Enhancing and solving linear programs with conjunctive queries

Thesis

to obtain the title of *PhD of Science* of the *Université de Lille*

Specialty : Computer Science

Defended on 27/02/2023 by

Nicolas Crosetti

Thesis Supervisors: Joachim Niehren, Sophie Tison
Thesis Advisors: Florent Capelli, Jan Ramon

Jury :

Reviewers :	Pierre Senellart	Professor	ENS Université PSL
	Bruno Zanuttini	Professor	Université de Caen Normandie
Supervisors :	Joachim Niehren	Research Director	Inria Université de Lille
	Sophie Tison	Professor Emeritus	Université de Lille
President :	Pierre Marquis	Professor	Université d'Artois
Examiner :	Luce Brotcorne	Research Director	Inria Université de Lille
Invited :	Florent Capelli	Assistant Professor	Université de Lille
	Jan Ramon	Research Director	Inria Université de Lille

Université de Lille

Enrichir et résoudre des programmes linéaires avec des requêtes conjonctives

Thèse

pour obtenir le titre de *docteur* de l' *Université de Lille*

Spécialité : Informatique

Soutenue le 27/02/2023 par

Nicolas Crosetti

Directeur·rice de thèse : Joachim Niehren, Sophie Tison
Encadrants de thèse : Florent Capelli, Jan Ramon

Jury :

Rapporteurs :	Pierre Senellart	Professeur	ENS Université PSL
	Bruno Zanuttini	Professeur	Université de Caen Normandie
Co-directeur :	Joachim Niehren	Directeur de recherche	Inria Université de Lille
Co-directrice :	Sophie Tison	Professeure émerite	Université de Lille
Président :	Pierre Marquis	Professeur	Université d'Artois
Examinatrice :	Luce Brotcorne	Directrice de recherche	Inria Université de Lille
Invités :	Florent Capelli	Maître de conférences	Université de Lille
	Jan Ramon	Directeur de recherche	Inria Université de Lille

Abstract

Mathematical optimization and data management are two major fields of computer science that are widely studied by mostly separate communities. However complex optimization problems often depend on large datasets that may be cumbersome to manage, while managing large amounts of data is only useful insofar as one analyzes this data to extract some knowledge in order to solve some practical problem, so these fields are often actually intertwined in practice. This thesis places itself at the crossroads between these two fields by studying linear programs that reason about the answers of database queries.

The first contribution of this thesis is the definition of the so-called language of linear programs with conjunctive queries, or $LP(CQ)$ for short. It is a language to model linear programs with constructs that allow one to express linear constraints and linear sums that reason over the answer sets of database queries in the form of conjunctive queries. We then describe the natural semantics of the language by showing how such models can be interpreted, in conjunction with a database, into actual linear programs that can then be solved by any standard linear program solver and discuss the hardness of solving $LP(CQ)$ models.

Motivated by the hardness of solving $LP(CQ)$ models in general, we then introduce a process based on the so-called T-factorized interpretation to solve such models more efficiently. This approach is based on classical techniques from database theory to exploit the structure of the queries using hypertree decompositions of small width. The T-factorized interpretation yields a linear program that has the same optimal value as the natural semantics of the model but fewer variables which can thus be used to solve the model more efficiently.

The third contribution is a generalization of the previous result to the framework of factorized databases. We introduce a specific circuit data-structure to succintly encode relations. We the define the so-called C-factorized interpretation that leverages the succintness of these circuits to yield a linear program that has the same optimal value as the natural semantics of the model but fewer variables similarly to the T-factorized interpretation. Finally we show that we can explicitly compile the answer sets of conjunctive queries with small fractional hypertreewidth into succinct circuits, thus allowing us to recapture the T-factorized interpretation.

Keywords: Database theory, Linear programing, Conjunctive Queries, Factorized databases, Knowledge compilation

Résumé

L'optimisation mathématique et la gestion des données sont deux domaines majeurs de l'informatique qui sont largement étudiés par des communautés essentiellement distinctes. Cependant, les problèmes d'optimisation complexes dépendent souvent de grands jeux de données qui peuvent être difficiles à gérer, alors que la gestion de grandes quantités de données n'est utile que dans la mesure où l'on analyse ces données pour en extraire des connaissances afin de résoudre un problème pratique, de sorte que ces domaines sont souvent entremêlés en pratique. Cette thèse se place à la croisée de ces deux domaines en étudiant les programmes linéaires qui raisonnent sur les réponses de requêtes de bases de données.

La première contribution de cette thèse est la définition de ce que nous appelons le langage des programmes linéaires avec requêtes conjonctives (que nous noterons $LP(CQ)$). Il s'agit d'un langage de modélisation de programmes linéaires avec des constructions permettant d'exprimer des contraintes et sommes linéaires qui raisonnent sur les ensembles de réponses de requêtes de bases de données sous forme conjonctive. Nous décrivons ensuite la sémantique naturelle du langage en montrant comment de tels modèles peuvent être interprétés, en conjonction avec une base de données, en de vrais programmes linéaires qui peuvent ensuite être résolus par tout solveur de programmes linéaires standard et nous discutons de la difficulté de résoudre les modèles $LP(CQ)$.

Motivés par la difficulté de résoudre les modèles $LP(CQ)$ en général, nous introduisons ensuite un processus basé sur ce que nous appelons l'interprétation T-factorisée pour résoudre de tels modèles plus efficacement. Cette approche est basée sur des techniques classiques en théorie des bases de données pour exploiter la structure des requêtes en utilisant des décompositions arborescentes de petite largeur. L'interprétation T-factorisée produit un programme linéaire qui a la même valeur optimale que la sémantique naturelle du modèle mais moins de variables et qui peut donc être utilisé pour résoudre le modèle plus efficacement.

La troisième contribution est une généralisation du résultat précédent au cadre des bases de données factorisées. Nous introduisons une structure de données spécifique pour coder succinctement les relations sous forme de circuit. Nous définissons ensuite l'interprétation dite C-factorisée qui exploite le caractère succinct de ces circuits pour produire un programme linéaire qui a la même valeur optimale que la sémantique naturelle du modèle mais avec moins de variables de manière similaire à l'interprétation T-factorisée. Enfin, nous montrons que nous pouvons explicitement compiler les ensembles de réponses de requêtes conjonctives admettant une décomposition de petite largeur en circuits succincts, ce qui nous permet de récapturer l'interprétation T-factorisée.

Mots-clés: Théorie des bases de données, Programmation linéaire, Requêtes conjonctives, Bases de données factorisées, Compilation de connaissances

Acknowledgments

Je remercie mes rapporteurs, Pierre Senellart et Bruno Zanuttini, pour avoir accepté de relire ce manuscrit. Je remercie également Luce Brotcorne et Pierre Marquis d'avoir accepté de faire partie de mon jury.

Je remercie mes encadrants pour leur soutien indéfectible malgré les revers et les difficultés. Je remercie les membres de l'équipe Links pour leur soutien et leurs conseils. Je remercie plus particulièrement les doctorants, présents et passés, de l'équipe avec qui nous avons perpétué une ambiance éminemment studieuse dans notre bureau afin de mener nos thèses à bien.

Je remercie finalement ma famille et mes amis pour m'avoir soutenu dans les moments difficiles et sans qui je ne serai pas arrivé jusque là.

Funding

Cette thèse a été cofinancée par l'ANR Headwork et par le conseil régional Hauts-de-France.

Contents

Chapter 1
Introduction

Mathematical optimization and data management are two major fields of computer science that are widely studied by mostly separate communities. However complex optimization problems often depend on large datasets that may be cumbersome to manage, while managing large amounts of data is only useful insofar as one analyzes this data to extract some information(s) to solve some practical problem, so these fields are often actually intertwined in practice. This thesis places itself at the crossroads between these two fields by studying linear programs that reason about the answers of database queries. In this introduction we will gradually introduce notions from both fields to motivate and contextualize our contributions.

We will begin by giving some short context about linear programming. We will then discuss how linear programs are usually modeled. This then brings us to discussing some existing links between linear programs and databases to motivate the introduction of the $LP(CQ)$ language as the first contribution of this thesis in order to model linear programs that reason about the answers of database queries. We then briefly discuss the complexity of solving linear programs based on databases to motivate our second contribution which is a process based on the so-called T-factorized interpretation to solve such models more efficiently. Finally the third contribution is a generalization of the second result using so-called $\{\uplus, \times\}$-circuits to succinctly encode the answer sets of queries.

Linear programming and modelization

Linear programming is a subfield of mathematical optimization that aims at maximizing (or minimizing) a linear function under a system of linear constraints. This term was coined by George Dantzig in 1947 but linear programming had been independently studied by Leonid Kantorovitch since 1939. In both cases linear programing was motivated by logistical problems encountered by armed forces. From then on linear programing has been widely used in economics, logistics, scheduling etc...

An interesting property of optimization problems is that it is often possible to describe the logical constraints of the system independently from the underlying data. We illustrate this with a very simplistic example. Consider an unspecified firm that wants to optimize its production to maximize its commercial revenue while accounting for its storage capacity. We assume for brevity that every product is equivalent in terms of required storage even though this makes the problem trivial to solve. This problem can be expressed by a simple model without any specific prior knowledge about the products:

$$\begin{aligned}\textbf{maximize} \quad & \sum_{p \in products} quantity(p) \times price(p) \\ \textbf{subject to} \quad & \sum_{p \in products} quantity(p) \leq max\_quantity\end{aligned}$$

Note that in this case we would consider the *price* of each product p to be a parameter of the problem to be provided as part of the input dataset. On the other hand the *quantity* of produced p would be an unknown variable that should be optimized to maximize the revenue. Observe that this sum is indeed linear under these assumptions.

An advantage of modelling problems this way is that such a model can then easily be reused with different datasets. This idea is at the core of modelling languages for linear programming such as AMPL [FGK90] and GNU Mathprog and more general languages for constraint programming such as MiniZinc [NSB+07]. Using GNU Mathprog we can write our partial linear program model very similarly to the formula we used previously:

```
set Products;
param Price{p in Products};
param Max_quantity;
var quantity{p in Products} >= 0;

maximize revenue:
    sum{p in Products} quantity[p] * Price[p];
s.t. max_production:
    sum{p in Products} quantity[p] <= Max_quantity;
```

The dataset can then be provided in a *data* section that follows the model. For example, we might provide a simple dataset inline as follows:

```
set Products := Baguette Croissant;

param Price :=
    Baguette    1.1
    Croissant   0.9;

param Max_quantity := 100;
```

Observe that specifying a large dataset in this manner would be impractical. To avoid this issue, modeling languages propose bindings to CSV files or databases. We will now discuss the links betweeen linear programs and databases.

Linear programming and databases

Linear programming has been used in the context of database research, when using integer linear programming for finding optimal database repairs as proposed by Kolaitis, Pema and Tan [KPT13], or when using linear optimization to explain the result of a database query to the user as proposed by Meliou and Suciu [MS12].

In general, given the common separation between the logical model of a linear program and its dataset, it seems natural to be able to store this data in a database. Thus, being able to bind a table of parameters or a set of variables to the answer of an SQL query is a feature of linear programming modeling languages such as AMPL or GNU Mathprog.

```
table products IN "ODBC" "Driver=SQLITE3;Database=bakery.sqlite"
"SELECT name, price FROM Products":
    Products <- [name], Price ~ price;
```

Better integration of optimization problems directly into DBMS has also already been investigated. Cadoli and Mancini [CM07] introduced an extension of `SQL` called `ConSQL` that allows to select optimal solutions to constraint problems directly in `SQL`-like queries. Šikšnys and Pedersen introduced `SolveDB` in [ŠP16] which is also an extension of `SQL` that allows to solve optimization problems directly in the querying language thus simplifying the usual workflow of feeding an `AMPL` programs with data extracted from a database.

Both languages proposed in these works are extensions of SQL that aim to integrate optimization problems into queries. In this thesis we follow a different philosophy by introducing the $LP(CQ)$ language which aims to integrate queries into LP models. At its core the $LP(CQ)$ language is a formalization of the mathematical notations one could use to model an optimization problem, similar to the AMPL and GNU Mathprog languages. A key difference between our language and existing LP modeling languages is in the way we connect sets of variables of the linear program with the database through queries. Models expressed in existing languages are based on sets variables and parameters that can then be populated by in-line values, the content of CSV files or the answers of SQL queries on a database. We instead limit ourselves to populating our sets of variables using only queries but we also offer a way to reason about the answers of these queries inside the model while still separating it from the database. In order to make use of known theoretical results we use *Conjunctive Queries* rather than SQL queries without losing much expressivity[1].

This is possible by writing linear sums with the **weight** operator that allows one to refer to a specific subset of the variables bound to the answer of a query. Our previous partial LP example can be expressed in the $LP(CQ)$ language as follows:

[1] Indeed conjunctive queries cover simple SELECT . . . FROM . . . WHERE . . . queries

$$\begin{array}{rl}
\textbf{maximize} & \sum_{(n',p'):Q'} \textbf{num}(p')\ \textbf{weight}_{(n):n \doteq n'}(Q) \\
\textbf{subject to} & \textbf{weight}_{(n):true}(Q) \leq Max\_quantity \\
& \quad \text{where } Q(n) = \exists p.Products(n,p) \\
& \quad \text{and } Q'(n',p') = Products(n',p')
\end{array}$$

Intuitively, the query $Q(n)$ selects the names of all the products and a $\textbf{weight}_{(n):q}(Q)$ operator then refers to the combined produced quantity of every product that satisfies the restriction expressed with q. Thus each occurrence of $\textbf{weight}_{(n):n \doteq n'}(Q)$ in the sum captures one specific product named n' while $\textbf{weight}_{(n):true}(Q)$ captures every product. The $LP(CQ)$ language is the first contribution of this thesis and we will describe its semantics in details in Chapter 3.

Now while it may appear at first glance that we have simply duplicated[2] and moved our query to the model, this change actually allows us to exploit the structure of the queries to get some interesting efficiency results when solving $LP(CQ)$ models.

Tractable results on conjunctive queries

It is well known that solving a linear program is polynomial in its number of variables [Kar84]. Thus the complexity of solving an LP model actually depends on the size of the dataset it is solved with. This issue is further intensified when the dataset is described by a database query as it is well known that answering queries is NP-hard in general which impacts the linear program. However it is known that some classes of queries are tractable and the $LP(CQ)$ language allows us to to exploit the structure of the queries. In this thesis we will present a technique to solve some $LP(CQ)$ models more efficiently as a second contribution. These first two contributions were previously published in [CCNR22]. This technique builds on well-known techniques using dynamic programming on tree decompositions of the hypergraph of conjunctive queries.

These techniques were first introduced by Yannakakis [Yan81] who observed that so-called acyclic conjunctive queries could be answered in linear time using dynamic programming on a tree whose nodes are in correspondence with the atoms of the query. Generalizations have followed in two directions: on the one hand, generalizations of acyclicity such as notions of hypertree width [GLS02, GLS99, Gro06] have been introduced and on the other hand enumeration and aggregation problems have been shown to be tractable on these families of queries such as finding the size of the answer set [PS13] or enumerating it with small delay [BDG07]. The T-factorized interpretation mentioned above heavily draws inspiration from this approach as we use bottom up dynamic programming on a hypertree decomposition of each input query Q to construct a partial representation of the answers set of

[2] The queries Q and Q' are equivalent to "SELECT name FROM Products" and "SELECT name, price FROM Products" in SQL.

Q on database $\mathbb{D}$ that we then use to construct the more succinct T-factorized interpretation of the linear program that can thus be solved more efficiently.

We have seen that to exploit the structure of queries through their decomposition is general enough that it can be adapted to various problems. However, in practice, each of these algorithms reimplements the same common dynamic programming framework with only a few differences to handle the specific problem at hand. This idea has thus been generalized with *factorized databases* which allows on to abstract away this cumbersome framework to a sort of precomputation phase.

Generalization to factorized databases

The tractability results presented previously can be obtained in a unified and generalized manner by using factorized databases introduced by Olteanu and Závodný [OZ12, OZ15a], from which our third contribution is inspired. Factorized databases provide succinct representations for answer sets of queries on databases. The representation enjoys interesting syntactic properties allowing to efficiently solve numerous aggregation problems on answer sets in polynomial time in the size of the representation. Olteanu and Závodný [OZ15a] have shown that when the fractional hypertree width of a query Q is bounded, then one can construct, given a hypertree decomposition of Q and a database $\mathbb{D}$, a factorized databases representing the answers of Q on $\mathbb{D}$ of polynomial size. They also give a $O(1)$ delay enumeration algorithm on factorized databases. Combining both results gives a generalization of the result of Bagan, Durand and Grandjean [BDG07] on the complexity of enumerating the answers of conjunctive queries.

Our third contribution is thus a generalization of the T-factorized interpretation (dubbed the C-factorized interpretation) that leverages a factorized representation of the answer sets of the queries to avoid repeating the reoccuring dynamic programming framework. However, rather than using factorized representations as defined by Olteanu and Závodný [OZ15a], we instead introduce our own factorized representation, the so-called $\{\uplus, \times\}$-circuits after which the circuit-based factorized interpretation takes its name. While $\{\uplus, \times\}$-circuits are very similar to these factorized representations, they are slightly more general. More importantly, they are the target of a specific compilation algorithm that allows us to encode some additional informations that are necessary to solve some $LP(CQ)$ models. This allows us to recapture the result of the T-factorized interpretation.

Outline of the thesis

We now outline the contents of this thesis.

Chapter 2 introduces the basic notions we will use throughout this thesis, namely conjunctive queries and linear programs.

Chapter 3 is dedicated to the first contribution of this thesis, the $LP(CQ)$ language. We present the syntax and semantics of this language in details. We also formally study the complexity and hardness of solving $LP(CQ)$ models. Finally we formally introduce the notion of alternate interpretations of $LP(CQ)$ models to prepare the following chapters.

Motivated by the general hardness of solving $LP(CQ)$ language, we characterize a tractable a fragment of $LP(CQ)$ models in Chapter 4. We then present the so-called T-factorized interpretation as the second contribution of this thesis. This interpretation leverages the tree-decomposition of conjunctive queries to build a linear program that is smaller than the semantics of the model while having the same optimal value thus providing a way to solve the model more efficiently. We also show that we can then efficiently compute a solution of the semantics of the model.

In Chapter 5 we introduce the so-called $\{\uplus, \times\}$-circuits as a succinct structure to encode relations (and in particular the answer sets of conjunctive queries with bounded width). We then use the $\{\uplus, \times\}$-circuits to define the so-called C-factorized interpretation as the third main contribution of this thesis. Finally we show that the C-factorized interpretation generalizes the T-factorized interpretation of the previous chapter.

Finally in Chapter 6 we present a few ways in which our three main contributions can be taken further by relaxing or lifting some of the restrictions we worked with in the previous chapters.

Chapter 2
Preliminaries

Contents

2.1 General notations

2.1.1 Sets, Functions and Relations

Let $\mathbb{B} = \{0, 1\}$ be the set of Booleans, $\mathbb{N}$ the set of natural numbers including 0 and $\mathbb{Z}$ the set of integers. Let $\mathbb{R}^+$ be the set of positive reals including 0 and subsuming $\mathbb{N}$ and $\mathbb{R}$ the set of all reals.

Given any set S and $n \in \mathbb{N}$ we denote by S^n the set of all n-tuples over S and by $S^* = \cup_{n \in \mathbb{N}} S^n$ the set of all words over S. A *weighting* on S is a (total) function $f : S \to \mathbb{R}^+$.

Given a set of (total) functions $A \subseteq D^S = \{f \mid f : S \to D\}$ and a subset $S' \subseteq S$, we define the set of restrictions $A_{|S'} = \{f_{|S'} \mid f \in A\}$. For any binary relation $R \subseteq S \times S$, we denote its transitive closure by $R^+ \subseteq S \times S$ and the reflexive transitive closure by $R^* = R^+ \cup \{(s, s) \mid s \in S\}$.

2.1.2 Rooted trees

A digraph is a pair $(\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge sets $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. A digraph is acyclic if there is no $v \in \mathcal{V}$ for which $(v, v) \in \mathcal{E}^+$. For any node $u \in \mathcal{V}$, we denote by $\downarrow u = \{v \in \mathcal{V} \mid (u, v) \in \mathcal{E}^*\}$ the set of nodes in $\mathcal{V}$ reachable over some downwards path from u, and by $\uparrow u = (\mathcal{V} \setminus \downarrow u) \cup \{u\}$ the context of u. A *rooted tree* is an acyclic digraph where $(u,v),(u',v) \in \mathcal{E}$ implies $u = u'$, and there exists a node $r \in \mathcal{V}$ such that $\mathcal{V} = \downarrow r$. In this case, r is unique and called the root of the tree. Observe that in this tree, the paths are oriented from the root to the leaves of the tree.

2.1.3 Variable assignments

We fix a countably infinite set of variables $\mathcal{X}$. For any domain *dom*, an assignment of variables to domain elements is a function $\alpha : X \to dom$ that maps elements of a finite subset of variables $X \subseteq \mathcal{X}$ to values of *dom*. We denote the empty assignment by ϵ (that is $\epsilon : \emptyset \to dom$). For any two sets of variable assignments $A_1 \subseteq dom^{X_1}$ and $A_2 \subseteq dom^{X_2}$ we define their join $A_1 \bowtie A_2 = \{\alpha_1 \cup \alpha_2 \mid \alpha_1 \in A_1, \alpha_2 \in A_2, \alpha_{1|I} = \alpha_{2|I}\}$ where $I = X_1 \cap X_2$.

Let $A \subseteq dom^X = \{\alpha \mid \alpha : X \to dom\}$ be a set of variable assignments on a finite set of variables X. Let $X' \subseteq X \subseteq \mathcal{X}$. For any $\alpha' : X' \to dom$ we define the set of its extensions into A by $A[\alpha'] = \{\alpha \in A \mid \alpha_{|X'} = \alpha'\}$. Moreover, given a weighting $\omega : A \to \mathbb{R}^+$ of A, we define its projection on X' as $\pi_{X'}(\omega) : A_{|X'} \to \mathbb{R}^+$ such that for all $\alpha' \in A_{|X'}$: $\pi_{X'}(\omega)(\alpha') = \sum_{\alpha \in A[\alpha']} \omega(\alpha)$.

We also use a few vector notations. Given a vector of variables $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ we denote by $set(\mathbf{x}) = \{x_1, \ldots, x_n\}$ the set of the elements of $\mathbf{x}$. For any variable assignment $\alpha : X \to dom$ with $set(\mathbf{x}) \subseteq X$ we denote the application of the assignment α on $\mathbf{x}$ by $\alpha(\mathbf{x}) = (\alpha(x_1), \ldots, \alpha(x_n))$.

2.2 Conjunctive queries

2.2.1 Relational Databases

A *database schema* is a pair $\Sigma = (R, \mathcal{C})$ where $\mathcal{C}$ is a finite set of constants and $R \subset \cup_{n \in \mathbb{N}} \mathcal{R}^{(n)}$ is a finite set of relation symbols. The elements $r \in \mathcal{R}^{(n)}$ are called relation symbols of arity $n \in \mathbb{N}$.

A *database* $\mathbb{D} \in db_\Sigma$ is a tuple $\mathbb{D} = (\Sigma, D, \cdot^{\mathbb{D}})$, where Σ is a schema, D a finite set of database elements, and $r^{\mathbb{D}} \subseteq D^n$ a relation for any relation symbol $r \in \mathcal{R}^{(n)}$ and $a^{\mathbb{D}} \in D$ a database element for any constant $a \in \mathcal{C}$. We also define the database's domain $dom(\mathbb{D}) = D$.

A *database with real numbers* is a tuple $\mathbb{D} = (\Sigma, D, \cdot^{\mathbb{D}}, \mathbf{num}^{\mathbb{D}})$ such that $\mathbb{D} = (\Sigma, D, \cdot^{\mathbb{D}})$ is a relational database and $\mathbf{num}^{\mathbb{D}}$ a partial function from D to $\mathbb{R}$.

2.2.2 Conjunctive Queries

In Figure 2.1 we recall the notion of conjunctive queries on relational databases. An expression $E \in Ex_{\mathcal{C}}$ is either a (query) variable $x \in \mathcal{X}$ or a constant $a \in \mathcal{C}$. The set of conjunctive queries $Q \in CQ_\Sigma$ is built from equations $E_1 \doteq E_2$, atoms $r(E_1, \ldots, E_n)$, the logical operators of conjunction $Q \wedge Q'$, existential quantification

$$\begin{array}{lll}
\text{Expressions} & E_1, \dots, E_n \in Ex_{\mathcal{C}} \quad ::= & x \mid a \\
\text{Conjunctive queries} & Q, Q' \in CQ_{\Sigma} \quad ::= & E_1 \doteq E_2 \mid r(E_1, \dots, E_n) \\
 & & \mid Q \wedge Q' \mid \exists x.Q \mid \mathit{true}
\end{array}$$

Figure 2.1: The set of conjunctive queries CQ_Σ with schema $\Sigma = ((\mathcal{R}^{(n)})_{n\in\mathbb{N}}, \mathcal{C})$ where $x \in \mathcal{X}$, $a \in \mathcal{C}$, and $r \in \mathcal{R}^{(n)}$.

$$\begin{array}{rcl}
\mathit{eval}^{\mathbb{D},\alpha}(x) & = & \alpha(x) \\
\mathit{eval}^{\mathbb{D},\alpha}(a) & = & a^{\mathbb{D}} \\
\mathit{ans}^{\mathbb{D}}_{X}(E_1 \doteq E_2) & = & \{\alpha : X \to D \mid \mathit{eval}^{\mathbb{D},\alpha}(E_1) = \mathit{eval}^{\mathbb{D},\alpha}(E_2)\} \\
\mathit{ans}^{\mathbb{D}}_{X}(r(E_1, \dots, E_n)) & = & \{\alpha : X \to D \mid (\mathit{eval}^{\mathbb{D},\alpha}(E_1), \dots, \mathit{eval}^{\mathbb{D},\alpha}(E_n)) \in r^{\mathbb{D}}\} \\
\mathit{ans}^{\mathbb{D}}_{X}(Q_1 \wedge Q_2) & = & \mathit{ans}^{\mathbb{D}}_{X}(Q_1) \cap \mathit{ans}^{\mathbb{D}}_{X}(Q_2) \\
\mathit{ans}^{\mathbb{D}}_{X}(\exists x.Q) & = & \begin{cases} \{\alpha_{|X} \mid \alpha \in \mathit{ans}^{\mathbb{D}}_{X\cup\{x\}}(Q)\} & \text{if } x \notin X \\ \text{undefined} & \text{otherwise} \end{cases} \\
\mathit{ans}^{\mathbb{D}}_{X}(\mathit{true}) & = & \mathit{dom}(\mathbb{D})^X
\end{array}$$

Figure 2.2: Answer sets of conjunctive queries.

$\exists x.Q$ and the tautology *true*. Given a vector $\mathbf{x} = (x_1, \dots, x_n) \in \mathcal{X}^n$ and a query Q, we write $\exists \mathbf{x}.Q$ instead of $\exists x_1. \dots .\exists x_n.Q$.

The set of free variables $\mathit{fv}(Q) \subseteq \mathcal{X}$ are those variables that occur in Q outside the scope of an existential quantifier. A conjunctive query Q is said to be *quantifier free* if it does not contain any existential quantifier.

For any conjunctive query $Q \in CQ_\Sigma$, set $X \supseteq \mathit{fv}(Q)$ and database $\mathbb{D} \in db_\Sigma$ we define the answer set $\mathit{ans}^{\mathbb{D}}_{X}(Q)$ in Figure 2.2. It contains all those assignments $\alpha : X \to \mathit{dom}(\mathbb{D})$ for which Q becomes true on $\mathbb{D}$. We also write $\mathit{ans}^{\mathbb{D}}(Q)$ instead of $\mathit{ans}^{\mathbb{D}}_{\mathit{fv}(Q)}(\mathbb{D})(Q)$. Observe that $\mathit{ans}^{\mathbb{D}}(\exists \mathbf{x}.Q) = \mathit{ans}^{\mathbb{D}}(Q)_{|\mathit{fv}(Q)\setminus \mathit{set}(\mathbf{x})}$.

2.2.3 Tree decompositions

In this section we define hypertree decompositions of conjunctive queries which are a way of laying out the structure of a conjunctive query in a tree. They allow one to solve many aggregation problems (such as checking the existence of a solution, counting or enumerating the solutions etc.) on quantifier free conjunctive queries in polynomial time where the degree of the polynomial is given by the width of the decomposition which we will define in the following section.

First we define decomposition trees of finite sets of variables that we will then lift to conjunctive queries.

Definition 2.1.
Let $X \subseteq \mathcal{X}$ be a finite set of variables. A decomposition tree T of X is a tuple $(\mathcal{V}, \mathcal{E}, \mathcal{B})$ such that:

- *$(\mathcal{V}, \mathcal{E})$ is a finite directed rooted tree with edges from the root to the leaves,*
- *the bag function $\mathcal{B} : \mathcal{V} \to 2^X$ maps nodes to subsets of variables in X,*
- *for all $x \in X$ the subset of nodes $\{u \in \mathcal{V} \mid x \in \mathcal{B}(u)\}$ is connected in the tree $(\mathcal{V}, \mathcal{E})$,*
- *each variable of X appears in some bag, that is $\bigcup_{u \in \mathcal{V}} \mathcal{B}(u) = X$.*

Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a decomposition tree for a finite set of variables X. Given two nodes $u, v \in \mathcal{V}$ we denote the intersection of their bags by $\mathcal{B}^{uv} = \mathcal{B}(u) \cap \mathcal{B}(v)$. For any decomposition tree $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ and subset $V \subseteq \mathcal{V}$ we define the set of variables:

$$\mathsf{Attr}(V) = \bigcup_{v \in V} \mathcal{B}(v)$$

In particular, this defines for any $v \in \mathcal{V}$ the union $\mathsf{Attr}(\uparrow v)$ of bags of vertices in-the-context-or-equal-to v, and the union $\mathsf{Attr}(\downarrow v)$ of bags of vertices that are descendants-or-equal-to v.

We now lift the notion of decomposition trees to conjunctive queries. A hypertree decomposition of a quantifier free conjunctive query is a decomposition tree where for each atom of the query there is at least one bag that covers its variables.

Definition 2.2 (Hypertree decompositions of quantifier free conjunctive queries)**.** *Let $Q \in CQ_\Sigma$ be a quantifier free conjunctive query.*

A generalized hypertree decomposition of *Q is a decomposition tree $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of $fv(Q)$ such that for each atom $r(\mathbf{x})$ of Q there is a vertex $u \in \mathcal{V}$ such that $set(\mathbf{x}) \subseteq \mathcal{B}(u)$.*

2.2.4 Width of tree decompositions

In this section we define the width of a query and discuss its influence on the complexity of query answering. We begin by defining the generalized hypertree width of a quantifier free query.

