \in var(\tau')$, where the head predicate in the root of τ' is $E_j(x_{j-1}, x_j)$;
- if $w = x_i$ or, for some $j \in \{i+1, \ldots, n\}, w = x_j$ or $v \in var(\tau'')$, where the head predicate in the root of τ'' is $E_j(x_{j-1}, x_j)$;

then $v \not\equiv_{(N,l')} w$, where $l' = (\tau, S \backslash \{E_i(x_{i-1}, x_i)\}, \rho, V, \varphi)$. ∎

Another definition that may be useful it the notion of the distance from a variable to a node. In fact, it is the distance from the domain of the variable to the considered node. This definition will be helpful when the time will come to delay the splitting operation because all the interesting variables are in the same sub-tree.

Definition A.4.17.
Let τ be an unfolding tree of a DATALOG program Π, $v \in var(\tau)$ and $N \in \mathcal{N}_\tau$. The *distance* $d(v, N)$ from v to N is the least integer $n \in \mathbb{N}$ such that $\exists N_0 = (R_0(\mathbf{t}_0), \rho_0), N_1 = (R_1(\mathbf{t}_1), \rho_1), \ldots, N_n = (R_n(\mathbf{t}_n), \rho_n) \in \mathcal{N}_\tau$ where $N = N_0$ and $v \in var(\rho_n)$. ∎

Now, we use immediately the notion of distance defined just above, to create a mapping between positive instances and polynomials of $\mathbb{N}[X]$, which will allow us to run proofs by induction.

Definition A.4.18.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. Let $\Sigma \in p\_trees(Q, \Pi)$ be a proof tree of Π, $\kappa = \mathcal{U}(\Sigma) \in u_{trees}(Q, \Pi)$, $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$ such that $\mathcal{I}_N(l) \equiv \top$ and (N, l, ϕ) a positive instance of (N, l).

Let $P(\mathbf{v})$ be an atom in S. The *weight* of $P(\mathbf{v})$ is the integer $\mathcal{W}_{P(\mathbf{v})}$ defined as follows:

- If P is a predicate $G^\diamond$, let be $z \in \mathbb{N}$ such that $G = P_z$. Then, $\mathcal{W}_{(N,l,\phi)}(P(\mathbf{v}))$ is the least integer $k \in \mathbb{N}$ such that $\exists w_0, w_1, \ldots, w_k \in var(\kappa)$ where $G^*(\phi(\mathbf{w})) = G^*(w_0, w_k)$ and
$$\top \equiv (\forall \mathbf{P}_1, \ldots, \mathbf{P}_n)\,(\forall v_1, \ldots, v_m)$$
$$\left(\left(\bigwedge_{i=1}^{\|\gamma\|} \mathcal{F}_i(\mathbf{P}_1, \ldots, \mathbf{P}_n) \wedge \bigwedge_{i=1}^{a} \mathbf{P}_{s_i}(\mathbf{v}_i) \right) \Rightarrow \bigwedge_{i=0}^{k-1} \mathbf{P}_z(w_i, w_{i+1}) \right)$$
- Otherwise, $\mathcal{W}_{(N,l,\phi)}(P(\mathbf{v})) = 1$.

The *weight* of (N, l, ϕ) is the polynomial of $\mathbb{N}[X]$

$$\mathcal{W}_{(N,l,\phi)}(X) = \sum_{P(\mathbf{v}) \in S} \left(\mathcal{W}_{(N,l,\phi)}(P(\mathbf{v}))^2 X^{d^\circ P} + \sum_{v \in \mathbf{v}} d(\phi(\Lambda_{\varphi,\kappa}(v)), N) X \right) + |\varphi(var_\diamond(\gamma^\diamond)) \cap var_\diamond(\gamma^\diamond)|$$

The notion of *weight* induces an ordering on positive instances. Therefore, if (N_1, l_1, ϕ_1) and (N_2, l_2, ϕ_2) are positive instances of couples $(N_1, l_1), (N_2, l_2) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$, we denote by $(N_1, l_1, \phi_1) \leq_{\mathcal{W}} (N_2, S_2, \phi_2)$ the inequality $\mathcal{W}_{(N_1,l_1,\phi_1)}(X) \leq \mathcal{W}_{(N_2,l_2,\phi_2)}(X)$. ∎

With the notion of *eccentricity*, introduced hereafter, we point out the existence of predicates involving variables that do not appear in the same sub-trees. These are the predicates of particular interest that were the subject of the discussion of the page 29.

Definition A.4.19.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. Let $\Sigma \in p\_trees(Q,\Pi)$ be a proof tree of Π, $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q,\Pi)$, $(N,l) = ((R(\mathbf{t}),\rho),(\tau,S,\rho,V,\varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma,\Pi)$ such that $\mathcal{I}_N(l) \equiv \top$, (N,l,ϕ) a positive instance of (N,l) and $\Lambda_{\varphi,\kappa}$ a κ-mapping.

Let $\mathfrak{N} \subseteq \mathcal{N}_\kappa$ be a set of nodes in κ, $\overline{\mathfrak{N}} = \mathcal{N}_\kappa \backslash \mathfrak{N} \subseteq \mathcal{N}_\kappa$, and $\mathfrak{N}_V \subseteq var(\kappa)$ (resp. $\overline{\mathfrak{N}}_V \subseteq var(\kappa)$) the set of variables that have an occurrence appearing in the rule of some node $n \in \mathfrak{N}$ (resp. $n \in \overline{\mathfrak{N}}$). The *eccentric part* of S is the set $S_\mathfrak{N}$ of atoms $G^\diamond(x,y) \in S$ that are in one of the following two cases:

- $\phi(\Lambda_{\varphi,\kappa}(x)) \notin \overline{\mathfrak{N}}_V$ and $\phi(\Lambda_{\varphi,\kappa}(y)) \notin \mathfrak{N}_V$.
- $\phi(\Lambda_{\varphi,\kappa}(x)) \notin \mathfrak{N}_V$ and $\phi(\Lambda_{\varphi,\kappa}(y)) \notin \overline{\mathfrak{N}}_V$.

(N,l,ϕ) is said *eccentric* relatively to $\mathfrak{N}$ if $S_\mathfrak{N} \neq \emptyset$. ∎

We define now the concept of *smooth partition* of a positive instance and of *smooth positive instance*, that are analogous to the concepts of natural partition and natural label introduced in the Definitions A.4.15 and A.4.16, but in the context of the existence of a sub-tree of particular interest.

Definition A.4.20.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. Let $\Sigma \in p\_trees(Q,\Pi)$ be a proof tree of Π, $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q,\Pi)$, $(N,l) = ((R(\mathbf{t}),\rho),(\tau,S,\rho,V,\varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma,\Pi)$ such that $\mathcal{I}_N(l) \equiv \top$, (N,l,ϕ) a positive instance of (N,l), $\Lambda_{\varphi,\kappa}$ a κ-mapping, $\mathfrak{N} \subseteq \mathcal{N}_\kappa$ be a set of nodes in κ, $\overline{\mathfrak{N}} = \mathcal{N}_\kappa \backslash \mathfrak{N} \subseteq \mathcal{N}_\kappa$, and $\mathfrak{N}_V \subseteq var(\kappa)$ (resp. $\overline{\mathfrak{N}}_V \subseteq var(\kappa)$ the set of variables that have an occurrence appearing in the rule of some node $n \in \mathfrak{N}$ (resp. $n \in \overline{\mathfrak{N}}$).

We define the symmetric relationship $\approx_{(N,l,\phi),\mathfrak{N}}$ between variables $a,b \in \varphi^{-1}(var_\diamond(\gamma^\diamond))$ by: $a \approx_{(N,l,\phi),\mathfrak{N}} b \Leftrightarrow \exists P(\mathbf{v}) \in S \backslash S_\mathfrak{N}, \exists c,d \in var(\mathbf{v})$ such that $\varphi(a) = \varphi(c)$ and $\varphi(b) = \varphi(d)$. Then, we define the equivalence relationship $\equiv_{(N,l,\phi),\mathfrak{N}}$ between variables $a,b \in \varphi^{-1}(var_\diamond(\gamma^\diamond))$ as the reflexive transitive closure of $\approx_{(N,l,\phi),\mathfrak{N}}$. This lets us define classes of equivalence $\mathcal{C}_1,\ldots,\mathcal{C}_k \subseteq \varphi^{-1}(var_\diamond(\gamma^\diamond))$ such that $\forall i \in \{1,\ldots,k\}, \forall a \in \mathcal{C}_i, \forall b \in \varphi^{-1}(var_\diamond(\gamma^\diamond)), b \in \mathcal{C}_i \Leftrightarrow a \equiv_{(N,l,\phi),\mathfrak{N}} b$.

We can now split S into sets $S_0,S_1,\ldots,S_k \subseteq S$ such that $S_0 = \{P(\mathbf{v}) \in S_i | var(\mathbf{v}) \subseteq V\}$ and $\forall i \in \{1,\ldots,k\}, S_i = \{P(\mathbf{v}) \in S_i | var(\mathbf{v}) \cap \mathcal{C}_i \neq \emptyset\}$. This partition $(S_0,\ldots,S_k)$ of S is called *smooth partition* of (N,l,ϕ) relatively to $\mathfrak{N}$. ∎

Definition A.4.21.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. Let $\Sigma \in p\_trees(Q,\Pi)$ be a proof tree of Π, $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q,\Pi)$, $(N,l) = ((R(\mathbf{t}),\rho),(\tau,S,\rho,V,\varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma,\Pi)$ such that $\mathcal{I}_N(l) \equiv \top$, (N,l,ϕ) a positive instance of (N,l), $\Lambda_{\varphi,\kappa}$ a κ-mapping, $\mathfrak{N} \subseteq \mathcal{N}_\kappa$ be a set of nodes in κ, $\overline{\mathfrak{N}} = \mathcal{N}_\kappa \backslash \mathfrak{N} \subseteq \mathcal{N}_\kappa$, and $\mathfrak{N}_V \subseteq var(\kappa)$ (resp. $\overline{\mathfrak{N}}_V \subseteq var(\kappa)$) the set of variables that have an occurrence appearing in the rule of some node $n \in \mathfrak{N}$ (resp. $n \in \overline{\mathfrak{N}}$).

We say that (N, l, ϕ) is *smooth* relatively to $\mathfrak{N}$ if, whenever τ contains a node n whose rule is

$$\rho = E_1(x_0, x_1) \wedge E_2(x_1, x_2) \wedge \ldots \wedge E_n(x_{n-1}, x_n) \Rightarrow G^*(x_0, x_n)$$

- if $n \geq 2$;
- if $E_i = G^\diamond$ for some $i \in \{1, \ldots, n\}$;
- if, for some $j \in \{1, \ldots, i\}, v = x_{j-1}$ or $v \in var(\tau')$, where the head predicate in the root of τ' is $E_j(x_{j-1}, x_j)$;
- if $w = x_i$ or, for some $j \in \{i+1, \ldots, n\}, w = x_j$ or $v \in var(\tau'')$, where the head predicate in the root of τ'' is $E_j(x_{j-1}, x_j)$;

then $v \not\equiv_{(N,l',\phi),\mathfrak{N}} w$, where $l' = (\tau, S \backslash \{E_i(x_{i-1}, x_i)\}, \rho, V, \varphi)$. ∎

Then, we introduce the concept of *regularisable positive instances*, which denote the positive instances for which, in every diamond-rule containing two diamond-predicates, the variables of one of the diamond-predicates are in the considered sub-tree and the variables of the other diamond-predicate are not.

Definition A.4.22.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. Let $\Sigma \in p\_trees(Q, \Pi)$ be a proof tree of Π, $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q, \Pi)$, $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$ such that $\mathcal{I}_N(l) \equiv \top$, (N, l, ϕ) a positive instance of (N, l), $\Lambda_{\varphi,\kappa}$ a κ-mapping, $\mathfrak{N} \subseteq \mathcal{N}_\kappa$ be a set of nodes in κ, $\overline{\mathfrak{N}} = \mathcal{N}_\kappa \backslash \mathfrak{N} \subseteq \mathcal{N}_\kappa$, and $\mathfrak{N}_V \subseteq var(\kappa)$ (resp. $\overline{\mathfrak{N}}_V \subseteq var(\kappa)$) the set of variables that have an occurrence appearing in the rule of some node $n \in \mathfrak{N}$ (resp. $n \in \overline{\mathfrak{N}}$).

Let us assume that

$$\exists n = (G^*(x_0, x_k), E_1(x_0, x_1) \wedge E_2(x_1, x_2) \wedge \ldots \wedge E_k(x_{k-1}, x_k) \Rightarrow G^*(x_0, x_k)) \in \mathcal{N}_\tau$$

such that $\exists i, j \in \{1, \ldots, n\}$ verifying

- $i < j$ and $E_i = E_j = G^\diamond$
- $E_i(x_{i-1}, x_i), E_j(x_{j-1}, x_j) \in S$

If we are in none of the following cases:

- $\phi(\Lambda_{\varphi,\kappa}(x_{i-1})), \phi(\Lambda_{\varphi,\kappa}(x_i)) \in \mathfrak{N}_V$ and $\phi(\Lambda_{\varphi,\kappa}(x_{j-1})), \phi(\Lambda_{\varphi,\kappa}(x_j)) \in \overline{\mathfrak{N}}_V$.
- $\phi(\Lambda_{\varphi,\kappa}(x_{i-1})), \phi(\Lambda_{\varphi,\kappa}(x_i)) \in \overline{\mathfrak{N}}_V$ and $\phi(\Lambda_{\varphi,\kappa}(x_{j-1})), \phi(\Lambda_{\varphi,\kappa}(x_j)) \in \mathfrak{N}_V$.

or if we are not in the two following cases:

- $\phi(\Lambda_{\varphi,\kappa}(x_{i-1})), \phi(\Lambda_{\varphi,\kappa}(x_j)) \notin \mathfrak{N}_V \cap \overline{\mathfrak{N}}_V$.
- if $j \neq i+1$, then $\phi(\Lambda_{\varphi,\kappa}(x_i)), \phi(\Lambda_{\varphi,\kappa}(x_{j-1})) \notin \mathfrak{N}_V \cap \overline{\mathfrak{N}}_V$.

then (N, l, ϕ) is said to be *not regularisable* relatively to $\mathfrak{N}$.

Every positive instance (N, l, ϕ) that is not in such a case is said to be *regularisable* relatively to $\mathfrak{N}$. ∎

Now, we define a *smoothing mapping*, which is an application that transforms a positive instance (N, l, ϕ) in a positive instance (N, l_k, ϕ_k) such that $(N, l) \in \Psi^{\omega}_{\Sigma}(\{(N, l_k)\})$, $(N, l_k, \phi_k) \leq_{\mathcal{W}} (N, l, \phi)$ and (N, l_k, ϕ_k) is not eccentric relatively to a given sub-tree.

Definition A.4.23.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. Let $\Sigma \in p\_trees(Q, \Pi)$ be a proof tree of Π, $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q, \Pi)$, $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \mathcal{N}_{\Sigma} \times \mathcal{L}(\gamma, \Pi)$ such that $\mathcal{I}_N(l) \equiv \top$, $\Lambda_{\varphi,\kappa}$ a κ-mapping, $\mathfrak{N} \subseteq \mathcal{N}_{\kappa}$, $\overline{\mathfrak{N}} = \mathcal{N}_{\kappa} \backslash \mathfrak{N} \subseteq \mathcal{N}_{\kappa}$ and (N, l, ϕ) a positive instance of (N, l) which is smooth and regularisable relatively to $\mathfrak{N}$.