Definition 2.3 (Generalized hypertree width of quantifier free conjunctive queries)**.** *Let $Q \in CQ_\Sigma$ be a quantifier free conjunctive query and $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a hypertree decomposition of Q.*

The generalized hypertree width *of T with respect to Q is the minimal number k such that every bag of T can be covered by the variables of k atoms of Q.*

The generalized hypertree width of a query *Q is the minimal width of a tree decomposition of Q.*

We call a conjunctive query α-acyclic if it has generalized hypertree width 1. The query $r(x, y) \wedge r(y, z)$ has a generalized hypertree decomposition $(\mathcal{V}, \mathcal{E}, \mathcal{B})$ with $\mathcal{V} = \{1, 2, 3\}$, $\mathcal{E} = \{(1, 2), (1, 3)\}$, and $\mathcal{B} = [1/\{y\}, 2/\{x, y\}, 3/\{y, z\}]$ of width 1, so it is α-acyclic.

While generalized hypertree width allows to obtain efficient algorithms on conjunctive queries, it can be generalized to *fractional hypertree width*, which consists in a fractional relaxation of the generalized hypertree width, to obtain better bounds in some cases. Let $Q \in CQ_\Sigma$ be a quantifier free conjunctive query, $\mathcal{A}$ be the atoms of Q and let $X \subseteq fv(Q)$. A *fractional cover of* X is a function $c\colon \mathcal{A} \to \mathbb{R}^+$ assigning positive weights to the atoms of Q such that for every $x \in X$, $\sum_{R\in\mathcal{A}, x\in fv(R)} c(R) \geq 1$. The *value of a fractional cover* c is defined as $\sum_{R\in\mathcal{A}} c(R)$.

For example, consider the query $Triangle = R(x,y) \wedge S(y,z) \wedge T(z,x)$ and $X = \{x,y,z\}$. The function c such that $c(R) = c(S) = c(T) = 1/2$ is a fractional cover of X of value $3/2$.

Definition 2.4 (Fractional hypertree width of quantifier free conjunctive queries)**.** *Let Q be a quantifier-free conjunctive query and $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a generalized hypertree decomposition of Q.*

The fractional hypertree width *of T is the smallest k such that for every $u \in \mathcal{V}$, there exists a fractional cover of $\mathcal{B}(u)$ of value smaller than k. The* fractional hypertree width *of Q, denoted by* $\mathsf{fhtw}(Q)$*, is the smallest k such that Q has a generalized hypertree decomposition of fractional hypertree width k.*

Observe that the fractional hypertree width of a query is never larger and sometimes smaller than its generalized hypertree width. Indeed the generalized hypertree width of $Triangle$ is 2 while its previously mentioned fractional cover had a value of $3/2$.

In the rest of this thesis we will simply refer to the fractional hypertree width of T (or Q) as the width of T (or Q).

The key observation making fractional hypertree width suitable for algorithmic purposes is due to Grohe and Marx [GM14] who proved that if the free variables of a quantifier free conjunctive query Q has a fractional cover of value k then $|ans^{\mathbb{D}}(Q)| \leq |\mathbb{D}|^k$. Hence, if $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ is a tree decomposition of Q of width k, then $ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ is of size at most $|\mathbb{D}|^k$ for any $u \in \mathcal{V}$. Moreover, it can be computed efficiently:

Lemma 2.5.
Given a tree decomposition $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of a quantifier free conjunctive query $Q \in CQ_\Sigma$ of width k and a database $\mathbb{D} \in db_\Sigma$, one can compute the collection of bag projections $(ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)})_{u\in\mathcal{V}}$ in time $O((|\mathbb{D}|^k \log(|\mathbb{D}|)) \cdot |T|)$.

Moreover, for every $u \in \mathcal{V}$, $ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ is of size at most $|\mathbb{D}|^k$.

Lemma 2.5 is folklore: it can be proven by computing the semi-join of every bag in a subtree in a bottom-up fashion, as it is done in [Lib13, Theorem 6.25] and using a worst-case optimal join algorithm such as Triejoin [Vel14] for computing the relation at each bag. This yields a superset S_u of $ans^{\mathbb{D}}(Q)|_{\mathcal{B}(u)}$ for every u. Then, with a second top-down phase, one can remove tuples from S_u that cannot be extended to a solution of $ans^{\mathbb{D}}(Q)$.

Given a query Q, we define $\mathsf{AGM}(Q)$ as $\min(\{v(c) \mid \text{c is a fractional cover of Q}\})$ where $v(c)$ is the value of c. By the previously mentioned upper bound of [GM14], we have $ans^{\mathbb{D}}(Q) \leq |\mathbb{D}|^{\mathsf{AGM}(Q)}$. Interestingly Asterias, Grohe and Marx later proved in [AGM13] that this upper bound, now usually referred to as the AGM bound, is tight (up to polynomial factors). More precisely, they prove that there exists a database $\mathbb{D}^*$ such that $ans^{\mathbb{D}^*}(Q)$ is of size greater than $\frac{|\mathbb{D}^*|^{\mathsf{AGM}(Q)}}{poly(|Q|)}$.

Lemma 2.5 is then particularly interesting as it gives a way of describing the set of solutions of Q that is of size $|\mathbb{D}|^k$ that can be order of magnitudes more succinct than representing $ans^{\mathbb{D}}(Q)$ explicitly when $k < \mathsf{AGM}(Q)$. We will exploit this succinctness to design efficient algorithms in this thesis.

2.2.5 Normalizing tree decompositions

In order to simplify the future proofs we will require the tree decompositions to be so-called normalized decomposition trees:

Definition 2.6.
Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a decomposition tree. We call a node $u \in \mathcal{V}$ of T:

- **an extend node** *if it has a single child u' and $\mathcal{B}(u) = \mathcal{B}(u') \cup \{x\}$ for some $x \in \mathcal{X} \setminus \mathcal{B}(u')$,*

- **a project node** *if it has a single child u' and $\mathcal{B}(u) = \mathcal{B}(u') \setminus \{x\}$ for some $x \in \mathcal{X} \setminus \mathcal{B}(u)$,*

- **a join node** *if it has $k \geq 1$ children $u_1, ..., u_k$ with $\mathcal{B}(u) = \mathcal{B}(u_1) = ... = \mathcal{B}(u_k)$.*

We call T normalized[1] *if all its nodes in $\mathcal{V}$ are either extend nodes, project nodes, join nodes, or leaves.*

It is well-known that tree decompositions can always be normalized without changing their width. Thus this requirement will not negatively impact the asymptotic complexity of the algorithms.

Lemma 2.7 (Lemma 13.1.2 of [Klo94])**.**
For every tree decomposition of $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of Q of width k, there exists a normalized tree decomposition $T' = (\mathcal{V}', \mathcal{E}', \mathcal{B}')$ of width k. Moreover, one can compute T' from T in polynomial time.

2.3 Linear programming

Let Ξ be a set of linear program variables. In Figure 2.3, we recall the definition of the sets of linear expressions $L\textsc{e}$, linear constraints $L\textsc{c}$ and linear programs $L\textsc{p}$ with variables in Ξ.

We consider the usual linear equations $S \doteq S'$ as syntactic sugar for the constraints $S \leq S' \wedge S' \leq S$. Similarly $S \geq S'$ is syntactic sugar for the constraint $-S \leq -S'$. For any linear program $L =$ **minimize** S **subject to** C we call S the

[1] In the literature this property is referred to as "nice" tree decompositions.

$$\begin{array}{llcl}
\text{Linear expressions} & S, S' \in L\textsc{e}_{\Xi} & ::= & c \mid \xi \mid cS \mid S + S' \\
\text{Linear constraints} & C, C' \in L\textsc{c}_{\Xi} & ::= & S \leq S' \mid C \wedge C' \mid \mathit{true} \\
\text{Linear programs} & L \in L\textsc{p}_{\Xi} & ::= & \textbf{minimize } S \textbf{ subject to } C
\end{array}$$

Figure 2.3: The set of linear programs $L\textsc{p}$ with variables $\xi \in \Xi$ and constants $c \in \mathbb{R}$.

objective function of L and C the constraint of L. Note that the linear program **maximize** S **subject to** C can be expressed by **minimize** (-1) S **subject to** C.

The formal semantics of linear programs is recalled in Figure 2.4. Since we will only be interested in variables for positive real numbers – and do not want to impose positivity constraints all over – we will always implicitly consider the variables of linear programs to take values over $\mathbb{R}^+$.

For any weighting $\omega : \Xi \to \mathbb{R}^+$, the value of a sum $S \in L\textsc{e}$ is the real number $[\![S]\!]_\omega \in \mathbb{R}$, and the value of a constraint $C \in L\textsc{c}$ is the truth value $[\![C]\!]_\omega \in \mathbb{B}$. We denote the variables of a linear expression, constraint or program E by $\mathit{var}(E)$. We define the (feasible) solutions of a linear program L with objective function S and constraint C as $\mathit{sol}(L) = \{\omega : \mathit{var}(L) \to \mathbb{R}^+ \mid [\![C]\!]_\omega = 1\}$. Its optimal value $\mathit{opt}(L) \in \mathbb{R}$ is $\mathit{opt}(L) = \min(\{[\![S]\!]_\omega \mid \omega \in \mathit{sol}(L)\})$.

In practice the optimal solution of a linear program is computed via the exponential time Simplex method (Dantzig, 47) or a polynomial time interior point method [Kar84]. However the best theoretical complexity that we can get follows from combining [CLS21] and [AW21] which allows to prove the following theorem:

Theorem 2.8.
A linear program L can be solved in time $O(|L|_b \cdot n^\ell)$ where $|L|_b$ is the size in bits of the encoding of L and $\ell = 2.37286$.

More accurately, [CLS21] states that a linear program can be solved in time $O(|L|_b \cdot n^e)$ for $e > \max(2 + 1/6, \omega)$ where ω is the best known exponent of matrix multiplication complexity, which is known to be greater than 2.37286 by [AW21].

$$
\begin{aligned}
\llbracket c \rrbracket_\omega &= c \\
\llbracket \xi \rrbracket_\omega &= \omega(\xi) \\
\llbracket cS \rrbracket_\omega &= c \cdot \llbracket S \rrbracket_\omega \\
\llbracket S + S' \rrbracket_\omega &= \llbracket S \rrbracket_\omega + \llbracket S' \rrbracket_\omega \\
\\
\llbracket \mathit{true} \rrbracket_\omega &= 1 \\
\llbracket S \leq S' \rrbracket_\omega &= \begin{cases} 1 & \text{if } \llbracket S \rrbracket_\omega \leq \llbracket S' \rrbracket_\omega \\ 0 & \text{otherwise.} \end{cases} \\
\llbracket C \wedge C' \rrbracket_\omega &= \llbracket C \rrbracket_\omega \wedge \llbracket C' \rrbracket_\omega \\
\\
\llbracket \textbf{minimize } S \textbf{ subject to } C \rrbracket &= \min(\{\llbracket S \rrbracket_\omega \mid \omega : \Xi \to \mathbb{R}^+, \llbracket C \rrbracket_\omega \doteq 1\})
\end{aligned}
$$

Figure 2.4: Evaluation of linear expressions, constraints and programs.

Chapter 3

Linear programs on conjunctive queries

Contents

3.1 Introduction

In this chapter we introduce the so-called $LP(CQ)$ language that allows one to describe a linear program model that can then be automatically instatiated using a database. We first consider a task assignment problem to illustrate this notion. This problem will also serve as a running example throughout the chapter.

Consider a situation where an organization has to see some tasks to completion by assigning employees to work on them. Each task requires a specific skill and

Tasks	tid	skill	duration
	T_1	Python	45
	T_2	C	10
	T_3	SQL	20

Skills	eid	skill
	Alice	Python
	Alice	C
	Bob	SQL
	Bob	Python

Figure 3.1: Example for the *Tasks* and *Skills* tables.

Q_{assign}	t	s	e	d	var
	T_1	Python	Alice	45	θ_1
	T_1	Python	Bob	45	θ_2
	T_2	C	Alice	10	θ_3
	T_3	SQL	Bob	20	θ_4

Figure 3.2: Answers to the Q_{assign} query on the example database with the associated variables

takes a set amount of time to be completed. For example the task "Update the website" might take 20 hours of combined work from one or several HTML developers. Furthermore we need to account for the fact that employees cannot work over the legal limit.

We can store the information about the tasks and employees in two tables as follows: The first table $Skills(\underline{eid, skill})$ represents the skills possessed by the employees. The second table $Tasks(\underline{tid}, skill, duration)$ represents the skill and time required for each task. An example database is given in Figure 3.1.

Now in order to find out how to assign the employees to the tasks we can model this problem as a linear program. First we need to list every valid assignment by joining these tables as follows:

$$Q_{assign} = Tasks(t, s, d) \wedge Skills(e, s).$$

Then we create a variable for each assignment that will represent the duration of the assignment. The answers of Q_{assign} and associated variables are listed in Figure 3.2. For example the value of θ_3 will represent the time Alice will spend working on the task T_2.

We can now use these variables to express the constraints of the problem. First we want to ensure that the total time assigned to each task is sufficient. For example to make sure that task T_1 is worked on for at least 45 hours we can enforce the following constraint:

$$\theta_1 + \theta_2 \geq 45.$$

Then we also need to make sure no employee is overworked. For example to make sure that Alice doesn't work for more than 40 hours a week we can enforce the following constraint:

$$\theta_1 + \theta_3 \leq 40.$$

Finally we will minimize the total time spent by employees. The problem can thus be modelled as follows:

Example 3.1 (Task assignment linear program)**.**

$$\begin{aligned}
\textit{minimize} \quad & \theta_1 + \theta_2 + \theta_3 + \theta_4 \\
\textit{subject to} \quad & \theta_1 + \theta_3 \leq 40 \\
& \theta_2 + \theta_4 \leq 40 \\
& \theta_1 + \theta_2 \geq 45 \\
& \theta_3 \geq 10 \\
& \theta_4 \geq 20.
\end{aligned}$$

One possible answer to this linear program would be $\theta_1 = 25, \theta_2 = 20, \theta_3 = 10$ and $\theta_4 = 20$ which amounts to an objective value of 75. This solution means that Alice would work 25 hours on task T_1 and 10 hours on T_2 while Bob would work 20 hours on T_1 and 20 hours on T_3.

Suppose now that the database was updated with new tasks and employees. We would need to write a new linear program from scratch even though it would share the same structure as the underlying problem stays the same. To avoid repeating this process every time the input database changes, the natural approach is to write a linear program model that describes a linear program once it is combined with the input data. We thus introduce the $LP(CQ)$ language in this chapter to model linear programs with conjunctive queries.

Outline of the chapter

In Section 3.2 we give an overview of the features of the $LP(CQ)$ language. We then formally present its syntax in Section 3.3. In Section 3.4 we explain its semantics in details. In Section 3.5 we study the complexity of solving $LP(CQ)$ models. Finally in Section 3.6 we introduce a notion of alternate interpretations of $LP(CQ)$ models that we will use in the following chapters.

3.2 Overview of the language

We want to model linear programs that assign weights to the answers of conjunctive queries on a database. For this, we introduce the language $LP(CQ)$ of linear programs with conjunctive queries. This language is based on the standard linear program syntax and has some additional constructs to access data from a database. Namely it allows one to:

- assign weights to select subsets of query answers,
- universally quantify constraints,

- sum linear expressions,
- retrieve constants from the database.

We will give some intuition about these features of our language using our task assignment example.

3.2.1 Weighting subsets of answers

In our task assignment example the first constraint in Example 3.1 ensured that Alice worked at most 40 hours in total. It was written as $\theta_1 + \theta_3 \leq 40$ where θ_1 and θ_3 correspond to both answers of Q_{assign} such that $e \doteq Alice$.

In order to describe the weight of subset of answers without having to explicitly answer the query we introduce the **weight** construct. The previous constraint can then be rewritten as follows:

$$\mathbf{weight}_{(t,s,e,d):e \doteq Alice}(Q_{assign}) \leq 40.$$

Here the weight expression means that among all the answers of Q_{assign} on variables (t, s, e, d) we select only the tuples that satisfy $e \doteq Alice$. The constraint then means that the total weight of these tuples should be lesser than or equal to 40 and it can interpreted back to our initial constraint $\theta_1 + \theta_3 \leq 40$.

Similarly the second constraint can be expressed as

$$\mathbf{weight}_{(t,s,e,d):e \doteq Bob}(Q_{assign}) \leq 40$$

which means that the assignments of Bob should amount to at most 40 hours.

Observe that these three constraints are very similar, only differing on the value that the variable e should be equal to.

3.2.2 Quantifying contraints

Observe that when we defined the two constraints mentioned earlier we defined one constraint for each employee. We can obtain these values from the database with the query $\exists s'.Skills(e', s')$ which gives the answer

$\exists s'.Skills(e', s')$	e'
	Alice
	Bob

In order to describe such groups of constraints we introduce a $\forall$ construct that will iterate over the answers of the query to produce a constraint for each answer.

The two constraints mentioned earlier can be expressed as follows:

$$\forall (e'){:}\exists s'.Skills(e', s').\mathbf{weight}_{(t,s,e,d):e \doteq e'}(Q_{assign}) \leq 40.$$

This means that for each value e' taken by eid in $Skills$ the total weight of the answers of Q_{assign} such that $e \doteq e'$ should be at most 40. This can then be partially interpreted as

$$\begin{aligned} & \mathbf{weight}_{(t,s,e,d):e \doteq Alice}(Q_{assign}) \leq 40 \\ \wedge \quad & \mathbf{weight}_{(t,s,e,d):e \doteq Bob}(Q_{assign}) \leq 40 \end{aligned}$$

and then further interpreted as

$$\begin{aligned} & \theta_1 + \theta_3 \leq 40 \\ \wedge \quad & \theta_2 + \theta_4 \leq 40. \end{aligned}$$

Now we will look at the other three constraints that ensured enough time was assigned to each task:

$$\begin{aligned} & \theta_1 + \theta_2 \geq 45 \\ \wedge \quad & \theta_3 \geq 10 \\ \wedge \quad & \theta_4 \geq 20. \end{aligned}$$

Observe that we have one such constraint for each task while accounting for the duration of the task in the right-hand side. We can obtain the tuples we are interested in with the query $\exists s'.Tasks(t', s', d')$ which gives the following answers:

$\exists s'.Tasks(t', s', d')$	tid	duration
	T_1	45
	T_2	10
	T_3	20

We can then use this query and a ∀ construct to iterate over each task. Finally we are interested in bounding the total time assigned to the task which we can represent using a **weight** construct. We obtain a constraint that would look like this:

$$\forall (t', d'){:}\exists s'.Tasks(t', s', d').\mathbf{weight}_{(t,s,e,d):t \doteq t'}(Q_{assign}) \geq \textbf{d'}.$$

However the right-hand side constant depends on the value of the database variable d' but we cannot use it directly so we introduce a new operator to retrieve its value.

3.2.3 Retrieving values from the database

In order to retrieve a value from the database and use it in the linear program we simply introduce the **num** keyword. This operator takes a database variable (that was bound by a ∀ for instance) and converts it to a numerical value.

We can thus express the last three constraints of the task assignment problem as follows:

$$\forall(t',d'){:}\exists s'.Tasks(t',s',d').\mathbf{weight}_{(t,s,e,d):t\doteq t'}(Q_{assign}) \geq \mathbf{num}(d').$$

This means that we iterate over each pair of values (t',d') taken by *tid* and *duration* in $Tasks$. We then ensure that for each such pair that the total weight of the answers of Q_{assign} such that $t\doteq t'$ should be at least the numerical value of d'. Using our database, this constraint will thus be interpreted as

$$\begin{array}{rl} & \theta_1 + \theta_2 \geq 45 \\ \wedge & \theta_3 \geq 10 \\ \wedge & \theta_4 \geq 20. \end{array}$$

3.2.4 Summing linear expressions

We will now look at the objective function of our example. For the sake of demonstration we will consider that the objective function should be written as the sum of the workloads of the employees[1].

To do so we introduce the $\sum$ operators which we can use to write the objective function as follows:

$$\sum_{(e'):\exists s'.Skills(e',s')} \mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign}).$$

This expression represents the sum over all the values e' taken by *eid* of the total weight of the answers of Q_{assign} such that $e\doteq e'$. It will be interpreted as $\theta_1 + \theta_2 + \theta_3 + \theta_4$ using our database like we explicitly defined in the introduction.

3.2.5 Full linear program model

Finally we get with the following linear program model:

Example 3.2 (Task assignment $LP(CQ)$ model)**.**

$$\begin{array}{rl} \textbf{\textit{minimize}} & \displaystyle\sum_{(e'):\exists s'.Skills(e',s')} \mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign}) \\ \textbf{\textit{subject to}} & \forall(t',d'){:}\exists s'.Tasks(t',s',d').\mathbf{weight}_{(t,s,e,d):t\doteq t'}(Q_{assign}) \geq \mathbf{num}(d') \\ \wedge & \forall(e'){:}\exists s'.Skills(e',s').\mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign}) \leq 40. \end{array}$$

In Section 3.4 we will define the semantics of $LP(CQ)$ models. Interpreting the semantics of this model using our initial database will yield our original linear program of Example 3.1.

[1] Even though we may notice that the objective function could also be written as $\mathbf{weight}_{(t,s,e,d):true}(Q_{assign})$ (i.e., the total weight of all the answers of Q_{assign}).

3.3 Full syntax of $LP(CQ)$

Constant numbers	$N \in \mathit{Num}(CQ)$	$::=$	$c \mid \mathbf{num(E)}$
$LP(CQ)$ expressions	$S, S' \in \textsc{Le}(CQ)$	$::=$	$N \times S \mid S + S' \mid N$
			$\mid \mathbf{weight}_{\mathbf{x}:\mathbf{Q}'}(\mathbf{Q}) \mid \sum_{\mathbf{x}:\mathbf{Q}} \mathbf{S}$
$LP(CQ)$ constraints	$C, C' \in \textsc{Lc}(CQ)$	$::=$	$S \leq S' \mid C \wedge C' \mid \mathit{true} \mid \forall \mathbf{x}{:}\mathbf{Q}.\mathbf{C}$
$LP(CQ)$ models	$L \in LP(CQ)$	$::=$	$\mathbf{minimize}\ S\ \mathbf{subject\ to}\ C$

Figure 3.3: The syntax of $LP(CQ)$ models where $c \in \mathbb{R}$, $E \in \mathit{Ex}_{\mathcal{C}}$, $\mathbf{x} \in \mathcal{X}^*$ and $Q, Q' \in CQ_{\Sigma}$.

We now formally define the syntax of the $LP(CQ)$ language by adding the additional features we saw previously to the standard definition of linear programs. Let $c \in \mathbb{R}$ be a constant, $E \in \mathit{Ex}_{\mathcal{C}}$ be an expression as defined in Figure 2.1, $\mathbf{x} \in \mathcal{X}^*$ be a tuple of variables and $Q, Q' \in CQ_{\Sigma}$ be conjunctive queries. We define constant numbers $N \in \mathit{Num}(CQ)$, $LP(CQ)$ expressions $S, S' \in \textsc{Le}(CQ)$, $LP(CQ)$ constraints $C, C' \in \textsc{Lc}(CQ)$ and $LP(CQ)$ models $L \in LP(CQ)$ in Figure 3.3. The specific features of the $LP(CQ)$ language are highlighted in red.

As it was hinted at in the previous section, **weight** operators may contain free variables that can then be bound in the context by $\forall$ or $\sum$ operators. For instance in the objective function of Example 3.2 we had

$$\sum_{(\mathbf{e}'):\exists s'.\mathit{Skills}(e',s')} \mathbf{weight}_{(\mathbf{t},\mathbf{s},\mathbf{e},\mathbf{d}):e\doteq\mathbf{e}'}(\mathit{Tasks}(\mathbf{t},\mathbf{s},\mathbf{d}) \wedge \mathit{Skills}(\mathbf{e},\mathbf{s}))$$

where the free variable e' of the **weight** is bound to the variable e' quantified by the $\forall$. However the variable t, s, e and d are not free in the **weight** operator because they are guarded by the tuple (t, s, e, d). In general we define the free variables of a **weight** operator $\mathit{fv}(\mathbf{weight}_{\mathbf{x}:Q'}(Q)) = \mathit{fv}(Q) \cup \mathit{fv}(Q') \setminus \mathit{set}(\mathbf{x})$.

Similarly $\forall$ and $\sum$ operators may also have free variables. For instance consider the constraint

$$\forall(\mathbf{z}){:}r(z).\ \sum_{(x,y):r'(x,y,\mathbf{z})} S \leq C$$

where z is a free variable of the $\sum$ operator that is bound to the z variable quantified by the $\forall$ operator. Similarly to the **weight** operators, the free variables of $\sum$ and $\forall$ operators are defined as $\mathit{fv}(\sum_{\mathbf{x}:Q} S) = \mathit{fv}(S) \cup \mathit{fv}(Q) \setminus \mathit{set}(\mathbf{x})$ and $\mathit{fv}(\forall \mathbf{x}{:}Q.C) = \mathit{fv}(Q) \cup \mathit{fv}(C) \setminus \mathit{set}(\mathbf{x})$.

The free variables of the other elements of the $LP(CQ)$ language are straightforward and are given in the full definition in Figure 3.4.

Overall we say that a $LP(CQ)$ model L is interpretable and thus well-formed iff $\mathit{fv}(L) = \emptyset$. In this thesis we will only consider well-formed $LP(CQ)$ models.

$$\begin{array}{ll}
\mathit{fv}(c) = \emptyset & \mathit{fv}(\mathbf{num}(E)) = \mathit{fv}(E) \\
\mathit{fv}(\mathbf{weight}_{\mathbf{x}:Q'}(Q)) = (\mathit{fv}(Q) \cup \mathit{fv}(Q')) \setminus \mathit{set}(\mathbf{x}) & \mathit{fv}(\sum_{\mathbf{x}:Q} S) = (\mathit{fv}(S) \cup \mathit{fv}(Q)) \setminus \mathit{set}(\mathbf{x}) \\
\mathit{fv}(N \times S) = \mathit{fv}(N) \cup \mathit{fv}(S) & \mathit{fv}(S \leq S') = \mathit{fv}(S) \cup \mathit{fv}(S') \\
\mathit{fv}(S + S') = \mathit{fv}(S) \cup \mathit{fv}(S') & \mathit{fv}(C \wedge C') = \mathit{fv}(C) \cup \mathit{fv}(C') \\
\mathit{fv}(\forall \mathbf{x}{:}Q.C) = (\mathit{fv}(C) \cup \mathit{fv}(Q)) \setminus \mathit{set}(\mathbf{x}) & \mathit{fv}(\mathit{true}) = \emptyset \\
\mathit{fv}(\mathbf{maximize}\ S\ \mathbf{subject\ to}\ C) = \mathit{fv}(S) \cup \mathit{fv}(C) &
\end{array}$$

Figure 3.4: Free variables of $LP(CQ)$ expressions, constraints, and programs.

3.4 Semantics

We now define the semantics of an $LP(CQ)$ model $L \in LP(CQ)$ with respect to a database $\mathbb{D} \in db_\Sigma$ with real numbers by an interpretation to a linear program $\langle L \rangle^{\mathbb{D}} \in L\textsc{p}$. We first look at a smaller subclass of $LP(CQ)$ models called *closed $LP(CQ)$ models* and defined as follows:

Definition 3.3.
An $LP(CQ)$ model L, constraint $C \in L\textsc{c}(CQ)$ or expression $S \in L\textsc{e}(CQ)$ is closed *iff*

- *no $\forall$ or $\sum$ quantifiers appear in it,*
- *no* **num** *operators appear in it,*
- *every* $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ *operator that appears in it is closed i.e., $\mathit{fv}(Q) \cup \mathit{fv}(Q') \subseteq \mathit{set}(\mathbf{x})$.*

Oberve that closed $LP(CQ)$ models are actually quite similar to linear programs as all their constraints are inequalities of linear combinations of closed **weight** operators. Thus to interpret such models we only need to give a semantic to the **weight** operators which we will do by instantiating them into linear sums of linear program variables.

3.4.1 Instantiating closed weight operators

In this section we will consider a simple closed $LP(CQ)$ model to demonstrate the instantiation of **weight** operators. Let $Q = R(x) \wedge R(y)$ be a conjunctive query and $\mathbb{D}$ be a database with the following table R:

$R^{\mathbb{D}}$	x
	0
	1

We then consider the following closed $LP(CQ)$ program L:

$$\begin{array}{rl}\textbf{maximize} & \textbf{weight}_{(x,y):\mathit{true}}(Q)\\ \textbf{subject to} & \textbf{weight}_{(x,y):x\doteq 0}(Q) \leq 1\\ \wedge & \textbf{weight}_{(x,y):x\doteq 1}(Q) \leq 1.\end{array}$$

As we saw earlier when we use an expression $\textbf{weight}_{\mathbf{x}:Q'}(Q)$ our intent is to describe the weight of the answers of Q on $\mathit{set}(\mathbf{x})$ that also satisfy Q'. Thus we need to introduce variables to refer to the weight of these answers. However rather than arbitrarily numbering our variables as we did in Section 3.2 we will instead index our variables with the query and an answer of the query.

Here the set of answers of Q and their associated variables is as follows

Q	x	y	var
	0	0	$\theta_Q^{[x/0,y/0]}$
	0	1	$\theta_Q^{[x/0,y/1]}$
	1	0	$\theta_Q^{[x/1,y/0]}$
	1	1	$\theta_Q^{[x/1,y/1]}$

The expression $\textbf{weight}_{(x,y):\mathit{true}}(Q)$ in the objective function of the model represents the total weight of the answers of Q on variables (x, y) that also satisfy the query *true*, that is every answer of Q. Thus this expression is instantiated as follows:

$$\theta_Q^{[x/0,y/0]} + \theta_Q^{[x/0,y/1]} + \theta_Q^{[x/1,y/0]} + \theta_Q^{[x/1,y/1]}.$$

The expression $\textbf{weight}_{(x,y):x\doteq 0}(Q)$ in the first constraint represents the total weight of the answers of Q on variables (x, y) that also satisfy the query $x = 0$, that is $[x/0, y/0]$ and $[x/0, y/1]$. Thus the constraint is instantiated as follows:

$$\theta_Q^{[x/0,y/0]} + \theta_Q^{[x/0,y/1]} \leq 1.$$

Observe that these sums share the linear program variables $\theta_Q^{[x/0,y/0]}$ and $\theta_Q^{[x/0,y/1]}$, so these two weight expressions of L are semantically related.

Similarly the second constraint can instantiated as follows:

$$\theta_Q^{[x/1,y/0]} + \theta_Q^{[x/1,y/1]} \leq 1.$$

Finally the instantiation of L is

$$\begin{array}{rl}\textbf{maximize} & \theta_Q^{[x/0,y/0]} + \theta_Q^{[x/0,y/1]} + \theta_Q^{[x/1,y/0]} + \theta_Q^{[x/1,y/1]}\\ \textbf{subject to} & \theta_Q^{[x/0,y/0]} + \theta_Q^{[x/0,y/1]} \leq 1\\ \wedge & \theta_Q^{[x/1,y/0]} + \theta_Q^{[x/1,y/1]} \leq 1.\end{array}$$

Formally, given a conjunctive query Q, the set of variables it can describe is

$$\Theta_Q^{\mathbb{D}} = \{\theta_Q^{\alpha} \mid \alpha \in \mathit{ans}_{\mathbf{x}}^{\mathbb{D}}(Q)\}.$$

We can then define the instantiation of closed **weight** operators with $\mathbb{D}$ as the function $\mathrm{Inst}_{\mathbb{D}}$ that maps any **weight** operator to the total weight of the variables it represents.

$$\mathrm{Inst}_{\mathbb{D}}(\mathbf{weight}_{\mathbf{x}:Q'}(Q)) = \sum_{\tau \in ans^{\mathbb{D}}_{\mathbf{x}}(Q \wedge Q')} \theta^{\tau}_{Q}.$$

Observe that Q defines the set $\Theta^{\mathbb{D}}_{Q}$ from which this instantiation pulls its variables while Q' selects a subset of $\Theta^{\mathbb{D}}_{Q}$. Thus $\mathrm{Inst}_{\mathbb{D}}(\mathbf{weight}_{\mathbf{x}:Q'}(Q))$ and $\mathrm{Inst}_{\mathbb{D}}(\mathbf{weight}_{\mathbf{x}:Q''}(Q))$ can share variables independently of whether Q' and Q'' are equal, as was evidenced in the above example.

Let $cq(L)$ be the set of base queries that appear in L i.e., the set of queries Q such that there is a $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ in L. Note that we consider queries to be equal if and only if they are syntactically identical so $cq(L)$ could contain two queries $Q = \mathit{true}$ and $Q' = \mathit{true} \wedge \mathit{true}$ for instance. In general the set of variables of a $LP(CQ)$ model L are defined as follows:

$$\Theta^{\mathbb{D}}_{L} = \biguplus_{Q \in cq(L)} \Theta^{\mathbb{D}}_{Q}.$$

Finally we lift the notion of instantiation to closed $LP(CQ)$ models.

Definition 3.4.
Given a closed $LP(CQ)$ model L and database $\mathbb{D}$, we denote the instantiation of L with $\mathbb{D}$ by $Inst_{\mathbb{D}}(L)$.

It is the linear program obtained by replacing every $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ in L with $Inst_{\mathbb{D}}(\mathbf{weight}_{\mathbf{x}:Q'}(Q))$ defined as $\sum_{\tau \in ans^{\mathbb{D}}_{\mathbf{x}}(Q \wedge Q')} \theta^{\tau}_{Q}$.

Observe that $var(\mathrm{Inst}_{\mathbb{D}}(L)) \subseteq \Theta^{\mathbb{D}}_{L}$.

3.4.2 Closing the model

In order to provide the semantics of $LP(CQ)$ models we define a semantics of the quantifiers of the language that allows us to close any[2] $LP(CQ)$ model.

To get some intuition about this closure, let us isolate the following constraint from Example 3.2:

$$\forall(e'){:}\exists s'.Skills(e', s').\mathbf{weight}_{(t,s,e,d):e \doteq e'}(Q_{assign}) \leq 40.$$

The central part of this transformation is the *substitution* of the free variable e' of the **weight** expression with a value provided by the $\forall$ quantifier it is bound to. Thus, as a prerequisite to this substitution, we need to iterate over the answers of $\exists s'.Skills(e', s')$ and unfold the quantifier into a conjunction of subconstraints while passing the corresponding answer to each subconstraint. In this case, the query has two answers $[e'/Alice]$ and $[e'/Bob]$ so we will generate two subconstraints while propagating these two answers as follows:

[2] Recall that we consider only well-formed $LP(CQ)$ models with no free variables.

$$\begin{array}{rll} & \mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign}) \leq 40 & \text{(with } [e'/\mathit{Alice}]\text{)} \\ \wedge & \mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign}) \leq 40 & \text{(with } [e'/\mathit{Bob}]\text{)}. \end{array}$$

In general multiple quantifiers might be nested so we pass the variable assignments as an argument of the closure, which we will call an *environment*, that can be enriched as we traverse quantifiers. Given a database $\mathbb{D}$ and an environment $\gamma : Y \to \mathit{dom}(\mathbb{D})$ with $\mathit{fv}(C) \subseteq Y$, we denote the closure of a constraint $C \in L\textsc{c}(CQ)$ by $\mathit{close}^{\mathbb{D},\gamma}(C)$. Formally we thus define the closure of this constraint (with an empty initial environment as it is a top-level constraint in the model) as follows:

$$\begin{array}{rl} & \mathit{close}^{\mathbb{D},\emptyset}(\forall(e'){:}\exists s'.\mathit{Skills}(e',s').\mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign}) \leq 40) \\ = & \mathit{close}^{\mathbb{D},[e'/\mathit{Alice}]}(\mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign}) \leq 40) \\ \wedge & \mathit{close}^{\mathbb{D},[e'/\mathit{Bob}]}(\mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign}) \leq 40). \end{array}$$

Then the next step of unfolding the inequality is straightforward:

$$\begin{array}{rl} & \mathit{close}^{\mathbb{D},[e'/\mathit{Alice}]}(\mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign})) \leq \mathit{close}^{\mathbb{D},[e'/\mathit{Alice}]}(40) \\ \wedge & \mathit{close}^{\mathbb{D},[e'/\mathit{Bob}]}(\mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign})) \leq \mathit{close}^{\mathbb{D},[e'/\mathit{Bob}]}(40). \end{array}$$

Finally we handle the **weight** operators by substituting their free variable e' with the value provided by the environment while the right-hand side constants are already closed:

$$\begin{array}{rl} & \mathbf{weight}_{(t,s,e,d):e\doteq \mathit{Alice}}(Q_{assign}) \leq 40 \\ \wedge & \mathbf{weight}_{(t,s,e,d):e\doteq \mathit{Bob}}(Q_{assign}) \leq 40. \end{array}$$

In general we define for any conjunctive query Q and variable assignment $\gamma : Y \to \mathit{dom}$ a conjunctive query $\mathit{subs}_\gamma(Q)$ by replacing all free occurrences of variables $y \in Y$ in Q by $\gamma(y)$. The formal definition is given in Figure 3.5.

As we saw earlier the closure of a constraint $\forall\mathbf{x}{:}Q.C$ given a prior environment $\gamma : Y \to \mathit{dom}$ is defined as a conjunction over the answers of Q of the closures of C with updated environments.

To build this new environment we first consider a restriction of γ that does not assign values to variables in $\mathbf{x}$, $\tilde{\gamma} = \gamma_{|Y\setminus \mathit{set}(\mathbf{x})}$, in order to avoid ambiguities when multiple quantifiers refer to the same variable(s)[3]. The second part of our updated

[3] See Section 3.4.4 for more details.

environments is the answer of Q being iterated upon by the quantifier. However Q might have free variables that are not covered by $\mathbf{x}$ and are thus bound to the context therefore we substitute $\tilde{\gamma}$ in Q before enumerating its answers. Finally our updated environments are $\tilde{\gamma} \cup \gamma'$ with $\gamma' \in ans_{\mathbf{x}}^{\mathbb{D}}(subs_{\tilde{\gamma}}(Q))$. Thus the closure of this constraint in environment γ can be formally defined as follows:

$$close^{\mathbb{D},\gamma}(\forall \mathbf{x}{:}Q.C) = \bigwedge_{\gamma' \in ans^{\mathbb{D}}_{set(\mathbf{x})}(subs_{\tilde{\gamma}}(Q))} close^{\mathbb{D},\tilde{\gamma}\cup\gamma'}(C).$$

The closure of an expression $\sum_{\mathbf{x}:Q} S$ is very similar to the closure of a $\forall$ constraint with the only difference being that it generates a sum of closures of the subexpression S. It is defined formally as follows:

$$close^{\mathbb{D},\gamma}(\sum_{\mathbf{x}:Q} S) = \sum_{\gamma' \in ans^{\mathbb{D}}_{set(\mathbf{x})}(subs_{\tilde{\gamma}}(Q))} close^{\mathbb{D},\tilde{\gamma}\cup\gamma'}(S).$$

To define the closure of an expression $\mathbf{num}(E)$ we first need to substitute γ in E to change it to a database constant $a \in dom$ if it was a query variable $x \in \mathcal{X}$. We can then apply the $\mathbf{num}^{\mathbb{D}}$ operator to retrieve the value $\mathbf{num}^{\mathbb{D}}(a) \in \mathbb{R}$ of the database constant a. It is defined formally as follows:

$$close^{\mathbb{D},\gamma}(\mathbf{num}(E)) = \mathbf{num}^{\mathbb{D}}(subs_{\gamma}(E)).$$

The rest of the inductive definition of the closure of $LP(CQ)$ models is straightforward and is given in Figure 3.6. As an example the closure of the $LP(CQ)$ model from Example 3.2 is given in Example 3.5.

Example 3.5 (Closure of the task assignment $LP(CQ)$ model on $\mathbb{D}$)**.**

$$\begin{aligned}
\textbf{\textit{minimize}}\ & \mathbf{weight}_{(t,s,e,d):e\doteq Alice}(Q_{assign}) \\
& + \mathbf{weight}_{(t,s,e,d):e\doteq Bob}(Q_{assign}) \\
\textbf{\textit{subject to}}\ & \mathbf{weight}_{(t,s,e,d):t\doteq T_1}(Q_{assign}) \geq 45 \\
& \wedge\ \mathbf{weight}_{(t,s,e,d):t\doteq T_2}(Q_{assign}) \geq 10 \\
& \wedge\ \mathbf{weight}_{(t,s,e,d):t\doteq T_3}(Q_{assign}) \geq 20 \\
& \wedge\ \mathbf{weight}_{(t,s,e,d):e\doteq Alice}(Q_{assign}) \leq 40 \\
& \wedge\ \mathbf{weight}_{(t,s,e,d):e\doteq Bob}(Q_{assign}) \leq 40
\end{aligned}$$

We now show that our closure operation is indeed a closure by showing that the closure of any $LP(CQ)$ model is indeed closed in Proposition 3.8 and by showing that it is the identity when applied to a closed model in Proposition 3.11.

We begin by showing that the closure of a well-formed $LP(CQ)$ produces a closed $LP(CQ)$ model as defined in Definition 3.3, that is a $LP(CQ)$ model that contains

$$\begin{array}{rcl}
subs_\gamma(Q \wedge Q') & = & subs_\gamma(Q) \wedge subs_\gamma(Q') \\
subs_\gamma(\exists x.Q) & = & \exists x.subs_{\gamma_{|dom(\gamma)\setminus\{x\}}}(Q) \\
subs_\gamma(r(t_1,\ldots,t_n)) & = & r(subs_\gamma(t_1),\ldots,subs_\gamma(t_n)) \\
subs_\gamma(t = t') & = & subs_\gamma(t) \dot{=} subs_\gamma(t') \\
subs_\gamma(x) & = & \begin{cases} \gamma(x) & \text{if } x \in dom(\gamma) \\ x & \text{otherwise} \end{cases} \\
subs_\gamma(a) & = & a
\end{array}$$

Figure 3.5: Lifting substitutions $\gamma : fv(Q) \to \mathcal{C}$ to queries Q.

no $\forall$, $\sum$ or **num** operators and whose every **weight** expression is closed. Observe that by definition the closure removes any $\forall$, $\sum$ or **num** operators it encounters so the proof will mostly focus on making sure the model is traversed properly as well as on checking the free variables of **weight** expressions. For clarity we first prove this property on $LP(CQ)$ expressions and $LP(CQ)$ constraints in the following lemmas.

Lemma 3.6.
Let $S \in \textsc{Le}(CQ)$ be an $LP(CQ)$ expression and $\mathbb{D}$ be a database. Given an environment γ such that $fv(S) \subseteq var(\gamma)$, $close^{\mathbb{D},\gamma}(S)$ is closed.

Proof. We show this by bottom-up induction on the structure of S. Observe that the case $S = N$ can be split into two cases $S = c$ and $S = \mathbf{num}(E)$.

Base case 1: $S = c \in \mathbb{R}$

By definition $close^{\mathbb{D},\gamma}(S) = c$ so it is closed.

Base case 2: $S = \mathbf{num}(E)$

By definition $close^{\mathbb{D},\gamma}(S) = \mathbf{num}^{\mathbb{D}}(subs_\gamma(E))$ which returns a real $c \in \mathbb{R}$ so $close^{\mathbb{D},\gamma}(S)$ is closed.

Base case 3: $S = \mathbf{weight}_{\mathbf{x}:Q'}(Q)$

By definition $close^{\mathbb{D},\gamma}(\mathbf{weight}_{\mathbf{x}:Q'}(Q)) = \mathbf{weight}_{\mathbf{x}:subs_{\tilde{\gamma}}(Q')}(subs_{\tilde{\gamma}}(Q))$.

Recall that $\gamma : Y \to dom(\mathbb{D})$ such that $fv(S) \subseteq Y$ so $fv(Q) \setminus set(\mathbf{x}) \subseteq Y$. Observe that when we consider $\tilde{\gamma} = \gamma_{|Y\setminus set(\mathbf{x})}$ it still covers every variable in $fv(Q) \setminus set(\mathbf{x})$. Thus when we compute $subs_{\tilde{\gamma}}(Q)$ we replace every variable in $fv(Q) \setminus set(\mathbf{x})$ with a database constant so $fv(subs_{\tilde{\gamma}}(Q)) \setminus set(\mathbf{x}) = \emptyset$. Similarly $fv(subs_{\tilde{\gamma}}(Q')) \setminus set(\mathbf{x}) = \emptyset$ so $fv(\mathbf{weight}_{\mathbf{x}:subs_{\tilde{\gamma}}(Q')}(subs_{\tilde{\gamma}}(Q))) = \emptyset$ thus $\mathbf{weight}_{\mathbf{x}:subs_{\tilde{\gamma}}(Q')}(subs_{\tilde{\gamma}}(Q))$ is closed.

Inductive case 1: $S = \sum_{\mathbf{x}:Q} S'$.

By definition $close^{\mathbb{D},\gamma}(\sum_{\mathbf{x}:Q} S') = \sum_{\gamma' \in ans^{\mathbb{D}}_{set(\mathbf{x})}(subs_{\tilde{\gamma}}(Q))} close^{\mathbb{D},\tilde{\gamma}\cup\gamma'}(S')$

Recall that by definition $fv(S) = fv(S') \cup fv(Q) \setminus set(\mathbf{x})$ and by hypothesis $fv(S) \subseteq var(\gamma)$ so $fv(S') \setminus set(\mathbf{x}) \subseteq var(\gamma)$. Observe that $var(\tilde{\gamma}) = var(\gamma) \setminus$

$$\begin{aligned}
&\mathit{close}^{\mathbb{D},\gamma}(\mathbf{num}(E)) = \begin{cases} \mathbf{num}^{\mathbb{D}}(\mathit{subs}_{\gamma}(E)) & \text{if defined,} \\ 0 & \text{otherwise} \end{cases} \\
&\mathit{close}^{\mathbb{D},\gamma}(c) = c \\[1em]
&\mathit{close}^{\mathbb{D},\gamma}(\mathbf{weight}_{\mathbf{x}:Q'}(Q)) = \mathbf{weight}_{\mathbf{x}:\mathit{subs}_{\tilde{\gamma}}(Q')}(\mathit{subs}_{\tilde{\gamma}}(Q)) \\
&\mathit{close}^{\mathbb{D},\gamma}(S_1 + S_2) = \mathit{close}^{\mathbb{D},\gamma}(S_1) + \mathit{close}^{\mathbb{D},\gamma}(S_2) \\
&\mathit{close}^{\mathbb{D},\gamma}(N \times S) = \mathit{close}^{\mathbb{D},\gamma}(N)\mathit{close}^{\mathbb{D},\gamma}(S) \\
&\mathit{close}^{\mathbb{D},\gamma}(N) = N \\
&\mathit{close}^{\mathbb{D},\gamma}(\mathit{true}) = \mathit{true} \\[1em]
&\mathit{close}^{\mathbb{D},\gamma}(\textstyle\sum_{\mathbf{x}:Q} S) = \sum_{\gamma' \in \mathit{ans}^{\mathbb{D}}_{\mathit{set}(\mathbf{x})}(\mathit{subs}_{\tilde{\gamma}}(Q))} \mathit{close}^{\mathbb{D},\tilde{\gamma}\cup\gamma'}(S) \\
&\mathit{close}^{\mathbb{D},\gamma}(\forall \mathbf{x}{:}Q.C) = \textstyle\bigwedge_{\gamma' \in \mathit{ans}^{\mathbb{D}}_{\mathit{set}(\mathbf{x})}(\mathit{subs}_{\tilde{\gamma}}(Q))} \mathit{close}^{\mathbb{D},\tilde{\gamma}\cup\gamma'}(C) \\
&\mathit{close}^{\mathbb{D},\gamma}(S_1 \leq S_2) = \mathit{close}^{\mathbb{D},\gamma}(S_1) \leq \mathit{close}^{\mathbb{D},\gamma}(S_2) \\
&\mathit{close}^{\mathbb{D},\gamma}(C_1 \wedge C_2) = \mathit{close}^{\mathbb{D},\gamma}(C_1) \wedge \mathit{close}^{\mathbb{D},\gamma}(C_2) \\[1em]
&\mathit{close}^{\mathbb{D}}(\textbf{maximize } S \textbf{ subject to } C) = \textbf{maximize } \mathit{close}^{\mathbb{D},\emptyset}(S) \textbf{ subject to } \mathit{close}^{\mathbb{D},\emptyset}(C)
\end{aligned}$$

Figure 3.6: Closure of linear expressions (constraints and programs) with conjunctive queries F over database $\mathbb{D}$ as closed linear expression (constraints and programs respectively) $\mathit{close}^{\mathbb{D},\gamma}(F)$, where $\gamma : Y \to \mathit{dom}(\mathbb{D})$ and $\mathit{fv}(F) \subseteq Y \subseteq \mathcal{X}$ and $\tilde{\gamma} = \gamma_{|Y \setminus \mathit{set}(\mathbf{x})}$.

$set(\mathbf{x})$ so $fv(S') \setminus set(\mathbf{x}) \subseteq var(\tilde{\gamma})$. Finally $var(\gamma') = set(\mathbf{x})$ so $fv(S') \subseteq var(\tilde{\gamma}) \cup var(\gamma')$ so by induction $close^{\mathbb{D},\tilde{\gamma}\cup\gamma'}(S')$ is closed.

Thus $close^{\mathbb{D},\gamma}(S)$ is closed as it is a sum of closed expressions.

Inductive case 2: $S = N \times S'$

By definition $close^{\mathbb{D},\gamma}(N \times S) = close^{\mathbb{D},\gamma}(N) close^{\mathbb{D},\gamma}(S)$.

Recall that by definition $fv(S) = fv(N) \cup fv(S')$ and by hypothesis $fv(S) \subseteq var(\gamma)$ so $fv(N) \subseteq var(\gamma)$ and $fv(S') \subseteq var(\gamma)$.

Thus by induction $close^{\mathbb{D},\gamma}(N)$ and $close^{\mathbb{D},\gamma}(S')$ are closed so $close^{\mathbb{D},\gamma}(S)$ is closed.

Inductive case 3: $S = S' + S''$

By definition $close^{\mathbb{D},\gamma}(S' + S'') = close^{\mathbb{D},\gamma}(S') + close^{\mathbb{D},\gamma}(S'')$.

Recall that by definition $fv(S) = fv(S') \cup fv(S'')$ and by hypothesis $fv(S) \subseteq var(\gamma)$ so $fv(S') \subseteq var(\gamma)$ and $fv(S'') \subseteq var(\gamma)$.

Thus by induction $close^{\mathbb{D},\gamma}(S')$ and $close^{\mathbb{D},\gamma}(S'')$ are closed so $close^{\mathbb{D},\gamma}(S)$ is closed.

□

Lemma 3.7.
Let $C \in L\textsc{c}(CQ)$ be an $LP(CQ)$ constraint and $\mathbb{D}$ be a database. Given an environment γ such that $fv(C) \subseteq var(\gamma)$, $close^{\mathbb{D},\gamma}(S)$ is closed.

Proof. We show this by bottom-up induction on the structure of S.

Base case 1: $C = true$

By definition $close^{\mathbb{D},\gamma}(C) = true$ which is closed.

Base case 2: $C = S \leq S'$

By definition $close^{\mathbb{D},\gamma}(S \leq S') = close^{\mathbb{D},\gamma}(S) \leq close^{\mathbb{D},\gamma}(S')$.

Recall that by definition $fv(C) = fv(S) \cup fv(S')$ and by hypothesis $fv(S) \subseteq var(\gamma)$ so $fv(S) \subseteq var(\gamma)$ and $fv(S') \subseteq var(\gamma)$.

Thus by Lemma 3.6 $close^{\mathbb{D},\gamma}(S)$ and $close^{\mathbb{D},\gamma}(S')$ are closed so $close^{\mathbb{D},\gamma}(C)$ is closed.

Inductive case 1: $\forall \mathbf{x}{:}Q.C'$

By definition $close^{\mathbb{D},\gamma}(\forall \mathbf{x}{:}Q.C') = \bigwedge_{\gamma' \in ans^{\mathbb{D}}_{set(\mathbf{x})}(subs_{\tilde{\gamma}}(Q))} close^{\mathbb{D},\tilde{\gamma}\cup\gamma'}(C')$.

Recall that by definition $fv(C) = fv(C') \cup fv(Q) \setminus set(\mathbf{x})$ and by hypothesis $fv(C) \subseteq var(\gamma)$ so $fv(C') \setminus set(\mathbf{x}) \subseteq var(\gamma)$. Observe that $var(\tilde{\gamma}) = var(\gamma) \setminus set(\mathbf{x})$ so $fv(C') \setminus set(\mathbf{x}) \subseteq var(\tilde{\gamma})$. Finally $var(\gamma') = set(\mathbf{x})$ so $fv(C') \subseteq var(\tilde{\gamma}) \cup var(\gamma')$ so by induction $close^{\mathbb{D},\tilde{\gamma}\cup\gamma'}(C')$ is closed.

Thus $close^{\mathbb{D},\gamma}(S)$ is closed as it is a conjunction of closed constraints.

Inductive case 2: $C = C' \wedge C''$

By definition $close^{\mathbb{D},\gamma}(C' \wedge C'') = close^{\mathbb{D},\gamma}(C') \wedge close^{\mathbb{D},\gamma}(C'')$.

Recall that by definition $fv(C) = fv(C') \cup fv(C'')$ and by hypothesis $fv(C) \subseteq var(\gamma)$ so $fv(C') \subseteq var(\gamma)$ and $fv(C'') \subseteq var(\gamma)$.

Thus by induction $close^{\mathbb{D},\gamma}(C')$ and $close^{\mathbb{D},\gamma}(C'')$ are closed so $close^{\mathbb{D},\gamma}(C)$ is closed.

□

Proposition 3.8.
*Let $L =$ **maximize** S **subject to** C be an $LP(CQ)$ model and $\mathbb{D}$ be a database then $close^{\mathbb{D}}(L)$ is closed.*

Proof. By definition $close^{\mathbb{D}}($**maximize** S **subject to** $C) =$ **maximize** $close^{\mathbb{D},\emptyset}(S)$ **subject to** $close^{\mathbb{D},\emptyset}(C)$. Observe that $fv(S) = \emptyset$ because L is well-formed. Thus by Lemma 3.6, $close^{\mathbb{D},\emptyset}(S)$ is closed. Similarly $fv(C) = \emptyset$ so by Lemma 3.7 $close^{\mathbb{D},\emptyset}(S)$ is closed. Thus $close^{\mathbb{D}}(L)$ is closed.

□

We now show that the $LP(CQ)$ closure is the identity when applied to a closed model. Observe that by definition the environment of a closure only changes when encoutering a $\forall$ or $\sum$ which will not happen in a closed $LP(CQ)$ model so we will only consider closures with an empty environment. For clarity we begin by proving this property on $LP(CQ)$ expressions and constraints in the following lemmas.

Lemma 3.9.
Let $S \in \mathrm{LE}(CQ)$ be a closed $LP(CQ)$ expression. Given a database $\mathbb{D}$, $close^{\mathbb{D},\emptyset}(S) = S$.

Proof. We show this by induction on the structure of S. Observe that because S is closed there are no cases where $S = \mathbf{num}(E)$ or $S = \sum_{\mathbf{x}:Q} S'$.

Base case 1: $S = N \in Num(CQ)$**.**

Observe that because S is closed then S is necessarily a constant $c \in \mathbb{R}$. Then by definition $close^{\mathbb{D},\emptyset}(S) = S$.

Base case 2: $S = \mathbf{weight}_{\mathbf{x}:Q'}(Q)$**.**

By definition $close^{\mathbb{D},\gamma}(S) = \mathbf{weight}_{\mathbf{x}:subs_{\tilde{\gamma}}(Q')}(subs_{\tilde{\gamma}}(Q))$ with $\gamma = \emptyset$ and $\tilde{\gamma} = \gamma_{|var(\gamma)\setminus set(\mathbf{x})} = \emptyset$. Thus $close^{\mathbb{D},\emptyset}(S) = \mathbf{weight}_{\mathbf{x}:subs_{\emptyset}(Q')}(subs_{\emptyset}(Q)) = S$.

Inductive case: $S = N \times S'$ **or** $S = S' + S''$**.**

Observe that because S then N, S' and S'' are also closed thus by induction $close^{\mathbb{D},\emptyset}(N) = N$, $close^{\mathbb{D},\emptyset}(S') = S'$ and $close^{\mathbb{D},\emptyset}(S'') = S''$. Then $close^{\mathbb{D},\emptyset}(N \times S') = N \times S'$ and $close^{\mathbb{D},\emptyset}(S' + S'') = S' + S''$ □

Lemma 3.10.
Let C be a closed $LP(CQ)$ constraint. Given a database $\mathbb{D}$, $close^{\mathbb{D},\emptyset}(C) = C$.

Proof. We show this by induction on the structure of C.

Observe that because C is closed there are no cases where $C = \forall \mathbf{x}{:}Q.C'$.

Base case 1: $C = \mathit{true}$**.**

By definition $\mathit{close}^{\mathbb{D},\emptyset}(\mathit{true}) = \mathit{true}$.

Base case 2: $C = S \leq S'$**.**

By definition $\mathit{close}^{\mathbb{D},\emptyset}(S \leq S') = \mathit{close}^{\mathbb{D},\emptyset}(S) \leq \mathit{close}^{\mathbb{D},\emptyset}(S')$. However C is closed so S and S' are also closed. Thus by Lemma 3.9 $\mathit{close}^{\mathbb{D},\emptyset}(S) = S$ and $\mathit{close}^{\mathbb{D},\emptyset}(S') = S'$ so $\mathit{close}^{\mathbb{D},\emptyset}(S \leq S') = S \leq S'$.

Inductive case: $C = C' \wedge C''$**.**

Observe that because C then C' and C'' are also closed thus by induction $\mathit{close}^{\mathbb{D},\emptyset}(C') = C'$, $\mathit{close}^{\mathbb{D},\emptyset}(C'') = C''$. Then $\mathit{close}^{\mathbb{D},\emptyset}(C' \wedge C'') = C' \wedge C''$. □

Proposition 3.11.
Let L be a closed $LP(CQ)$ model. Given a database $\mathbb{D}$, $\mathit{close}^{\mathbb{D}}(L) = L$.

Proof. Let S be a $LP(CQ)$ expression and C be a $LP(CQ)$ constraint such that $L =$ **maximize** S **subject to** C.

By defintion $\mathit{close}^{\mathbb{D}}(L) =$ **maximize** $\mathit{close}^{\mathbb{D},\emptyset}(S)$ **subject to** $\mathit{close}^{\mathbb{D},\emptyset}(C)$.

Observe that because L is closed then S and C are also closed so by Lemma 3.9 $\mathit{close}^{\mathbb{D},\emptyset}(S) = S$ and by Lemma 3.10 $\mathit{close}^{\mathbb{D},\emptyset}(C) = C$. Thus it follows that $\mathit{close}^{\mathbb{D}}(L) = \mathit{close}^{\mathbb{D}}($**maximize** S **subject to** $C) =$ **maximize** S **subject to** $C = L$. □

3.4.3 Semantics of $LP(CQ)$ models

As we saw in the two previous sections, the semantic of a $LP(CQ)$ model is defined by a two-step process. The model must first be closed before so that its **weight** expressions can be instantiated as linear program variables. Thus the semantic of a $LP(CQ)$ model L is formally defined as:

$$\langle L \rangle^{\mathbb{D}} = \mathrm{Inst}_{\mathbb{D}}(\mathit{close}^{\mathbb{D}}(L))$$

We let the reader check that the semantics of the model in Example 3.2 with the database from Figure 3.1 is indeed the linear program seen in Example 3.1. Instead we will look at the semantic of this model on a different database given in Figure 3.7.

Tasks	tid	skill	duration
	T_1	Python	45
	T_2	C	10
	T_3	SQL	20
	T_4	Java	30

Skills	eid	skill
	Alice	Python
	Alice	C
	Bob	SQL
	Bob	Python
	Carol	Python
	Carol	Java

Q_{assign}	t	s	e	d	var
	T_1	Python	Alice	45	$\theta^{(T_1,Alice)}_{Q_{assign}}$
	T_1	Python	Bob	45	$\theta^{(T_1,Bob)}_{Q_{assign}}$
	T_1	Python	Carol	20	$\theta^{(T_1,Carol)}_{Q_{assign}}$
	T_2	C	Alice	10	$\theta^{(T_2,Alice)}_{Q_{assign}}$
	T_3	SQL	Bob	20	$\theta^{(T_3,Bob)}_{Q_{assign}}$
	T_4	Java	Carol	30	$\theta^{(T_4,Carol)}_{Q_{assign}}$

Figure 3.7: Example for the *Tasks* and *Skills* tables. Answers of Q_{assign} query.