If (N, l, ϕ) is not eccentric relatively to $\mathfrak{N}$, we set $\mathfrak{R}_{\mathfrak{N}}((N, l, \phi)) = (N, l, \phi)$.

If (N, l, ϕ) is eccentric relatively to $\mathfrak{N}$, let $\mathcal{G}^{\diamond}(x, y)$ be an atom in $S_{\mathfrak{N}}$. Let be $z \in \mathbb{N}$ such that $G = P_z$ and $k = \mathcal{W}_{(N,l,\phi)}(G^{\diamond}(x, y))$. $\exists w_0, \ldots, w_k \in var(\kappa)$ such that $w_0 = \phi(\Lambda_{\varphi,\kappa}(x)), w_k = \phi(\Lambda_{\varphi,\kappa}(y))$ and

$$\top \equiv (\forall \mathbf{P}_1, \ldots, \mathbf{P}_n)\, (\forall v_1, \ldots, v_m)$$

$$\left(\left(\bigwedge_{i=1}^{\|\gamma\|} \mathcal{F}_i(\mathbf{P}_1, \ldots, \mathbf{P}_n) \wedge \bigwedge_{i=1}^{a} \mathbf{P}_{s_i}(\mathbf{v}_i) \right) \Rightarrow \bigwedge_{i=0}^{k-1} \mathbf{P}_z(w_i, w_{i+1}) \right)$$

- If $\exists i \in \{0, \ldots, k-1\}$ such that $w_i \in \mathfrak{N}_V \cap \overline{\mathfrak{N}}_V$, we choose some $z \in var_{\diamond}(\gamma^{\diamond}) \backslash var(\tau)$. Let ρ_0 be a rule containing $\mathcal{G}^{\diamond}(x, y)$ and $n_0 \in \mathcal{N}_{\tau}$ the node whose rule is ρ_0. We define:
  - The set $S_1 = \{G^{\diamond}(x, z), G^{\diamond}(z, y)\} \cup S \backslash \{G^{\diamond}(x, y)\}$.
  - The rule ρ_1, obtained from ρ_0 by replacing, in the body of ρ_0, the atom $G^{\diamond}(x, y)$ by the conjunction of atoms $G^{\diamond}(x, z), G^{\diamond}(z, y)$.
  - The tree $\tau_1 \in u\_trees(\Gamma, \gamma^{\nabla})$, obtained from τ by replacing the node n_0 by a node n_1 whose rule is ρ_1.
  - A total mapping $\varphi_1 : var_{\diamond}(\gamma^{\diamond}) \to var_{\diamond}(\gamma^{\diamond}) \cup V$, such that $\varphi_{1|var(S)} = \varphi_{|var(S)}$, $\varphi(z) \in var_{\diamond}(\gamma^{\diamond})$ and $\varphi(z) \notin \varphi(var(\tau))$.
  - The couple $(N, l_1) \in \mathcal{N}_{\Sigma} \times \mathcal{L}(\gamma, \Pi)$, where $l_1 = (\tau_1, S_1, \rho, V, \varphi_1)$.
  - A total mapping $\phi_1 : var(\kappa) \cup var_{\diamond}(\gamma^{\diamond}) \to var(\kappa)$, such that $\phi_{1|var(\kappa)} = Id_{var(\kappa)}$, $\phi_{1|var(S)} = \phi_{|var(S)}$ and $\phi_1(\varphi(z)) = w_i$.

  It appears that (N, l_1, ϕ_1) is a positive instance of (N, l_1), which is why $\mathcal{I}_N(l_1) \equiv \top$. We set $\mathfrak{R}_{\mathfrak{N}}((N, l, \phi)) = (N, l_1, \phi_1)$.
  By rule 12., $(N, (\tau, S, \rho, V, \varphi_1)) \in \Psi^{\omega}_{\Sigma}(\{(N, l_1)\})$.
  By rule 9., $(N, l) = (N, (\tau, S, \rho, V, \varphi)) \in \Psi^{\omega}_{\Sigma}(\{(N, l_1)\})$.
- If, $\forall i \in \{0, \ldots, k-1\}, w_i \notin \mathfrak{N}_V \cap \overline{\mathfrak{N}}_V$, we still know that $\exists i \in \{0, \ldots, k-1\}$ such that
  - if $w_0 \in \mathfrak{N}_V$ and $w_k \in \overline{\mathfrak{N}}_V$, then $w_i \in \mathfrak{N}_V$ and $w_{i+1} \in \overline{\mathfrak{N}}_V$
  - if $w_0 \in \overline{\mathfrak{N}}_V$ and $w_k \in \mathfrak{N}_V$, then $w_i \in \overline{\mathfrak{N}}_V$ and $w_{i+1} \in \mathfrak{N}_V$

  Let ρ_0 be a rule containing $\mathbf{G}^{\diamond}(x, y)$ and $n_0 \in \mathcal{N}_{\tau}$ the node whose rule is ρ_0. γ contains an instance of a star-free rule $\mathcal{R} = E_1(\mathbf{v}^1) \wedge \ldots \wedge E_k(\mathbf{v}^k) \wedge G^*_1(x^1_1, x^1_2) \wedge \ldots \wedge G^*_l(x^l_1, x^l_2) \Rightarrow G(x^0_1, x^0_2)$ such that $\exists S_1, S_2, S_3, \rho_1, \tau_1, \tau_2, \varphi_1, \phi_1$ satisfying:

- $var(\mathcal{R}) \cap var(\tau) = \emptyset$.
- $S_1 = \{G^\diamond(x, x_1^0), G^\diamond(x_2^0, y), E_1(\mathbf{v}^1), \ldots, E_k(\mathbf{v}^k), G_1^*(x_1^1, x_2^1), \ldots, G_l^*(x_1^l, x_2^l)\}$.
- $S_2 = S_1 \cup S \backslash \{G^\diamond(x, y)\}$.
- $S_3 = \{G(x_1^0, x_2^0), G^\diamond(x, x_1^0), G^\diamond(x_2^0, y)\} \cup S \backslash \{G^\diamond(x, y)\}$.
- ρ_1 is obtained from ρ_0 by replacing, in the body of ρ_0, the atom $G^\diamond(x, y)$ by the conjunction of atoms $G^\diamond(x, x_1^0), G(x_1^0, x_2^0), G^\diamond(x_2^0, y)$.
- $\tau_2 \in u\_trees(G, \gamma^\nabla)$ has a root n_2 whose rule is $\mathcal{R}$ and l other nodes $n_1', \ldots, n_l'$ which are leaves, children of n_2, and whose rules are $G_i^\diamond(x_1^i, x_2^i) \Rightarrow G_i^*(x_1^i, x_2^i)$.
- $\tau_1 \in u\_trees(\Gamma, \gamma^\nabla)$ is the tree built from τ by replacing the node n_0 by a node n_1 whose rule is ρ_1, and by giving to n_1, in addition to the children of n_0, a new child, which is n_2.
- $\varphi_1 : var_\diamond(\gamma^\diamond) \to var_\diamond(\gamma^\diamond) \cup V$ verifies $\varphi_{1|var(S)} = \varphi_{|var(S)}$, $\varphi_1(v) \in var_\diamond(\gamma^\diamond)$ if $v \notin var(S)$ and $\forall v, v' \in var_\diamond(\gamma^\diamond), (v \neq v' \wedge \varphi_1(v) = \varphi_1(v')) \Rightarrow v, v' \in var(S)$.
- $(N, l_1) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$, where $l_1 = (\tau_1, S_2, \rho, V, \varphi_1)$, and $\mathcal{I}_N(l_1) \equiv \top$.
- $\phi_1 : var(\kappa) \cup var_\diamond(\gamma^\diamond) \to var(\kappa)$ verifies $\phi_{1|var(\kappa)} = Id_{var(\kappa)}$, $\phi_{1|var(S)} = \phi_{|var(S)}$, $\phi_1(x_1^0) = w_i$ and $\phi_1(x_2^0) = w_{i+1}$.
- (N, l_1, ϕ_1) is a positive instance of (N, l_1).

Then, we set $\mathfrak{R}_\mathfrak{N}((N, l, \phi)) = (N, l_1, \phi_1)$.
By rule 10., $(N, (\tau_1, S_2 \cup \{G(x_1^0, x_2^0)\}, \rho, V, \varphi_1)) \in \Psi_\Sigma^\omega(\{(N, l_1)\})$.
By rule 4, $(N, (\tau_1, S_3, \rho, V, \varphi_1)) \in \Psi_\Sigma^\omega(\{(N, l_1)\})$.
By rules 11., 12. and 12. again, $(N, (\tau, S, \rho, V, \varphi_1)) \in \Psi_\Sigma^\omega(\{(N, l_1)\})$.
By rule 9., $(N, l) = (N, (\tau, S, \rho, V, \varphi)) \in \Psi_\Sigma^\omega(\{(N, l_1)\})$.

It follows directly from this definition that $\mathfrak{R}_\mathfrak{N}((N, l, \phi))$ is regularisable and smooth relatively to $\mathfrak{N}$, and that $\mathfrak{R}_\mathfrak{N}((N, l, \phi)) \leq_\mathcal{W} (N, l, \phi)$, with equality if and only if (N, l, ϕ) is not eccentric relatively to $\mathfrak{N}$. Let now (N, l_i, ϕ_i) be the sequence defined by:

- $(N, l_0, \phi_0) = (N, l, \phi)$.
- $\forall i \in \mathbb{N}, (N, l_{i+1}, \phi_{i+1}) = \mathfrak{R}_\mathfrak{N}((N, l_i, \phi_i))$.

We know that $\exists k \in \mathbb{N}$ such that $\forall i \geq k, (N, l_i, \phi_i) = (N, l_k, \phi_k)$. The *smoothing mapping* relatively to $\mathfrak{N}$ is the partial mapping $\mathfrak{S}_\mathfrak{N}$ such that $\mathfrak{S}_\mathfrak{N}((N, l, \phi)) = (N, l_k, \phi_k)$.

It is straightforward that $(N, l_k, \phi_k) = \mathfrak{S}_\mathfrak{N}((N, l, \phi))$ is regularisable and smooth relatively to $\mathfrak{N}$, that $(N, l) \in \Psi_\Sigma^\omega(\{(N, l_k)\})$ and that $\mathfrak{S}_\mathfrak{N}((N, l, \phi)) \leq_\mathcal{W} (N, l, \phi)$, with equality if and only if (N, l, ϕ) is not eccentric relatively to $\mathfrak{N}$. ∎

A very interesting class of labels is the class of labels that, locally, look like labels $(\tau, S, \rho, V, \varphi)$ whose tree τ would be in $u\_trees(\Gamma, \gamma^\nabla)$. Indeed, this local property has effects on a global way. The only problem is that, in order to avoid an explosion of the number of indices used, I had to work on a disjunction of cases, which remained possible here, but appears to be particularly heavy to manipulate. However, at least, the properties that will be proved later in this section, so far they concern the hereafter defined *locally triangular labels*, will often be quite easy to prove, since each case of the disjunction is quite simple.

Definition A.4.24.
Let Π be a DATALOG Program with goal predicate Q and γ a transitive DATALOG program. Let $\Sigma \in p\_trees(Q, \Pi)$ be a proof tree of Π and $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$.

(N, l) is said do be *locally triangular* if τ does not contain any rule ρ' containing more than 2 diamond predicates, and if the 8 following properties are verified:

1. If the rule ρ' of a node $n \in \mathcal{N}_\tau$ is of the type

$$G(x, z_1) \wedge G^\diamond(z_1, z_2) \wedge G(z_2, z_3) \wedge G^\diamond(z_3, z_4), G(z_4, y) \Rightarrow G^*(x, y)$$

and $G(x, z_1), G(z_2, z_3), G(z_4, y)$ are respectively the atoms in the head of the children n_1, n_2, n_3 of n, then we are in one of the following 2 cases:
   - $var(\tau_{(n_1)}) \cap var(S) = \emptyset$ and $z_2 \notin var(S)$.
   - $var(\tau_{(n_3)}) \cap var(S) = \emptyset$ and $z_3 \notin var(S)$.

2. If the rule ρ' of a node $n \in \mathcal{N}_\tau$ is of the type

$$G^\diamond(x, z_1) \wedge G(z_1, z_2) \wedge G^\diamond(z_2, z_3) \wedge G(z_3, y) \Rightarrow G^*(x, y)$$

and $G(z_1, z_2), G(z_3, y)$ are respectively the atoms in the head of the children n_1, n_2 of n, then we are in one of the following 2 cases:
   - $z_1 \notin var(S)$.
   - $var(\tau_{(n_2)}) \cap var(S) = \emptyset$ and $z_2 \notin var(S)$.

3. If the rule ρ' of a node $n \in \mathcal{N}_\tau$ is of the type

$$G(x, z_1) \wedge G^\diamond(z_1, z_2) \wedge G(z_2, z_3) \wedge G^\diamond(z_3, y) \Rightarrow G^*(x, y)$$

and $G(x, z_1), G(z_2, z_3)$ are respectively the atoms in the head of the children n_1, n_2 of n, then we are in one of the following 2 cases:
   - $var(\tau_{(n_1)}) \cap var(S) = \emptyset$ and $z_2 \notin var(S)$.
   - $z_3 \notin var(S)$.

4. If the rule ρ' of a node $n \in \mathcal{N}_\tau$ is of the type

$$G^\diamond(x, z_1) \wedge G(z_1, z_2) \wedge G^\diamond(z_2, y) \Rightarrow G^*(x, y)$$

and $G(z_1, z_2)$ is the atom in the head of the child n_1 of n, then we are in one of the following 2 cases:
   - $z_1 \notin var(S)$.
   - $z_2 \notin var(S)$.

5. If the rule ρ' of a node $n \in \mathcal{N}_\tau$ is of the type

$$G(x, z_1) \wedge G^\diamond(z_1, z_2) \wedge G^\diamond(z_2, z_3) \wedge G(z_3, y) \Rightarrow G^*(x, y)$$

and $G(x, z_1), G(z_2, y)$ are respectively the atoms in the head of the children n_1, n_2 of n, then we are in one of the following 2 cases:
   - $var(\tau_{(n_1)}) \cap var(S) = \emptyset$.
   - $var(\tau_{(n_2)}) \cap var(S) = \emptyset$.

6. If the rule ρ' of a node $n \in \mathcal{N}_\tau$ is of the type

$$G^\diamond(x, z_1) \wedge G^\diamond(z_1, z_2) \wedge G(z_2, y) \Rightarrow G^*(x, y)$$

and $G(z_2, y)$ is the atom in the head of the child n_1 of n, then we are in one of the following 2 cases:

- $x \notin var(S)$.
- $var(\tau_{(n_1)}) \cap var(S) = \emptyset$.

7. If the rule ρ' of a node $n \in \mathcal{N}_\tau$ is of the type

$$G(x, z_1) \wedge G^\diamond(z_1, z_2) \wedge G^\diamond(z_2, y) \Rightarrow G^*(x, y)$$

and $G(x, z_1)$ is the atom in the head of the child n_1 of n, then we are in one of the following 2 cases:

- $var(\tau_{(n_1)}) \cap var(S) = \emptyset$.
- $y \notin var(S)$.

8. If the rule ρ' of a node $n \in \mathcal{N}_\tau$ is of the type

$$G^\diamond(x, z_1) \wedge G^\diamond(z_1, y) \Rightarrow G^*(x, y)$$

then we are in one of the following 2 cases:

- $x \notin var(S)$.
- $y \notin var(S)$.