Observe that the answers of $\exists s'.Skills(e',s')$ and $\exists s'.Tasks(t',d')$ are as follows:

$\exists s'.Skills(e',s')$	e'
	Alice
	Bob
	Carol

$\exists s'.Tasks(t',s',d')$	tid	duration
	T_1	45
	T_2	10
	T_3	20
	T_4	30

Thus the closure $close^{\mathbb{D}}(L)$ of L on $\mathbb{D}$ is:

$$\begin{aligned}
\textbf{minimize } & \textbf{weight}_{(t,s,e,d):e\doteq Alice}(Q_{assign}) + \textbf{weight}_{(t,s,e,d):e\doteq Bob}(Q_{assign}) \\
+ \quad & \textbf{weight}_{(t,s,e,d):e\doteq Carol}(Q_{assign}) \\
\textbf{subject to } & \textbf{weight}_{(t,s,e,d):t\doteq T_1}(Q_{assign}) \geq 45 \\
\wedge \quad & \textbf{weight}_{(t,s,e,d):t\doteq T_2}(Q_{assign}) \geq 10 \\
\wedge \quad & \textbf{weight}_{(t,s,e,d):t\doteq T_3}(Q_{assign}) \geq 20 \\
\wedge \quad & \textbf{weight}_{(t,s,e,d):t\doteq T_4}(Q_{assign}) \geq 30 \\
\wedge \quad & \textbf{weight}_{(t,s,e,d):e\doteq Alice}(Q_{assign}) \leq 40 \\
\wedge \quad & \textbf{weight}_{(t,s,e,d):e\doteq Bob}(Q_{assign}) \leq 40 \\
\wedge \quad & \textbf{weight}_{(t,s,e,d):e\doteq Carol}(Q_{assign}) \leq 40.
\end{aligned}$$

Finally we instantiate the **weight** operators to obtain the semantics of L. For brevity we denote any variable $\theta^{[t/t',s/s',e/e',d/d']}_{Q_{assign}}$ by $\theta^{(t',e')}_{Q_{assign}}$ (for example $\theta^{[t/T_1,s/Python,e/Alice,d/45]}_{Q_{assign}}$ is written $\theta^{(T_1,Alice)}_{Q_{assign}}$).

$$\begin{aligned}
\textbf{minimize } & \theta_{Q_{assign}}^{(T_1,Alice)} + \theta_{Q_{assign}}^{(T_2,Alice)} + \theta_{Q_{assign}}^{(T_1,Bob)} + \theta_{Q_{assign}}^{(T_3,Bob)} + \theta_{Q_{assign}}^{(T_1,Carol)} + \theta_{Q_{assign}}^{(T_4,Carol)} \\
\textbf{subject to } & \theta_{Q_{assign}}^{(T_1,Alice)} + \theta_{Q_{assign}}^{(T_1,Bob)} + \theta_{Q_{assign}}^{(T_1,Carol)} \geq 45 \\
\wedge \quad & \theta_{Q_{assign}}^{(T_2,Alice)} \geq 10 \\
\wedge \quad & \theta_{Q_{assign}}^{(T_3,Bob)} \geq 20 \\
\wedge \quad & \theta_{Q_{assign}}^{(T_4,Carol)} \geq 30 \\
\wedge \quad & \theta_{Q_{assign}}^{(T_1,Alice)} + \theta_{Q_{assign}}^{(T_2,Alice)} \leq 40 \\
\wedge \quad & \theta_{Q_{assign}}^{(T_1,Bob)} + \theta_{Q_{assign}}^{(T_3,Bob)} \leq 40 \\
\wedge \quad & \theta_{Q_{assign}}^{(T_1,Carol)} + \theta_{Q_{assign}}^{(T_4,Carol)} \leq 40.
\end{aligned}$$

The optimal value of this linear program is 105 which can be achieved with the following assignments:

	Alice	Bob	Carol
T_1	15	20	10
T_2	10	-	-
T_3	-	20	-
T_4	-	-	30

3.4.4 Intricacies of the semantics

Scope of database variables

We now explain the use of $\tilde{\gamma} = \gamma_{|var(\gamma)\setminus\mathbf{x}}$ when defining the closure of $\forall$ and $\sum$ quantifiers.

Let C be the following $LP(CQ)$ constraint:

$$\forall(x'){:}r(x'.) \sum_{(x'):r'(x')} \mathbf{weight}_{(x,y):x\doteq x'}(q(x,y)) \leq \mathbf{num}(x').$$

Observe that there are two instances of the variable x' in **weight** expressions but that, as indicated by the highlighting, they are bound to different quantifiers. Indeed the occurence of x' in the **weight** operator is bound to the $\sum$ quantifier as it is its closest quantifier. Similarly the second occurrence of x' in the **num** operator is bound to its own closest quantifier which is the $\forall$. We will show on an example that the use of $\tilde{\gamma}$ in the closure operation allows us to properly handle such cases.

We consider a very simple database $\mathbb{D}$:

r	x
	13
	37

r’	x
	0
	1

If we close the top-level $\forall$ quantifier of C with $\mathbb{D}$ and an empty prior environment we obtain the following:

$$\begin{aligned} & close^{\mathbb{D},[x'/13]}(\sum_{(x'):r'(x')} \mathbf{weight}_{(x,y):x\doteq x'}(q(x,y)) \leq \mathbf{num}(x')) \\ \wedge \quad & close^{\mathbb{D},[x'/37]}(\sum_{(x'):r'(x')} \mathbf{weight}_{(x,y):x\doteq x'}(q(x,y)) \leq \mathbf{num}(x')). \end{aligned}$$

Then closing the inequalities yields:

$$\begin{aligned} & close^{\mathbb{D},[x'/13]}(\sum_{(x'):r'(x')} \mathbf{weight}_{(x,y):x\doteq x'}(q(x,y))) \leq close^{\mathbb{D},[x'/13]}(\mathbf{num}(x')) \\ \wedge \quad & close^{\mathbb{D},[x'/37]}(\sum_{(x'):r'(x')} \mathbf{weight}_{(x,y):x\doteq x'}(q(x,y))) \leq close^{\mathbb{D},[x'/37]}(\mathbf{num}(x')). \end{aligned}$$

Finally we close both operands of the inequality. In the right-hand operands we will simply substitute the values of x' (here 13 and 37) provided by the environment. However for the left-hand side, the values of x' will be overriden by new values yielded by the $\sum$ operator which we will then substitute into the **weight** operators as follows:

$$\begin{aligned} & \mathbf{weight}_{(x,y):x\doteq 1}(q(x,y)) + \mathbf{weight}_{(x,y):x\doteq 2}(q(x,y)) \leq 13 \\ \wedge \quad & \mathbf{weight}_{(x,y):x\doteq 1}(q(x,y)) + \mathbf{weight}_{(x,y):x\doteq 2}(q(x,y)) \leq 37. \end{aligned}$$

α-Renaming may change the semantics of $LP(CQ)$ programs.

We note that α-renaming the bound variables in weight expressions does *not* always preserve the semantics of $LP(CQ)$ programs. It may make previously equal queries different, so that different weights may be assigned to their answer sets.

To illustrate this let $\mathbf{x} = (x_1, x_2)$ and L be the following $LP(CQ)$ program:

$$\textbf{maximize } \mathbf{weight}_{\mathbf{x}:x_2\doteq 1}(r(\mathbf{x})) \textbf{ subject to } \mathbf{weight}_{\mathbf{x}:true}(r(\mathbf{x})) \leq 1.$$

Let $\mathbb{D}$ be a database with signature $\Sigma = \{r^{(2)}\}$ and interpretation $r^{\mathbb{D}} = \{(0,0),(0,1)\}$. If Q is the query $r(\mathbf{x})$ then the semantics of this $LP(CQ)$ program $\langle L\rangle^{\mathbb{D}}$ is the linear program:

$$\textbf{maximize } \theta_Q^{(0,1)} \textbf{ subject to } \theta_Q^{(0,0)} + \theta_Q^{(0,1)} \leq 1.$$

The optimal value of $\langle L\rangle^{\mathbb{D}}$ is $[\![\langle L\rangle^{\mathbb{D}}]\!]_{\mathbb{D}} = 1$ since $\theta_Q^{(0,0)} + \theta_Q^{(0,1)} \leq 1$ and $\theta_Q^{(0,0)} \geq 0$. Now let us α-rename the second occurrence of $\mathbf{x}$ in L to $\mathbf{x}' = (x_1', x_2')$ yielding the following $LP(CQ)$ program L':

$$\textbf{maximize } \mathbf{weight}_{\mathbf{x}:x_2\doteq 1}(r(\mathbf{x})) \textbf{ subject to } \mathbf{weight}_{\mathbf{x}':true}(r(\mathbf{x}')) \leq 1.$$

The semantics $\langle L' \rangle^{\mathbb{D}}$ is the following linear program where Q is $r(\mathbf{x})$ and Q' is $r(\mathbf{x}')$:

$$\textbf{maximize } \theta_Q^{(0,1)} \textbf{ subject to } \theta_{Q'}^{(0,0)} + \theta_{Q'}^{(0,1)} \leq 1.$$

The optimal value of $\langle L' \rangle^{\mathbb{D}}$ is ∞ since $\theta_Q^{(0,1)}$ is no longer constrained. Indeed the queries $r(\mathbf{x})$ and $r(\mathbf{x}')$ are syntactically different so we consider them to be different and to have distinct answer sets. Therefore the two **weight** operators are instantiated with distinct sets of LP variables.

3.5 Solving $LP(CQ)$ programs

In this section we discuss the combined complexity then the data complexity of computing the optimal value of closed $LP(CQ)$ models. Finally we briefly discuss the more general case of solving open $LP(CQ)$ models.

3.5.1 Solving closed $LP(CQ)$ models

Solving a closed $LP(CQ)$ model with a database $\mathbb{D}$ is quite straightforward. Indeed, the natural interpretation of L with $\mathbb{D}$ yields a linear program $L' = \langle L \rangle^{\mathbb{D}}$ which can then be handed to a linear program solver to obtain its optimal value $opt(L')$ and an optimal solution $w : var(L') \rightarrow \mathbb{R}^+$.

3.5.1.1 Combined complexity

We formally define the problem of solving a $LP(CQ)$ model L on a database $\mathbb{D}$. We consider both L and $\mathbb{D}$ to be a part of the input in order to study the combined complexity of this problem.

Definition 3.12 ($\mathsf{SOLVE}(LP(CQ))$ problem)**.**
Input: *A $LP(CQ)$ model L and database $\mathbb{D}$.*
Output: *The optimal value of L on $\mathbb{D}$, $opt(\llbracket L \rrbracket_{\mathbb{D}})$.*

Intuitively this problem is NP-hard as instantiating L requires computing the answer sets of the base queries $cq(L)$. In order to formally study the hardness of this problem, we introduce the following associated decision problem:

Definition 3.13 ($\mathsf{DECIDE}_{\neq 0}(LP(CQ))$ problem)**.**
Input: *A $LP(CQ)$ model L and database $\mathbb{D}$.*
Output: *True if $opt(\llbracket L \rrbracket_{\mathbb{D}}) \neq 0$, false otherwise..*

We now show that this problem is NP-hard by reduction from query satisfiability.

Theorem 3.14.
The $\mathsf{DECIDE}_{\neq 0}(LP(CQ))$ problem is NP-hard.

Proof. The proof follows by reduction from the problem of deciding whether $ans^{\mathbb{D}}(Q) \neq \emptyset$ given a conjunctive query Q and a database $\mathbb{D}$

in the input, which is known to be NP-complete [CM77]. We consider, given a conjunctive query Q, the following $LP(CQ)$: $L_Q :=$ **maximize** $\mathbf{weight}_{\mathbf{x}:true}(Q)$ **subject to** $\mathbf{weight}_{\mathbf{x}:true}(Q) \leq 1$.

The hardness now follows from the fact that for every Q and $\mathbb{D}$, $\langle L_Q \rangle^{\mathbb{D}} \neq 0$ if and only if $ans^{\mathbb{D}}(Q) \neq \emptyset$. Indeed, if $ans^{\mathbb{D}}(Q) = \emptyset$, then $\langle L_Q \rangle^{\mathbb{D}} =$ **maximize** 0 **subject to** $0 \leq 1$. The optimal value of $\langle L_Q \rangle^{\mathbb{D}}$ is thus 0. However, if $ans^{\mathbb{D}}(Q) \neq \emptyset$, we have $\langle L_Q \rangle^{\mathbb{D}} =$ **maximize** ϕ **subject to** $\phi \leq 1$ where $\phi = \sum_{\alpha \in ans^{\mathbb{D}}(Q)} \theta_Q^{\alpha}$. Let $\alpha \in ans^{\mathbb{D}}(Q)$ and consider the weighting ω_α such that $\omega_\alpha(\theta_Q^\alpha) := 1$ and for every $\alpha' \in ans^{\mathbb{D}}(Q)$ such that $\alpha' \neq \alpha$, $\omega_\alpha(\theta_Q^{\alpha'}) := 0$. This weighting clearly respects the constraints of $\langle L_Q \rangle^{\mathbb{D}}$ and has value 1, showing that the optimal value of $\langle L_Q \rangle^{\mathbb{D}} \geq 1$. □

3.5.1.2 Data complexity

The hardness from Theorem 3.14 mainly stems from the hardness of answering conjunctive queries that is only relevant in the context of combined complexity.

It is often assumed however that the size of the query is small with respect to the size of the data, hence one can study the data complexity of the problem, that is, the complexity of the problem when we the query is fixed. We lift this notion to our problem by fixing the linear program L. In this case, computing $opt(\langle L \rangle^{\mathbb{D}})$ can be done in time polynomial in $|\mathbb{D}|$ using the following procedure:

1. Explicitly compute $ans^{\mathbb{D}}(Q)$ for every Q appearing in $cq(L)$,
2. Compute $L' = \langle L \rangle^{\mathbb{D}}$,
3. Solve L' in time polynomial in $|L'|$ using an LP-solver.

The exact complexity of this procedure is however dependent on the size of $\langle L \rangle^{\mathbb{D}}$ whose number of variables is bounded by the sum of $|ans^{\mathbb{D}}(Q)|$ for every $Q \in cq(L)$. We lift the AGM bound presented in Section 2.2.4 to closed $LP(CQ)$ models by defining $\mathsf{AGM}(L)$ to be the maximum of $\mathsf{AGM}(Q)$ for every query $Q \in cq(L)$. Observe that any $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ can be instantiated with $|ans^{\mathbb{D}}(Q)|$ variables at most so the size of the encoding of $\langle L \rangle^{\mathbb{D}}$ can now be upper bounded by $O(|L| \cdot |\mathbb{D}|^{\mathsf{AGM}(L)} \cdot \log(|\mathbb{D}|^{\mathsf{AGM}(L)}))$. Using a worst-case optimal join algorithm such as Triejoin [Vel14] to compute $ans^{\mathbb{D}}(Q)$ in time $O(|\mathbb{D}|^{\mathsf{AGM}(Q)})$ and Theorem 2.8, we conclude that one can compute $opt(\langle L \rangle^{\mathbb{D}})$ in time $O(|\langle L \rangle^{\mathbb{D}}|_b \cdot |\mathbb{D}|^{\mathsf{AGM}(L) \cdot \ell})$ where $|\langle L \rangle^{\mathbb{D}}|_b = O((|L| \cdot |\mathbb{D}|^{\mathsf{AGM}(L)} \cdot \log(|\mathbb{D}|^{\mathsf{AGM}(L)}))$.

In data complexity this corresponds to $O(|\mathbb{D}|^{\mathsf{AGM}(L)} \cdot \log(|\mathbb{D}|^{\mathsf{AGM}(L)}) \cdot |\mathbb{D}|^{\mathsf{AGM}(L) \cdot \ell}) = O(|\mathbb{D}|^{\mathsf{AGM}(L) \cdot (\ell+1)} \cdot \log(\mathbb{D}))$.

3.5.2 Solving open $LP(CQ)$ models

We now discuss the problem of solving an open $LP(CQ)$ program K. To do so we can simply close K yielding a closed $LP(CQ)$ program $L = close^{\mathbb{D}}(K)$. The procedure described previously can then be applied to solve L.

Thus it follows that one can compute the optimal value of K in $O(|\langle L\rangle^{\mathbb{D}}|_b \cdot |\mathbb{D}|^{\mathsf{AGM}(L)\cdot\ell})$. Recall that when computing L, closing a constraint $\forall\mathbf{x}{:}Q.C$ generates $|ans^{\mathbb{D}}(Q)|$ constraints $close^{\mathbb{D},\gamma}(C)$ which can be further compounded if C contains more $\forall$ quantifiers. Thus L can be quickly grow much bigger than K depending on the number and nesting depth of quantifiers in K as well as the number of answers of the queries that appear in these quantifiers.

There are thus two levers that could be considered when trying to optimize the solving time of an open $LP(CQ)$ model: the closing phase and the instantiation phase. Optimizing the closing phase falls out of the scope of this thesis and we will instead focus on improving the instantiation phase to generate linear programs with fewer variables. Thus, throughout this thesis we will study the problem of solving closed $LP(CQ)$ models and consider the closing phase to be a form of implicit pre-computation.

3.6 Alternate interpretations

As we saw in Section 3.5 the natural semantics of a $LP(CQ)$ model on a database can be exponentially larger than the model and input database which makes the resulting linear program hard to solve. More specifically recall that, even when considering a closed $LP(CQ)$ model, the instantiation step alone causes a blow up in the number of variables.

In the following chapters we will show that we can solve some $LP(CQ)$ models more efficiently by defining alternate interpretations of closed models that are equivalent to their natural semantics.

In Section 3.6.1 we formalize the notions of $LP(CQ)$ interpretations and instantiations. Then, in Section 3.6.2, we provide a general framework to prove that two interpretations are equivalent i.e., that they have the same optimal value.

3.6.1 Definitions

Recall that we defined the natural instantiation in Section 3.4.1 as a function that maps **weight** operators to sums of linear program variables so we begin by introducing a few notations to refer to **weight** operators more easily. Given a closed $LP(CQ)$ model L we denote by $\mathbf{W}(L)$ the set of all the closed **weight** operators that appear in L. Additionally we define $\mathbf{W}_Q(L) = \{\mathbf{weight}_{\mathbf{x}:Q''}(Q') \in \mathbf{W}(L) \mid Q = Q'\}$ the set of all the closed **weight** operators of L that refer to Q. We extend these notations to closed linear sums in $L\textsc{e}(CQ)$ and linear constraints in $L\textsc{c}(CQ)$ in a straightforward manner. We sometimes refer to **weight** operators of L by $\mathbf{w} \in \mathbf{W}(L)$ for short.

Let $\mathcal{W}$ be a set of closed **weight** operators and Ξ be a set of linear program variables. We call a function $\mathrm{Inst} : \mathcal{W} \to L\textsc{e}_{\Xi}$ an *instantiation function.* Such a function can then be applied to $LP(CQ)$ sums, constraints and models to define an interpretation of $LP(CQ)$ models. Observe that the function $\mathrm{Inst}_{\mathbb{D}}$ we defined in Section 3.4.1 to describe the natural semantics of $LP(CQ)$ models is indeed an instantiation function.

Finally we formalize what it means to interpret a $LP(CQ)$ model as a linear program. When we define more efficient interpretations in the following they will include additional constraints to ensure their equivalence with the natural semantics. Thus we define the interpretation of a $LP(CQ)$ model L given an instantiation function Inst and additional constraint ac as

$$\langle L\rangle^{\mathrm{Inst},ac} = \mathrm{Inst}(L) \wedge ac.$$

Observe that the natural semantics is indeed an interpretation with instantiation $\mathrm{Inst}_{\mathbb{D}}$ and additional constraint *true*. In the rest of this thesis we will sometimes refer to it as the *natural interpretation*.

3.6.2 Equivalence of $LP(CQ)$ interpretations

We now show how to prove that two interpretations are equivalent i.e., that they have the same optimal values. We consider a closed $LP(CQ)$ model L and two interpretations $\rho_1(L) = \langle L\rangle^{\mathrm{Inst}_1,ac_1}$ and $\rho_2(L) = \langle L\rangle^{\mathrm{Inst}_2,ac_2}$. Intuitively the equivalence of $\rho_1(L)$ and $\rho_2(L)$ will follow from linking pairs of weightings w_1 and w_2 of $\rho_1(L)$ and $\rho_2(L)$ respectively such that

$$w_1(\mathrm{Inst}_1(\mathbf{w})) = w_2(\mathrm{Inst}_2(\mathbf{w}))$$

for any $\mathbf{w} \in \mathbf{W}(L)$.

We fix two sets of linear program variables Ξ_1 and Ξ_2, two instantiations $\mathrm{Inst}_1 : \mathbf{W}(S) \to L\textsc{e}_{\Xi_1}$ and $\mathrm{Inst}_2 : \mathbf{W}(S) \to L\textsc{e}_{\Xi_2}$ and two weightings $w_1 : \Xi_1 \to \mathbb{R}^+$ and $w_2 : \Xi_2 \to \mathbb{R}^+$.

We can then easily extend this correspondence to linear sums and linear constraints in the two following lemmas.

Lemma 3.15.
Let S be a closed $L\textsc{e}(CQ)$ expression.

If $w_1(Inst_1(\mathbf{w})) = w_2(Inst_2(\mathbf{w}))$ for any $\mathbf{w} \in \mathbf{W}(S)$ then $[\![Inst_1(S)]\!]_{w_1} = [\![Inst_2(S)]\!]_{w_2}$.

Proof. We show this by induction on the structure of S.

Base case 1 $S = \mathbf{weight}_{\mathbf{x}:Q'}(Q)$

$$\begin{array}{rcll} [\![\mathrm{Inst}_1(S)]\!]_{w_1} & = & w_1(\mathrm{Inst}_1(S)) & \\ & = & w_2(\mathrm{Inst}_2(S)) & \text{by hypothesis} \\ & = & [\![\mathrm{Inst}_2(S)]\!]_{w_2} & \end{array}$$

Base case 2 $S = N$

Observe that because S is closed then N is a constant c. Thus it is obvious that $[\![\mathrm{Inst}_1(S)]\!]_{w_1} = [\![\mathrm{Inst}_2(S)]\!]_{w_2}$.

Inductive cases $S = N \times S'$ or $S = S' + S''$.

These are straightforward by induction.

□

Lemma 3.16.
Let C be a closed $\textsc{Lc}(CQ)$ *expression.*

If $w_1^Q(\mathit{Inst}_1(\mathbf{w})) = w_2^Q(\mathit{Inst}_2(\mathbf{w}))$ *for any* $\mathbf{w} \in \mathbf{W}(S)$ *then* $[\![\mathit{Inst}_1(C)]\!]_{w_1} = [\![\mathit{Inst}_2(C)]\!]_{w_2}$.

Proof. By induction on the structure of C.

Base case 1 $C = \mathit{true}$

Obvious.

Base case 2 $C = S \leq S'$

This follows directly from Lemma 3.15.

Induction step $C = C' \wedge C''$

This is straightforward by induction.

□

Finally, in order to simplify future proofs, we show that we can deduce a relationship between $\mathit{opt}(\rho_1(L))$ and $\mathit{opt}(\rho_2(L))$ from a correspondence on partial weightings of these linear programs by considering each $Q \in \mathit{cq}(L)$ independently.

We denote the set of variables of the interpretation $\rho_1(L)$ related to a query $Q \in \mathit{cq}(L)$ by $\mathit{var}_Q(\rho_1(L)) = \bigcup_{\mathbf{w} \in \mathbf{W}_Q(\rho_1(L))} \mathit{var}(\mathrm{Inst}_1(\mathbf{w}))$. Observe that given two distinct queries Q and Q', the domains of w_1^Q and $w_1^{Q'}$ are disjoint by definition.

Proposition 3.17.
Let L be a closed $LP(CQ)$ model.

Let $\rho_1(L) = \langle L \rangle^{\mathit{Inst}_1, ac_1}$ *and* $\rho_2(L) = \langle L \rangle^{\mathit{Inst}_2, ac_2}$.

If it holds for any $Q \in \mathit{cq}(L)$ *and weighting* w_1^Q *of* $\mathit{var}_Q(\rho_1(L))$ *such that* $[\![ac_1]\!]_{w_1^Q} = 1$ *that there exists a weighting* w_2^Q *of* $\mathit{var}_Q(\rho_2(L))$ *such that*

- $[\![ac_2]\!]_{w_2^Q} = 1$,
- $\forall \mathbf{w} \in \mathbf{W}_Q(L)$, $[\![\mathit{Inst}_1(\mathbf{w})]\!]_{w_1^Q} = [\![\mathit{Inst}_2(\mathbf{w})]\!]_{w_2^Q}$

then $\mathit{opt}(L_1) \leq \mathit{opt}(L_2)$.

Proof. For any $Q \in \mathit{cq}(L)$, let w_1^Q be a weighting of $\mathit{var}_Q(\rho_1(L))$ such that $[\![ac_1]\!]_{w_1^Q} = 1$ It follows by hypothesis that there is a weighting w_2^Q of $\mathit{var}_Q(\rho_2(L))$ such that $[\![ac_2]\!]_{w_2^Q} = 1$ and $[\![\mathrm{Inst}_1(\mathbf{w})]\!]_{w_1^Q} = [\![\mathrm{Inst}_2(\mathbf{w})]\!]_{w_2^Q}$ for any $\mathbf{w} \in \mathbf{W}_Q(L)$.

Let $w_1 : \Xi_1 \to \mathbb{R}^+$ be the weighting defined such that $w_1(\mathbf{w}) = w_1^Q(\mathbf{w})$ for any $Q \in cq(L)$ and $\mathbf{w} \in \mathbf{W}_Q(L)$.

Let $w_2 : \Xi_2 \to \mathbb{R}^+$ be the weighting defined such that $w_2(\mathbf{w}) = w_2^Q(\mathbf{w})$ for any $Q \in cq(L)$ and $\mathbf{w} \in \mathbf{W}_Q(L)$.

Observe that $[\![ac_1(Q)]\!]_{w_1} = 1$, $[\![ac_2(Q)]\!]_{w_2} = 1$ and for any $\mathbf{w} \in \mathbf{W}(L)$, $[\![\mathrm{Inst}_1(\mathbf{w})]\!]_{w_1^Q} = [\![\mathrm{Inst}_2(\mathbf{w})]\!]_{w_2^Q}$.

Suppose now that w_1 also satisfies $\mathrm{Inst}_1(C)$ (making it a solution of L_1) then by Lemma 3.16 w_2 satisfies $\mathrm{Inst}_2(C)$ and is a solution of L_2. Moreover by Lemma 3.15 it follows that $[\![\mathrm{Inst}_1(S)]\!]_{w_1} = [\![\mathrm{Inst}_2(S)]\!]_{w_2}$. Thus for any optimal solution of L_1 there is a solution of L_2 with the same objective value so $opt(L_1) \leq opt(L_2)$. □

Observe that if $opt(L_1) \leq opt(L_2)$ and $opt(L_2) \leq opt(L_1)$ then it follows that $opt(L_1) = opt(L_2)$.

3.7 Conclusion

In this chapter we have extensively discussed the syntax and semantics of the $LP(CQ)$ language that allows one to model linear programs whose underlying data is represented by conjunctive queries. We then studied the complexity of solving $LP(CQ)$ models and remarked that this problem is NP-hard in general. Finally we introduced the notion of alternate interpretations of $LP(CQ)$ models to pave the way for more efficient interpretations that we will present in the following chapters.

Chapter 4

Tractable fragment of $LP(CQ)$

Contents

4.1 Introduction

In this chapter we introduce an alternate tractable interpretation based on hypertree decompositions to solve some closed $LP(CQ)$ models more efficiently.

Recall that we remarked in Section 3.5 that the complexity of solving $LP(CQ)$ models stems from answering the query Q for each $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ term. Classically, from Yannakakis' algorithm [Yan81] to factorized databases [OZ12, Olt16], tractable results for various operations[1] on conjunctive queries rely on hypertree decompositions. In line with these results we will lift the notions of hypertree decomposition and hypertree width to closed $LP(CQ)$ models. We will then show that we can exploit the decomposition of a closed $LP(CQ)$ model to construct a linear program with fewer variables than its natural semantics.

[1] Satisfiability, enumeration, counting, . . .

Intuitively we reduce the number of variables in the linear program by instantiating each **weight** operator in a closed $LP(CQ)$ model with a single variable. We give a broad intuition of this approach on a very basic example.

Intuition of the tractable interpretation

Consider a conjunctive query $Q = R(x) \wedge R(y)$ and a database $\mathbb{D}$ with a single table $R^{\mathbb{D}} = \{(0),(1)\}$. We then consider the following closed $LP(CQ)$ program L:

$$\begin{array}{rl} \textbf{maximize} & \textbf{weight}_{(x,y):y\doteq 0}(Q) + \textbf{weight}_{(x,y):y\doteq 1}(Q) \\ \textbf{subject to} & \textbf{weight}_{(x,y):x\doteq 0}(Q) \leq 1 \\ \wedge & \textbf{weight}_{(x,y):x\doteq 1}(Q) \leq 1 \end{array}$$

The answer set of Q is $ans^{\mathbb{D}}(Q) = \{\alpha \mid \alpha : \{x,y\} \to \{0,1\}\}$. The natural interpretation $\langle L\rangle^{\mathbb{D}}$ is thus the following linear program with variables in $\Theta_L^{\mathbb{D}}$, where we denote any query answer $\alpha \in ans^{\mathbb{D}}(Q)$ by a pair $(\alpha(x'),\alpha(y'))$ in the Cartesian product $\{0,1\}^2$ for brevity:

$$\begin{array}{rl} \textbf{maximize} & \theta_Q^{(0,0)} + \theta_Q^{(0,1)} + \theta_Q^{(1,0)} + \theta_Q^{(1,1)} \\ \textbf{subject to} & \theta_Q^{(0,0)} + \theta_Q^{(0,1)} \leq 1 \\ \wedge & \theta_Q^{(1,0)} + \theta_Q^{(1,1)} \leq 1 \end{array}$$

Observe that this linear program has an optimal value of 2.

The main idea of the tractable interpretation is to instantiate each $\textbf{weight}_{\mathbf{x}:v\doteq d}(Q)$ with a single variable $\xi_{[v/d]}$. In this case this yields the following:

$$\begin{array}{rl} \textbf{maximize} & \xi_{[y/0]} + \xi_{[y/1]} \\ \textbf{subject to} & \xi_{[x/0]} \leq 1 \\ \wedge & \xi_{[x/1]} \leq 1 \end{array}$$

However observe that this new linear program is now unbounded as its constraints are now independent from its objective function. In this case we can restore the link between the variables by adding the following linear constraint to the LP:

$$\xi_{[y/0]} + \xi_{[y/1]} = \xi_{[x/0]} + \xi_{[x/1]}$$

Note that while this linear program is simple enough that fixing it only required adding a single so-called soundness constraint, their general construction is more complex. Intuitively we rely on the fact that the projections of the answers of Q are a succinct representation of the answer set of Q (see Lemma 2.5) to bound the number of variables of the T-factorized interpretation. We then exploit the structure exposed by the tree decomposition of the query to reconstruct the link between the variables through the soundness constraints mentioned above.