∎

Now, as I told just above, lies a global property on locally triangular labels:

Proposition A.4.25.
Let Π *be a* Datalog *Program with goal predicate* Q *and* γ *a transitive* Datalog *program with goal predicate* Γ. *Let* $\Sigma \in p\_trees(Q, \Pi)$ *be a proof tree of* Π, (N, l) *be a label* $(N, l) = ((R(\boldsymbol{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$ *such that* $\mathcal{I}_N(l) \equiv \top$, *and* (N, l, ϕ) *a positive instance of* (N, l).

If (N, l) *is locally triangular,* $\exists \tau' \in u\_trees(\Gamma, \gamma^\nabla)$ *such that* $l' = (\tau', S, \rho, V, \varphi)$ *verify* $(N, l') \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$, $\mathcal{I}_N(l') \equiv \top$ *and* (N, l', ϕ) *is a positive instance of* (N, l'). ∎

∎

Proof. Let us assume that $(N, (\tau', S, \rho, V, \varphi))$ is a triangular label for some tree $\tau' \in u\_trees(\Gamma, \gamma)$ which contains $k \geq 1$ occurrences of diamond rules where 2 diamond predicates are present. Let n be a node whose rule ρ' is such a rule.

1. If ρ' is of the type

$$G(x, z_1) \wedge G^\diamond(z_1, z_2) \wedge G(z_2, z_3) \wedge G^\diamond(z_3, z_4) \wedge G(z_4, y) \Rightarrow G^*(x, y)$$

and $G(x, z_1), G(z_2, z_3), G(z_4, y)$ are respectively the atoms in the head of the children n_1, n_2, n_3 of n, then we are in one of the following 2 cases:

- $var(\tau_{(n_1)}) \cap var(S) = \emptyset$ and $z_2 \notin var(S)$.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G(x, z_3) \wedge G^{\diamond}(z_3, z_4) \wedge G(z_4, y) \Rightarrow G^*(x, y)$, by erasing $\tau'_{(n_1)}$ and by replacing every occurrence of z_2 in $\tau'_{(n_2)}$ by an occurrence of x.
- $var(\tau_{(n_3)}) \cap var(S) = \emptyset$ and $z_3 \notin var(S)$.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G(x, z_1) \wedge G^{\diamond}(z_1, z_2) \wedge G(z_2, y) \Rightarrow G^*(x, y)$, by erasing $\tau'_{(n_3)}$ and by replacing every occurrence of z_3 in $\tau'_{(n_2)}$ by an occurrence of y.

2. If ρ' is of the type

$$G^{\diamond}(x, z_1) \wedge G(z_1, z_2) \wedge G^{\diamond}(z_2, z_3) \wedge G(z_3, y) \Rightarrow G^*(x, y)$$

and $G(z_1, z_2), G(z_3, y)$ are respectively the atoms in the head of the children n_1, n_2 of n, then we are in one of the following 2 cases:

- $z_1 \notin var(S)$.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G(x, z_2) \wedge G^{\diamond}(z_3, z_3) \wedge G(z_3, y) \Rightarrow G^*(x, y)$ and by replacing every occurrence of z_1 in $\tau'_{(n_1)}$ by an occurrence of x.
- $var(\tau_{(n_2)}) \cap var(S) = \emptyset$ and $z_2 \notin var(S)$.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G^{\diamond}(x, z_1) \wedge G(z_1, y) \Rightarrow G^*(x, y)$, by erasing $\tau'_{(n_2)}$ and by replacing every occurrence of z_2 in $\tau'_{(n_1)}$ by an occurrence of y.

3. If ρ' is of the type

$$G(x, z_1) \wedge G^{\diamond}(z_1, z_2) \wedge G(z_2, z_3) \wedge G^{\diamond}(z_3, y) \Rightarrow G^*(x, y)$$

and $G(x, z_1), G(z_2, z_3)$ are respectively the atoms in the head of the children n_1, n_2 of n, then we are in one of the following 2 cases:

- $var(\tau_{(n_1)}) \cap var(S) = \emptyset$ and $z_2 \notin var(S)$.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G(x, z_3) \wedge G^{\diamond}(z_3, y) \Rightarrow G^*(x, y)$, by erasing $\tau'_{(n_1)}$ and by replacing every occurrence of z_2 in $\tau'_{(n_2)}$ by an occurrence of x.
- $z_3 \notin var(S)$.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G(x, z_1) \wedge G^{\diamond}(z_1, z_2) \wedge G(z_2, y) \Rightarrow G^*(x, y)$ and by replacing every occurrence of z_3 in $\tau'_{(n_2)}$ by an occurrence of y.

4. If ρ' is of the type

$$G^{\diamond}(x, z_1) \wedge G(z_1, z_2) \wedge G^{\diamond}(z_2, y) \Rightarrow G^*(x, y)$$

and $G(z_1, z_2)$ is the atom in the head of the child n_1 of n, then we are in one of the following 2 cases:

- $z_1 \notin var(S)$.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node whose rule is $G(x, z_2) \wedge G^{\diamond}(z_2, y) \Rightarrow G^*(x, y)$ and by replacing every occurrence of z_1 in $\tau'_{(n_1)}$ by an occurrence of x.

- $z_2 \notin var(S)$.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G^{\diamond}(x, z_1) \wedge G(z_1, y) \Rightarrow G^{*}(x, y)$ and by replacing every occurrence of z_2 in $\tau'_{(n_1)}$ by an occurrence of y.

5. If ρ' is of the type

$$G(x, z_1) \wedge G^{\diamond}(z_1, z_2) \wedge G^{\diamond}(z_2, z_3) \wedge G(z_3, y) \Rightarrow G^{*}(x, y)$$

and $G(x, z_1), G(z_2, y)$ are respectively the atoms in the head of the children n_1, n_2 of n, then we are in one of the following 2 cases:

- $var(\tau_{(n_1)}) \cap var(S) = \emptyset$.
  Let $\tau_1 \in u\_trees(G, \gamma^{\nabla})$ such that $var(\tau_1) \cap var(\tau') = \{x, z_2\}$ and such that $G(x, z_2)$ is the atom in the head of the rule of the root n'_1 of τ_1.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G(x, z_2) \wedge G^{\diamond}(z_2, z_3) \wedge G(z_3, y) \Rightarrow G^{*}(x, y)$ and by setting n'_1 to be a new child of n'.
- $var(\tau_{(n_2)}) \cap var(S) = \emptyset$.
  Let $\tau_1 \in u\_trees(G, \gamma^{\nabla})$ such that $var(\tau_1) \cap var(\tau') = \{z_2, y\}$ and such that $G(z_2, y)$ is the atom in the head of the rule of the root n'_1 of τ_1.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G(x, z_1) \wedge G^{\diamond}(z_1, z_2) \wedge G(z_2, y) \Rightarrow G^{*}(x, y)$ and by setting n'_1 to be a new child of n'.

6. If ρ' is of the type

$$G^{\diamond}(x, z_1) \wedge G^{\diamond}(z_1, z_2) \wedge G(z_2, y) \Rightarrow G^{*}(x, y)$$

and $G(z_2, y)$ is the atom in the head of the child n_1 of n, then we are in one of the following 2 cases:

- $x \notin var(S)$.
  Let $\tau_1 \in u\_trees(G, \gamma^{\nabla})$ such that $var(\tau_1) \cap var(\tau') = \{x, z_1\}$ and such that $G(x, z_1)$ is the atom in the head of the rule of the root n'_1 of τ_1.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G(x, z_1) \wedge G^{\diamond}(z_1, z_2) \wedge G(z_2, y) \Rightarrow G^{*}(x, y)$ and by setting n'_1 to be a new child of n'.
- $var(\tau_{(n_1)}) \cap var(S) = \emptyset$.
  Let $\tau_1 \in u\_trees(G, \gamma^{\nabla})$ such that $var(\tau_1) \cap var(\tau') = \{z_1, y\}$ and such that $G(z_1, y)$ is the atom in the head of the rule of the root n'_1 of τ_1.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G^{\diamond}(x, z_1) \wedge G(z_1, y) \Rightarrow G^{*}(x, y)$ and by setting n'_1 to be a new child of n'.

7. If ρ' is of the type

$$G(x, z_1) \wedge G^{\diamond}(z_1, z_2) \wedge G^{\diamond}(z_2, y) \Rightarrow G^{*}(x, y)$$

and $G(x, z_1)$ is the atom in the head of the child n_1 of n, then we are in one of the following 2 cases:

- $var(\tau_{(n_1)}) \cap var(S) = \emptyset$.
  Let $\tau_1 \in u\_trees(G, \gamma^{\nabla})$ such that $var(\tau_1) \cap var(\tau') = \{x, z_2\}$ and such that $G(x, z_2)$ is the atom in the head of the rule of the root n'_1 of τ_1.

Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G(x, z_2) \wedge G^{\diamond}(z_2, y) \Rightarrow G^*(x, y)$ and by setting n'_1 to be a new child of n'.

- $y \notin var(S)$.
  Let $\tau_1 \in u\_trees(G, \gamma^{\nabla})$ such that $var(\tau_1) \cap var(\tau') = \{z_2, y\}$ and such that $G(z_2, y)$ is the atom in the head of the rule of the root n'_1 of τ_1.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G(x, z_1) \wedge G^{\diamond}(z_1, z_2) \wedge G(z_2, y) \Rightarrow G^*(x, y)$ and by setting n'_1 to be a new child of n'.

8. If ρ' is of the type
$$G^{\diamond}(x, z_1) \wedge G^{\diamond}(z_1, y) \Rightarrow G^*(x, y)$$
then we are in one of the following 2 cases:

- $x \notin var(S)$.
  Let $\tau_1 \in u\_trees(G, \gamma^{\nabla})$ such that $var(\tau_1) \cap var(\tau') = \{x, z_1\}$ and such that $G(x, z_1)$ is the atom in the head of the rule of the root n'_1 of τ_1.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G(x, z_1) \wedge G^{\diamond}(z_1, y) \Rightarrow G^*(x, y)$ and by setting n'_1 to be a new child of n'.
- $y \notin var(S)$.
  Let $\tau_1 \in u\_trees(G, \gamma^{\nabla})$ such that $var(\tau_1) \cap var(\tau') = \{z_1, y\}$ and such that $G(z_1, y)$ is the atom in the head of the rule of the root n'_1 of τ_1.
  Let $\tau'' \in u\_trees(\Gamma, \gamma)$ be the tree built from τ' by replacing n by a node n' whose rule is $G^{\diamond}(x, z_1) \wedge G(z_1, y) \Rightarrow G^*(x, y)$ and by setting n'_1 to be a new child of n'.

In each case, we have built a tree $\tau'' \in u\_trees(\Gamma, \gamma)$ such that $l'' = (\tau'', S, \rho, V, \varphi)$ verify $(N, l'') \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$, (N, l'') be locally triangular, $\mathcal{I}_N(l'') \equiv \top$, (N, l'', ϕ) be a positive instance of (N, l'') and τ'' contain at most $k - 1$ occurrences of diamond rules where 2 diamond predicates are present.

Thus, by a direct induction, $\exists \tau''' \in u\_trees(\Gamma, \gamma)$ such that $l''' = (\tau''', S, \rho, V, \varphi)$ verify $(N, l''') \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$, (N, l''') be locally triangular, $\mathcal{I}_N(l''') \equiv \top$, (N, l''', ϕ) be a positive instance of (N, l''') and τ''' contain no occurrence of diamond rules where 2 diamond predicates are present. This means that $\tau''' \in u\_trees(\Gamma, \gamma^{\nabla})$, which was the statement of the above proposition. □

Here comes the last definition of this part of the proof. In order to prove that the labels that must be contained in every fixpoint γ-labelled proof tree are sufficiently numerous, we look at a subset of those labels that are not committed to be contained in every fixpoint γ-labelled proof tree. We want to show that this subset is empty, and to that extent we will successively find more and more properties of its supposed minimal element, to finally reach a point where these properties cannot hold at the same time, this showing that the minimal element does not exist.

Definition A.4.26.
Let Π be a DATALOG program with goal predicate Q and γ a transitive DATALOG program with goal predicate Γ. If $\Sigma \in p\_trees(Q, \Pi)$, let $\mathbb{S}(\Sigma)$ be the set of couples $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$ such that

- $\mathcal{I}_N(l) \equiv \top$.
- $\tau \in u\_trees(\Gamma, \gamma^\nabla)$.
- S is descendant-free in τ.
- (N, l) is natural.
- $(N, l) \notin \Psi^\omega_\Sigma(\emptyset)$.

If $\mathbb{S}(\Sigma) \neq \emptyset$, since $\mathbb{S}(\Sigma)$ is finite, we can, without ambiguity, select a couple $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) \in \mathbb{S}(\Sigma)$ such that (N, l) has a positive instance (N, l, ϕ) of minimal weight among the positive instances of elements in $\mathbb{S}(\Sigma)$. This couple is denoted $\min(\mathbb{S}(\Sigma))$. ∎

As said just above, here starts the length list of properties that must hold on the supposedly existing $\min(\mathbb{S}(\Sigma))$.

First, we prove that $\min(\mathbb{S}(\Sigma))$ does not contain any star-free IDB predicate:

Lemma A.4.27.
If $\Sigma \in p\_trees(Q, \Pi)$, $\mathbb{S}(\Sigma) \neq \emptyset$ and $(N, l) = ((R(\boldsymbol{t}), \rho), (\tau, S, \rho, V, \varphi)) = \min(\mathbb{S}(\Sigma))$, then S does not contain any star-free IDB predicate. ∎

Proof. We prove the above statement by *reductio ad absurdum*.

Let us assume that S contains an atom $P(\mathbf{v})$, where P is a star-free IDB predicate. Let n_0 be a node of τ whose rule r_0 contains an occurrence $\mathbf{P}(\mathbf{v})$ of $P(\mathbf{v})$ such that $\mathbf{P}(\mathbf{v})$ is not overwhelmed by S in τ. Let n be a child of n_0 such that $P(\mathbf{v})$ is the atom in the head of the rule r of n.

Since $\mathcal{I}_N(l) \equiv \top$, there exists an instance of a rule $\mathcal{R} = E_1(\mathbf{v}^1) \wedge \ldots \wedge E_k(\mathbf{v}^k) \wedge G^*_1(x^1_1, x^1_2) \wedge \ldots \wedge G^*_l(x^l_1, x^l_2) \Rightarrow P(\mathbf{v})$ in γ such that $\exists S_1, \tau_1, h, \varphi_1$ satisfying:

- $var(\mathcal{R}) \cap var(S) = var(\mathbf{v})$.
- $S_1 = \{E_1(\mathbf{v}^1), \ldots, E_k(\mathbf{v}^k), G^*_1(x^1_1, x^1_2), \ldots, G^*_l(x^l_1, x^l_2)\}$.
- $\tau_1 \in u\_trees(\Gamma, \gamma^\nabla)$ is the tree built from τ by replacing $\tau_{(n)}$ by a tree τ_2 such that $\mathcal{R}$ is the rule of the root n' of τ_2, $var(\tau_2) \cap var(\tau) = var(\mathbf{v})$ and every diamond-rule in τ_2 is of the type $G^\diamond(x, y) \Rightarrow P(\mathbf{v})$.
- $\varphi_1 : var_\diamond(\gamma^\diamond) \rightarrow var_\diamond(\gamma^\diamond) \cup V$ verifies $\varphi_{1|var(S)} = \varphi_{|var(S)}$, $\varphi_1(v) \in var_\diamond(\gamma^\diamond)$ if $v \notin var(S)$ and $\forall v, v' \in var_\diamond(\gamma^\diamond), (v \neq v' \wedge \varphi_1(v) = \varphi_1(v')) \Rightarrow v, v' \in var(S)$.
- $(N, l_1) \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$, where $l_1 = (\tau_1, S_1 \cup S \backslash \{P(\mathbf{v})\}, \rho, V', \varphi_1)$.
- $\mathcal{I}_N(l_1) \equiv \top$.

Let (N, l, ϕ) be a positive instance of (N, l) of minimal weight. (N, l_1) has a positive instance (N, l_1, ϕ) such that $(N, l_1, \phi) <_\mathcal{W} (N, l, \phi)$. Moreover, if $Q(\mathbf{u}) \in S_1$, then the occurrence $\mathbf{Q}(\mathbf{u})$ of $Q(\mathbf{u})$ is not overwhelmed by S in τ_1. Every atom $Q(\mathbf{u}) \in S \backslash \{P(\mathbf{v})\}$ had an occurrence $\mathbf{Q}(\mathbf{u})$ that was not overwhelmed by S in τ; therefore, this occurrence was not in $\tau_{(n)}$, and still exists in τ_1, being not overwhelmed by S in τ_1. Therefore, $S_1 \cup S \backslash \{P(\mathbf{v})\}$ is descendant-free in τ_1 and, since (N, l) is natural, so is (N, l_1). Therefore, $(N, l_1) \in \Psi^\omega_\Sigma(\emptyset)$.

Then, by rule 10., $(N, (\tau_1, S_1 \cup S, \rho, V, \varphi_1)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
By rule 4., $(N, (\tau_1, S, \rho, V, \varphi_1)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
By rule 6., $(N, (\tau, S, \rho, V, \varphi_1)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
By rule 9., $(N, (\tau, S, \rho, V, h \circ \varphi)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.

But, by definition of $\mathbb{S}(\Sigma)$, this is impossible. Therefore, our first assumption was false, which proves the statement of the above lemma. □

We prove now that $\min(\mathbb{S}(\Sigma))$ does not contain any star IDB predicate:

Lemma A.4.28.
If $\Sigma \in p\_trees(Q, \Pi)$, $\mathbb{S}(\Sigma) \neq \emptyset$ *and* $(N, l) = ((R(\boldsymbol{t}), \rho), (\tau, S, \rho, V, \varphi)) = \min(\mathbb{S}(\Sigma))$, *then* S *does not contain any star IDB predicate.* ∎

Proof. Similarly to how we proved the precedent lemma, we prove the above statement by *reductio ad absurdum*.

Let us assume that S contains an atom $G^*(v_1, v_2)$, where G^* is a star IDB predicate. Let n_0 be a node of τ whose rule r_0 contains an occurrence $\mathbf{G}^*(v_1, v_2)$ of $G^*(v_1, v_2)$ such that $\mathbf{G}^*(v_1, v_2)$ is not overwhelmed by S in τ. Let n be a child of n_0 such that $G^*(v_1, v_2)$ is the atom in the head of the rule r of n.

We introduce:

- a rule $\mathcal{R} = G^{\diamond}(v_1, v_2) \Rightarrow G^*(v_1, v_2)$ in γ^{∇}.
- the set $S_1 = \{G^{\diamond}(v_1, v_2)\}$.
- the tree $\tau_1 \in u\_trees(\Gamma, \gamma^{\nabla})$ built from τ by replacing $\tau_{(n)}$ by a tree τ_2 such that $\mathcal{R}$ is the rule of the root n' of τ_2 and every diamond-rule in τ_2 is of the type $G^{\diamond}(x, y) \Rightarrow G^*(x, y)$.
- the labelling $(N, l_1) \in \mathcal{N}_{\Sigma} \times \mathcal{L}(\gamma, \Pi)$, where $l_1 = (\tau_1, S_1 \cup S \backslash \{G^*(v_1, v_2)\}, \rho, V, \varphi)$.

Then, $\mathcal{I}_N(l_1) \equiv \mathcal{I}_N(l) \equiv \top$. Let (N, l, ϕ) be a positive instance of (N, l) of minimal weight. (N, l_1, ϕ) is a positive instance of (N, l_1) such that $(N, l_1, \phi) <_{\mathcal{W}} (N, l, \phi)$. Moreover, the occurrence $\mathbf{G}^{\diamond}(v_1, v_2)$ of $G^{\diamond}(v_1, v_2)$ is not overwhelmed by S in τ_1. Every atom $Q(\mathbf{u}) \in S \backslash \{G^*(v_1, v_2)\}$ had an occurrence $\mathbf{Q}(\mathbf{u})$ that was not overwhelmed by S in τ; therefore, this occurrence was not in $\tau_{(n)}$, and still exists in τ_1, being not overwhelmed by S in τ_1. Therefore, $S_1 \cup S \backslash \{P(\mathbf{v})\}$ is descendant-free in τ_1 and, since (N, l) is natural, so is (N, l_1). Therefore, $(N, l_1) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
Then, by rule 10., $(N, (\tau_1, S_1 \cup S, \rho, V, \varphi)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
By rule 4., $(N, (\tau_1, S, \rho, V, \varphi)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
By rule 6., $(N, l) = (N, (\tau, S, \rho, V, \varphi)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.

But, by definition of $\mathbb{S}(\Sigma)$, this is impossible. Therefore, our first assumption was false, which proves the statement of the above lemma. □

We prove now that $\min(\mathbb{S}(\Sigma))$ cannot consist in a unique EDB atom whose variables would be all identified:

Lemma A.4.29.
If $\Sigma \in p\_trees(Q, \Pi)$, $\mathbb{S}(\Sigma) \neq \emptyset$, $(N, l) = ((R(\boldsymbol{t}), \rho), (\tau, S, \rho, V, \varphi)) = \min(\mathbb{S}(\Sigma))$ *and if* P *is an EDB predicate of* γ*, then* S *is not a singleton* $\{P(\boldsymbol{v})\}$ *such that* $\varphi(var(\boldsymbol{v})) \subseteq V$. ∎

Proof. We prove the above statement by *reductio ad absurdum*.

Let us assume that $S = \{P(\mathbf{v})\}$, where $V' = \varphi(var(\mathbf{v})) \subseteq V$ and P is an EDB predicate of γ. $\mathcal{I}_N(l) \equiv \top$, which proves that $\exists N_0 = (R_0(\mathbf{t}_0), \rho_0), N_1 = (R_1(\mathbf{t}_1), \rho_1), \ldots, N_k = (R_k(\mathbf{t}_k), \rho_k) \in \mathcal{N}_\Sigma$ such that

- $N = N_k$.
- the body of ρ_0 contains $P(\mathbf{v})$.
- $\forall i \in \{0, \ldots, k-1\}$, N_i is the father or a child of N_{i+1}.
- $\forall i \in \{0, \ldots, k\}, var(\mathbf{v}) \in var(\rho_i)$.

Let $\varphi_1 : var_\diamond(\gamma^\diamond) \to var_\diamond(\gamma^\diamond) \cup V'$ be a total mapping such that $\varphi_{1|var(\mathbf{v})} = \varphi_{|var(\mathbf{v})}$, $\varphi_1(v) \in var_\diamond(\gamma^\diamond)$ if $v \notin var(\mathbf{v})$.
By rule 14., $(N_0, (\tau, S, \rho_0, V', \varphi_1)) \in \Psi^\omega_\Sigma(\emptyset)$.
Then, $\forall i \in \{1, \ldots, k\}$, by rules 2. and 3., $(N_i, (\tau, S, \rho_i, V', \varphi_1)) \in \Psi^\omega_\Sigma(\emptyset)$.
By rule 7., $(N, (\tau, S, \rho, V, \varphi_1)) \in \Psi^\omega_\Sigma(\emptyset)$.
By rule 9., $(N, l) = (N, (\tau, S, \rho, V, \varphi)) \in \Psi^\omega_\Sigma(\emptyset)$.

But, by definition of $\mathbb{S}(\Sigma)$, this is impossible. Therefore, our first assumption was false, which proves the statement of the above lemma. □

We prove now that $\min(\mathbb{S}(\Sigma))$ cannot consist in a unique diamond atom whose variables would be all identified:

Lemma A.4.30.
If $\Sigma \in p\_trees(Q, \Pi)$, $\mathbb{S}(\Sigma) \neq \emptyset$, $(N, l) = ((R(\boldsymbol{t}), \rho), (\tau, S, \rho, V, \varphi)) = \min(\mathbb{S}(\Sigma))$ *and* P *is a diamond atom of* $\gamma^\diamond$*, then* S *is not a singleton* $\{P(\boldsymbol{v})\}$ *such that* $\varphi(\boldsymbol{v}) \subseteq V$. ∎

Proof. We prove the above statement by *reductio ad absurdum*.

Let us assume that $S = \{G^\diamond(v_1, v_2)\}$, where $\varphi(\{v_1, v_2\}) \subseteq V$ and $G^\diamond$ is a diamond atom. Let (N, l, ϕ) be a positive interpretation of (N, l) of least weight.

- If $\varphi(v_1) = \varphi(v_2)$, then, by rule 13., $(N, l) \in \Psi^\omega_\Sigma(\emptyset)$, which is impossible by definition of $\mathbb{S}(\Sigma)$.
- If $\mathcal{W}_{(N,l,\phi)}(G^\diamond(v_1, v_2)) = 1$, let $\tau_1 \in u\_trees(\Gamma, \gamma^\nabla)$ be a tree containing a node n whose rule ρ is $G^\diamond(v_1, v_2) \Rightarrow G^*(v_1, v_2)$, and such that no diamond-rule ρ' whose head predicate is $G^*(v_1, v_2)$ and whose body is not $G^\diamond(v_1, v_2) \Rightarrow G^*(v_1, v_2)$ may appear in τ_1. We set $l_1 = (\tau_1, \{G(v_1, v_2)\}, \rho, V, \varphi)$.
  Since $\mathcal{W}_{(N,l,\phi)}(G^\diamond(v_1, v_2)) = 1$, (N, l_1, ϕ) is a positive instance of (N, l_1) $(N, l_1, \phi) <_\mathcal{W} (N, l, \phi)$, $\{G(v_1, v_2)\}$ is descendant-free in τ_1, $\mathcal{I}_N(l_1) \equiv \top$ and (N, l_1) is natural. Therefore, $(N, l_1) \notin \mathbb{S}(\Sigma)$, and $(N, l_1) \in \Psi^\omega_\Sigma(\emptyset)$.
  By rule 11., $(N, l) \in \Psi^\omega_\Sigma(\emptyset)$, which is impossible by definition of $\mathbb{S}(\Sigma)$.

- If $k = \mathcal{W}_{(N,l,\phi)}(G^{\diamond}(v_1, v_2)) \geq 2$, let be $\kappa = \mathcal{U}(\Sigma)$. If $P_y = G$, then $\exists w_0, \ldots, w_k \in var(\kappa)$ such that $w_0 = \phi(\Lambda_{\varphi,\kappa}(v_1)), w_k = \phi(\Lambda_{\varphi,\kappa}(v_2))$ and

$$\top \equiv (\forall \mathbf{P}_1, \ldots, \mathbf{P}_n)\,(\forall v_1, \ldots, v_m)$$
$$\left(\left(\bigwedge_{i=1}^{\|\gamma\|} \mathcal{F}_i(\mathbf{P}_1, \ldots, \mathbf{P}_n) \wedge \bigwedge_{i=1}^{a} \mathbf{P}_{s_i}(\mathbf{v}_i)\right) \Rightarrow \bigwedge_{i=0}^{k-1} \mathbf{P}_y(w_i, w_{i+1})\right)$$

  Let be $v_3 \in var_{\diamond}(\gamma^{\diamond})\backslash\{v_1, v_2\}$, and $\varphi_1 : var_{\diamond}(\gamma^{\diamond}) \to var_{\diamond}(\gamma^{\diamond}) \cup V$ such that $\varphi_1(v_1) = \varphi(v_1), \varphi_1(v_2) = \varphi(v_2)$ and $\varphi_1(v_3) = v_3$. Let $\tau_2 \in u\_trees(\Gamma, \gamma^{\nabla})$ be a tree containing a node n_2 whose rule is $G(v_1, v_3) \wedge G^{\diamond}(v_3, v_2) \Rightarrow G^{*}(v_1, v_2)$ and $\tau_3 \in u\_trees(\Gamma, \gamma^{\nabla})$ be a tree containing a node n_3 whose rule is $G^{\diamond}(v_1, v_3) \wedge G^{\diamond}(v_3, v_2) \Rightarrow G^{*}(v_1, v_3)$. We also want that every diamond-rule in $\tau_{2(n_2)}$ be of the type $G^{\diamond}(x, y) \Rightarrow G^{*}(x, y)$. We set $l_2 = (\tau_2, \{G(v_1, v_3), G^{\diamond}(v_3, v_2)\}, \rho, V', \varphi_1)$ and $\phi_2 : var_{\diamond}(\gamma^{\diamond}) \cup var(\kappa) \to var(\kappa)$, a total mapping such that $\phi_2(v_3) = w_1$.
  $\{G(v_1, v_3), G^{\diamond}(v_3, v_2)\}$ is descendant-free in τ_2, (N, l_2) is natural, (N, l_2, ϕ_2) is a positive instance of (N, l_2) and $(N, l_2, \phi) <_{\mathcal{W}} (N, l, \phi)$. Therefore, $(N, l_2) \notin \mathbb{S}(\Sigma)$, and $(N, l_2) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
  By rule 11., $(N, (\tau_3, \{G^{\diamond}(v_1, v_3), G^{\diamond}(v_3, v_2)\}, \rho, V, \varphi_1)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
  By rule 12., $(N, (\tau, S, \rho, V, \varphi_1)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
  By rule 9., $(N, l) = (N, (\tau, S, \rho, V, \varphi)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$, which is impossible by definition of $\mathbb{S}(\Sigma)$.

No integer value of $\mathcal{W}_{(N,l,\phi)}(G^{\diamond}(v_1, v_2))$ is possible: therefore, our first assumption was false, which proves the statement of the above lemma. □

We prove now that $\min(\mathbb{S}(\Sigma))$ cannot contain any atom whose variables would be all identified:

Proposition A.4.31.
If $\Sigma \in p\_trees(Q, \Pi)$, $\mathbb{S}(\Sigma) \neq \emptyset$ and $(N, l) = ((R(\boldsymbol{t}), \rho), (\tau, S, \rho, V, \varphi)) = \min(\mathbb{S}(\Sigma))$, then S does not contain any atom $P(\boldsymbol{v})$ such that $\varphi(\boldsymbol{v}) \subseteq V$. ∎

Proof. We prove the above statement by *reductio ad absurdum.*

Let us assume that S contains an atom $P(\mathbf{v})$ such that $\varphi(\mathbf{v}) \subseteq V$. Let (N, l, ϕ) be an interpretation of (N, l) of least weight. Let be $l_1 = (\tau, \{P(\mathbf{v})\}, \rho, V, \varphi)$ and $l_2 = (\tau, S\backslash\{P(\mathbf{v})\}, \rho, V, \varphi)$: $(N, l_1), (N, l_2) \in \mathcal{N}_{\Sigma} \times \mathcal{L}(\gamma, \Pi)$. Since $\{P(\mathbf{v})\} \subseteq S$ and $S\backslash\{P(\mathbf{v})\} \subsetneq S$, we know that

- $\{P(\mathbf{v})\}$ and $S\backslash\{P(\mathbf{v})\}$ are descendant-free in τ.
- $\mathcal{I}_N(l) \Rightarrow \mathcal{I}_N(l_1)$ and $\mathcal{I}_N(l) \Rightarrow \mathcal{I}_N(l_2)$. Therefore, $\top \equiv \mathcal{I}_N(l_1) \equiv \mathcal{I}_N(l_2)$.
- (N, l_1) and (N, l_2) are natural.
- (N, l_2, ϕ) is a positive instance of (N, l_2) and $(N, l_2, \phi) <_{\mathcal{W}} (N, l, \phi)$.