Overview of the tractable interpretation

We have seen in Section 3.5 that using the natural interpretation to solve closed $LP(CQ)$ models is inefficient as instantiating models can generate a large amount of variables related to the answers of the queries that appear in the model. Indeed, the number of variables of $\langle L\rangle^{\mathbb{D}}$ may contain up to $D^{\mathsf{AGM}(L)}$ variables. In this chapter, we propose the so-called T-factorized interpretation that, for a fragment of closed $LP(CQ)$ models, yields a linear program equivalent to their natural interpretation (i.e., that has the same optimal value) but only $\mathbb{D}^k$ variables where k is a generalization of hypertree-width to closed $LP(CQ)$ models that may possibly be smaller than $\mathsf{AGM}(Q)$ leading to a more efficient method for solving $LP(CQ)$ models.

Outline of the chapter

In Section 4.2 we define the so-called $LP(CQ)_{proj}$ fragment and lift the notion of fractional hypertree width to models in this fragment, we then define the so-called T-factorized interpretation. Then in Section 4.3 we describe how the T-factorized interpretation can be used to solve $LP(CQ)_{proj}$ models. Finally Section 4.4 is dedicated to proving that this approach is correct.

4.2 A tractable interpretation of $LP(CQ)$ models

In this section we formally define the *tree decomposition-based factorized interpretation* or T-factorized interpretation for short that will allow us to solve closed $LP(CQ)$ models more efficiently.

4.2.1 Characterizing tractable $LP(CQ)$ models and their width

We start by defining the fragment of projecting $LP(CQ)$ programs, whose main restriction resides on how the **weight** operators can select subset of answers of their queries. This restriction will allow us to apply the T-factorized interpretation and bound the number of its variables.

Definition 4.1.
The fragment $LP(CQ)_{proj}$ is the set of closed $LP(CQ)$ programs L such that every one of its $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ satisfies the following:

- *$fv(Q) = set(\mathbf{x})$,*

- *Q' is of the form $\mathbf{x}' \doteq \mathbf{d}$ with $fv(Q') = set(\mathbf{x}') \subseteq set(\mathbf{x})$ and $set(\mathbf{d}) \subset dom$.*

Additionally we denote by $LP(CQ_{qf})_{proj}$ the subset of $LP(CQ)_{proj}$ where every conjunctive query $Q \in cq(L)$ is quantifier free.

We now lift the notions of tree decompositions and (fractional hypertree) width to closed $LP(CQ_{qf})_{proj}$ programs.

The T-factorized interpretation of a closed $LP(CQ)$ model L will exploit tree decompositions of the queries in $cq(L)$ that have some additional structure. The complexity of this new approach will be tied to the width of these decompositions. Hence, to present our algorithm, we first need to lift the concept of tree decompositions of queries to $LP(CQ)$ models to integrate this additional structure. Moreover we also lift the notion of width in order to analyze its complexity.

Definition 4.2.
Let L be an $LP(CQ_{qf})_{proj}$ program and $\mathcal{T} = (T_Q)_{Q \in cq(L)}$ a collection of decomposition trees.

We call $\mathcal{T}$ a tree decomposition of L if for any expression $\textbf{weight}_{\mathbf{x}:\mathbf{x}' \doteq \mathbf{y}}(Q)$ in L, $T_Q = (\mathcal{V}_Q, \mathcal{E}_Q, \mathcal{B}_Q)$ is a tree decomposition of Q and there is a node u of T_Q such that $\mathcal{B}_Q(u) = set(\mathbf{x}')$.

We define the width of $\mathcal{T}$ to be the maximal (fractional hypertree) width of T_Q for $Q \in cq(L)$. The size of $\mathcal{T}$ is defined to be $|\mathcal{T}| = \sum_{Q \in cq(L)} |\mathcal{V}_Q|$.

Observe that the width of the decompositions in $\mathcal{T}$ may be greater than the minimal width of the queries in $cq(L)$ as they might require bigger bags in some places in order to cover the **weight** expressions in L. We show a tree decomposition as part of an example in Section 4.2.3.

4.2.2 Tree decomposition-based factorized interpretation

In this section we define a more succinct interpretation – the so-called *tree decomposition-based interpretation* or *T-factorized interpretation* for short – that leverages the efficiency afforded by the tree decompositions of the base queries in the $LP(CQ)$ model. As we saw in the introduction of this chapter, the general idea is to instantiate each **weight** operator with (at most) one variable then add some soundness constraints to restore the semantical link between the new variables.

Recall that in Section 3.6 we defined interpretations of closed $LP(CQ)$ models as pairs of instantiations and additional constraints.

4.2.2.1 T-factorized instantiation

We begin by defining the T-factorized instantiation. Intuitively this instantiation maps every $\textbf{weight}_{\mathbf{x}:Q'}(Q) \in \mathbf{W}(L)$ to a variable associated to a tuple in the projection of $ans^{\mathbb{D}}(Q)$ on the bag of some node of the decomposition of Q. Formally the set of variables of the T-factorized instantiation is $\Xi_L = \biguplus_{Q \in cq(L)} \Xi_{L,Q}$ where $\Xi_{L,Q} = \biguplus_{u \in \mathcal{V}_Q} \{\xi_{Q,u,\beta} \mid \beta \in ans^{\mathbb{D}}_{\mathbf{x}}(Q)_{|\mathcal{B}_Q(u)}\}$. Observe that every $\Xi_{L,Q}$ can be computed efficiently:

Lemma 4.3.
Let k be the width of T.

The size of $\Xi_{L,Q}$ is at most $|\mathcal{V}| \cdot |\mathbb{D}|^k$ and one can compute $\Xi_{L,Q}$ in time $O(|T| \cdot |\mathbb{D}|^k \log(|\mathbb{D}|))$.

Proof. It follows directly by Lemma 2.5 in Section 2.2.4. □

We now define the T-factorized instantiation:

Definition 4.4 (T-factorized instantiation)**.**
Given a closed $LP(CQ_{qf})_{proj}$ model L, database $\mathbb{D}$ and decomposition $\mathcal{T}$ of L, we denote the instantiation of L with $\mathbb{D}$ and $\mathcal{T}$ by $FInst_{\mathbb{D},\mathcal{T}}(L)$.

It is the linear program obtained by replacing every $\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q)$ in L with $FInst_{\mathcal{T},\mathbb{D}}(\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q))$ where $FInst_{\mathcal{T},\mathbb{D}}$ is the function that maps any $\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q)$ to $\xi_{Q,u,\beta}$ with u the vertex of T_Q such that $\mathcal{B}_Q(u) = fv(Q')$ and $\beta = [\mathbf{x}'/\mathbf{y}]$ if $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ or 0 otherwise.

Observe that $\beta \notin ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ iff $ans^{\mathbb{D}}_{\mathbf{x}}(Q \wedge \mathbf{x}'\doteq\mathbf{y}) = \emptyset$ so the factorized instantiation of $\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q)$ is 0 iff its natural instantiation is also 0.

Observe that each operator $\mathbf{w}$ in L is instantiated with one variable at most so it can be encoded in $\log(|\mathbb{D}|^k)$ bits from which the following lemma follows:

Lemma 4.5.
Given a closed $LP(CQ_{qf})_{proj}$ model L, database $\mathbb{D}$ and decomposition $\mathcal{T}$ of L of width k, $FInst_{\mathbb{D},\mathcal{T}}(L)$ can be encoded in $O(|L| \cdot \log(|\mathbb{D}|^k))$ bits.

4.2.2.2 Local soundness constraints

While the T-factorized interpretation yields a more concise linear program, it is not yet sufficient to solve $LP(CQ)$ models efficiently as this new linear program does not necessarily have the same objective value as the natural interpretation of the model[2].

Thus we need to add additional constraints to link our new variables together. Given a query Q and its tree decomposition $T_Q \in \mathcal{T}$, we define the so-called *local soundness constraints* on an edge (u, v) of T_Q.

Definition 4.6 (Local soundness)**.**
Given a query Q, its tree decomposition T_Q and database $\mathbb{D}$.

Let $A = ans^{\mathbb{D}}_{\mathbf{x}}(Q)$ and $e = (u, v)$ be an edge of T_Q.

$$lsc^{T_Q,\mathbb{D}}(e) = \bigwedge_{\gamma \in A_{|\mathcal{B}_Q(u)\cap\mathcal{B}_Q(v)}} \sum_{\beta \in A_{|\mathcal{B}_Q(u)}[\gamma]} \xi_{Q,u,\beta} \doteq \sum_{\beta' \in A_{|\mathcal{B}_Q(v)}[\gamma]} \xi_{Q,v,\beta'}$$

We denote the local soundness constraint for a query Q by $lsc(Q) = \bigwedge_{e\in T_Q} lsc^{T_Q,\mathbb{D}}(e)$. It can also be computed efficienctly with respect to the width of its decomposition:

Lemma 4.7.
Let k be the width of T.

The constraint $lsc(Q)$ can be encoded in $O(|T| \cdot |\mathbb{D}|^k \cdot \log(|\mathbb{D}|^k))$ bits and it can be computed in time $O(|Q| \cdot |T| \cdot |\mathbb{D}|^k \log(|\mathbb{D}|))$.

[2] We observed this in the example of the introduction of this chapter.

Proof. In order to compute $\mathit{lsc}(Q)$, we start by computing $(\mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(u)})_{u \in \mathcal{V}}$ using Lemma 2.5. Then for each edge $e = (u, v)$ of T, we construct $\mathit{lsc}^{T_Q,\mathbb{D}}(e)$ as follows: We begin by iterating over the tuples $\beta \in \mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$. Let $\gamma = \beta_{|\mathcal{B}(v)}$, we create a linear sum $S_u^\gamma = \xi_{Q,u,\beta}$ if it does not already exist or we append $+\xi_{Q,u,\beta}$ if it does. Similarly we construct sums S_v^γ by iterating over the tuples $\beta' \in \mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(v)}$.

Finally, we obtain $\mathit{lsc}^{T_Q,\mathbb{D}}(e) = \bigwedge_{\gamma \in A_{|\mathcal{B}_Q(u) \cap \mathcal{B}_Q(v)}} S_u^\gamma \doteq S_v^\gamma$ by pairing each S_u^γ with the corresponding S_v^γ. Observe that each sum S_u^γ (resp. S_v^γ) has a counterpart S_u^γ (resp. S_u^γ) because the β and β' are projections of the tuples of $\mathit{ans}^{\mathbb{D}}(Q)$.

Observe that each $\beta \in \mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ and $\beta' \in \mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(v)}$ is listed at most once for each edge of T so $\mathit{lsc}^{T_Q,\mathbb{D}}(e)$ can be encoded in $O(2 \cdot |\mathbb{D}|^k \cdot \log(|\mathbb{D}|^k))$ bits from which it follows that $\mathit{lsc}(Q)$ can be encoded in $O(|T| \cdot |\mathbb{D}|^k \cdot \log(|\mathbb{D}|^k))$ bits.

Moreover observe that the projection γ can be constructed in time $O(|Q|)$ so the total time required to construct $\mathit{lsc}(Q)$ is $O(|Q| \cdot |T| \cdot |\mathbb{D}|^k \log(|\mathbb{D}|))$. □

4.2.2.3 T-factorized interpretation

Finally we define the factorized interpretation of a closed $LP(CQ_{qf})_{proj}$ L with a tree decomposition $\mathcal{T}$ and database $\mathbb{D}$ denoted by $\rho^{\mathcal{T},\mathbb{D}}(L)$. It is obtained by combining the local soudness constraint of any edge e in any decomposition $T_Q \in \mathcal{T}$ with the factorized instantiation of L as follows:

Definition 4.8 (Factorized interpretation)**.**
Given a closed $LP(CQ_{qf})_{proj}$ model L, database $\mathbb{D}$ and decomposition $\mathcal{T}$ of L,

$$\rho^{\mathcal{T},\mathbb{D}}(L) = \langle L \rangle^{FInst_{\mathcal{T},\mathbb{D}},C}$$

with $C = \bigwedge_{T_Q \in \mathcal{T}} \bigwedge_{e \in \mathcal{E}_Q} \mathit{lsc}^{T_Q,\mathbb{D}}(e)$.

The factorized interpretation $\rho^{\mathcal{T},\mathbb{D}}(L)$ is interesting because it is smaller than the natural interpretation $\langle L \rangle^{\mathbb{D}}$. Indeed, while $\langle L \rangle^{\mathbb{D}}$ may have up to $O(|\mathbb{D}|^{\mathsf{AGM}(L)})$ variables and $O(|L|)$ constraints (see Section 2.2.4), one can show that if k is the width of $\mathcal{T}$ then the size of $\rho^{\mathcal{T},\mathbb{D}}(L)$ is $O(|\mathbb{D}|^k)$ in the data complexity model (where L is considered constant). It follows from the following, more precise, combined complexity analysis:

Theorem 4.9.
Given $L \in LP(CQ_{qf})_{proj}$, its tree decomposition $\mathcal{T}$ of width k and a database $\mathbb{D}$, Let q be the sum of the sizes of the queries in $cq(L)$.

The T-factorized interpretation $\rho^{\mathcal{T},\mathbb{D}}(L)$ can be encoded in $O((|L| + |\mathcal{T}| \cdot |\mathbb{D}|^k) \cdot \log(|\mathbb{D}|^k))$ bits and it can be computed in time $O(|L| + q|\mathcal{T}| \cdot |\mathbb{D}|^k \log |\mathbb{D}|)$.

Proof. This is a direct consequence of applying Lemma 4.3 and Lemma 4.7 to each query in $cq(L)$ as well as Lemma 4.5. □

We will show in Section 4.3 that the T-factorized interpretation can indeed be used to efficiently solve closed $LP(CQ_{qf})_{proj}$ models.

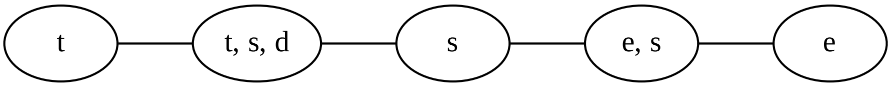
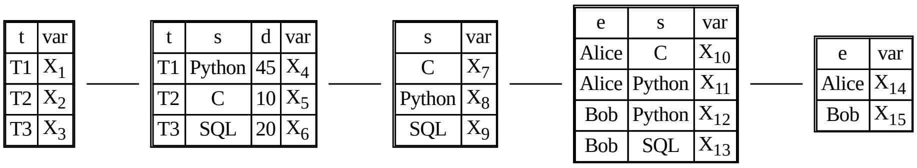


Figure 4.1: Tree decomposition for the task assignment $LP(CQ)$ model



t	var
T1	X_1
T2	X_2
T3	X_3

t	s	d	var
T1	Python	45	X_4
T2	C	10	X_5
T3	SQL	20	X_6

s	var
C	X_7
Python	X_8
SQL	X_9

e	s	var
Alice	C	X_{10}
Alice	Python	X_{11}
Bob	Python	X_{12}
Bob	SQL	X_{13}

e	var
Alice	X_{14}
Bob	X_{15}

Figure 4.2: Variables for the factorized interpretation of L.

4.2.3 Example

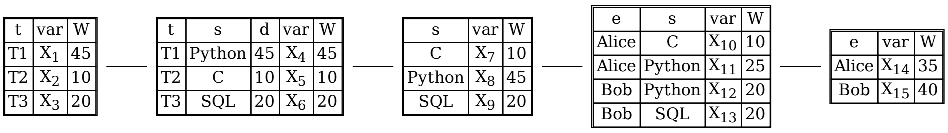
We now demonstrate the T-factorized interpretation on the task assignment problem from Chapter 3. We consider the closed $LP(CQ)$ model L from Example 3.5 and we give its tree decomposition in Figure 4.1. Observe that it is indeed a decomposition of L as the three middle nodes are a normalized[3] decomposition of the query $Q_{assign} = Tasks(t, s, d) \wedge Skills(e, s)$ and the additional t and e bags cover all the **weight** operators of L.

Now, in order to compute the factorized instantiation of L and its soundness constraints, we first need to define its variables. For brevity we use shortened names detailed in Figure 4.2.

Applying the factorized instantiation to the model yields the following:

$$\begin{aligned} \textbf{minimize } & X_{14} + X_{15} \\ \textbf{subject to } & X_1 \geq 45 \\ & \wedge\ X_2 \geq 10 \\ & \wedge\ X_3 \geq 20 \\ & \wedge\ X_{14} \leq 40 \\ & \wedge\ X_{15} \leq 40 \end{aligned}$$

[3] For brevity we consider a relaxation of normalized tree decompositions where we allow the extension and projection of multiple variables at once under which our result is still correct.



t	var	W
T1	X_1	45
T2	X_2	10
T3	X_3	20

t	s	d	var	W
T1	Python	45	X_4	45
T2	C	10	X_5	10
T3	SQL	20	X_6	20

s	var	W
C	X_7	10
Python	X_8	45
SQL	X_9	20

e	s	var	W
Alice	C	X_{10}	10
Alice	Python	X_{11}	25
Bob	Python	X_{12}	20
Bob	SQL	X_{13}	20

e	var	W
Alice	X_{14}	35
Bob	X_{15}	40

Figure 4.3: Solution W of the factorized interpretation of L.

We can then construct the local soundness constraints on the decomposition tree as follows:

$$
\begin{aligned}
& X_1 = X_4 \land X_2 = X_5 \land X_3 = X_6 \\
\land\ & X_5 = X_7 \land X_4 = X_8 \land X_6 = X_9 \\
\land\ & X_7 = X_{10} \land X_8 = X_{11} + X_{12} \land X_9 = X_{13} \\
\land\ & X_{10} + X_{11} = X_{14} \land X_{12} + X_{13} = X_{15}
\end{aligned}
$$

Finally this yields the following linear program:

$$
\begin{aligned}
\textbf{minimize}\ & X_{14} + X_{15} \\
\textbf{subject to}\ & X_1 \geq 45 \\
& \land X_2 \geq 10 \\
& \land X_3 \geq 20 \\
& \land X_{14} \leq 40 \\
& \land X_{15} \leq 40 \\
& \land X_1 = X_4 \quad \land \quad X_2 = X_5 \quad \land \quad X_3 = X_6 \\
& \land X_5 = X_7 \quad \land \quad X_4 = X_8 \quad \land \quad X_6 = X_9 \\
& \land X_7 = X_{10} \quad \land \quad X_8 = X_{11} + X_{12} \quad \land \quad X_9 = X_{13} \\
& \land X_{10} + X_{11} = X_{14} \quad \land \quad X_{12} + X_{13} = X_{15}
\end{aligned}
$$

This program has an optimal value of 75, which is the same value we obtained in Section 3.1, with a weighting W given in Figure 4.3.

In the next section we will formally show that the T-factorized can be used to solve closed $LP(CQ)$ models.

4.3 Solving $LP(CQ)$ models efficiently

4.3.1 Computing the optimal value of a $LP(CQ)$ model

The minimum expectation when solving a linear program is to get its optimal value. Thanks to the following theorem we can do so directly by solving the T-factorized interpretation of a closed $LP(CQ_{qf})_{proj}$ model which has the same optimal value as its natural interpretation.

The correctness of this approach is expressed by the following theorem:

Theorem 4.10 (Equivalence of factorized interpretation and natural semantics)**.** *Let L be a $LP(CQ_{qf})_{proj}$ program, $\mathcal{T}$ a decomposition of L of width k and $\mathbb{D}$ a database.*

The T-factorized interpretation $\rho^{\mathcal{T},\mathbb{D}}(L)$ has the same optimal value as its natural semantics $\langle L\rangle^{\mathbb{D}}$.

Moreover $opt(\rho^{\mathcal{T},\mathbb{D}}(L))$ can be computed in $O(|\rho^{\mathcal{T},\mathbb{D}}(L)|_b \cdot |\mathcal{T}| \cdot |\mathbb{D}|^{k\cdot\ell})$ where $|\rho^{\mathcal{T},\mathbb{D}}(L)|_b = O((|L| + |\mathcal{T}| \cdot |\mathbb{D}|^k) \cdot \log(|\mathbb{D}|^k))$ in combined complexity which corresponds to $O(|\mathbb{D}|^{k\cdot(\ell+1)} \cdot \log(\mathbb{D}))$ in data complexity.

The complexity of solving $\rho^{\mathcal{T},\mathbb{D}}(L)$ follows from Theorem 4.9 and Theorem 2.8. Observe that it does indeed improve on the complexity stated in Section 3.5.1.2.

We will prove the correctness of this theorem in Section 4.4 using the framework introduced in Section 3.6.2. We give an overview of this proof here. Recall that in order to use Proposition 3.17 we fix query Q and need to reconstruct a weighting w_1^Q of the variables of the natural interpretation from a weighting w_2^Q of the variables of the T-factorized interpretation and vice-versa. The key point of these constructions is that the following correspondence must hold:

$$\forall \mathbf{w} \in \mathbf{W}_Q(L). w_1^Q(\mathrm{Inst}_{\mathbb{D}}(\mathbf{w})) = w_2^Q(\mathrm{FInst}_{\mathcal{T},\mathbb{D}}(\mathbf{w})).$$

Recall that we only consider $LP(CQ_{qf})_{proj}$ models whose **weight** operators are of the form $\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{d}}(Q)$. The factorized instantiation of such a weight is a variable $\xi_{Q,u,\beta}$ with $\beta = [\mathbf{x}'/\mathbf{d}]$. The natural instantiation of this weight is the sum $\sum_{\alpha\in ans^{\mathbb{D}}(Q\wedge\mathbf{x}'=\mathbf{d})} \theta_Q^\alpha$ which can be alternatively described as $\sum_{\substack{\alpha\in ans^{\mathbb{D}}(Q)\\ \alpha_{|\mathbf{x}'}=\beta}} \theta_Q^\alpha$.

In order to simplify the constructions and proofs, we further decompose w_1^Q into a family of weightings for each node u in the tree decomposition T of Q:

Definition 4.11.
A family $W = (W_v)_{v\ \in\mathcal{V}}$ is a weighting collection on T of *$ans^{\mathbb{D}}(Q)$ if it satisfies the following conditions for any edge $(u,v) \in \mathcal{E}$:*

- **-** W_u **is a weighting of** $ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$**,** *i.e.,* $W_u : ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)} \to \mathbb{R}^+$.

- **-** W_u **is sound for** T **at** (u,v)**,** *i.e.,* $\pi_{\mathcal{B}^{uv}}(W_u) = \pi_{\mathcal{B}^{uv}}(W_v)$.

Observe that the soundness property of a TWD of $ans^{\mathbb{D}}(Q)$ corresponds to satisfying the soundness constraints $lsc(Q)$ which will allow us to seamlessly convert it to a weighting w_1^Q of $\Xi_{L,Q}$ that satisfies $lsc(Q)$ in the later parts of the proof.

Given a query Q, the value assigned by W for a given bag u and tuple β is simply $W_u(\beta) = w_2^Q(\xi_{Q,u,\beta})$. Similarly for the natural interpretation we consider weightings $\omega : ans^{\mathbb{D}}(Q) \to \mathbb{R}^+$ with $\omega(\alpha) = w_1^Q(\theta_Q^\alpha)$. Observe that the value of the natural instantiation is now $\sum_{\substack{\alpha \in ans^{\mathbb{D}}(Q) \\ \alpha_{|\mathbf{x}'} = \beta}} \omega(\alpha) = \pi_{\mathcal{B}(u)}(\omega)(\beta)$. Thus the correspondence we need to upkeep while reconstructing weightings back and forth is

$$\pi_{\mathcal{B}(u)}(\omega)(\beta) = W_u(\beta).$$

Observe that this correspondence gives us an obvious way to reconstruct a weighting collection on T W from a weighting ω by projecting it on every bag of u. The inverse construction is a bit more complex however. We present it in the next section as a minor contribution whose uses go further than the proof of Theorem 4.10.

4.3.2 Computing a full solution of the natural interpretation

While computing the T-factorized interpretation of a closed $LP(CQ_{qf})_{proj}$ model yields a smaller linear program with the same optimal value as its natural semantics, this comes at the cost of losing some information on the weights of the individual θ_Q^τ variables of the natural semantics. However one might still be interested in the values of these individual variables. Fortunately, as we mentioned before, we have to reconstruct a weighting of the variables of the natural semantics of L given a solution of its T-factorized interpretation as part of the proof of its correctness.

For simplicity we construct weightings restricted to a single query that we can then recombine into a full weighting of the interpretation.

Definition 4.12.
Let Q be a conjunctive query, $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a normalized decomposition tree of $fv(Q)$ and $W = (W_u)_{u \in \mathcal{V}}$ be a weighting collection on T for $ans^{\mathbb{D}}(Q)$. Let r be the root of T.

We construct a weighting $\amalg(W) : \Theta_Q^{\mathbb{D}} \to \mathbb{R}^+$ such that $\amalg(W)(\theta_Q^\alpha) = \omega_r(\alpha)$ for any $\theta_Q^\alpha \in \Theta_Q^{\mathbb{D}}$ where $\omega_u : ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow u)} \to \mathbb{R}^+$ is constructed through bottom-up induction on the nodes $u \in \mathcal{V}$:

***If** u **is a leaf of** T, we define ω_u such that for all $\alpha \in ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow u)}$, $\omega_u(\alpha) := W_u(\alpha)$.*

Now, assume $\omega_{u'}$ is defined for all children u' of u. Let $\alpha \in ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow u)}$ and $\beta = \alpha_{|\mathcal{B}(u)}$.

We define $\omega_u(\alpha)$ as follows:

***If** u **is an extend node** with a single child v then $\omega_u(\alpha) = \frac{W_u(\beta)}{W_v(\alpha_{|\mathcal{B}(v)})}\omega_v(\alpha_{|\mathsf{Attr}(\downarrow v)})$ if $W_v(\alpha_{|\mathcal{B}(v)}) > 0$ and $\omega_u(\alpha) = 0$ otherwise.*

If *u* ***is a project node*** *with a single child v then $\omega_u(\alpha) = \omega_v(\alpha)$*[4]*.*

If *u* ***is a join node*** *with children $v_1, \ldots, v_k$ then $\omega_u(\alpha) = \frac{\prod_{i=1}^{k} \omega_{v_i}(\alpha_{|\mathsf{Attr}(\downarrow v_i)})}{W_u(\beta)^{k-1}}$ if $W_u(\beta) > 0$ and $\omega_u(\alpha) = 0$ otherwise.*

Finally, given a solution w_2 of L_{fact}, we obtain a solution w_1 of L_{nat} by simply defining $w_1(\theta_Q^\alpha) = \amalg(W^Q)(\theta_Q^\alpha)$ where W^Q is a weighting collection on T of $ans^{\mathbb{D}}(Q)$ with $W_u^Q(\beta) = w_2(\xi_{Q,u,\beta})$.

We will prove that this reconstruction is correct in Section 4.3.2 as a simple byproduct of the proof of Theorem 4.10:

Lemma 4.13.
Let L be a $LP(CQ_{qf})_{proj}$ program, $\mathcal{T}$ a decomposition of L of width k and $\mathbb{D}$ a database. Let S be the objective function of L and w_2 be an optimal solution of $\rho^{\mathcal{T},\mathbb{D}}(L)$.

We can construct a solution w_1 of $\langle L \rangle^{\mathbb{D}}$ in $O(|w_1|)$ such that $[\![Inst_{\mathbb{D}}(S)]\!]_{w_1} = [\![FInst_{\mathcal{T},\mathbb{D}}(S)]\!]_{w_2}$.

Observe that given a query Q and node $u \in T_Q$, $|ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow u)}| \leq |ans^{\mathbb{D}}(Q)|$ so we can compute $\amalg(W)$ in $O(|T_Q| \cdot |\mathbb{D}|^{\mathsf{AGM}(Q)})$ and w_1 in $O(|\mathcal{T}| \cdot |\mathbb{D}|^{\mathsf{AGM}(L)})$. While this is slower than simply computing $opt(\rho^{\mathcal{T},\mathbb{D}}(L))$, this is still more efficient than explicitly solving $\langle L \rangle^{\mathbb{D}}$ in $O((|L| \cdot |\mathbb{D}|^{\mathsf{AGM}(L)})^{\ell})$.

Moreover, if we sort the tuples in $(ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)})_{u \in \mathcal{V}}$, then w_2 can be seen as a succinct encoding of w_1 that allows one to compute a specific $w_1(\theta_Q^\theta)$ in $O(|T_Q| \cdot \log(|\mathbb{D}|^k))$. This could be useful in applications where one might want to provide users with access to the value of a few variables θ_Q^θ without materializing the whole solution at once.

Example We illustrate this process using the example from Section 4.2.3 and the solution W of Figure 4.3 in particular.

We will compute the value taken by the answer $[T_1, Python, Alice, 45]$[5]of $Q = Q_{assign}$.

We denote each node of the decomposition of Q by the variables of its bag and we consider the left-most node t in Figure 4.1 to be the root of the decomposition.

We begin with the leaf e where we have $\omega_e([Alice]) = W_e([Alice]) = 35$.

We then have $\omega_{es}([Alice, Python]) = \frac{W_{es}([Alice,Python])}{W_e([Alice])} \omega_e([Alice]) = \frac{25}{35} \cdot 35 = 25$ since es is an extend node.

At the project node s we simply have $\omega_s([Alice, Python]) = \omega_{es}([Alice, Python]) = 25$.

The next node tsd is an extend node so we have $\omega_{tsd}([T_1, Python, Alice, 45]) = \frac{W_{tsd}([T_1,Python,45])}{W_s([Python])} \omega_s([Alice, Python]) = \frac{45}{45} \cdot 25 = 25$

[4] Observe that in this case $\mathsf{Attr}(\downarrow u) = \mathsf{Attr}(\downarrow v)$.

[5] We omit the variables for brevity since the values are not ambiguous.

Finally the root s is a project node so we simply obtain $\omega_s([T_1, Python, Alice, 45]) = \omega_{tsd}([T_1, Python, Alice, 45]) = 25$. Thus we know that Alice should be assigned to work 25 hours on the task T_1.

4.3.3 Handling conjunctive queries with existential quantifiers

The previous method of factorized interpretation only works for the $LP(CQ_{qf})_{proj}$ fragment, where conjunctive queries are supposed to be quantifier free. It turns out that one can similarly solve linear programs of $LP(CQ)_{proj}$ programs by applying a simple transformation.

For any $LP(CQ)_{proj}$ program L we can move the existential quantifiers of the conjunctive query into the weight expression as follows, yielding an $LP(CQ_{qf})_{proj}$ program $mvq(L)$: we replace every subexpression $\mathbf{weight}_{\mathbf{x}:Q'}(\exists\mathbf{z}.Q)$ of L, where Q is quantifier free, by $\mathbf{weight}_{\mathbf{xz}:Q'}(Q)$ where $\mathbf{xz}$ is the concatenation of vectors $\mathbf{x}$ and $\mathbf{z}$. We have:

Theorem 4.14 (Removing Existential Quantifiers).
For any projecting $LP(CQ)$ program, the $LP(CQ_{qf})_{proj}$ program $mvq(L)$ has the same optimal value as L.