Therefore, $(N, l_2) \notin \mathbb{S}(\sigma)$, and $(N, l_2) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.

If $S \neq \{P(\mathbf{v})\}$, (N, l_1, ϕ) is also an interpretation of (N, l_1) such that $(N, l_1, \phi) <_{\mathcal{W}} (N, l, \phi)$; then, $(N, l_1) \notin \mathbb{S}(\sigma)$, and $(N, l_1) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
By rule 5., $(N, l) \in \Psi^{\omega}_{\Sigma}(\emptyset)$, which is impossible since $(N, l) \in \mathbb{S}(\Sigma)$.

This is why S is the singleton $\{P(\mathbf{v})\}$. Then, by Lemma A.4.27, Lemma A.4.28, Lemma A.4.29 and Lemma A.4.30, P cannot be a predicate in γ^{∇}, which is a non-sense. Therefore, our first assumption was false, which proves the statement of the above proposition. □

We prove now that $\min(\mathbb{S}(\Sigma))$ cannot have any positive instance ins which some non-identified variable would be mapped to a variable of the current node:

Lemma A.4.32.
If $\Sigma \in p\_trees(Q,\Pi)$, $\mathbb{S}(\Sigma) \neq \emptyset$ and $(N,l) = ((R(\boldsymbol{t}),\rho),(\tau,S,\rho,V,\varphi)) = \min(\mathbb{S}(\Sigma))$, let (N,l,ϕ) be a positive instance of (N,l) of least weight, $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q,\Pi)$ and θ the unfolding mapping of Σ. $\forall v \in \Lambda_{\varphi,\kappa}(var(S)), \phi(v) \in \theta^{-1}([var(\rho)]) \Rightarrow v \in var(\kappa)$. ∎

Proof. We prove the above statement by *reductio ad absurdum*.

Let us assume that $\exists v \in \Lambda_{\varphi,\kappa}(var(S))$ such that $\phi(v) \in \theta^{-1}([var(\rho)])$ and $v \notin var(\kappa)$. Then, $v \in var_{\diamond}(\gamma^{\diamond})$. We set $V' = var(\rho)$ and $\varphi_1 : var_{\diamond}(\gamma^{\diamond}) \to var_{\diamond}(\gamma^{\diamond}) \cup V'$ such that $\varphi_1(v) = \theta(\phi(v)) \in V'$, and $\varphi_1(w) = \varphi(w)$ if $v \neq w$. We also set $l_1 = (\tau, S, \rho, V', \varphi_1)$.

(N, l_1, ϕ_1) is a positive instance of (N, l_1), and therefore $\mathcal{I}_N(l_1) \equiv \top$. Moreover, (N, l_1) is natural and $(N, l_1, \phi_1) <_{\mathcal{W}} (N, l, \phi)$, which proves that $(N, l_1) \notin \mathbb{S}(\Sigma)$ and $(N, l_1) \in \Phi^{\omega}_{\Sigma}(\emptyset)$.
By rule 8., $(N, (\tau, S, \rho, V', \varphi)) \in \Phi^{\omega}_{\Sigma}(\emptyset)$.
By rule 7., $(N, l) = (N, (\tau, S, \rho, V, \varphi)) \in \Phi^{\omega}_{\Sigma}(\emptyset)$.

But, by definition of $\mathbb{S}(\Sigma)$, this is impossible. Therefore, our first assumption was false, which proves the statement of the above lemma. □

We prove now that $\min(\mathbb{S}(\Sigma))$ cannot be eccentric relatively to any sub-tree:

Proposition A.4.33.
If $\Sigma \in p\_trees(Q,\Pi)$, $\mathbb{S}(\Sigma) \neq \emptyset$ and $(N,l) = ((R(\boldsymbol{t}),\rho),(\tau,S,\rho,V,\varphi)) = \min(\mathbb{S}(\Sigma))$, let (N,l,ϕ) be an interpretation of (N,l) of least weight, $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q,\Pi)$ and h the unfolding node mapping of Σ.

By removing the node $h^{-1}(N)$ from κ, one splits $\mathcal{N}_{\kappa}\backslash\{h^{-1}(N)\}$ in connex parts. Let $\mathfrak{N}$ be one of these parts. Then, (N,l,ϕ) is not eccentric relatively to $\mathfrak{N}$. ∎

Proof. Since (N,l,ϕ) is natural, (N,l,ϕ) is smooth relatively to $\mathfrak{N}$. Moreover, since $\tau \in u\_trees(\Gamma,\gamma^{\diamond})$, (N,l,ϕ) is regularisable relatively to $\mathfrak{N}$. We set $(N,l_1,\phi_1) = \mathfrak{S}_{\mathfrak{N}}((N,l,\phi))$, and $(\tau_1, S_1, \rho, V, \varphi_1) = l_1$. Let θ be the unfolding mapping of Σ, $\Lambda_{\varphi_1,\kappa}$ a κ-mapping and $\varphi_2 : var_{\diamond}(\gamma^{\diamond}) \to var_{\diamond}(\gamma^{\diamond}) \cup V$ a total mapping such that

- if $\phi_1(\Lambda_{\varphi_1,\kappa}(v)) \in \theta^{-1}([var(\rho)])$, then $\varphi_2(v) = \theta(\phi_1(\Lambda_{\varphi_1,\kappa}(v))) \in V$.
- otherwise, $\varphi_2(v) = \varphi(v)$.

It is clear that $\phi_1 \circ \Lambda_{\varphi_1,\kappa} = \phi_1 \circ \Lambda_{\varphi_2,\kappa}$. We define now $V_1 = \{v \in var_{\diamond}(\gamma^{\diamond}) | \phi_1(\Lambda_{\varphi_1,\kappa}(v)) \in \mathfrak{N}_V\}$ and $V_2 = \{v \in var_{\diamond}(\gamma^{\diamond}) | \phi_1(\Lambda_{\varphi_1,\kappa}(v)) \in \overline{\mathfrak{N}}_V\}$: $V_1 \cup V_2 = var_{\diamond}(\gamma^{\diamond})$. This lets us split S_1 in two parts: $S_2 = \{P(\mathbf{v}) \in S_1 | var(\mathbf{v}) \not\subseteq V_1\}$ and $S_3 = \{P(\mathbf{v}) \in S_1 | var(\mathbf{v}) \subseteq V_1\}$.

Let us assume that some $v \in var(S_2) \cap var(S_3)$. Let be $P(\mathbf{v}) \in S_2$ and $Q(\mathbf{w}) \in S_3$ such that $v \in var(\mathbf{v}) \cap var(\mathbf{w})$. $v \in var(\mathbf{w})) \subseteq V_1$. Now, we have two possible cases:

- if P is an EDB predicate of γ, then $P(\phi_1(\Lambda_{\varphi_1,\kappa}(var(\mathbf{v}))))$ is an atom in κ. Therefore, $\exists N' \in \mathcal{N}_\kappa$ whose rule ρ' contains $P(\phi_1(\Lambda_{\varphi_1,\kappa}(\mathbf{v})))$. Since $\phi_1(\Lambda_{\varphi_1,\kappa}(var(\mathbf{v}))) \not\subseteq \mathfrak{N}_V$, $N' \in \overline{\mathfrak{N}}$, and $\phi_1(\Lambda_{\varphi_1,\kappa}(var(\mathbf{v}))) \subseteq \overline{\mathfrak{N}}_V$: $v \in var(\mathbf{v}) \subseteq V_2$.
- if P is a diamond predicate $G^\diamond$, since (N, l_1, ϕ_1) is not eccentric relatively to $\mathfrak{N}$, then we have $P(\mathbf{v}) = G^\diamond(x, y)$, where both $\phi_1(\Lambda_{\varphi,\kappa}(x)), \phi(\Lambda_{\varphi,\kappa}(y)) \in \mathfrak{N}_V$ or both $\phi_1(\Lambda_{\varphi,\kappa}(x)), \phi(\Lambda_{\varphi,\kappa}(y)) \in \overline{\mathfrak{N}}_V$. Therefore, we know that $\phi_1(\Lambda_{\varphi_1,\kappa}(var(\mathbf{v}))) \not\subseteq \mathfrak{N}_V$, and that $\phi_1(\Lambda_{\varphi_1,\kappa}(var(\mathbf{v}))) \subseteq \overline{\mathfrak{N}}_V$, which means that $v \in var(\mathbf{v}) \subseteq V_2$.

In both cases, $v \in V_1 \cap V_2$. Let $\mathcal{S} \subseteq \mathcal{N}_\kappa$ be the set of nodes whose rules contain $\phi_1(\Lambda_{\varphi_1,\kappa}(v))$. Since κ is an unfolding tree, $\mathcal{S}$ is connex. Furthermore, $\mathcal{S} \cap \mathfrak{N} \neq \emptyset$ and $\mathcal{S} \cap \overline{\mathfrak{N}} \neq \emptyset$. Therefore, $h^{-1}(N) \in \mathcal{S}$, $\phi(\Lambda_{\varphi_1,\kappa}(v)) \in \theta^{-1}([var(\rho)])$ and $\varphi_2(v) \in V$, which implies that $\varphi_2(var(S_2) \cap var(S_3)) \subseteq V$.

That is why we can, in fact, split S_1 in 3 parts:

- $T_1 = \{P(\mathbf{v}) \in S_1 | var(\mathbf{v}) \not\subseteq V_1\}$.
- $T_2 = \{P(\mathbf{v}) \in S_1 | var(\mathbf{v}) \not\subseteq V_2\}$.
- $T_3 = \{P(\mathbf{v}) \in S_1 | var(\mathbf{v}) \subseteq V_1 \cap V_2\}$.

such that $\varphi_2(var(T_1) \cap var(T_2)) \subseteq V$, $T_1 \cup T_3 = \{P(\mathbf{v}) \in S_1 | var(\mathbf{v}) \subseteq V_2\}$ and $T_2 \cup T_3 = \{P(\mathbf{v}) \in S_1 | var(\mathbf{v}) \subseteq V_1\}$.

Now, let be $l_2 = (\tau_1, S_1, \rho, V, \varphi_2)$ and $(S'_0, \ldots, S'_n)$ be the natural partition of (N, l_2). We know that $\forall i \in \{i, \ldots, n\}, S'_i \subseteq T_1, T_2$ or T_3. Moreover, (N, l_2, ϕ_1) is still a positive instance of (N, l_2). Since (N, l_1, ϕ_1) is both smooth and not eccentric relatively to $\mathcal{N}$, it means that (N, l_2, ϕ_1) is natural, as well as each (N, l'_i, ϕ_1), where $l'_i = (\tau_1, S'_i, \rho, V, \varphi_2)$.

Now, let us assume that some rule ρ' in τ_1 contains 2 diamond predicates. Let us look at the node $n \in \mathcal{N}_\tau$ whose rule is ρ' and at a set S'_i. Since (N, l_1) is regularisable relatively to $\mathfrak{N}$, we are in one of the 8 following cases:

1. If ρ' is of the type

$$G(x, z_1) \wedge G^\diamond(z_1, z_2) \wedge G(z_2, z_3) \wedge G^\diamond(z_3, z_4) \wedge G(z_4, y) \Rightarrow G^*(x, y)$$

   and $G(x, z_1), G(z_2, z_3), G(z_4, y)$ are respectively the atoms in the head of the children n_1, n_2, n_3 of n, then we know that $\exists j \in \{1, 2\}$ such that $z_2 \in V_j \backslash V_{j'}$ and $z_3 \in V_{j'} \backslash V_j$, where $j' = 3 - j$.

   - If $z_2 \in var(S'_i)$, then $S'_i \subseteq T_{j'}$, $z_3 \notin var(S'_i), G^\diamond(z_3, z_4) \notin S'_i$ and $var(\tau_{(n_3)}) \cap var(S'_i) = \emptyset$.
   - If $var(\tau_{(n_1)}) \cap var(S'_i) \neq \emptyset$ and $z_2 \notin var(S'_i)$, then $G^\diamond(z_1, z_2) \notin S'_i, z_3 \notin var(S'_i)$ and $var(\tau_{(n_3)}) \cap var(S'_i) = \emptyset$.

   Therefore, we must be in one of the following 2 cases:

   - $var(\tau_{(n_1)}) \cap var(S'_i) = \emptyset$ and $z_2 \notin var(S'_i)$.
   - $var(\tau_{(n_3)}) \cap var(S'_i) = \emptyset$ and $z_3 \notin var(S'_i)$.

2. If ρ' is of the type

$$G^\diamond(x, z_1) \wedge G(z_1, z_2) \wedge G^\diamond(z_2, z_3) \wedge G(z_3, y) \Rightarrow G^*(x, y)$$

and $G(z_1, z_2), G(z_3, y)$ are respectively the atoms in the head of the children n_1, n_2 of n, then we know that $\exists j \in \{1, 2\}$ such that $z_1 \in V_j \backslash V_{j'}$ and $z_2 \in V_{j'} \backslash V_j$, where $j' = 3 - j$.
If $z_1 \in var(S'_i)$, then $S'_i \subseteq T_{j'}$, $z_2 \notin var(S'_i), G^\diamond(z_2, z_3) \notin S'_i$ and $var(\tau_{(n_2)}) \cap var(S'_i) = \emptyset$.
Therefore, we must be in one of the following 2 cases:

- $z_1 \notin var(S'_i)$.
- $var(\tau_{(n_2)}) \cap var(S'_i) = \emptyset$ and $z_2 \notin var(S'_i)$.

3. If ρ' is of the type

$$G(x, z_1) \wedge G^\diamond(z_1, z_2) \wedge G(z_2, z_3) \wedge G^\diamond(z_3, y) \Rightarrow G^*(x, y)$$

and $G(x, z_1), G(z_2, z_3)$ are respectively the atoms in the head of the children n_1, n_2 of n, then we know that $\exists j \in \{1, 2\}$ such that $z_3 \in V_j \backslash V_{j'}$ and $z_2 \in V_{j'} \backslash V_j$, where $j' = 3 - j$.
If $z_3 \in var(S'_i)$, then $S'_i \subseteq T_{j'}$, $z_2 \notin var(S'_i), G^\diamond(z_1, z_2) \notin S'_i$ and $var(\tau_{(n_1)}) \cap var(S'_i) = \emptyset$.
Therefore, we must be in one of the following 2 cases:

- $var(\tau_{(n_1)}) \cap var(S'_i) = \emptyset$ and $z_2 \notin var(S'_i)$.
- $z_3 \notin var(S'_i)$.

4. If ρ' is of the type

$$G^\diamond(x, z_1) \wedge G(z_1, z_2) \wedge G^\diamond(z_2, y) \Rightarrow G^*(x, y)$$

and $G(z_1, z_2)$ is the atom in the head of the child n_1 of n, then we know that $\exists j \in \{1, 2\}$ such that $z_1 \in V_j \backslash V_{j'}$ and $z_2 \in V_{j'} \backslash V_j$, where $j' = 3 - j$.
If $z_1 \in var(S'_i)$, then $S \subseteq T_{j'}$ and $z_2 \notin var(S'_i)$.
Therefore, we must be in one of the following 2 cases:

- $z_1 \notin var(S'_i)$.
- $z_2 \notin var(S'_i)$.

5. If ρ' is of the type

$$G(x, z_1) \wedge G^\diamond(z_1, z_2) \wedge G^\diamond(z_2, z_3) \wedge G(z_3, y) \Rightarrow G^*(x, y)$$

and $G(x, z_1), G(z_2, y)$ are respectively the atoms in the head of the children n_1, n_2 of n, then we know that $\exists j \in \{1, 2\}$ such that $z_1 \in V_j \backslash V_{j'}$ and $z_3 \in V_{j'} \backslash V_j$, where $j' = 3 - j$.

- If $z_1 \in var(S'_i)$, then $S'_i \subseteq T_{j'}$, $z_3 \notin var(S'_i), G^\diamond(z_2, z_3) \notin S'_i$ and $var(\tau_{(n_2)}) \cap var(S'_i) = \emptyset$.
- If $var(\tau_{(n_1)}) \cap var(S'_i) \neq \emptyset$ and $z_2 \notin var(S'_i)$, then $G^\diamond(z_1, z_2) \notin S'_i, z_3 \notin var(S'_i)$ and $var(\tau_{(n_2)}) \cap var(S'_i) = \emptyset$.

Therefore, we must be in one of the following 2 cases:

- $var(\tau_{(n_1)}) \cap var(S'_i) = \emptyset$ and $z_1 \notin var(S'_i)$.
- $var(\tau_{(n_2)}) \cap var(S'_i) = \emptyset$ and $z_3 \notin var(S'_i)$.

6. If ρ' is of the type

$$G^{\diamond}(x, z_1) \wedge G^{\diamond}(z_1, z_2) \wedge G(z_2, y) \Rightarrow G^{*}(x, y)$$

and $G(z_2, y)$ is the atom in the head of the child n_1 of n, then we know that $\exists j \in \{1, 2\}$ such that $x \in V_j \backslash V_{j'}$ and $z_2 \in V_{j'} \backslash V_j$, where $j' = 3 - j$.
If $x \in var(S'_i)$, then $S'_i \subseteq T_{j'}$, $z_2 \notin var(S'_i)$, $G^{\diamond}(z_1, z_2) \notin S'_i$ and $var(\tau_{(n_1)}) \cap var(S'_i) = \emptyset$.
Therefore, we must be in one of the following 2 cases:

- $x \notin var(S'_i)$.
- $var(\tau_{(n_1)}) \cap var(S'_i) = \emptyset$ and $z_2 \notin var(S'_i)$.

7. If ρ' is of the type

$$G(x, z_1) \wedge G^{\diamond}(z_1, z_2) \wedge G^{\diamond}(z_2, y) \Rightarrow G^{*}(x, y)$$

and $G(x, z_1)$ is the atom in the head of the child n_1 of n, then we know that $\exists j \in \{1, 2\}$ such that $y \in V_j \backslash V_{j'}$ and $z_1 \in V_{j'} \backslash V_j$, where $j' = 3 - j$.
If $y \in var(S'_i)$, then $S'_i \subseteq T_{j'}$, $z_1 \notin var(S'_i)$, $G^{\diamond}(z_1, z_2) \notin S'_i$ and $var(\tau_{(n_1)}) \cap var(S'_i) = \emptyset$.
Therefore, we must be in one of the following 2 cases:

- $var(\tau_{(n_1)}) \cap var(S'_i) = \emptyset$ and $z_1 \notin var(S'_i)$.
- $y \notin var(S'_i)$.

8. If ρ' is of the type

$$G^{\diamond}(x, z_1) \wedge G^{\diamond}(z_1, y) \Rightarrow G^{*}(x, y)$$

then we know that $\exists j \in \{1, 2\}$ such that $x \in V_j \backslash V_{j'}$ and $y \in V_{j'} \backslash V_j$, where $j' = 3 - j$.
If $x \in var(S'_i)$, then $S \subseteq T_{j'}$ and $y \notin var(S'_i)$.
Therefore, we must be in one of the following 2 cases:

- $x \notin var(S'_i)$.
- $y \notin var(S'_i)$.

We have just proved that (N, l'_i) is locally triangular: let $\tau'_i \in u\_trees(\Gamma, \gamma^{\nabla})$ such that, if $l''_i = (\tau'_i, S'_i, \rho, V, \varphi_2)$, then $(N, l''_i) \in \mathcal{N}_{\Sigma} \times \mathcal{L}(\gamma, \Pi)$, $\mathcal{I}_N(l''_i) \equiv \top$ and (N, l''_i, ϕ_1) is a positive instance of (N, l''_i). S'_i contains only EDB predicates of γ^{∇}, so that (N, l''_i) is descendant-free. Moreover, since (N, l'_i, ϕ_1) is natural, so is (N, l''_i, ϕ_1).

By rule 6., $(N, l'_i) \in \Psi^{\omega}_{\Sigma}(\{(N, l''_i, \phi_1)\})$.
Then, by applying n times the rule 5., $(N, l_2) \in \Psi^{\omega}_{\Sigma}(\{(N, l''_i, \phi_1) | 0 \leq i \leq n\})$.
By rule 8., $(N, l_1) \in \Psi^{\omega}_{\Sigma}(\{(N, l''_i, \phi_1) | 0 \leq i \leq n\})$.
By the property mentioned in Definition A.4.23, $(N, l) \in \Psi^{\omega}_{\Sigma}(\{(N, l''_i, \phi_1) | 0 \leq i \leq n\}$.

Therefore, $\exists i \in \{0, \ldots, n\}$ such that $(N, l''_i, \phi_1) \notin \Psi^{\omega}_{\Sigma}(\emptyset)$, which means that $(N, l, \phi) \leq_{\mathcal{W}} (N, l''_i, \phi_1)$. But, if (N, l, ϕ) were eccentric relatively to $\mathfrak{N}$, then, $\forall i \in \{0, \ldots, n\}$, we would have

$$(N, l''_i, \phi_1) \leq_{\mathcal{W}} (N, l'_i, \phi_1) \leq_{\mathcal{W}} (N, l_2, \phi_1) \leq_{\mathcal{W}} (N, l_1, \phi_1) <_{\mathcal{W}} (N, l, \phi)$$

That is why (N, l, ϕ) is not eccentric relatively to $\mathfrak{N}$, which is the statement of the proposition above. □

Finally, we prove that $\min(\mathbb{S}(\Sigma))$ cannot exist, which means that $\mathbb{S}(\Sigma)$ is empty:

Proposition A.4.34.
$\forall \Sigma \in p\_trees(Q, \Pi), \mathbb{S}(\Sigma) = \emptyset$. ∎

Proof. We prove the above statement by *reductio ad absurdum.*

Let be $\Sigma \in p\_trees(Q, \Pi)$. If $\mathbb{S}(\Sigma) \neq \emptyset$, we set $(N, l) = ((R(\mathbf{t}), \rho), (\tau, S, \rho, V, \varphi)) = \min(\mathbb{S}(\Sigma))$. Let (N, l, ϕ) be a positive instance of (N, l) of least weight, $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q, \Pi)$, θ the unfolding mapping of Σ, h the unfolding node mapping of Σ and $\Lambda_{\varphi,\kappa}$ a κ-mapping.

$(N, (\tau, \emptyset, \rho, V, \varphi)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$, and therefore $S \neq \emptyset$. By Lemma A.4.32 and Proposition A.4.31, we know that $\exists v \in var(S)$ such that $\phi_{\Lambda_{\varphi,\kappa}(v)} \notin \theta^{-1}([var(\rho)])$.

By removing the node $h^{-1}(N)$ from κ, one splits $\mathcal{N}_\kappa \backslash \{h(N)\}$ in connex parts. Then, such a part, $\mathfrak{N}$, must verify $\phi_{\Lambda_{\varphi,\kappa}(v)} \in \mathfrak{N}_V \backslash \overline{\mathfrak{N}}_V$. Since (N, l, ϕ) is not eccentric relatively to $\mathfrak{N}$, $\phi_{\Lambda_{\varphi,\kappa}(var(S))} \subseteq \mathfrak{N}_V$.

Here, we are in one of the 2 following cases:

- If the father n of $h^{-1}(N)$ verifies $n \in \mathfrak{N}$, then $\varphi(var(S)) \cap V \subseteq var(\mathbf{t})$. Let $N' = h(n) = (R'(\mathbf{t}'), \rho')$ be the father of N, $V' = var(\mathbf{t}) \subseteq V$, $\varphi' : var_\diamond(\gamma^\diamond) \to var_\diamond(\gamma^\diamond) \cup V'$ be a total mapping such that $\varphi'_{|var(S)} = \varphi_{|var(S)}$ and $\varphi'(var_\diamond(\gamma^\diamond) \backslash var(S)) \subseteq var_\diamond(\gamma^\diamond)$ and $l' = (\tau, S, \rho', V', \varphi')$. Then, $(N', l') \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$, $\mathcal{I}_{N'}(l') \equiv \top$ and (N', l', ϕ) is a positive instance of (N', l'), with $(N', l', \phi) <_{\mathcal{W}} (N, l, \phi)$. Moreover, since (N, l) is natural, so is (N', l'). Therefore, since $(N', l') \notin \mathbb{S}(\Sigma)$, we know that $(N', l') \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
  By rule 3., $(N, (\tau, S, \rho, V', \varphi')) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
  By rule 7., $(N, (\tau, S, \rho, V, \varphi')) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
  By rule 9., $(N, l) = (N, (\tau, S, \rho, V, \varphi)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
- If a child n of $h^{-1}(N)$ verifies $n \in \mathfrak{N}$, then $\varphi(var(S)) \cap V \subseteq var(\mathbf{t}')$. Let $N' = h(n) = (R'(\mathbf{t}'), \rho')$ be a child of N, $V' = var(\mathbf{t}') \subseteq V$, $\varphi' : var_\diamond(\gamma^\diamond) \to var_\diamond(\gamma^\diamond) \cup V'$ be a total mapping such that $\varphi'_{|var(S)} = \varphi_{|var(S)}$, $\varphi'(var_\diamond(\gamma^\diamond) \backslash var(S)) \subseteq var_\diamond(\gamma^\diamond)$ and $l' = (\tau, S, \rho', V', \varphi')$. Then, $(N', l') \in \mathcal{N}_\Sigma \times \mathcal{L}(\gamma, \Pi)$, $\mathcal{I}_{N'}(l') \equiv \top$ and (N', l', ϕ) is a positive instance of (N', l'), with $(N', l', \phi) <_{\mathcal{W}} (N, l, \phi)$. Moreover, since (N, l) is natural, so is (N', l'). Therefore, since $(N', l') \notin \mathbb{S}(\Sigma)$, we know that $(N', l') \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
  By rule 2., $(N, (\tau, S, \rho, V', \varphi')) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
  By rule 7., $(N, (\tau, S, \rho, V, \varphi')) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.
  By rule 9., $(N, l) = (N, (\tau, S, \rho, V, \varphi)) \in \Psi^{\omega}_{\Sigma}(\emptyset)$.

In both cases, $(N, l) \in \Psi^{\omega}_{\Sigma}(\emptyset)$, which is impossible by definition of $\mathbb{S}(\Sigma)$. Therefore, our first assumption was false, which proves the above proposition. □

This emptiness directly implies the following theorem, which expresses the other of the two implications Theorem 5.2.3 consists in:

Theorem A.4.35.
Let Π *be a* Datalog *program with goal predicate* Q, γ *a transitive* Datalog *program*

with goal predicate Γ. *Let be* $\Sigma \in p\_trees(Q, \Pi), \sigma = l_{min}(\Sigma) \in p\_label^{fp}(\Sigma, \gamma, Q, \Pi)$, *and* $r = (N, L)$ *the root of* σ. *Let* $\Gamma(\boldsymbol{v})$ *be an instance of* Γ *with variables among* $var_\diamond(\gamma^\diamond)$. *If there exists a containment mapping from a tree* $\nu \in u\_trees(\Gamma, \gamma)$ *to* $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q, \Pi)$, *then* $N = (R(\boldsymbol{t}), \rho)$ *is such that, for some tuple of variables* $\boldsymbol{v}$, *and some* $l = (\tau, \{\Gamma(\boldsymbol{v})\}, \rho, V, \varphi) \in L$ *verifies* $\varphi(\boldsymbol{v}) = \boldsymbol{t}$. ∎

Proof. Let be some $l = (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi)$ such that $\varphi(\mathbf{v}) = \mathbf{t}$. Since there exists a containment mapping from a tree $\nu \in u\_trees(\Gamma, \gamma)$ to $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q, \Pi)$, $\mathcal{I}_N(l) \equiv \top$. Let be $\tau' \in u\_trees(\Gamma, \gamma^\nabla)$ such that $l' = (\tau', \{\Gamma(\mathbf{v})\}, \rho, V, \varphi\}) \in \mathcal{L}(\gamma, \Pi)$. We know that

- $\mathcal{I}_N(l') \equiv \top$.
- $\tau' \in u\_trees(\Gamma, \gamma^\nabla)$.
- $\{\Gamma(\mathbf{v})\}$ is descendant-free in τ'.
- (N, l') is natural.
- $(N, l') \notin \mathbb{S}(\Sigma) = \emptyset$.

Therefore, $(N, l) \in \Psi^\omega_\Sigma(\emptyset)$, which proves the above theorem. □

A.4.4 Proof of Theorem 5.2.3 : An Equivalent Condition

Both Theorems A.4.13 and A.4.35 are now sufficient to prove the following theorem, which expresses a condition equivalent to the existence of a containment mapping.

Theorem A.4.36.
Let Π *be a* Datalog *program with goal predicate* Q, γ *be a transitive* Datalog *program with goal predicate* Γ, $\Sigma \in p\_trees(Q, \Pi)$ *be a proof tree of* Π. *The two following conditions are equivalent:*

- *There exists a containment mapping from a tree* $\nu \in u\_trees(\Gamma, \gamma)$ *to* $\mathcal{U}(\Sigma)$.
- $\forall \sigma \in p\_label^{fp}(\Sigma, \gamma, Q, \Pi)$, *the root* $(N, L) = ((R(\boldsymbol{t}), \rho), L)$ *of* σ *is such that, for some tuple of variables* $\boldsymbol{v}$ *and some* $l = (\tau, \{\Gamma(\boldsymbol{v})\}, \rho, V, \varphi) \in L$, $\varphi(\boldsymbol{v}) = \boldsymbol{t}$.

∎

Proof. If there exists a containment mapping from a tree $\nu \in u\_trees(\Gamma, \gamma)$ to $\mathcal{U}(\Sigma)$, then, by Theorem A.4.35, for some tuple of variables $\mathbf{v}$ and some $l = (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi)$ such that $\varphi(\mathbf{v}) = \mathbf{t}$, $(N, l) \in \Psi^\omega_\Sigma(\emptyset)$. Furthermore, $i(\sigma) = (\Sigma, \mathcal{E})$ verifies $\mathcal{E} = \Psi^\omega_\Sigma(\mathcal{E}) \supseteq \Psi^\omega_\Sigma(\emptyset) \ni (N, l)$. Therefore, the root (N, L) if σ verifies $l \in L$.

Conversely, if the root $(N, L) = (R(\mathbf{t}), \rho), L)$ of $l_{min}(\Sigma)$ is such that, for some tuple of variables $\mathbf{v}$ and some $l = (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi) \in L$, $\varphi(\mathbf{v}) = \mathbf{t}$, then, by Theorem A.4.13, there exists a containment mapping from a tree $\nu \in u\_trees(\Gamma, \gamma)$ to $\kappa = \mathcal{U}(\Sigma)$. □

Finally, since the containment of a program in another one can be reduced to the existence of containment mappings, we obtain a demonstration of Theorem 5.2.3:

Theorem A.4.37. [Theorem 5.2.3]
Let Π *be a* DATALOG *program with goal predicate* Q, γ *be a transitive* DATALOG *program with goal predicate* Γ.