Proof (Sketch). It is clear that every Q appearing in a subexpression $\mathbf{weight}_{\mathbf{xz}:Q'}(Q)$ of $mvq(L)$ is quantifier free by definition. Now, since L is in $LP(CQ)_{proj}$, Q' is of the form $\mathbf{x}' = \mathbf{y}$ where $\mathbf{x}'$ only contains free variables of $\exists\mathbf{z}.Q$. Since $fv(\exists\mathbf{z}.Q) \subseteq fv(Q)$, we have that $\mathbf{x}'$ only contains free variables of Q. Moreover, the other condition of $LP(CQ)_{proj}$ concerning the sum and universal quantification are still respected in $mvq(L)$, thus L is in $LP(CQ_{qf})_{proj}$.

Now, let $w_1 : \Theta_L^{\mathbb{D}} \to \mathbb{R}^+$ be a solution of L. We define $w_1' : \Theta_{mvq(L)}^{\mathbb{D}} \to \mathbb{R}^+$ as follows: $w_1'(\theta_Q^\alpha) = \frac{1}{N} w_1(\theta_{\exists\mathbf{z}.Q}^{\alpha|U})$ where $U = fv(\exists\mathbf{z}.Q)$ and $N = \#\{\beta : \mathbf{z} \to dom \mid \alpha \cup \beta \in ans^{\mathbb{D}}(Q)\}$. It is readily verified that the value $\llbracket \mathbf{weight}_{\mathbf{x}:Q'}(\exists\mathbf{z}.Q) \rrbracket_{w_1}$ is the same as $\llbracket \mathbf{weight}_{\mathbf{x},\mathbf{z}:Q'}(Q) \rrbracket_{w_1'}$ and thus that w_1' is a solution of $mvq(L)$ and $\llbracket L \rrbracket_{w_1} = \llbracket mvq(L) \rrbracket_{w_1'}$.

For the other way around, given $w_1' : \Theta_{mvq(L)}^{\mathbb{D}} \to \mathbb{R}^+$ a solution of $mvq(L)$, we construct $w_1 : \Theta_L^{\mathbb{D}} \to \mathbb{R}^+$ as $w_1(\theta_{\exists\mathbf{z}.Q}^\alpha) = \sum_{\beta|\alpha\cup\beta\in ans^{\mathbb{D}}(Q)} w_1'(\theta_Q^{\alpha\cup\beta})$. Again, it is readily verified that the value $\llbracket \mathbf{weight}_{\mathbf{x}:Q'}(\exists\mathbf{z}.Q) \rrbracket_{w_1}$ is the same as $\llbracket \mathbf{weight}_{\mathbf{x},\mathbf{z}:Q'}(Q) \rrbracket_{w_1'}$ and thus that w_1 is a solution of L and $\llbracket L \rrbracket_{w_1} = \llbracket mvq(L) \rrbracket_{w_1'}$. □

4.4 Proof of equivalence between the T-factorized and natural interpretations

4.4.1 Weighting correspondence

We will actually show a slightly more general correspondence by generalizing it from $ans^{\mathbb{D}}(Q)$ to so-called conjunctively decomposed sets of variables assignments.

As we mentioned in Section 4.3.1 we will consider a fixed conjunctively decomposed set A as a stand-in for the answer set of a fixed query Q.

We now define what it means for a set of variable assignments to be conjunctively decomposed. Note that the answer set of a query $ans^{\mathbb{D}}(Q)$ is a conjunctively decomposed set of variable assignments as we will show later in Proposition 4.27.

Definition 4.15.
Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a decomposition tree of $X \subseteq \mathcal{X}$. We call a subset of variable assignments $A \subseteq D^X$ conjunctively decomposed by T *iff for any $u \in \mathcal{V}$ and $\beta \in A_{|\mathcal{B}(u)}$:*

$$\{\alpha_1 \cup \alpha_2 \mid \alpha_1 \in A_{|\mathsf{Attr}(\uparrow u)}[\beta],\ \alpha_2 \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]\} = A[\beta].$$

Finally we fix a set of variable assignments A (which can be seen as fixing a query Q) and reason about weightings $\omega : A \to \mathbb{R}^+$ for the natural interpretation. For the T-factorized interpretation we will use weighting collections of A[6].

The correspondence theorem is then expressed as follows:

Theorem 4.16 (Correspondence)**.**
Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a normalized decomposition tree of $X \subseteq \mathcal{X}$ and $A \subseteq D^X$ be a set of variable assignments that is conjunctively decomposed by T.

1. *For every weighting ω of A, $(\pi_{\mathcal{B}(u)}(\omega))_{u \in \mathcal{V}}$ is a weighting collection on T for A.*
2. *For any weighting collection W on T for A there exists a weighting $\omega = \amalg(W)$ such that $\forall u : W_u = \pi_{\mathcal{B}(u)}(\omega)$.*

We will prove Theorem 4.16 1 in Section 4.4.2 by projecting ω on every bag of T. We will then prove Theorem 4.16 2 in Section 4.4.3 using the reconstruction of Definition 4.12. Finally we will show that Theorem 4.10 follows from Theorem 4.16 and Proposition 3.17 in Section 4.4.4.

4.4.2 Reconstructing a weighting collection on T of A

The first item of the correspondence theorem directly follows from the following proposition.

Proposition 4.17.
Let $A \subseteq D^X$ and let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a decomposition tree for X. For any weigthing $\omega : A \to \mathbb{R}^+$, the family $(\pi_{\mathcal{B}(v)}(\omega))_{v \in \mathcal{V}}$ is a weighting collection on T for A.

Proof. For any $u \in \mathcal{V}$ let $W_u = \pi_{\mathcal{B}(u)}(\omega)$. The first condition on weighting projections holds trivially so we only have to show that the soundness constraint holds. By definition of W_u, $\pi_{\mathcal{B}^{uv}}(W_u) = \pi_{\mathcal{B}^{uv}}(\pi_{\mathcal{B}(u)}(\omega))$. Observe that $\mathcal{B}^{uv} \subseteq \mathcal{B}(u)$ so by Proposition 4.20 $\pi_{\mathcal{B}^{uv}}(W_u) = \pi_{\mathcal{B}^{uv}}(\omega)$. Similarly $\pi_{\mathcal{B}^{uv}}(W_v) = \pi_{\mathcal{B}^{uv}}(\omega)$. □

[6] We generalize the notion from Definition 4.11 to weighting collections of A in a straightforward manner

If T is normalized then the local soundness constraint (4.11) of W at $(u, v) \in \mathcal{E}$ can be rewritten equivalently into a simpler form as follows:

- if u is an extend node with unique child v then: $\forall \beta \in A_{|\mathcal{B}(v)}$: $\sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} W_u(\beta') = W_v(\beta)$,
- if u is a project node with unique child v then $\forall \beta \in A_{|\mathcal{B}(u)}$: $W_u(\beta) = \sum_{\beta' \in A_{|\mathcal{B}(v)}[\beta]} W_v(\beta')$,
- if u is a join node with child v then $\forall \beta \in A_{|\mathcal{B}(u)} : W_u(\beta) = W_v(\beta)$.

4.4.3 Reconstructing a weighting of A

In this section we prove the second item of Theorem 4.16 by reconstructing a weighting of A by Definition 4.12 generalised to A in a straight-forward manner. Observe that this also implies that the reconstruction is indeed correct.

The proof that $\forall u : W_u = \pi_{\mathcal{B}(u)}(\omega)$ is done via two inductions. The first one is a bottom-up induction to prove that $W_u = \pi_{\mathcal{B}(u)}(\omega_u)$ for every node u in the tree decomposition. Then, by top-down induction, one can prove that $\omega_u = \pi_{\mathsf{Attr}(\downarrow u)}(\omega_r)$. The proof is tedious and mainly relies on calculations and careful analysis on how A is decomposed along T. We begin by showing a few lemmas on weightings and set of tuples.

First we show that the extensions of two different tuples are distinct:

Lemma 4.18.
For any two $\alpha_1, \alpha_2 \in A_{|X'}$, if $\alpha_1 \neq \alpha_2$ then $A[\alpha_1] \cap A[\alpha_2] = \emptyset$.

Proof. If $\alpha_1 \neq \alpha_2 \in A_{|X'}$, then there exists $x' \in X'$ such that $\alpha_1(x') \neq \alpha_2(x')$, so if $\gamma_1 \in A[\alpha_1]$ and $\gamma_2 \in A[\alpha_2]$ then $\gamma_1(x') = \alpha_1(x') \neq \alpha_2(x') = \gamma_2(x')$. □

Lemma 4.19.
For $A \subseteq D^X$, $X'' \subseteq X' \subseteq X$, $\alpha'' \in A_{|X''}$:

$$A[\alpha''] = \biguplus_{\alpha' \in A_{|X'}[\alpha'']} A[\alpha'].$$

Proof. First note that the union on the right is disjoint by Lemma 4.18.

For left-to-right inclusion, let $\alpha \in A[\alpha'']$ and $\alpha' = \alpha_{|X'}$. By definition, $\alpha' \in A_{|X'}$ so $\alpha \in A[\alpha']$. Furthermore, $\alpha' \in A_{|X'}[\alpha'']$ so $\alpha \in \biguplus_{\tilde{\alpha}' \in A_{|X'}[\alpha'']} A[\tilde{\alpha}']$.

For right-to-left inclusion, let $\alpha \in \biguplus_{\alpha' \in A_{|X'}[\alpha'']} A[\alpha']$ and let $\alpha' \in A_{|X'}[\alpha'']$ be such that $\alpha \in A[\alpha']$. By definition, $\alpha_{|X'} = \alpha'$ and $\alpha'_{|X''} = \alpha''$. Since $X'' \subseteq X'$, $\alpha_{|X''} = \alpha'_{|X''} = \alpha''$. Thus $\alpha \in A[\alpha'']$. □

Proposition 4.20.
For $A \in D^X$, $\omega : A \to \mathbb{R}^+$, $X'' \subseteq X' \subseteq X$:

$$\pi_{X''}(\omega) = \pi_{X''}(\pi_{X'}(\omega)).$$

Proof. Indeed, let $\alpha'' \in A_{|X''}$. We have:

$$\begin{aligned}
\pi_{X''}(\omega)(\alpha'') &= \sum_{\alpha \in A[\alpha'']} \omega(\alpha) && \text{by definition}\\
&= \sum_{\alpha' \in A_{|X'}[\alpha'']} \sum_{\alpha \in A[\alpha']} \omega(\alpha) && \text{by Lemma 4.19}\\
&= \sum_{\alpha' \in A_{|X'}[\alpha'']} \pi_{X'}(\omega)(\alpha') && \text{by definition of } \pi_{X'}(\omega)\\
&= \pi_{X''}(\pi_{X'}(\omega))(\alpha'') && \text{by definition of } \pi_{X''}(\pi_{X'}(\omega)).
\end{aligned}$$

The last equality is well defined since $\alpha'' \in A_{|X''} = (A_{|X'})_{|X''}$. □

Lemma 4.21.
Let T be a decomposition tree of X, u an extend node of T with child v, and $A \subseteq D^X$ a subset of variable assignments. If A is conjunctively decomposed by T then any assignment $\beta \in A_{|\mathcal{B}(u)}$ satisfies:

$$A_{|\mathsf{Attr}(\downarrow u)}[\beta]_{|\mathsf{Attr}(\downarrow v)} = A_{|\mathsf{Attr}(\downarrow v)}[\beta_{|\mathcal{B}(v)}]$$

Proof. For left-to-right inclusion, let $\alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]_{|\mathsf{Attr}(\downarrow v)}$. Since $\alpha \in A_{|\mathsf{Attr}(\downarrow v)}$ and $\alpha_{|\mathcal{B}(v)} = \beta_{|\mathcal{B}(v)}$ it follows that $\alpha \in A_{|\mathsf{Attr}(\downarrow v)}[\beta_{|\mathcal{B}(v)}]$.

For right-to-left inclusion, let $\alpha \in A_{|\mathsf{Attr}(\downarrow v)}[\beta_{|\mathcal{B}(v)}]$. Let $\gamma \in A_{|\mathsf{Attr}(\uparrow v)}[\beta]$ be arbitrary and $\tau = \gamma \cup \alpha$.

Note that $(\tau_{|\mathsf{Attr}(\downarrow u)})_{|\mathsf{Attr}(\downarrow v)} = \alpha$, so it is sufficient to show $\tau_{|\mathsf{Attr}(\downarrow u)} \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]$.

Since u is an extend node with child v it follows that $\mathsf{Attr}(\uparrow u) = \mathsf{Attr}(\uparrow v)$, and thus $\gamma \in A_{|\mathsf{Attr}(\uparrow v)}[\beta]$. By conjunctive decomposition of A by T it follows that $\tau \in A[\beta]$. Hence, $\tau_{|\mathsf{Attr}(\downarrow u)} \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]$ as required. □

Lemma 4.22.
Let T be a decomposition tree of X, u a join node of T with children $v_1, \dots, v_k$ where $k \geq 1$, and $A \subseteq D^X$ a subset of variable assignments. If A is conjunctively decomposed by T then any $\beta \in A_{|\mathcal{B}(u)}$ satisfies:

$$A_{|\mathsf{Attr}(\downarrow u)}[\beta] = A_{|\mathsf{Attr}(\downarrow v_1)}[\beta] \bowtie \dots \bowtie A_{|\mathsf{Attr}(\downarrow v_k)}[\beta]$$

Proof. The inclusion from the left to the right is obvious by projecting an element of $A_{|\mathsf{Attr}(\downarrow u)}[\beta]$ to $\mathsf{Attr}(\downarrow v_1) \dots \mathsf{Attr}(\downarrow v_k)$.

For the inclusion from the right to the left let $\alpha_1 \in A_{|\mathsf{Attr}(\downarrow v_1)}[\beta], \dots \alpha_k \in A_{|\mathsf{Attr}(\downarrow v_k)}[\beta]$. We show by induction that $\forall p \in [1, k]$, $\tau_p = \alpha_1 \bowtie \dots \bowtie \alpha_p \in A_{|Y_p}[\beta]$ where $Y_p = \bigcup_{i=1}^{p} \mathsf{Attr}(\downarrow v_i)$.

Base case $p = 1$: Obvious.

Inductive case: Recall that by induction $\tau_p \in A_{|Y_p}[\beta]$ and observe that $Y_p \subseteq \mathsf{Attr}(\uparrow v_{p+1})$ so there exists $\gamma \in \mathsf{Attr}(\uparrow v_{p+1})[\beta]$ such that $\gamma_{|Y_p} = \tau_p$.

By conjunctive decomposition on v_{p+1}, $\alpha = \gamma \bowtie \alpha_{p+1} \in A$. Finally we have $\alpha_{|Y_{p+1}} = (\gamma \bowtie \alpha_{p+1})_{|Y_p \cup \mathsf{Attr}(\downarrow v_{p+1})} = \gamma_{|Y_p} \bowtie \alpha_{p+1|\mathsf{Attr}(\downarrow v_{p+1})} = \tau_p \bowtie \alpha_{p+1} = \tau_{p+1}$ so $\tau_{p+1} \in A_{|Y_{p+1}}$. Thus $\tau_{p+1} \in Y_p[\beta]$ because $\tau_{p+1|\mathcal{B}(u)} = \beta$.

□

Proposition 4.23.
For all $u \in \mathcal{V}$, $W_u = \pi_{\mathcal{B}(u)}(\omega_u)$.

Proof. We show by bottom-up induction on the nodes of T that for all $u \in \mathcal{V}$ and $\beta \in A_{|\mathcal{B}(u)}$, $\sum_{\alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]} \omega_u(\alpha) = W_u(\beta)$.

The base case is clearly true by the definition of ω_u when u is a leaf.

Case 1 u is an extend node with v its only child.

Let $\beta \in A_{|\mathcal{B}(u)}$ and $\beta' = \beta_{|\mathcal{B}(v)}$.

Case 1.1 $W_v(\beta') = 0$.

By definition $\forall \alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta], \omega_u(\alpha) = 0$.

Recall that by soundness $\sum_{\beta'' \in A_{|\mathcal{B}(u)}[\beta']} W_u(\beta'') = W_v(\beta') = 0$. Observe that $\beta \in A_{|\mathcal{B}(u)}[\beta']$ so $W_u(\beta) = 0 = \sum_{\alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]} \omega_u(\alpha)$.

Case 1.2 $W_v(\beta') > 0$.

$$\begin{aligned}
&\textstyle\sum_{\alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]} \omega_u(\alpha) \\
&\quad = \textstyle\sum_{\alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]} \frac{W_u(\beta)}{W_v(\beta')} \omega_v(\alpha_{|\mathsf{Attr}(\downarrow v)}) && \text{by definition} \\
&\quad = \textstyle\frac{W_u(\beta)}{W_v(\beta')} \sum_{\alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]} \omega_v(\alpha_{|\mathsf{Attr}(\downarrow v)}) \\
&\quad = \textstyle\frac{W_u(\beta)}{W_v(\beta')} \sum_{\alpha' \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]_{|\mathsf{Attr}(\downarrow v)}} \omega_v(\alpha') \\
&\qquad \text{because every } \alpha \text{ is equal to } \beta(x) \bowtie \alpha_{|\mathsf{Attr}(\downarrow v)} \text{ with } x = \mathcal{B}(u) \setminus \mathcal{B}(v) \\
&\quad = \textstyle\frac{W_u(\beta)}{W_v(\beta')} \sum_{\alpha' \in A_{|\mathsf{Attr}(\downarrow v)}[\beta']} \omega_v(\alpha') && \text{by Lemma 4.21} \\
&\quad = \textstyle\frac{W_u(\beta)}{W_v(\beta')} W_v(\beta') && \text{by induction} \\
&\quad = W_u(\beta)
\end{aligned}$$

Case 2 u is a project node with only child v.

$$\begin{aligned}
&\textstyle\sum_{\alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]} \omega_u(\alpha) \\
&\quad = \textstyle\sum_{\alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]} \omega_u(\alpha) && \text{by definition} \\
&\quad = \textstyle\sum_{\beta' \in A_{|\mathcal{B}(v)}[\beta]} \sum_{\alpha' \in A_{|\mathsf{Attr}(\downarrow u)}[\beta']} \omega_v(\alpha) && \text{by Proposition 4.20 and } \mathcal{B}(v) \subseteq \mathsf{Attr}(\downarrow u) \\
&\quad = \textstyle\sum_{\beta' \in A_{|\mathcal{B}(v)}[\beta]} W_v(\beta') && \text{by induction and } \mathsf{Attr}(\downarrow u) = \mathsf{Attr}(\downarrow v) \\
&\quad = W_u(\beta) && \text{by soundness at } (u, v)
\end{aligned}$$

Case 3 u is a join node with children $v_1, \ldots, v_k$.

Let $\beta \in A_{|\mathcal{B}(u)}$.

Case 3.1 $W_u(\beta) = 0$.

By definition $\forall \alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta], \omega_u(\alpha) = 0$.

Thus $\sum_{\alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]} \omega_u(\alpha) = 0 = W_u(\beta)$.

Case 3.2 $W_u(\beta) > 0$.

$$\begin{aligned}
&\textstyle\sum_{\alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]} \omega_u(\alpha) \\
&= \textstyle\sum_{\alpha \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]} \frac{\prod_{i=1}^{k} \omega_{v_i}(\alpha_{|\mathsf{Attr}(\downarrow v_i)})}{W_u(\beta)^{k-1}} && \text{by definition} \\
&= \textstyle\sum_{\alpha_1 \in A_{|\mathsf{Attr}(\downarrow v_1)}[\beta]} \cdots \sum_{\alpha_k \in A_{|\mathsf{Attr}(\downarrow v_k)}[\beta]} \frac{\prod_{i=1}^{k} \omega_{v_i}(\alpha_i)}{W_u(\beta)^{k-1}} && \text{by Lemma 4.22} \\
&= \frac{\prod_{i=1}^{k} \sum_{\alpha_i \in A_{|\mathsf{Attr}(\downarrow v_i)}[\beta]} \omega_{v_i}(\alpha_i)}{W_u(\beta)^{k-1}} \\
&= \frac{\prod_{i=1}^{k} W_{v_i}(\beta)}{W_u(\beta)^{k-1}} && \text{by induction} \\
&= \frac{W_u(\beta)^k}{W_u(\beta)^{k-1}} && \text{by soundness at } (u, v_i) \\
&= W_u(\beta)
\end{aligned}$$

□

Lemma 4.24.
Let v be the child of an extend node u, $\forall \alpha \in A_{|\mathsf{Attr}(\downarrow v)}$ with $\beta = \alpha_{|\mathcal{B}(v)}$:

$$A[\alpha] = \biguplus_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} A[\alpha \cup \beta']$$

Proof. For left-to-right inclusion, let $\tau \in A[\alpha]$ and $\beta' = \tau_{|\mathcal{B}(u)}$. Observe that $\beta' \in A_{|\mathcal{B}(u)}[\beta]$. Moreover $\mathsf{Attr}(\downarrow u) = \mathsf{Attr}(\downarrow v) \cup \mathcal{B}(u)$ so $\tau_{|\mathsf{Attr}(\downarrow u)} = \alpha \cup \beta'$ so $\tau \in A[\alpha \cup \beta']$.

For right-to-left inclusion, let $\tau \in \biguplus_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} A[\alpha \cup \beta']$. By definition $\tau \in A$ and $\tau_{|\mathsf{Attr}(\downarrow v)} = \alpha$ so $\tau \in A[\alpha]$. □

Lemma 4.25.
Given a join node u and its children $v_1, \ldots, v_k$, let $\alpha \in A_{|\mathsf{Attr}(\downarrow v_1)}$ and $\beta = \alpha_{|\mathcal{B}(u)}$.

$$A_{|\mathsf{Attr}(\downarrow u)}[\alpha] = \{\alpha\} \bowtie A_{|\mathsf{Attr}(\downarrow v_2)}[\beta] \bowtie \ldots \bowtie A_{|\mathsf{Attr}(\downarrow v_k)}[\beta]$$

Proof. Clearly $A_{|\mathsf{Attr}(\downarrow u)}[\alpha] = \{\tau \in A_{|\mathsf{Attr}(\downarrow u)}[\beta] \mid \tau_{|\mathsf{Attr}(\downarrow v_1)} = \alpha\}$ because $\beta = \alpha_{|\mathcal{B}(u)}$.

Thus by Lemma 4.22, $A_{|\mathsf{Attr}(\downarrow u)}[\alpha] = \{\tau \in A_{|\mathsf{Attr}(\downarrow v_1)}[\beta] \bowtie \ldots \bowtie A_{|\mathsf{Attr}(\downarrow v_k)}[\beta] \mid \tau_{|\mathsf{Attr}(\downarrow u)} = \alpha\} = \{\alpha\} \bowtie A_{|\mathsf{Attr}(\downarrow v_2)}[\beta] \bowtie \ldots \bowtie A_{|\mathsf{Attr}(\downarrow v_k)}[\beta]$ □

Proposition 4.26.
For all $u \in \mathcal{V}$, $\omega_u = \pi_{\mathsf{Attr}(\downarrow u)}(\omega_r)$.

Proof. We show by top-down induction on the nodes of T that for all $v \in \mathcal{V}$ and $\alpha \in A_{|\mathsf{Attr}(\downarrow v)}$, $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \omega_v(\alpha)$.

The base case is clearly true when v is the root r of T.

In the following we consider a given $\alpha \in A_{|\mathsf{Attr}(\downarrow v)}$. and we let $\beta = \alpha_{|\mathcal{B}(v)}$

Case 1 v is the only child of an extend node u.

By Lemma 4.24, $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \sum_{\tau \in A[\alpha \cup \beta']} \omega_r(\tau)$. By induction this is equal to $\sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \omega_u(\alpha \bowtie \beta')$.

Case 1.1 $W_v(\beta) = 0$.

By definition of ω_u, $\sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \omega_u(\alpha \bowtie \beta') = 0$.

Observe that by Proposition 4.23, $\sum_{\alpha' \in A_{|\mathsf{Attr}(\downarrow u)}[\beta]} \omega_v(\alpha') = W_v(\beta) = 0$. However ω_v is non-negative so $\omega_v(\alpha) = 0 = \sum_{\tau \in A[\alpha]} \omega_r(\tau)$.

Case 1.2 $W_v(\beta) > 0$.

$$\begin{aligned}
&\textstyle\sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \omega_u(\alpha \bowtie \beta') \\
&= \textstyle\sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \frac{W_u(\beta')}{W_v(\beta)} \omega_v((\alpha \bowtie \beta')_{|\mathsf{Attr}(\downarrow v)}) && \text{by definition} \\
&= \frac{\sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} W_u(\beta')}{W_v(\beta)} \omega_v(\alpha) \\
&= \omega_v(\alpha) && \text{by soundness at } (u, v)
\end{aligned}$$

Case 2 v is the only child of a project node u.

By induction $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \omega_u(\alpha)$ so it follows by definition of ω_u that $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \omega_v(\alpha)$.

Case 3 v is the child of a join node u.

Let $v_1, \ldots, v_n$ be the children of u, we assume wlog that v is v_1.

By Proposition 4.20, $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \sum_{\alpha' \in A_{|\mathsf{Attr}(\downarrow u)}[\alpha]} \sum_{\tau \in A[\alpha']} \omega_r(\tau)$.

By induction we obtain $\sum_{\alpha' \in A_{|\mathsf{Attr}(\downarrow u)}[\alpha]} \omega_u(\alpha')$.

Case 3.1 $W_u(\beta) = 0$.

By definition of ω_u, $\sum_{\alpha' \in A_{|\mathsf{Attr}(\downarrow u)}[\alpha]} \omega_u(\alpha') = 0$.

Recall that because u is a join node, $W_v(\beta) = W_u(\beta) = 0$ so similarly to Case 1.2, $\omega_v(\alpha) = 0 = \sum_{\tau \in A[\alpha]} \omega_r(\tau)$.

Case 3.2 $W_v(\beta) > 0$.

By definition of ω_u, $\sum_{\alpha' \in A_{|\mathsf{Attr}(\downarrow u)}[\alpha]} \omega_u(\alpha') = \sum_{\alpha' \in A_{|\mathsf{Attr}(\downarrow u)}[\alpha]} \frac{\prod_{i=1}^{k} \omega_{v_i}(\alpha'_{|\mathsf{Attr}(\downarrow v_i)})}{W_u(\beta)^{k-1}}$. Moreover by Lemma 4.25 we can split α' into $\alpha \times \alpha_2 \times \cdots \times \alpha_k$ and the sum into $\sum_{\alpha_2 \in A_{|\mathsf{Attr}(\downarrow v_2)}[\beta]} \cdots \sum_{\alpha_k \in A_{|\mathsf{Attr}(\downarrow v_k)}[\beta]} \frac{\prod_{i=1}^{k} \omega_{v_i}(\alpha'_{|\mathsf{Attr}(\downarrow v_i)})}{W_u(\beta)^{k-1}}$. Observe that each term in the product only depends on α_i (or α for $i = 1$) and that the denominator only depends on the fixed β so we can rewrite the formula into the following $\omega_v(\alpha) \cdot \frac{\prod_{i=2}^{k} \sum_{\alpha_i \in A_{|\mathsf{Attr}(\downarrow v_i)}[\beta]} \omega_{v_i}(\alpha_i)}{W_u(\beta)^{k-1}}$ which is equal to $\omega_v(\alpha) \cdot \frac{\prod_{i=2}^{k} W_{v_i}(\beta)}{W_u(\beta)^{k-1}}$ by Proposition 4.23. Finally observe that by soundness, $\prod_{i=2}^{k} W_{v_i}(\beta) = W_u(\beta)^{k-1}$.

Thus $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \omega_v(\alpha)$.

□

Finally we are ready to prove the second item of Theorem 4.16. We know by Proposition 4.23 that $W_u = \pi_{\mathcal{B}(u)}(\omega_u)$ which is equal to $\pi_{\mathcal{B}(u)}(\pi_{\mathsf{Attr}(\downarrow u)}(\omega_r))$ by Proposition 4.26. Thus $W_u = \pi_{\mathcal{B}(u)}(\omega_r)$ by Proposition 4.20.

4.4.4 Proof of the equivalence of the natural and T-factorized interpetations

Now armed with Theorem 4.16 and Proposition 3.17 we are ready to prove Theorem 4.10. First we need to verify that the answer set of a conjunctive query is indeed conjunctively decomposed by its decomposition T thus allowing us to apply the correspondence theorem.

Proposition 4.27.
For any tree decomposition T of a quantifier free conjunctive query $Q \in CQ_\Sigma$ and database $\mathbb{D} \in db_\Sigma$, the answer set $ans^{\mathbb{D}}(Q)$ is conjunctively decomposed by T.

Proof. Let u be a node of T. Let $R(\mathbf{x})$ be an atom of Q, observe that there exists v in T such that $\mathbf{x} \subseteq \mathcal{B}(v)$ by definition. Thus we either have $\mathbf{x} \subseteq \mathsf{Attr}(\downarrow u)$ or $\mathbf{x} \subseteq \mathsf{Attr}(\uparrow u)$ (or both).

Moreover, recall that $\mathcal{B}(u) = \mathsf{Attr}(\downarrow u) \cap \mathsf{Attr}(\uparrow u)$ by the connectedness of tree decompositions. Let $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$, $\alpha_1 \in ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow u)}[\beta]$ and $\alpha_2 \in ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\uparrow u)}[\beta]$. We have to show that $\alpha = \alpha_1 \cup \alpha_2 \in ans^{\mathbb{D}}(Q)[\beta]$. Clearly, $\alpha_{|\mathcal{B}(u)} = \beta$ by construction so it remains to show that $\alpha \in ans^{\mathbb{D}}(Q)$. To do so, we fix an atom $R(\mathbf{x})$ of Q and suppose by our earlier observation that $\mathbf{x} \in \mathsf{Attr}(\downarrow u)$ (the other case is symmetric). Observe that by definition there is some $\alpha' \in ans^{\mathbb{D}}(Q)$ such that $\alpha_1 = \alpha'_{|\mathsf{Attr}(\downarrow u)}$ so α_1 and by extensions α satisfies $R(\mathbf{x})$. Thus it follows that α satisfies every atom of Q so $\alpha \in ans^{\mathbb{D}}(Q)$.

□

Proposition 4.27 does not hold when Q is not quantifier free thus the equivalence theorem is restricted to the fragment $LP(CQ_{qf})_{proj}$. We however explained how to extend it to $LP(CQ)_{proj}$ in Section 4.3.3.

We are now ready to prove Theorem 4.10.

Let $L_1 = \langle L \rangle^{\mathbb{D}} = \langle L \rangle^{\mathrm{Inst}_{\mathbb{D}}, C_1}$ with $C_1 = \bigwedge_{Q \in cq(L)} true$. Let $L_2 = \rho^{\mathcal{T},\mathbb{D}}(L) = \langle L \rangle^{\mathrm{FInst}_{\mathcal{T},\mathbb{D}}, C_2}$ with $C_2 = \bigwedge_{Q \in cq(L)} lsc^{\mathcal{T},\mathbb{D}}(Q)$.