Π *is contained in* γ *if and only if, for every tree* $\sigma \in p\_label^{fp}(\gamma, Q, \Pi)$, *the root* $(N, L) = ((R(\boldsymbol{t}), \rho), L)$ *of* σ *is such that, for some tuple of variables* $\boldsymbol{v}$ *and some* $l = (\tau, \{\Gamma(\boldsymbol{v})\}, \rho, V, \varphi) \in L$, $\varphi(\boldsymbol{v}) = \boldsymbol{t}$. ∎

Proof. If Π is contained in γ and $\sigma \in p\_label^{fp}(\gamma, Q, \Pi)$, let us look at $i(\sigma) = (\Sigma, \mathcal{E})$ and at $\kappa = \mathcal{U}(\Sigma) \in u\_trees(Q, \Pi)$. Since Π is contained in γ, there exists a containment mapping from a tree $\nu \in u\_trees(\Gamma, \gamma)$ to $\kappa = \mathcal{U}(\Sigma)$. Therefore, by Theorem A.4.36, the root $(N, L) = ((R(\mathbf{t}), \rho), L)$ of σ is such that, for some tuple of variables $\mathbf{v}$ and some $l = (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi) \in L$, $\varphi(\mathbf{v}) = \mathbf{t}$.

Conversely, we suppose that, $\forall \sigma \in p\_label^{fp}(\gamma, Q, \Pi)$, the root $(N, L) = ((R(\mathbf{t}), \rho), L)$ of σ is such that, for some tuple of variables $\mathbf{v}$ and some $l = (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi) \in L$, $\varphi(\mathbf{v}) = \mathbf{t}$. Let be $\kappa \in u\_trees(Q, \Pi)$. $\exists \Sigma \in p\_trees(Q, \Pi)$ such that $\kappa = \mathcal{U}(\Sigma)$ (up to a renaming of the variables). Furthermore, $\forall \sigma \in p\_label^{fp}(\Sigma, \gamma, Q, \Pi)$, the root $(N, L) = ((R(\mathbf{t}), \rho), L)$ of σ is such that, for some tuple of variables $\mathbf{v}$ and some $l = (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi) \in L$, $\varphi(\mathbf{v}) = \mathbf{t}$. Therefore, by Theorem A.4.36, there exists a containment mapping from a tree $\nu \in u\_trees(\Gamma, \gamma)$ to $\mathcal{U}(\Sigma)$; it becomes straightforward that there exists also a containment mapping from a tree $\nu \in u\_trees(\Gamma, \gamma)$ to κ. This being true for every tree $\kappa \in u\_trees(Q, \Pi)$, we know that Π is contained in γ. □

A.4.5 Tree Automata

Theorem A.4.38. [Theorem 5.3.2]
Let Π *be a* DATALOG *program with goal predicate* Q, *and* γ *be a transitive* DATALOG *program with goal predicate* Γ. *There is an automaton* $\mathcal{A}^{Q,\Pi}_{\Gamma,\gamma}$, *whose size is doubly exponential in the size of* Π *and triply exponential in the size of* γ, *such that* $T(\mathcal{A}^{Q,\Pi}_{\Gamma,\gamma}) = \emptyset$ *if and only if* Π *is contained in* Γ. ∎

Proof. We describe hereafter the construction of the automaton

$$\mathcal{A}^{Q,\Pi}_{\Gamma,\gamma} = (A, E \cup \{accept\}, I, \delta, \{accept\})$$

- A is the set of couples $(N, L) = ((R(\mathbf{t}), \rho), L)$ such that ρ is a rule of Π with variables over $var(\Pi)$, $R(\mathbf{t})$ is the atom in the head of ρ, $L \subseteq \mathcal{L}(\gamma, \Pi)$, $\Phi(L) = L$ and, $\forall\, l = (\tau, S, r, V, \varphi) \in L, r = \rho$.
- $E = A$.
- I is the set of couples $(N, L) = ((R(\mathbf{t}), \rho), L) \in E$ such that $R = Q$ and L does not contain any tuple $(\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi)$ verifying $\varphi(\mathbf{v}) = \mathbf{t}$.

and where δ is constructed as follows:

- Let ρ be a rule instance

$$R_1(\mathbf{t}_1) \wedge \ldots \wedge R_m(\mathbf{t}_m) \Rightarrow R(\mathbf{t})$$

in Π, with IDB atoms $R_{i_1}(\mathbf{t}_{i_1}), \ldots, R_{i_k}(\mathbf{t}_{i_k})$ in its body.
Let be $s = ((R(\mathbf{t}), \rho), L), s_1 = ((R_{i_1}(\mathbf{t}_{i_1}), \rho_1), L_1), \ldots, s_k = ((R_{i_k}(\mathbf{t}_{i_k}), \rho_k), L_k) \in E$.
If, $\forall j \in \{1, \ldots, k\}$,

$$\{(\tau, S, V, \varphi) | V \subseteq var(\mathbf{t}_{i_j}), (\tau, S, \rho, V, \varphi) \in L\}$$

$$=$$

$$\{(\tau, S, V, \varphi) | V \subseteq var(\mathbf{t}_{i_j}), (\tau, S, \rho_j, V, \varphi) \in L_j\}$$

then $\langle s_1, \ldots, s_k \rangle \in \delta(s, s)$.

- Let ρ be a rule instance

$$R_1(\mathbf{t}_1) \wedge \ldots \wedge R_m(\mathbf{t}_m) \Rightarrow R(\mathbf{t})$$

in Π, with only EDB atoms in its body. Let be $s = ((R(\mathbf{t}), \rho), L) \in E$. Then, $\langle accept \rangle \in \delta(s, s)$.

It is straightforward to see that $\mathcal{A}^{Q,\Pi}_{\Gamma,\gamma}$ accepts a subset of $p\_label^{pf}(\gamma, Q, \Pi)$, and that a tree $\sigma \in p\_label^{pf}(\gamma, Q, \Pi)$ is in the accepted set if and only if the root $(N, L) = ((R(\mathbf{t}), \rho), L)$ of σ is such that L does not contain any tuple $l = (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi)$ such that $\varphi(\mathbf{v}) = \mathbf{t}$.

Therefore, $\mathcal{A}^{Q,\Pi}_{\Gamma,\gamma}$ accepts the empty language if and only if Π is contained in γ. □

Theorem A.4.39. [Theorem 5.3.3]
Containment of a Datalog *program in a monadic* Datalog *program is in 3EXPTIME.*
■

Proof. Let be Π a Datalog program with goal predicate Q, and γ a transitive Datalog program with goal predicate Γ.

By an immediate induction, we prove that the number of rules that contains any tree $\tau \in u\_trees(\gamma^\diamond)$ is at most doubly exponential in the size of γ. Therefore, $|var_\diamond(\gamma^\diamond)|$ is also at most doubly exponential in the size of γ. Then, the number of trees $\tau \in u\_trees(\gamma^\diamond)$ with variables among $var_\diamond(\gamma^\diamond)$ is at most triply exponential in the size of γ. And, almost immediately, it comes that the number of tuples $(\tau, S, \rho, V, \varphi) \in \mathcal{L}(\gamma, \Pi)$ is at most triply exponential in the size of Π and γ, and that the set of these tuples can be calculated in time triply exponential in the sizes of Π and γ.

Moreover, the set $\{(R(\mathbf{t}), \rho)\}$ where $R(\mathbf{t})$ is the head atom of a rule ρ with variables among $var(\Pi)$ A can be calculated in time exponential in the size of Π. Then, $\forall L \in \mathcal{L}(\gamma, \Pi)$, we can calculate whether $\phi(L) = L$ in time triply exponential in the sizes of Π and γ. Therefore, A, E and I can be calculated in time triply exponential in the sizes of Π and γ.

Finally, the computation of δ also takes a time triply exponential in the sizes of Π and γ, and the emptiness of $\mathcal{A}^{Q,\Pi}_{\Gamma,\gamma}$ is decidable in time triply exponential in the sizes of Π and γ.

All these assertions imply that our algorithm checks whether Π is contained in γ in time and place triply exponential in the sizes of Π and γ. $\square$

Appendix B

Decidability and Courcelle's Theorem

B.1 Courcelle's Theorem

We can view a conjunctive query $\phi(x_1, \ldots, x_k)$ with free variables among $x_1, \ldots, x_k$ as a 2-sorted relational structure $\mathbf{A}_\phi$ in the following sense:

The sorts V and F of $\mathbf{A}_\phi$ denote the set of variables and atomic formulæ in ϕ, respectively. For each l-ary predicate symbol P in the vocabulary of ϕ, we have a predicate symbol P' in the vocabulary of $\mathbf{A}_\phi$ of type $F \times V^l$. The vocabulary of $\mathbf{A}_\phi$ also has constant symbols $\mathbf{x}_1, \ldots, \mathbf{x}_k$. These constant and predicate symbols are interpreted in $\mathbf{A}_\phi$ as follows. First, the constant symbol $\mathbf{x}_i$ is interpreted as x_i. Second, if the atomic formula a_i is $P(z_1, \ldots, z_l)$ in ϕ, then we have a tuple $\langle a_i, z_1, \ldots, z_l \rangle$ in the interpretation of P'. Note that ϕ can have multiple occurrences of the same atomic formula, which explains why we need the sort F in $\mathbf{A}_\phi$.

Since a conjunctive query ϕ can be viewed as a 2-sorted relational structure $\mathbf{A}_\phi$, we can view $u\_trees(Q, \Pi)$ as a set of 2-sorted relational structures, which we denote as $str(Q, \Pi)$. If Q is k-ary, then we can assume that all conjunctive queries in $u\_trees(Q, \Pi)$ have free variables among $x_1, \ldots, x_k$. Thus, all structures in $str(Q, \Pi)$ have the same vocabulary, denoted $vocab(Q, \Pi)$. We can now express properties of DATALOG program in terms of properties of the associated collection of 2-sorted structures. If ψ is a monadic second-order (MSO) formula over $vocab(Q, \Pi)$, then we say that the program Π with goal predicate Q *satisfies* ψ if ψ holds in *all* structures in $str(Q, \Pi)$.

For example, consider the property of *strong non-redundancy* defined hereafter: We say that a DATALOG program Π with goal predicate Q is strongly no-redundant if no unfolding expansion tree contains two distinct occurrences of the same EDB atom. It is easy to see that this property can be expressed as a first-order property of the structures in $str(Q, \Pi)$. For simplicity assume that there is a single EDB predicate P, which happens to be k-ary. Then the desired property holds if the program Π with goal predicate Q satisfies the sentence

$$(\forall x_1, x_2 \in F)(\forall y_1, \ldots, y_k \in V)(P'(x_1, y_1, \ldots, y_k) \\ \wedge P'(x_2, y_1, \ldots, y_k) \Rightarrow x_1 = x_2).$$

Monadic second-order logic gives us a very powerful language to describe properties of DATALOG queries in terms of the associated set of structures. It is not clear, a priori, whether such properties can be effectively tested. After all, to check whether a DATALOG program Π with a goal Q satisfies a monadic second-order sentence ψ we have to check in principle the infinitely many structures in $str(Q, \Pi)$. The following powerful result by COURCELLE asserts that, nevertheless, monadic second-order properties of DATALOG programs can be effectively tested.

Theorem B.1.1. [7, 8]
There is an algorithm to decide, given a DATALOG *program* Π *with goal predicate* Q *and a monadic second-order sentence* ψ *over vocab*(Q, Π)*, whether* Π *satisfies* ψ*.* ∎

B.2 Monadic Programs

The decidability of containment in monadic programs follows now from Theorem B.1.1.

Theorem B.2.1. *Containment of* DATALOG *programs in monadic* DATALOG *programs is decidable.* ∎

Proof. Let us assume that Π is a DATALOG program with goal predicate Q. Let $\Pi_{\mathcal{M}}$ be an arbitrary monadic DATALOG program, with internal IDB predicates is $I_1, \ldots, I_n$ and k-ary goal predicate $Q_{\mathcal{M}}$. Let us assume that the internal rules of $\Pi_{\mathcal{M}}$ are $\mathcal{R}_i$:

$$R_1(\mathbf{v}^1) \wedge \ldots \wedge R_m(\mathbf{v}^m) \Rightarrow R(\mathbf{v})$$

with IDB atoms $R(\mathbf{v}), R_{i_1}(\mathbf{v}^{i_1}), \ldots, R_{i_{l'}}(\mathbf{v}^{i_{l'}})$ and EDB atoms $R_{j_1}(\mathbf{v}^{j_1}), \ldots, R_{j_l}(\mathbf{v}^{j_l})$. Each internal rule can be translated into a monadic second-order formula

$$\varphi_i(\mathbf{I}_1, \ldots, \mathbf{I}_n) = (\forall y_1, \ldots, y_m)((\bigwedge_{i=1}^{l} a_i \wedge \bigwedge_{i=1}^{l'} b_i) \Rightarrow b_0)$$

with free sets among $\mathbf{I}_1, \ldots, \mathbf{I}_n$ and no free variable, where each a_j is a first-order formula $p_j(z_1, \ldots, z_k)$ and each b_j is a monadic second-order atomic formula $\mathbf{I}_{j'}(z_{j''})$, all these formulæ being over the variables $y_1, \ldots, y_m$. Let us assume now that the goal rules of $\Pi_{\mathcal{M}}$ are $\mathcal{Q}_i$:

$$R_1(\mathbf{v}^1) \wedge \ldots \wedge R_m(\mathbf{v}^m) \Rightarrow Q_{\mathcal{M}}(\mathbf{v})$$

with head predicate $Q_{\mathcal{M}}(\mathbf{v})$, internal IDB atoms $R_{i_1}(\mathbf{v}^{i_1}), \ldots, R_{i_{l'}}(\mathbf{v}^{i_{l'}})$ and EDB atoms $R_{j_1}(\mathbf{v}^{j_1}), \ldots, R_{j_l}(\mathbf{v}^{j_l})$. Each goal rule can be translated into a monadic second-order formula

$$\overline{\varphi_i}(\mathbf{I}_1, \ldots, \mathbf{I}_n, x_1, \ldots, x_k) = (\exists y_1, \ldots, y_m)(\bigwedge_{i=1}^{l} a_i \wedge \bigwedge_{i=1}^{l'} b_i)$$

with free sets among $\mathbf{I}_1, \ldots, \mathbf{I}_n$ and free variables among $x_1, \ldots, x_k$, where each a_j is a first-order formula $p_j(z_1, \ldots, z_k)$ and each b_j is a monadic second-order atomic formula $\mathbf{I}_{j'}(z_{j''})$, all these formulæ being over the variables $y_1, \ldots, y_m, x_1, \ldots, x_k$. $\Pi_{\mathcal{M}}$ itself can be translated into a monadic second-order formula

$$\psi(x_1, \ldots, x_k) = (\forall \mathbf{I}_1, \ldots, \mathbf{I}_n)(\bigwedge_{i=1}^{s} \varphi_i(\mathbf{I}_1, \ldots, \mathbf{I}_n) \Rightarrow \bigvee_{i=1}^{s'} \overline{\varphi_i}(\mathbf{I}_1, \ldots, \mathbf{I}_n, x_1, \ldots, x_k))$$

with free variables among $x_1, \ldots, x_k$ and no free set. Given n sets $\mathbf{I}_1, \ldots, \mathbf{I}_n \subseteq V$, define $\varphi'_i(\mathbf{I}_1, \ldots, \mathbf{I}_n)$ to be the sentence

$$(\forall y_1, \ldots, y_m \in V)(\exists a_1, \ldots, a_l \in F)((\bigwedge_{i=1}^{l} a'_i \wedge \bigwedge_{i=1}^{l'} b'_i) \Rightarrow b'_0)$$

where a'_i is the atomic formula $p'_i(a_i, z'_1, \ldots, z'_l)$, b'_j is the atomic formula $z'_{j"} \in \mathbf{I}_{j'}$, and $z'_1, \ldots, z'_l$ are obtained from $z_1, \ldots, z_l$ by substituting $\mathbf{x}_j$ for x_j. We also define $\overline{\varphi'_i}(\mathbf{I}_1, \ldots, \mathbf{I}_n)$ to be the sentence

$$(\exists y_1, \ldots, y_m \in V)(\exists a_1, \ldots, a_l \in F)(\bigwedge_{i=1}^{l} a'_i \wedge \bigwedge_{i=1}^{l'} b'_i)$$

where a'_i is the atomic formula $p'_i(a_i, z'_1, \ldots, z'_l)$, b'_j is the atomic formula $z'_{j"} \in \mathbf{I}_{j'}$, and $z'_1, \ldots, z'_l$ are obtained from $z_1, \ldots, z_l$ by substituting $\mathbf{x}_j$ for x_j. Let now ψ' be the MSO formula

$$\psi' = (\forall \mathbf{I}_1, \ldots, \mathbf{I}_n \subseteq V)(\bigwedge_{i=1}^{s} \varphi'_i(\mathbf{I}_1, \ldots, \mathbf{I}_n) \Rightarrow \bigvee_{i=1}^{s'} \overline{\varphi'_i}(\mathbf{I}_1, \ldots, \mathbf{I}_n))$$

We claim that Π is contained in $\Pi_{\mathcal{M}}$ if and only if Π satisfies ψ'.