Fix a query $Q \in cq(L)$.

For any weighting w_1^Q of $var_Q(L_1)$ (that obviously satisfies *true*), we define a weighting ω of $ans^{\mathbb{D}}(Q)$ such that for any $\tau \in ans^{\mathbb{D}}(Q)$, $\omega(\tau) = w_1^Q(\theta_Q^\tau)$. By Proposition 4.27 and Theorem 4.16, $(\pi_{\mathcal{B}(u)}(\omega))_{u \in \mathcal{V}}$ is sound. Thus the weighting w_2^Q defined as $w_2^Q(\xi_{Q,u,\beta}) = \pi_{\mathcal{B}(u)}(\omega)(\beta)$ for any $u \in T_Q$ and $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ satisfies $lsc^{\mathcal{T},\mathbb{D}}(Q)$. Moreover by definition it follows that for any $\mathbf{w} \in \mathbf{W}_Q(L)$, $[\![\mathrm{FInst}_{\mathcal{T},\mathbb{D}}(\mathbf{w})]\!]_{w_2^Q} = w_2^Q(\xi_{Q,u,\beta}) = \pi_{\mathcal{B}(u)}(\omega)(\beta) = \sum_{\alpha \in ans^{\mathbb{D}}(Q)[\beta]} \omega(\alpha) = \sum_{\alpha \in ans^{\mathbb{D}}(Q)[\beta]} w_1^Q(\theta_Q^\alpha) = [\![\mathrm{Inst}_{\mathbb{D}}(\mathbf{w})]\!]_{w_1^Q}$ Thus by Proposition 3.17, $opt(L_1) \leq opt(L_2)$.

For any weighting w_2^Q of $var_Q(L_2)$ that satisfies $lsc^{\mathcal{T},\mathbb{D}}(Q)$ we define a weighting $W = (W_v)_{v\ \in \mathcal{V}}$ such that for any $u \in T_Q$ and $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$, $W_u(\beta) = w_2^Q(\xi_{Q,u,\beta})$. Observe that W is sound so by Theorem 4.16 there exists a weighting $\omega : ans^{\mathbb{D}}(Q) \rightarrow \mathbb{R}^+$ such that $W = (\pi_{\mathcal{B}(u)}(\omega))_{u \in \mathcal{V}}$. We define a weighting w_1^Q such that $w_1^Q(\theta_Q^\tau) = \omega(\tau)$ for any $\tau \in ans^{\mathbb{D}}(Q)$ that obviously satisfies *true*. Similarly to the above it follows that for any $\mathbf{w} \in \mathbf{W}_Q(L)$, $[\![\mathrm{Inst}_{\mathbb{D}}(\mathbf{w})]\!]_{w_1^Q} = [\![\mathrm{FInst}_{\mathcal{T},\mathbb{D}}(\mathbf{w})]\!]_{w_2^Q}$. Thus by Proposition 3.17, $opt(L_2) \leq opt(L_1)$. Additionally Lemma 4.13 holds by Lemma 3.15.

Finally $opt(L_1) = opt(L_2)$ and Theorem 4.10 holds.

4.5 Conclusion

In this chapter we have characterized the fragment of closed $LP(CQ)_{proj}$ models and lifted the notion of fractional hypertree width to these models. We then defined a more succinct interpretation of $LP(CQ)_{proj}$ models called the T-factorized interpretation. Finally we have showed that the T-factorized interpretation can be used to efficiently compute the optimal value and an optimal solution of $LP(CQ)_{proj}$ models.

In Figure 4.4, we give a summary of the solving process with both interpretations as well as their compared performances in data complexity by considering $|L|$ and $|\mathcal{T}|$ to be constants. On the left-hand side we sum up how a closed $LP(CQ)$ model can be naturally interpreted to a linear prograL L_n which can be solved to obtain its optimal value and an optimal solution w_n. On the right-hand side we represent the T-factorized interpretation of L to L_t using a collection of tree decompositions $\mathcal{T}$ of width k that then yields the optimal value and an optimal solution w_t of L_t from which we can reconstruct an optimal solution w_n of L_n.

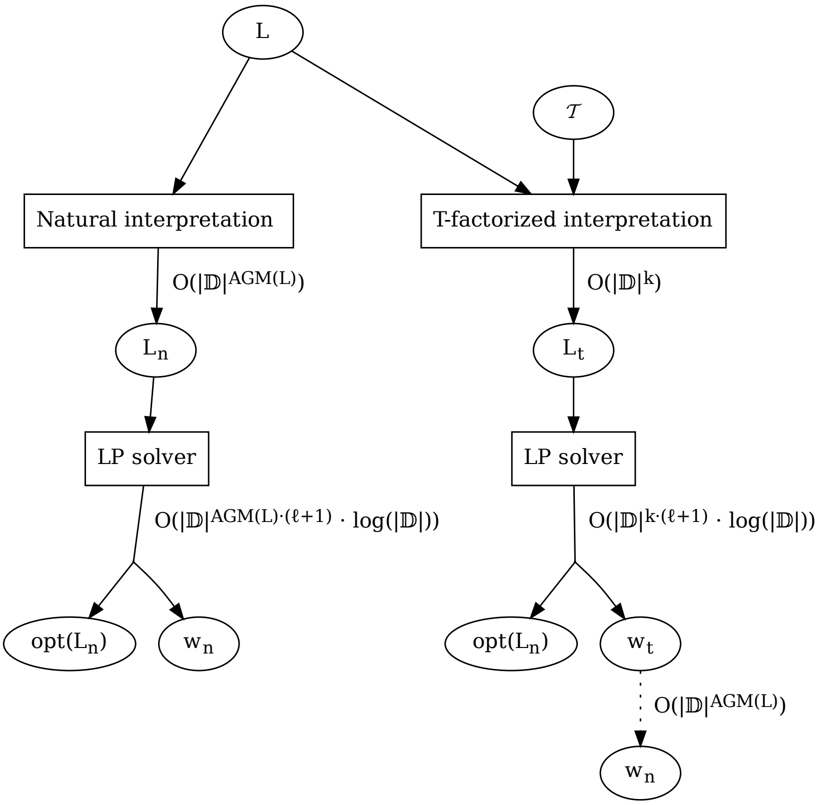

L
$\mathcal{T}$
Natural interpretation
T-factorized interpretation
$O(|\mathbb{D}|^{AGM(L)})$
$O(|\mathbb{D}|^{k})$
L_n
L_t
LP solver
LP solver
$O(|\mathbb{D}|^{AGM(L)\cdot(\ell+1)} \cdot \log(|\mathbb{D}|))$
$O(|\mathbb{D}|^{k\cdot(\ell+1)} \cdot \log(|\mathbb{D}|))$
$opt(L_n)$
w_n
$opt(L_n)$
w_t
$O(|\mathbb{D}|^{AGM(L)})$
w_n

Figure 4.4: Summary and data complexity of the solving process of a closed $LP(CQ)$ model L

Chapter 5

Linear programs on relational circuits

Contents

5.1 Introduction

In this chapter we introduce the circuit-based factorized interpretation (or C-factorized interpretation for short) as a more general alternate interpretation to solve $LP(CQ)$ models efficiently.

This interpretation is based on so-called $\{\uplus, \times\}$-circuits that succinctly encode relations with additional decomposability and determinism properties that we will define in this chapter. In our case we will consider circuits that encode the relations represented by the queries of a $LP(CQ)$ model.

This approach is very similar to the T-factorized interpretation of the previous chapter and can be summed up as instantiating to a smaller set of variables then restoring the semantical link between those variables with some soundness constraints. In this case we reduce the number of variables by exploiting the succinctness of the $\{\uplus, \times\}$-circuits and using the edges of the circuits as the variables of the C-factorized interpretation. We then define soundness constraints using the structure of the gates of the circuits.

We will then show that these similarities are not a coincindence by explicitly recapturing the result from the previous chapter with $\{\uplus, \times\}$-circuits by showing that the answer sets of the queries of any $LP(CQ)_{proj}$ model can be compiled to a $\{\uplus, \times\}$-circuit. Recall that in the previous chapter we took advantage of the oft-used fact that, given a query Q, it was possible to efficiently compute the projection of its answer set on the bags of its decomposition then reconstruct the full answer set through a bottom-up induction on the decomposition. In this case the compiled circuit acts both as a succinct encoding of the answers of Q and as a materialization of the generic bottom-up induction that is at the core of every algorithm that leverages the strucuture of tree decompositions to efficiently reason about conjunctive queries.

The relation between the T-factorized and C-factorized interpretations follows from a well-known connection between bounded width queries and the existence of small and tractable datastructures representing the answers of said queries on a given database.

It has indeed been proven in [OZ12, OZ15b] that the answer set of a conjunctive query Q on a database $\mathbb{D}$ can be represented as a *d-representation* of size $O(|\mathbb{D}|^{fhtw(Q)})$, where d-representations is a restricted family of the $\{\uplus, \times\}$-circuits we use in this thesis. Such circuits are tractable in the sense that many aggregation operations on the relation they represent can be solved in polynomial time in the size of the circuits. For example, given a $\{\uplus, \times\}$-circuit G, one can compute the size of the relation it represents in time linear in the size of G[1]. The well-known results by Skritek and Pichler [PS13] on the tractability of counting the number of answers of a conjunctive query Q when its width is bounded can be recovered by computing a $\{\uplus, \times\}$-circuit C representing $ans^{\mathbb{D}}(Q)$ then using the aforementioned algorithm on C. Many more known tractability results can be recovered with this approach, such as enumerating the answers of a query with a linear time preprocessing and a constant delay [BDG07]. This approach also led to the discovery of new interesting tractable operations on conjunctive queries such as efficiently performing linear regression [SOC16].

Using circuits (or factorized representations) is akin to the approach used in the area of knowledge compilation, where one is interested in transforming – in an offline phase – an intractable input into a more tractable representation. Traditionally, knowledge compilation has been mostly oriented toward compiling knowledge bases encoded as *CNFs* and has thus been more focused on data structures in the Boolean

[1] See the end of Section 5.2.2 for a short description of this algorithm.

domain. The relation between relational circuits and these data structures have been made formal in [Olt16], where it is shown that $\{\uplus, \times\}$-circuits can be seen as a generalization to non-binary domains of d-DNNF [Dar01, DM02, PD08] and it is not hard to see that $\{\uplus, \times\}$-circuits on the Boolean domain are exactly the same as d-DNNFs. Thus the C-factorized interpretation follows the usual approach of knowledge compilation in the sense that we show the tractability of an optimization problems on a restricted family of circuits. As a byproduct the results presented in this chapter can be directly generalized to solve linear programs whose variables are the solution sets of d-DNNF.

Outline of the chapter

In Section 5.2 we define Relational circuits and $\{\uplus, \times\}$-circuits. We then define the circuit-based factorized interpretation in Section 5.3. Section 5.4 is dedicated to proving that the C-factorized interpretation can indeed be used to solve $LP(CQ)$ models. Finally in Section 5.5 we show that the T-factorized interpretation can be captured by the C-factorized interpretation.

5.2 Relational circuits

5.2.1 Relational circuits

We start by defining a basic circuit structure to encode relations which we will enhance with additional properties in the following section.

Definition 5.1.
A relational circuit *with variable set $\mathcal{X}$ and domain dom is a tuple $G = (V, E, \rho, o)$ such that (V, E, ρ) is a directed acyclic graph rooted by $\rho \in V$ and $o : V \to \{\bowtie, \cup\} \cup \{[x/d] \mid x \in \mathcal{X}, d \in dom\} \cup \{[]\}$ is a function mapping nodes to operators or constants of the relational algebra.*

We define the type of a gate $g \in V$ with children $g_1, \ldots, g_n \in V$ as follows:

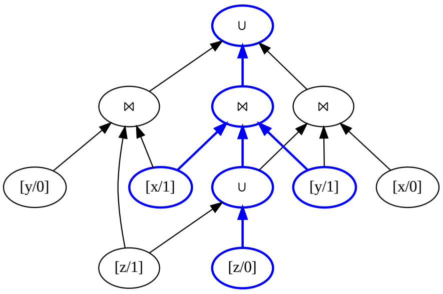
- *if $o(g) = \bowtie$ or $o(g) = \cup$ then $Attr(g) = Attr(g_1) \cup \ldots \cup Attr(g_n)$.*
- *if $o(g) = [x/d]$ then $Attr(g) = \{x\}$ and $n = 0$.*
- *if $o(g) = []$ then $Attr(g) = \emptyset$ and $n = 0$.*

We assume that G is well-typed i.e., for any $\cup$-gate $g \in V$ with children $g_1, \ldots, g_n$, $Attr(g) = Attr(g_1) = \ldots = Attr(g_n)$.

The size *$|G|$ of G is the cardinality of its edge set E.*

The nodes of a relational circuit are also called gates. An $\bowtie$-gate is a node g with $o(g) = \bowtie$ and an $\cup$-gate a node with $o(g) = \cup$.

For each node g of a circuit $G = (V, E, \rho, o)$ we can define a database relation $rel(G_g) \subseteq \{\alpha \mid \alpha : Attr(g) \to dom\}$ inductively as follows:



x	y	z
0	1	0
0	1	1
1	0	1
1	**1**	**0**
1	1	1

Figure 5.1: Example of a relational circuit with its database relation.

- if $o(g) = \bowtie$ then $rel(G_g) = rel(G_{g_1}) \bowtie \ldots \bowtie rel(G_{g_n})$.
- if $o(g) = \cup$ then $rel(G_g) = rel(G_{g_1}) \cup \ldots \cup rel(G_{g_n})$.
- if $o(g) = [x/d]$ then $rel(G_g) = \{[x/d]\}$ and $n = 0$.
- if $o(g) = []$ then $rel(G_g) = \{[]\}$ and $n = 0$.

Finally we define the relation represented by G as $rel(G) = rel(G_\rho)$.

An example for a relational circuit G is given in Figure 5.1. The domain of this circuit is $dom = \{0, 1\}$. The attributes of this circuit are $Attr(\rho) = \{x, y, z\}$. The nodes g with operator $o(g) = \bowtie$ all have $Attr(g) = \{x, y, z\}$. Figure 5.1 also contains the database relation $rel(G)$.

If the circuit G is not clear from the context, we will write $Attr_G$ for the typing function of G. For any gate g of a circuit G, we denote by G_g the subcircuit of G that is rooted in g, i.e., the restriction of G to descendant-or-self nodes of g.

5.2.2 $\{\uplus, \times\}$-Circuits

Unfortunately aribtrary relational circuits are not tractable for anything, even deciding whether their relation is non-empty, so we enhance them with additional properties which allow for tractable operations such as counting the answers of their induced relation which we will show as a short example at the end of this section.

We restrict relational circuits to $\{\uplus, \times\}$-circuits by imposing the restrictions of determinism and decomposability. These properties are well-known from d-DNNFs in the domain of knowledge compilation and our notion of $\{\uplus, \times\}$-circuits can be identified with multi-valued d-DNNFs [KLMT15, Olt16, Dar01, DM02].

Definition 5.2.
A $\{\uplus, \times\}$-circuit is a relational circuit $G = (V, E, \rho, o)$ such that for any node $g \in V$ with children $g_1, \ldots, g_n$:

decomposability *if $o(g) = \bowtie$ then $Attr(g_i) \cap Attr(g_j) = \emptyset$ for any $1 \leq i < j \leq n$.*

determinism *if* $o(g) = \cup$ *then* $rel(G_{g_i}) \cap rel(G_{g_j}) = \emptyset$ *for any* $1 \leq i < j \leq n$.

For $\{\uplus, \times\}$-circuits, the semantics of the join operation $\bowtie$ is a simple Cartesian product by decomposability and the semantics of the union operation $\cup$ is a disjoint union by determinism. Therefore, when considering $\{\uplus, \times\}$-circuits, we will freely identify the relation symbols $\bowtie$ with $\times$ and $\cup$ with $\uplus$.

Observe that the relational circuit in Figure 5.1 is a $\{\uplus, \times\}$-circuit.

Determinism is a semantical condition and it is coNP-hard to check the determinism of a gate given a relational circuit. In practice however, agorithms producing $\{\uplus, \times\}$-circuits, such as the one presented in Section 5.5.1, often ensure the determinism of all $\cup$ gates. This is usually achieved by building $\cup$ gates of the form: $\biguplus_{d \in dom}(G_d) \times [x/d]$, so that the different values d assigned to x ensure determinism.

Observe that for any node v in a $\{\uplus, \times\}$-circuit G we can compute the cardinality of its relation $|rel(G_v)|$ in time linear in the size of G by a simple inductive agorithm that adds the sizes of the children of an $\uplus$ gates, multiplies the sizes of the children of the $\times$ gates and returning a size of 1 (resp. 0) for leaves labelled with $[x/d]$ (resp. $[]$). It is easy to see that the correctness of this algorithm follows from the semantics of the nodes of G.

5.2.3 Proof trees

From now on we will only consider circuits that are decomposable and deterministic.

Let $G = (V, E, \rho, o)$ be a $\{\uplus, \times\}$-circuit and $\tau \in rel(G)$ be an assignment in the database relation of G.

The *proof-tree* of τ, denoted $pt_G(\tau)$, is a subcircuit of G participating in the computation of τ. More formally, $pt_G(\tau)$ is defined inductively by starting from the output as follows: the output of G is in $pt_G(\tau)$. Now if g is a gate in $pt_G(\tau)$ and g' is a child of g, then we add g' in $pt_G(\tau)$ if and only if $\tau_{|Attr(G'_g)} \in rel(G_{g'})$. A proof-tree may be seen as the only witness that a tuple belongs to $rel(G)$. An example of a proof-tree is depicted in blue in Figure 5.1.

We define the relation induced by an edge e as the set of tuples of $rel(G)$ such that their proof-tree contains the edge e, that is $rel(G, e) := \{\tau \in rel(G) \mid e \in pt_G(\tau)\}$.

In the rest of this chapter, to simplify the proofs, we assume wlog that every internal gate of a $\{\uplus, \times\}$-circuit has fan-in two. It is easy to see that, by associativity, $\times$-gates and $\uplus$-gates of fan-in $k > 2$ can be rewritten with $k - 1$ gates of fan-in 2 which is a polynomial size transformation of the circuit. We also assume that for every $x \in \mathsf{Attr}(G)$ and $d \in D$, we have at most one input labeled with x/d. This can be easily achieved by merging all such inputs. For the rest of this section, we fix a $\{\uplus, \times\}$-circuit G on attributes $\mathcal{X}$ and domain *dom*.

We now show that decomposability enforces that the underlying graph of every proof tree is indeed a tree while determinism ensures that every $\tau \in rel(G)$ has exactly one corresponding proof tree.

Proposition 5.3.
Given a $\{\uplus, \times\}$-circuit G and $\tau \in rel(G)$, $P := pt_G(\tau)$, the following holds:

- *every $\times$-gate of P has all its children in P,*
- *every $\uplus$-gate of P has exactly one of its children in P,*
- *P is connected and every gate of P has out-degree at most 1 in P,*
- *for any $x \in \mathsf{Attr}(G)$, P contains exactly one input labeled with $x/\tau(x)$.*

Proof. By induction, it is clear that for every gate g of P, $\tau_{|Attr(G_g)} \in rel(G_g)$. Thus, if g is a $\uplus$-gate of P, as g is disjoint, exactly one of it child v has $\tau_{|Attr(G_g)} \in rel(G_g)$.

If g is a $\times$-gate with children g_1, g_2, then by definition, $\tau_{|Attr(G_g)} \in rel(G_g)$ if and only if $\tau_{|Attr(G_{g_1})} \in rel(G_{g_1})$ and $\tau_{|Attr(G_{g_2})} \in rel(G_{g_2})$. Thus both g_1 and g_2 are in P.

It is clear from definition that P is connected since P is constructed by inductively adding children of gates that are already in P. Now assume that P has a gate g of out-degree greater than 1 in P. Let g_1, g_2 be two of its parents. By definition they are both connected to the root of P so the output of G is their common ancestor. Now let v be their least common ancestor in P. By definition, v has in-degree 2 and so is necessarily a $\times$-gates with children v_1, v_2. Thus g is both in G_{v_1} and G_{v_2}, which is impossible since they are disjoint subcircuits by determinism of v.

Finally, let $x \in \mathsf{Attr}(G)$. Observe that if P has an input labeled g with x/d then $d = \tau(x)$ since $\tau_{|Attr(G_g)} \in rel(G_g)$. Thus, if P contains two inputs v_1 and v_2 on attribute x, they are both labeled with $x/\tau(x)$ and are thus the same input. □

We present additional properties of proof-trees in 5.4.1 as part of the proof of the correctness of the C-factorized interpretation.

5.3 Circuit-based factorized interpretation

We now introduce a so-called circuit-factorized interpretation to solve closed $LP(CQ)$ models more efficiently by leveraging the conciseness of $\{\uplus, \times\}$-circuits representing the answer sets of the queries of the model. Intuitively this approach relies on the fact that each tuple in the relation described by a $\{\uplus, \times\}$-circuit is represented by a unique proof-tree in the circuit. Intuitively we will reduce the number of variables by taking advantage of the fact that each edge of the circuit can represent a subset of tuples of the relation whose proof-trees contain said edge.

5.3.1 Caracterizing informed circuits

We now introduce an additional property that we will require of our $\{\uplus, \times\}$-circuits called *informedness*. Recall that for the T-factorized interpretation we required our tree decompositions to also cover the **weight** operators of the model which allowed us to link each one to a variable $\xi_{Q,u,\beta}$. In a similar vein, the informedness property will allow us to link **weight** operators of $LP(CQ_{qf})_{proj}$ models with subsets of edges of the circuits.

Definition 5.4 (Informed circuit)**.**
Let $G = (V, E, \rho, o)$ be a $\{\uplus, \times\}$-circuit and $X \subseteq Attr(G)$.

We call G informed on X *if there exists a function $edgs_X$ mapping any assignment $\beta \in rel(G)_{|X}$ to a subset $edgs_X(\beta) \subseteq E$ such that:*

$$rel(G)[\beta] = \biguplus_{e \in edgs_X(\beta)} rel(G, e).$$

Observe that a circuit is always informed on sets of a single variable as we can then simply define $edgs_{\{x\}}(\beta)$ to be the outgoing edges of the inputs labelled with $[x/\beta(x)]$.

We then lift this definition to linear programs.

Definition 5.5 (Informed circuit)**.**
Let $\mathcal{G}$ be a collection of $\{\uplus, \times\}$-circuits and L a $LP(CQ_{qf})_{proj}$ model.

We say that $\mathcal{G}$ is informed on L *iff for any $Q \in cq(L)$:*

- *there is a circuit $G_Q \in \mathcal{G}$ such that $rel(G) = ans^{\mathbb{D}}(Q)$,*
- *for any $\mathbf{weight}_{\mathbf{x}:\mathbf{x}' \doteq \mathbf{y}}(Q)$ in $\mathbf{W}_Q(L)$, G_Q is informed on $\mathbf{x}'$.*

5.3.2 Circuit-based factorized interpretation

Intuitively the circuit-based factorized interpretation relies on having each edge of the circuits in $\mathcal{G}$ carry the weight of its induced relation. Thus, given a $LP(CQ_{qf})_{proj}$ model L, an informed collection of circuits $\mathcal{G}$ and a query $Q \in cq(L)$ we define a set of variables associated with the edges of the circuit that $G_Q \in \mathcal{G}$ as follows:

$$\Xi_{L,Q} = \{\xi_e^Q \mid e \in E_{G_Q}\}.$$

The variables of the full C-factorized interpretation are then $\Xi_L = \biguplus_{Q \in cq(L)} \Xi_{L,Q}$.

We now formally define the C-factorized instantiation as follows:

Definition 5.6 (Factorized instantiation)**.**
Given a closed $LP(CQ_{qf})_{proj}$ model L, database $\mathbb{D}$ and a collection of circuits $\mathcal{G}$ that is informed on L, we denote the instantiation of L with $\mathcal{G}$ by $Cinst^{\mathcal{G}}(L)$.

It is the linear program obtained by replacing every $\mathbf{weight}_{\mathbf{x}:\mathbf{x}' \doteq \mathbf{y}}(Q)$ in L with $Cinst^{\mathcal{G}}(\mathbf{weight}_{\mathbf{x}:\mathbf{x}' \doteq \mathbf{y}}(Q))$ where $Cinst^{\mathcal{G}}$ is the function that maps any $\mathbf{weight}_{\mathbf{x}:\mathbf{x}' \doteq \mathbf{y}}(Q)$ to $\sum_{e \in edgs_G([\mathbf{x}'/\mathbf{y}])} \xi_e^Q$ if $[\mathbf{x}'/\mathbf{y}] \in rel(G_Q)$ or 0 otherwise.

Given a query Q and its circuit $G_Q \in \mathcal{G}$ we then define the so-called *circuit soundness constraints* on a gate g of G_Q.

Definition 5.7 (Circuit soundness).
Given a query Q, its circuit G_Q and an internal gate g of G_Q.

$$csc^{G_Q}(g) = \begin{cases} \sum_{i \in \mathsf{In}(g)} \xi_i^Q = \sum_{o \in \mathsf{Out}(g)} \xi_o^Q & \text{if } o(g) = \uplus \\ \bigwedge_{i \in \mathsf{In}(g)} \xi_i^Q = \sum_{o \in \mathsf{Out}(g)} \xi_o^Q & \text{if } o(g) = \times. \end{cases}$$

Observe that any internal gate in a $\{\uplus, \times\}$-circuit is either a $\uplus$-gate or a $\times$-gate so the definition is complete.

Finally we define the factorized interpretation of a closed $LP(CQ_{qf})_{proj}$ L with informed circuits $\mathcal{G}$ which we denote by $\rho^{\mathcal{G}}(L)$. It is obtained by adding the local soudness constraint at any internal gate g in any decomposition $T_Q \in \mathcal{T}$ to the factorized instantiation of L as follows:

Definition 5.8 (Circuit-factorized interpretation).
Given a closed $LP(CQ_{qf})_{proj}$ model L and informed circuits $\mathcal{G}$.
$\rho^{\mathcal{G}}(L) = Cinst^{\mathcal{G}}(L) \wedge \bigwedge_{G_Q \in \mathcal{G}} \bigwedge_{g \in V_{G_Q}} csc^{G_Q}(g)$

The C-factorized interpretation can then be used in the exact same way as the T-factorized interpretation.

We now show how the size of $\rho^{\mathcal{G}}(L)$ depends on the size of $\mathcal{G}$.

Lemma 5.9.
Given a closed $LP(CQ_{qf})_{proj}$ model L and circuits $\mathcal{G}$ informed on L, $\rho^{\mathcal{G}}(L)$ can be encoded in $|\rho^{\mathcal{G}}(L)|_b = O((|L| \cdot I_{\mathcal{G}} + |\mathcal{G}|) \cdot \log(|\mathcal{G}|))$ bits where $|\mathcal{G}| = |\Xi_L|$ is the number of edges of $\mathcal{G}$ and $I_{\mathcal{G}} = max_{\beta \in rel(G)_{|X}} |edgs_X(\beta)|$ for any $G \in \mathcal{G}$ and X on which G is informed.

Proof. First we consider the size of the encoding of $\text{Cinst}^{\mathcal{G}}(L)$. Observe that each $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ in L can be instantiated with $|edgs_{set(\mathbf{x})}(\beta)|$ variables at most so $|\text{Cinst}^{\mathcal{G}}(L)|_b = O(|L| \cdot I_{\mathcal{G}} \cdot \log(|\mathcal{G}|)$.

For the soundness constraints, observe that we can use transitivity to write $csc^G(g) = (\xi_{i_1}^Q = \xi_{i_2}^Q) \wedge (\xi_{i_2}^Q = \xi_{i_3}^Q) \wedge \cdots \wedge (\xi_{i_n}^Q = \sum_{o \in \mathsf{Out}(g)} \xi_o^Q)$ for a $\times$-gate g with ingoing edges $i_1, \ldots, i_n$. Hence each variable ξ_e^Q can appear at most 3 times in the soudness constraints, twice as an ingoing edge of a $\times$-gate and once as an outgoing edge of any gate. Thus it follows that the soundess constraints of $\rho^{\mathcal{G}}(L)$ can be encoded in $O(|\mathcal{G}| \cdot \log(\mathcal{G}))$ bits □

The correctness of the C-based interpretation is expressed in the following theorem wich we prove in Section 5.4.

Theorem 5.10 (Equivalence of C-factorized interpretation and natural semantics).
Let L be a $LP(CQ_{qf})_{proj}$ program and $\mathcal{G}$ be circuits informed on L.

The C-factorized interpretation $\rho^{\mathcal{G}}(L)$ has the same optimal value as its natural semantics $\langle L \rangle^{\mathbb{D}}$.

Moreover $opt(\rho^L(\mathcal{G}))$ can be computed in $O(|\rho^L(\mathcal{G})|_b \cdot |\mathcal{G}|^{\ell})$ where $|\rho^L(\mathcal{G})|_b = (|L| \cdot I_{\mathcal{G}} + |\mathcal{G}|) \cdot \log(|\mathcal{G}|)$ in combined complexity.

We prove this theorem in Section 5.4 using the framework introduced in Section 3.6.2. The complexity of solving $\rho^L(\mathcal{G})$ follows from Lemma 5.9 and Theorem 2.8.

In order to easily reason about weightings of the C-factorized interpretation that concern a fixed query Q, we will consider *sound* edge-weightings W of G_Q by lifting the soundness constraints as follows:

Definition 5.11.
Let $G = (V, E, \rho, o)$ be a $\{\uplus, \times\}$-circuit.

An edge-weighting $W : E \to \mathbb{R}^+$ of G is sound if it holds for any $g \in V$ that:

- ***if*** $o(g) = \uplus$, $\sum_{i \in \mathsf{In}(g)} W(i) = \sum_{o \in \mathsf{Out}(g)} W(o)$,
- ***if*** $o(g) = \times$, $\forall i \in \mathsf{In}(g). W(i) = \sum_{o \in \mathsf{Out}(g)} W(o)$.

5.3.3 Computing a full solution of the natural interpretation

Similarly to the T-factorized interpretation, we provide a way to compute a solution of the natural interpretation from a solution of the C-factorized interpretation.

For simplicity we construct weightings restricted to a single query that we can then recombine into a full weighting of the interpretation.

Definition 5.12.
Let Q be a conjunctive query, $G = (V, E, \rho, o)$ be a $\{\uplus, \times\}$-circuit with $rel(G) = ans^{\mathbb{D}}(Q)$ and W be a sound edge-weighting of G. We assume wlog that the root r of G has a single ingoing edge which we call the output edge o_r.

We construct a weighting $\Pi(W) : \Theta_Q^{\mathbb{D}} \to \mathbb{R}^+$ such that $\Pi(W)(\theta_Q^\alpha) = \omega_{o_r}(\alpha)$ for any $\theta_Q^\alpha \in \Theta_Q^{\mathbb{D}}$ where $\omega_e : rel(G_g) \to \mathbb{R}^+$ is constructed through bottom-up induction on the edges $e = (g, g') \in E$. We distinguish the cases based on the type of g.