Let us assume that Π does not satisfy ψ' but is contained in $\Pi_{\mathcal{M}}$. There is a tree $\tau \in u\_trees(Q, \Pi)$ such that ψ' does not hold over $\mathbf{A}_\tau$. Therefore, there exists subsets $\mathbf{I}_1, \ldots, \mathbf{I}_n$ of $var(\tau)$ such that each $\varphi_i(\mathbf{I}_1, \ldots, \mathbf{I}_n)$ holds over $\mathbf{A}_\tau$, but no $\overline{\varphi'_i}(\mathbf{I}_1, \ldots, \mathbf{I}_n)$ does. Let $\mathcal{L} = \{I_1, \ldots, I_n\}$ and $\theta = \{(\mathbf{x}, I_k) \in \{\mathbf{x}_1, \ldots, \mathbf{x}_k\} \times \mathcal{L} | x \in \mathbf{I}_k\}$. Then, $\tau^{dec} = (\tau, \mathcal{L}, \theta) \in u\_dec(Q, \Pi)$. Since every $\varphi_i(\mathbf{I}_1, \ldots, \mathbf{I}_n)$ holds over $\mathbf{A}_\tau$, $\tau^{dec} \in u\_dec^{pf}(\mathcal{R}_i, Q_{\mathcal{M}}, \Pi_{\mathcal{M}})$ for every internal rule $\mathcal{R}_i$ of $\Pi_{\mathcal{M}}$, so that $\tau^{dec} \in u\_dec^{pf}(\Pi_{\mathcal{M}}, Q, \Pi)$. By Theorem 4.1.7, $\exists \sigma \in u\_trees(Q_{\mathcal{M}}, \Pi^*_{\mathcal{M}})$ whose root is a leaf and a decorating containment mapping $h : \sigma \to \tau^{dec}$. Let $\mathcal{Q}_i$ be the rule of the root of σ: $R_1(\mathbf{v}^1), \ldots, R_m(\mathbf{v}^m) \Rightarrow Q_{\mathcal{M}}(\mathbf{v})$. h maps the variables in σ to variables in τ, and preserves the distinguished variables. Therefore, $\overline{\varphi'_i}(\mathbf{I}_1, \ldots, \mathbf{I}_n)$ holds on $\mathbf{A}_\tau$, and so does ψ'. This contradicting our first assumption, we conclude that, if Π does not satisfy ψ', it is not contained in $\Pi_{\mathcal{M}}$.

Let us assume now that Π satisfies ψ'. Let $\tau^{dec} = (\tau, \mathcal{L}, \theta) \in u\_dec^{pf}(\Pi_{\mathcal{M}}, Q, \Pi)$ and, $\forall k, \mathbf{I}_k = \{v \in var(\tau) | (v, I_k) \in \theta\} \subseteq var(\tau)$. Since ψ' holds on $\mathbf{A}_\tau$, so does $\bigwedge_{i=1}^{s} \varphi'_i(\mathbf{I}_1, \ldots, \mathbf{I}_n) \Rightarrow \bigvee_{i=1}^{s'} \overline{\varphi'_i}(\mathbf{I}_1, \ldots, \mathbf{I}_n)$. $\tau^{dec} \in u\_dec^{pf}(\mathcal{R}_i, Q, \Pi)$, which proves that, for every internal rule $\mathcal{R}_i$ of $\Pi_{\mathcal{M}}$, $\varphi'_i(\mathbf{I}_1, \ldots, \mathbf{I}_n)$ holds on $\mathbf{A}_\tau$. Therefore, $\bigwedge_{i=1}^{s} \varphi'_i(\mathbf{I}_1, \ldots, \mathbf{I}_n)$ holds on $\mathbf{A}_\tau$, and so does $\bigvee_{i=1}^{s'} \overline{\varphi'_i}(\mathbf{I}_1, \ldots, \mathbf{I}_n)$, as well as some $\overline{\varphi'_i}(\mathbf{I}_1, \ldots, \mathbf{I}_n)$. Let $\mathcal{Q}_i$ be the corresponding rule of $\Pi_{\mathcal{M}}$: $R_1(\mathbf{v}^1), \ldots, R_m(\mathbf{v}^m) \Rightarrow Q_{\mathcal{M}}(\mathbf{v})$, with internal IDB predicates $R_{i_1}, \ldots, R_{i_l}$. Then, let $\mathcal{Q}^*_i$ be a rule of $\Pi^*_{\mathcal{M}}$: $R'_1(\mathbf{v}^1), \ldots, R'_m(\mathbf{v}^m) \Rightarrow Q_{\mathcal{M}}(\mathbf{v})$, where $R'_i = R^*_i$ if R_i is an internal IDB predicate of $\Pi_{\mathcal{M}}$, and $R'_i = R_i$ if R_i is an EDB predicate of $\Pi_{\mathcal{M}}$. If σ^* is the tree in $u\_trees(Q_{\mathcal{M}}, \Pi^*_{\mathcal{M}})$ whose only node is $(Q_{\mathcal{M}}(\mathbf{v}), \mathcal{Q}^*_i)$, then some decorating containment mapping maps σ^* to τ^{dec}. This statement is true $\forall \tau^{dec} \in u\_dec^{pf}(\Pi_2, Q, \Pi)$, and it follows from Theorem 4.1.7 that Π is contained in $\Pi_{\mathcal{M}}$. This completes our proof. □

Corollary B.2.2. *Equivalence of* DATALOG *programs to monadic programs is decidable.* ■

B.3 Transitive Programs

Similarly, the decidability of containment in transitive programs follows from Theorem B.1.1.

Theorem B.3.1. *Containment of* DATALOG *programs in transitive* DATALOG *programs is decidable.* ∎

Proof. Let us assume that Π is a DATALOG program with goal predicate Q. Let γ be an arbitrary transitive DATALOG program, with predicates $P_1, \ldots, P_n$ and with rules $\mathcal{R}_1, \ldots, \mathcal{R}_{|\gamma\|}$.

- Let P_a be an EDB predicate. We associate P_a to the MSO formula
$$\varphi_a(\mathbf{v}) \equiv P_a(\mathbf{v})$$
- Let $\mathcal{R}_i$ be a star rule $P_b(x,z) \wedge P_a(z,y) \Rightarrow P_a(x,y)$. We associate P_a to the monadic second-order formula
$$\varphi_a(x,y) \equiv (\forall \mathbf{S})\,(\exists u, v)\,(x \notin \mathbf{S} \vee y \in \mathbf{S} \vee (u \in \mathbf{S} \wedge v \notin \mathbf{S} \wedge \varphi_b(u,v)))$$
- Let $\mathcal{R}_i$ be a star-free rule $P_{n_1}(\mathbf{v}^1) \wedge \ldots \wedge P_{n_k}(\mathbf{v}^k) \Rightarrow P_{n_0}(\mathbf{v}^0)$. We associate $\mathcal{R}_i$ to the monadic second-order formula
$$\phi_i(\mathbf{v}^0) \equiv (\exists y_1, \ldots, y_m) \left(\bigwedge_{i=1}^{k} \varphi_{n_i}(\mathbf{v}^i) \right)$$
where $y_1, \ldots, y_m$ are the variables appearing in $\mathcal{R}_i$ but not in $\mathbf{v}^0$.
- Let P_a be a goal or star-free IDB predicate, and $\mathcal{R}_{n_1}, \ldots, \mathcal{R}_{n_k}$ the rules whose head predicates are P_a. We associate P_a to the MSO formula
$$\varphi_a(\mathbf{v}) \equiv \bigvee_{i=1}^{k} \phi_{n_i}(\mathbf{v})$$

It is straightforward that, if the goal predicate of γ is the k-ary predicate P_Γ, then $\gamma(x_1, \ldots, x_k)$ is equivalent to $\varphi_\Gamma(x_1, \ldots, x_k)$.

Now, we write the formula $\gamma(x_1, \ldots, x_k)$ in normal disjunctive form:

$$\begin{aligned} \gamma(x_1, \ldots, x_k) \;\equiv\; & (\forall \mathbf{S}_1^1, \ldots, \mathbf{S}_1^{k_1})(\exists v_1^1, \ldots, v_1^{k'_1}) \ldots \\ & (\forall \mathbf{S}_l^1, \ldots, \mathbf{S}_l^{k_l})(\exists v_l^1, \ldots, v_l^{k'_l})(\Phi(\mathbf{S}, \mathbf{v})) \end{aligned}$$

where $\Phi(\mathbf{S}, \mathbf{v})$ is a MSO formula $\Phi(\mathbf{S}, \mathbf{v}) \equiv \bigvee_{i=1}^{c} \left(\bigwedge_{j=1}^{d_i} \theta_{i,j}(\mathbf{S}, \mathbf{v}) \right)$ with free sets among $\mathbf{S}_1^1, \ldots, \mathbf{S}_1^{k_1}, \ldots, \mathbf{S}_l^1, \ldots, \mathbf{S}_l^{k_l}$, free variables among $x_1, \ldots, x_k, v_1^1, \ldots, v_1^{k'_1}, \ldots, v_l^1, \ldots, v_l^{k'_l}$ and such that each atom $\theta_{i,j}(\mathbf{S}, \mathbf{v})$ is an EDB atom $P_a(w_1, \ldots, w_l)$, an atom $w \in \mathbf{S}_i$ or an atom $w \notin \mathbf{S}_i$.

Now, we translate $\gamma(x_1, \ldots, x_k)$ into a formula $\gamma'(x_1, \ldots, x_k)$ over $vocab(Q, \Pi)$, where

$$\begin{aligned} \gamma'(\mathbf{x}_1, \ldots, \mathbf{x}_k) \;\equiv\; & (\forall \mathbf{S}_1^1, \ldots, \mathbf{S}_1^{k_1} \subseteq V)(\exists v_1^1, \ldots, v_1^{k'_1} \in V) \ldots \\ & (\forall \mathbf{S}_l^1, \ldots, \mathbf{S}_l^{k_l} \subseteq V)(\exists v_l^1, \ldots, v_l^{k'_l} \in V)(\Phi'(\mathbf{S}, \mathbf{v}')) \end{aligned}$$

where $\Phi'(\mathbf{S}, \mathbf{v}')$ is MSO a formula $\Phi'(\mathbf{S}, \mathbf{v}') \equiv \bigvee_{i=1}^{c} \left(\bigwedge_{j=1}^{d_i} \theta'_{i,j}(\mathbf{S}, \mathbf{v}') \right)$ with free sets among $\mathbf{S}_1^1, \ldots, \mathbf{S}_1^{k_1}, \ldots, \mathbf{S}_l^1, \ldots, \mathbf{S}_l^{k_l}$, free variables among $\mathbf{x}_1, \ldots, \mathbf{x}_k, v_1^1, \ldots, v_1^{k'_1}, \ldots, v_l^1, \ldots, v_l^{k'_l}$ and

- if $\theta_{i,j}(\mathbf{S}, \mathbf{v})$ is an EDB atom $P_a(w_1, \ldots, w_l)$, then we set, $\forall b \leq l, w'_b = \mathbf{x}_m$ if w_b is a variable x_m, and $w'_b = w_b$ if $w_b \notin \{x_1, \ldots, x_k\}$. Then, $\theta'_{i,j}(\mathbf{S}, \mathbf{v}') \equiv P'_a(P_a, w'_1, \ldots, w'_l)$.
- if $\theta_{i,j}(\mathbf{S}, \mathbf{v})$ is an atom $w \in \mathbf{S}_l$, then we set $w' = \mathbf{x}_m$ if w is a variable x_m, and $w' = w$ if $w \notin \{x_1, \ldots, x_k\}$. Then, $\theta'_{i,j}(\mathbf{S}, \mathbf{v}') \equiv w' \in \mathbf{S}_l$.
- if $\theta_{i,j}(\mathbf{S}, \mathbf{v})$ is an atom $w \notin \mathbf{S}_l$, then we set $w' = \mathbf{x}_m$ if w is a variable x_m, and $w' = w$ if $w \notin \{x_1, \ldots, x_k\}$. Then, $\theta'_{i,j}(\mathbf{S}, \mathbf{v}') \equiv w' \notin \mathbf{S}_l$.

We claim that Π is contained in γ if and only if Π satisfies γ'.

Indeed, let $\tau \in u\_trees(Q, \Pi)$ be an unfolding expansion tree. It is straightforward that γ' holds on $\mathbf{A}_\tau$ exactly $\exists l = (\tau, \{\Gamma(\mathbf{v})\}, \rho, V, \varphi) \in \mathcal{L}(\gamma, \Pi)$ such that $\varphi(\mathbf{v}) = \mathbf{t}$ and $\mathcal{I}_N(l) \equiv \top$. Therefore, by Proposition A.4.11, and since $\mathcal{U}(p\_trees(Q, \Pi)) = u\_trees(Q, \Pi)$, Π is contained in γ if and only if $\forall \tau \in u\_trees(Q, \Pi)$, γ' holds on $\mathbf{A}_\tau$, which means that Π satisfies γ'.

Since γ' is a monadic second-order formula over $vocab(Q, \Pi)$,our proof is complete. □

Corollary B.3.2. *Equivalence of* DATALOG *programs to transitive programs is decidable.*
■

B.4 Comments

Unfortunately, Theorem B.1.1 yields a very high upper bound; the algorithm described in [7] is of non-elementary time complexity, i.e., its time complexity cannot be bounded by any finite stack of exponentials.

This is why the decidability of the containment problem in monadic DATALOG programs and in transitive DATALOG programs was indeed proved by COURCELLE's theorem, but that we still had to look for the existence of algorithms that would give better complexity bounds.