- ***Case 1:*** *g is an input labeled with x/d (resp. $[]$): Observe that $rel(G_g)$ contains only the tuple $\tau = [x/d]$ (resp. $\tau = []$).*

  We define $\omega_e(\tau) := W(e)$.

- ***Case 2:*** *g is a $\uplus$-gate with children g_1, g_2.*

  Let $e_1 = \langle g_1, g \rangle$ and $e_2 = \langle g_2, g \rangle$. In this case, given $\tau \in rel(G_g)$, we have by definition that $\tau \in rel(G_{g_i})$ with $i \in \{1, 2\}$, and we then define:

$$\omega_e(\tau) = \begin{cases} W(e)\frac{\omega_{e_1}(\tau)}{W(e_1)+W(e_2)} & \textit{if } \sum_{f \in \mathsf{Out}(g)} W(f) \neq 0 \\ 0 & \textit{otherwise.} \end{cases}$$

  Observe that since W is sound, $W(e_1) + W(e_2) = \sum_{f \in \mathsf{Out}(g)} W(f) \neq 0$, so we never divide by 0.

- ***Case 3:*** *g is a $\times$-gate with children g_1, g_2.*

  Let $e_1 = \langle g_1, g\rangle$ and $e_2 = \langle g_2, g\rangle$. In this case, given $\tau \in rel(G_g)$, we have by definition that $\tau = \tau_1 \times \tau_2$ with $\tau_1 \in rel(G_{g_1})$ and $\tau_2 \in rel(G_{g_2})$. We define:

$$\omega_e(\tau) = \begin{cases} W(e)\frac{\omega_{e_1}(\tau_1)}{W(e_1)}\frac{\omega_{e_2}(\tau_2)}{W(e_2)} & \textit{if } W(e_1) \neq 0 \textit{ and } W(e_2) \neq 0, \\ 0 & \textit{otherwise.} \end{cases}$$

Finally, given a solution w_2 of $\rho^{\mathcal{G}}(L)$, we obtain a solution w_1 of $\langle L\rangle^{\mathbb{D}}$ by simply defining $w_1(\theta_Q^\alpha) = \amalg(W^Q)(\theta_Q^\alpha)$ where W^Q is a weighting of the edges of G_Q with $W^Q(e) = w_2(\xi_e^Q)$.

We will prove that this reconstruction is correct in Section 5.4 as a simple byproduct of the proof of Theorem 5.10:

Lemma 5.13.
Let L be a $LP(CQ_{qf})_{proj}$ program and $\mathcal{G}$ be a collection of circuits informed on L. Let S be the objective function of L and w_2 be an optimal solution of $\rho^{\mathcal{G}}(L)$.

We can construct a solution w_1 of $\langle L\rangle^{\mathbb{D}}$ in $O(|w_1|)$ such that $[\![Inst_{\mathbb{D}}(S)]\!]_{w_1} = [\![Cinst^{\mathcal{G}}(S)]\!]_{w_2}$.

5.4 Correctness

5.4.1 Proof trees properties

In this section we show a few properties of proof-trees that will be useful later.

Proposition 5.14.
Let u be a gate of G and $e_1, e_2 \in \mathsf{Out}(u)$ with $e_1 \neq e_2$. Then $rel(G, e_1) \cap rel(G, e_2) = \emptyset$.

Proof. Assume that $\tau \in rel(G, e_1)$ and $\tau \in rel(G, e_2)$ for $e_1, e_2 \in \mathsf{Out}(u)$ with $e_1 \neq e_2$. By definition, it means that both e_1 and e_2 are in $pt_G(\tau)$ which is a contradiction as $pt_G(\tau)$ has out-degree 1 by Proposition 5.3. □

Next we show that the disjoint union of the relations induced by each ingoing edge of a $\uplus$-gate is equal to the disjoint union of the relations induced by each outgoing edge of this gate.

Proposition 5.15.
Let u be an internal $\uplus$-gate of G,

$$\biguplus_{i \in \mathsf{In}(u)} rel(G, i) = \biguplus_{o \in \mathsf{Out}(u)} rel(G, o).$$

Proof. Let $S_u := \{\tau \mid u \in pt_G(\tau)\}$ and $\tau \in S_u$. It is clear from Proposition 5.3 that a proof tree contains u if and only if it contains at least an edge of $\mathsf{In}(u)$ and at least an edge of $\mathsf{Out}(u)$. Thus, both unions are equal to S_u.

The disjointness of the first union directly follows from the second item of Proposition 5.3 and the disjointness of the second union from Proposition 5.14. □

Finally we show that the disjoint union of the relations induced by the outgoing edges of a ×-gate is equal to the relation induced by each ingoing edge of this gate.

Proposition 5.16.
Let u be an internal $\times$-gate of G,

$$\forall i \in \mathsf{In}(u) : rel(G,i) = \biguplus_{o \in \mathsf{Out}(u)} rel(G,o)$$

Proof. Let $S_u := \{\tau \mid u \in pt_G(\tau)\}$. By Proposition 5.3, it is clear that if u is in a proof tree P, then every edge of $\mathsf{In}(u)$ is also in P and exactly one edge of $\mathsf{Out}(u)$ is in P. Thus, both sets in the statement are equal to S_u. The disjointness of the union follows directly from Proposition 5.14. □

5.4.2 Weighting correspondence

This section is dedicated to showing a weighting correspondence between weightings of the relation represented by a circuit and weightings of the edges of this circuit as expressed in Theorem 5.17. Similarly to the previous chapter, we will use Theorem 5.17 in combination with Proposition 3.17 in Section 5.4.3 in order to prove Theorem 5.10.

Given a circuit G and a weighting $\omega : rel(G) \to \mathbb{R}^+$ we define its projection on G as $\pi_G(\omega) : E_G \to \mathbb{R}^+$ such that for any $e \in E_G$, $\pi_G(\omega)(e) = \sum_{\tau \in rel(G,e)} \omega(\tau)$.

Theorem 5.17.
Let G be a $\{\uplus, \times\}$-circuit G.

1. *For every weighting ω of $rel(G)$, $\pi_G(\omega)$ is sound.*
2. *Given a sound edge-weighting W of G, there exists a weighting $\omega = \amalg(W)$ of $rel(G)$ such that $W = \pi_G(\omega)$.*

5.4.2.1 Proof ω to W

This section is dedicated to the proof of Theorem 5.17(1), namely that, given a $\{\uplus, \times\}$-circuit G and a tuple-weighting ω of G, the edge-weighting W of G induced by ω defined as $W(e) := \sum_{\tau \in rel(G,e)} \omega(\tau)$ is sound.

We will see that the soundness of W follows naturally from the properties of proof-trees of the previous section. We prove that W is sound by checking the case of $\uplus$-gates and $\times$-gates separately.

- We first have to show that for every $\uplus$-gate u of G, it holds that $\sum_{e \in \mathsf{Out}(u)} W(e) = \sum_{e \in \mathsf{In}(u)} W(e)$. This is a consequence of Proposition 5.15. Let u be a $\uplus$-gate of G.

$$\begin{aligned}\sum_{e\in\mathsf{Out}(u)} W(e) &= \sum_{e\in\mathsf{Out}(u)} \sum_{\tau\in rel(G,e)} \omega(\tau) \quad \text{(by definition of } W\text{)}\\ &= \sum_{\tau\in R} \omega(\tau) \quad \left(\text{where } R = \biguplus_{e\in\mathsf{Out}(u)} rel(G,e)\right)\end{aligned}$$

The disjointness of the union in R has been proven in Proposition 5.15, which also states that $R = \biguplus_{e\in\mathsf{In}(u)} rel(G,e)$. Thus, the last term in the sum can be rewritten as

$$\begin{aligned}\sum_{\tau\in R} \omega(\tau) &= \sum_{e\in\mathsf{In}(u)} \sum_{\tau\in rel(G,e)} \omega(\tau)\\ &= \sum_{e\in\mathsf{In}(u)} W(e) \quad \text{(by definition of } W\text{)}\end{aligned}$$

- We now show that for every ×-gate u of G and for every edge $i \in \mathsf{In}(u)$ going in u, it holds that $\sum_{e\in\mathsf{Out}(u)} W(e) = W(i)$. Let u be a ×-gate and $i \in \mathsf{In}(u)$. The proof is very similar to the previous case but is now a consequence of Proposition 5.16. As before, we have $\sum_{e\in\mathsf{Out}(u)} W(e) = \sum_{\tau\in R} \omega(\tau)$ where $R = \biguplus_{e\in\mathsf{Out}(u)} rel(G,e)$. The disjointness of the union in R has been proven in Proposition 5.16, which also implies that $R = rel(G,i)$. Thus, we have:

$$\begin{aligned}\sum_{e\in\mathsf{Out}(u)} W(e) &= \sum_{\tau\in rel(G,i)} \omega(\tau)\\ &= W(i) \quad \text{(by definition of } W\text{)}\end{aligned}$$

5.4.2.2 Proof W to ω

This section is dedicated to the proof of Theorem 5.17(2), namely that, given a circuit W and a sound edge-weighting W of G, there exists a tuple-weighting ω of G such that, for every edge e of G, $\sum_{\tau\in rel(G,e)} \omega(\tau) = W(e)$.

In this section, we fix a $\{\uplus, \times\}$-circuit $G = (V, E, \rho, o)$ and a sound edge-weighting W of its edges. We then construct weightings ω_e for the edges of G with Definition 5.12.

The following lemma which follows the inductive construction of ω_e is the first step of our proof and establishes the relation between ω_e and $W(e)$.

Lemma 5.18.
For every gate u of G and $e = \langle u, v\rangle \in E$,

$$W(e) = \sum_{\tau\in rel(G_u)} \omega_e(\tau).$$

Proof. Let $e = \langle u, v\rangle$ be an edge of G. Observe that when $\sum_{o\in\mathsf{Out}(u)} W(o) = 0$, then $W(e) = 0$ since $e \in \mathsf{Out}(u)$ and W has non-negative value. Moreover, by definition of ω_e, for every $\tau \in rel(G_u)$, $\omega_e(\tau) = 0$. In particular, $\sum_{\tau\in rel(G_u)} \omega_e(\tau) = 0 = W(e)$. In this case then, the lemma holds.

In the rest of the proof, we now assume that $\sum_{o\in\mathsf{Out}(u)} W(o) \neq 0$. We show the lemma by induction on the nodes of G from the leaves to the root.

Base case : u is a leaf labeled with $\tau = x/d$ or $[]$.

Let e be an outgoing edge of u. By definition of ω_e, $\omega_e(\tau) = W(e)$.

Moreover, observe that $rel(G_u) = \{\tau\}$ thus $W(e) = \sum_{\tau\in rel(G_u)} \omega_e(\tau)$.

Inductive case Now let u be an internal gate of G with children u_1, u_2 and let $e_1 = \langle u_1, u\rangle$ and $e_2 = \langle u_2, u\rangle$, as depicted in Figure 5.2.

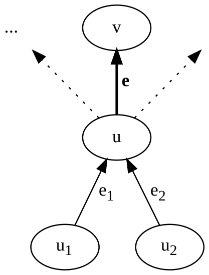


Figure 5.2: Inductive step notations.

Case 1: u is a $\uplus$-gate.

Let $W = W(e_1) + W(e_2)$. Since u is disjoint, given $\tau \in rel(G_u)$, either $\tau \in rel(G_{u_1})$ or $\tau \in rel(G_{u_2})$ but not both. It follows that

$$\begin{aligned}\sum_{\tau\in rel(G_u)} \omega_e(\tau) &= \sum_{\tau\in rel(G_{u_1})} \omega_e(\tau) + \sum_{\tau\in rel(G_{u_2})} \omega_e(\tau)\\ &= \sum_{\tau\in rel(G_{u_1})} \frac{W(e)}{W}\omega_{e_1}(\tau_1) + \sum_{\tau\in rel(G_{u_2})} \frac{W(e)}{W}\omega_{e_2}(\tau_2)\end{aligned}$$

by definition of ω_e.

Observe that by induction we have $W(e_1) = \sum_{\tau\in rel(G_{u_1})} \omega_{e_1}(\tau_1)$ and $W(e_2) = \sum_{\tau\in rel(G_{u_2})} \omega_{e_2}(\tau_2)$. Thus, by taking the constants out and using this identity, it follows that:

$$\begin{aligned}\sum_{\tau \in rel(G_u)} \omega_e(\tau) &= \frac{W(e)}{W} \sum_{\tau \in rel(G_{u_1})} \omega_{e_1}(\tau_1) + \frac{W(e)}{W} \sum_{\tau \in rel(G_{u_2})} \omega_{e_2}(\tau_2) \\ &= \frac{W(e)}{W} W(e_1) + \frac{W(e)}{W} W(e_2) \\ &= W(e) \frac{W(e_1)+W(e_2)}{W} \\ &= W(e).\end{aligned}$$

Case 2: u is a $\times$-gate.

By definition of ω_e, we have

$$\sum_{\tau \in rel(G_u)} \omega_e(\tau) = \sum_{\tau \in rel(G_u)} W(e) \frac{\omega_{e_1}(\tau_{|\mathsf{Attr}(G_{u_1})})}{W(e_1)} \frac{\omega_{e_2}(\tau_{|\mathsf{Attr}(G_{u_2})})}{W(e_2)}.$$

Remember that by definition of $\{\uplus, \times\}$-circuits, $rel(G_u) = rel(G_{u_1}) \times rel(G_{u_2})$. That is, $\tau \in rel(G_u)$ if and only if $\tau_1 = \tau_{|\mathsf{Attr}(G_{u_1})} \in rel(G_{u_1})$ and $\tau_2 = \tau_{|\mathsf{Attr}(G_{u_2})} \in rel(G_{u_2})$. Thus, we can rewrite the previous sum yielding

$$\sum_{\tau \in rel(G_u)} \omega_e(\tau) = \sum_{\tau_1 \in rel(G_{u_1})} \sum_{\tau_2 \in rel(G_{u_2})} W(e) \frac{\omega_{e_1}(\tau_1)}{W(e_1)} \frac{\omega_{e_2}(\tau_2)}{W(e_2)}.$$

By taking the constants $W(e), W(e_1)$ and $W(e_2)$ out of the sums and observing that the sum is now separated into two independent terms, we have:

$$\sum_{\tau \in rel(G_u)} \omega_e(\tau) = W(e) \frac{\sum_{\tau_1 \in rel(G_{u_1})} \omega_{e_1}(\tau_1)}{W(e_1)} \frac{\sum_{\tau_2 \in rel(G_{u_2})} \omega_{e_2}(\tau_2)}{W(e_2)}.$$

By induction, $W(e_i) = \sum_{\tau \in rel(G_{u_i})} \omega_{e_i}(\tau_i)$ for $i = 1, 2$. Thus both fractions are equal to 1, from which we conclude that

$$\sum_{\tau \in rel(G_u)} \omega_e(\tau) = W(e).$$

□

We choose $\omega = \omega_{o_r}$ where o_r is the output edge. Lemma 5.18 is however not enough to prove Theorem 5.17 as it only gives the equality $W(e) = \sum_{\tau \in rel(G,e)} \omega(\tau)$ for $e = o_r$. Fortunately we can show that it holds for every edge e of the circuit. We actually prove a stronger property, that ω_e is, in some sense, a projection of ω.

Given an edge $e = \langle u, v \rangle$ of G and $\tau' \in rel(G_u)$, we denote by $rel(G, e, \tau')$ the set of tuples τ of $rel(G, e)$ such that $\tau_{|\mathsf{Attr}(G_u)} = \tau'$. We prove the following:

Lemma 5.19.
For every $e = \langle u, v \rangle \in E$*, for every* $\tau' \in rel(G_u)$*,*

$$\omega_e(\tau') = \sum_{\tau \in rel(G,e,\tau')} \omega(\tau).$$

Proof. We prove this lemma by top-down induction on the edges of G.

Base case We prove the result for $e = o_r = \langle u, v \rangle$. Let $\tau' \in rel(G_u)$. Because u is the output gate, we have $\mathsf{Attr}(G_u) = \mathsf{Attr}(G)$ and thus $rel(G, e, \tau') = \{\tau'\}$. Recall that $\omega = \omega_{o_r}$ by definition. In other words, $\omega_e(\tau') = \omega(\tau') = \sum_{\tau \in rel(G,e,\tau')} \omega(\tau)$.

Inductive case: Now let $e = \langle u, v \rangle$ be an internal edge of G. Let $o_1, \ldots, o_n$ be the outgoing edges of v, u' be the only sibling of u and let $e' = \langle u', v \rangle$. See Figure 5.3 for a schema of these notations. We fix $\tau' \in rel(G_u)$ and prove the desired equality.

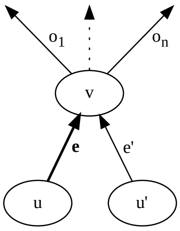


Figure 5.3: Notations for the inductive step.

Case 1 : v is a $\uplus$-gate.

In this case, $\tau' \in rel(G_v)$. We claim that

$$rel(G, e, \tau') = \biguplus_{o \in \mathsf{Out}(v)} rel(G, o, \tau').$$

For left-to-right inclusion, let $\tau \in rel(G, e, \tau')$. By definition, its proof tree $pt_G(\tau)$ contains e. Since $pt_G(\tau)$ is connected by Proposition 5.3, $pt_G(\tau)$ has to contain one edge of $\mathsf{Out}(v)$. The disjointness of the right-side union is a direct consequence of Proposition 5.14.

For the right-to-left inclusion, fix $o \in \mathsf{Out}(v)$ and let $\tau \in rel(G, o, \tau')$. By definition, its proof tree $pt_G(\tau)$ contains o, thus it also contains the vertex v. Now recall that $\tau_{|\mathsf{Attr}(G_u)} = \tau' \in rel(G_u)$. Thus, by definition of proof trees, u is also in $pt_G(\tau)$. In other words, $\tau \in rel(G, e, \tau')$.

Using this equality, we have

$$\begin{aligned}\sum_{\tau\in rel(G,e,\tau')}\omega(\tau) &= \sum_{o\in\mathsf{Out}(v)}\sum_{\tau\in rel(G,o,\tau')}\omega(\tau)\\ &= \sum_{o\in\mathsf{Out}(v)}\omega_o(\tau')\end{aligned}$$

since we know by induction that for every $o \in \mathsf{Out}(v)$, $\omega_o(\tau') = \sum_{\tau\in rel(G,o,\tau')}\omega(\tau)$.

Assume first that $\sum_{o\in\mathsf{Out}(v)} W(o) = 0$. In this case, by definition, for every o, $\omega_o(\tau') = 0$. However, since W is sound, it follows that $W(e) = 0$, which implies by Lemma 5.18 that $\omega_e(\tau') = 0$ as well. In this case, $\sum_{\tau\in rel(G,e,\tau')}\omega(\tau) = 0 = \omega_e(\tau')$ which is the induction hypothesis.

Now assume that $\sum_{o\in\mathsf{Out}(v)} W(o) \neq 0$. We can thus apply the definition of $\omega_o(\tau') = \frac{W(o)}{W(e)+W(e')}\omega_e(\tau')$ in the last sum. This yields

$$\begin{aligned}\sum_{o\in\mathsf{Out}(v)}\omega_o(\tau') &= \sum_{o\in\mathsf{Out}(v)}\frac{W(o)}{W(e)+W(e')}\omega_e(\tau')\\ &= \omega_e(\tau')\frac{1}{W(e)+W(e')}\sum_{o\in\mathsf{Out}(v)}W(o)\\ &= \omega_e(\tau')\end{aligned}$$

where the last equality follows from the fact that W is sound and that the ratio is thus 1.

Case 2: v is a $\times$-gate.

Similarly as before, we have:

$$rel(G,e,\tau') = \biguplus_{\tau''\in rel(G_{u'})}\ \biguplus_{o\in\mathsf{Out}(v)} rel(G,o,\tau'\times\tau'').$$

For left-to-right inclusion, let $\tau \in rel(G,e,\tau')$. By definition, its proof tree $pt_G(\tau)$ contains e. Since $pt_G(\tau)$ is connected by Proposition 5.3, $pt_G(\tau)$ has to contain one edge o of $\mathsf{Out}(v)$. Thus, $\tau \in rel(G,o,\tau' \times \tau_{|\mathsf{Attr}(G_{u'})})$.

The disjointness of the right-side union is a direct consequence of Proposition 5.14. For the right-to-left inclusion, we fix $o \in \mathsf{Out}(v)$ and $\tau'' \in rel(G_{u'})$. Let $\tau \in rel(G,o,\tau'\times\tau'')$. By definition, its proof tree $pt_G(\tau)$ contains o, thus it also contains the vertex v and by definition of proof trees, u is also in $pt_G(\tau)$. Moreover, since $\tau_{|\mathsf{Attr}(G_u)} = \tau'$, it follows that $\tau \in rel(G,e,\tau')$. Using this equality, we have

$$\begin{aligned}\sum_{\tau\in rel(G,e,\tau')}\omega(\tau) &= \sum_{o\in\mathsf{Out}(v)}\sum_{\tau''\in rel(G_{u'})}\sum_{\tau\in rel(G,o,\tau'\times\tau'')}\omega(\tau)\\ &= \sum_{o\in\mathsf{Out}(v)}\sum_{\tau''\in rel(G_{u'})}\omega_o(\tau'\times\tau'').\end{aligned}$$

since we know by induction that for every $o \in \mathsf{Out}(v)$, $\omega_o(\tau' \times \tau'') = \sum_{\tau \in rel(G,o,\tau' \times \tau'')} \omega(\tau)$.

Assume first that $\sum_{o \in \mathsf{Out}(v)} W(o) = 0$. In this case, by definition, for every o and τ'', $\omega_o(\tau' \times \tau'') = 0$.

However, since W is sound, we also have $W(e) = 0$ which implies by Lemma 5.18 that $\omega_e(\tau') = 0$ as well. In this case, $\sum_{\tau \in rel(G,e,\tau')} \omega(\tau) = 0 = \omega_e(\tau')$ wich is the induction hypothesis.

Now assume that $\sum_{o \in \mathsf{Out}(v)} W(o) \neq 0$. We can then apply the definition of $\omega_o(\tau' \times \tau'') = W(o)\frac{\omega_e(\tau')}{W(e)}\frac{\omega_{e'}(\tau'')}{W(e')}$ in the last sum. This yields

$$\begin{aligned}
&\sum_{o \in \mathsf{Out}(v)} \sum_{\tau'' \in rel(G_{u'})} \omega_o(\tau' \times \tau'') \\
&= \sum_{o \in \mathsf{Out}(v)} \sum_{\tau'' \in rel(G_{u'})} W(o)\frac{\omega_e(\tau')}{W(e)}\frac{\omega_{e'}(\tau'')}{W(e')} \\
&= \frac{\omega_e(\tau')}{W(e)}(\sum_{o \in \mathsf{Out}(v)} W(o))\frac{\sum_{\tau'' \in rel(G_{u'})} \omega_{e'}(\tau'')}{W(e')}
\end{aligned}$$

Since W is sound, it follows that $\sum_{o \in \mathsf{Out}(v)} W(o) = W(e)$. Moreover, by Lemma 5.18, $\sum_{\tau'' \in rel(G_{u'})} \omega_{e'}(\tau'') = W(e')$. Thus, the last sum is equal to $\omega_e(\tau')$ which concludes the proof.

□

Theorem 5.17 2 is now an easy consequence of Lemma 5.18 and Lemma 5.19:

$$\begin{aligned}
W(e) &= \sum_{\tau' \in rel(G_u)} \omega_e(\tau') \quad \text{(by Lemma 5.18)} \\
&= \sum_{\tau' \in rel(G_u)} \sum_{\tau \in rel(G,e,\tau')} \omega(\tau) \quad \text{(by Lemma 5.19)} \\
&= \sum_{\tau \in rel(G,e)} \omega(\tau) \quad (\text{since } rel(G,e) = \biguplus_{\tau' \in rel(G_u)} rel(G,e,\tau')).
\end{aligned}$$

Indeed, $rel(G,e) = \biguplus_{\tau' \in rel(G_u)} rel(G,e,\tau')$. For the left-to-right inclusion, if $\tau \in rel(G,e)$ then by definition $\tau' = \tau_{|\mathsf{Attr}(G_u)} \in rel(G_u)$ and thus $\tau \in rel(G,e,\tau')$. The other inclusion follows by definition since $rel(G,e,\tau') \subseteq rel(G,e)$ for every $\tau' \in rel(G_u)$.

5.4.3 Proof of the equivalence theorem

Now armed with Theorem 5.17 we are ready to prove Theorem 5.10.

Let $L_1 = \langle L \rangle^{\mathbb{D}} = \langle L \rangle^{\mathrm{Inst}_{\mathbb{D}},C_1}$ with $C_1 = \bigwedge_{Q \in cq(L)} true$.

Let $L_2 = \rho^{\mathcal{G}}(L) = \langle L \rangle^{\mathrm{Cinst}^{\mathcal{G}},C_2}$ with $C_2 = \bigwedge_{Q \in cq(L)} csc^{\mathcal{G}}(Q)$. We fix a query $Q \in cq(L)$.

For any weighting w_1^Q of $var_Q(L_1)$ (that obviously satisfies *true*), we define a weighting ω of $ans^{\mathbb{D}}(Q)$ such that for any $\tau \in ans^{\mathbb{D}}(Q)$, $\omega(\tau) = w_1^Q(\theta_Q^\tau)$. By Theorem 5.17, $\pi_G(\omega)$ is sound. Thus the weighting w_2^Q defined as $w_2^Q(\xi_e^Q) = \pi_G(\omega)(e)$ for any $u \in T_Q$ and $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ satisfies $lsc^{\mathcal{T},\mathbb{D}}(Q)$.

We now fix $\mathbf{weight}_{\mathbf{x}:\mathbf{x}' \doteq \mathbf{d}}(Q) \in \mathbf{W}_Q(L)$, let $\beta = [\mathbf{x}'/\mathbf{d}]$. By definition $[\![\mathrm{Cinst}^{\mathcal{G}}(\mathbf{w})]\!]_{w_2^Q} = \sum_{e \in edgs_G(\beta)} \xi_e^Q = \sum_{e \in edgs_G(\beta)} \sum_{\alpha \in rel(G,e)} \omega(\alpha)$. Observe that G_Q is informed on $\mathbf{x}$ so $rel(G)[\beta] = \biguplus_{e \in edgs_X(\beta)} rel(G, e)$ thus it follows that the previous sum is equal to $\sum_{\alpha \in ans^{\mathbb{D}}(Q)[\beta]} \omega(\alpha) = \sum_{\alpha \in ans^{\mathbb{D}}(Q)[\beta]} w_1^Q(\theta_Q^\alpha) = [\![\mathrm{Inst}_{\mathbb{D}}]\!]_{w_1^Q}$.

Thus by Proposition 3.17, $opt(L_1) \leq opt(L_2)$.

For any weighting w_2^Q of $var_Q(L_2)$ that satisfies $csc^{\mathcal{G}}(Q)$ we define a weighting W such that for any edge $e \in G_Q$, $W(e) = w_2^Q(\xi_e^Q)$. Observe that W is sound so by Theorem 5.17 there is a weighting $\omega = \amalg(W)$ such that $W = \pi_G(\omega)$. We define a weighting w_1^Q such that $w_1^Q(\theta_Q^\tau) = \omega(\tau)$ for any $\tau \in ans^{\mathbb{D}}(Q)$ that obviously satisfies *true*. Similarly to the above it follows that for any $\mathbf{w} \in \mathbf{W}_Q(L)$, $[\![\mathrm{Inst}_1(\mathbf{w})]\!]_{w_1^Q} = [\![\mathrm{Inst}_2(\mathbf{w})]\!]_{w_2^Q}$.

Thus by Proposition 3.17, $opt(L_2) \leq opt(L_1)$. Additionally Lemma 5.13 holds by Lemma 3.15.

Finally $opt(L_1) = opt(L_2)$ and Theorem 5.10 holds.

5.5 Recapturing the T-factorized interpretation

In this section we will show that the C-factorized interpretation is a generalization of the T-factorized interpretation.

First we show that the answer set of any conjunctive query Q with tree decomposition T of width k on the database $\mathbb{D}$ can be represented as a $\{\uplus, \times\}$-circuit of size $O(|T||\mathbb{D}|^k)$ that is informed on every bag of T. Thus, given a closed $LP(CQ_{qf})_{proj}$ model L and a tree decomposition of L, we are then able to compute an informed circuit for every $Q \in cq(L)$. Finally we can use this family of circuits to compute the C-factorized interpretation of L which allows us to compute $opt(L)$ with the same data complexity as computing it through T-factorized interpretation

5.5.1 Compiling a conjunctive query with a tree decomposition

We now describe how to compile a normalized tree decomposition T of a query Q and database D into a $\{\uplus, \times\}$-circuit that represents $ans^{\mathbb{D}}(Q)$. While a similar result appears in [OZ15b], we are looking for different properties in our circuits (particularly the informedness) so we define and prove our own process in this section to best suit our purposes.

We assume that the bag of the root of T is empty, that is $\mathcal{B}(r) = \emptyset$, which can easily be ensured by adding some project nodes on top of the tree. For simplicity we also assume that we have computed $ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t)}$ for any $t \in T$ which is relatively easy to do by Lemma 2.5.

For simplicity we denote by $g = l(g_1, \dots g_n)$ a gate with $o(g) = l$ and children $g_1, \dots, g_n$ and by $g = l$ a gate with $o(g) = l$ and no children.

Definition 5.20.
We compile a query Q with a tree decomposition T and database $\mathbb{D}$ into a circuit $compile(Q, T, \mathbb{D})$ by constructing a gate $g_{t,\beta}$ inductively for each vertex $t \in T$ and $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t)}$ as follows:

- *if t is a leaf then*
$$g_{t,\beta} = [],$$
- *if t is a join node with children t', t'' then*
$$g_{t,\beta} = \bowtie(g_{t',\beta}, g_{t'',\beta}),$$
- *if t is a extend node with child u' then*
$$g_{t,\beta} = \cup(g_{t',\beta'})$$
with $\beta' = \beta_{|\mathcal{B}(t')}$,
- *if t is a project node with child t' and $\mathcal{B}(t') \setminus \mathcal{B}(t) = \{x\}$ then*
$$g_{t,\beta} = \cup(g_{t,\beta,\beta_1(x)}, \dots g_{t,\beta,\beta_l(x)})$$
with $g_{t,\beta,\beta_i(x)} = \bowtie(g_{t',\beta_i}, [x/\beta_i(x)])$ for any $\beta_i = \beta \bowtie [x/\beta_i(x)] \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t')}[\beta]$.

Finally $compile(Q, T, \mathbb{D})$ is the relational circuit G rooted in $g_{r,[]}$.

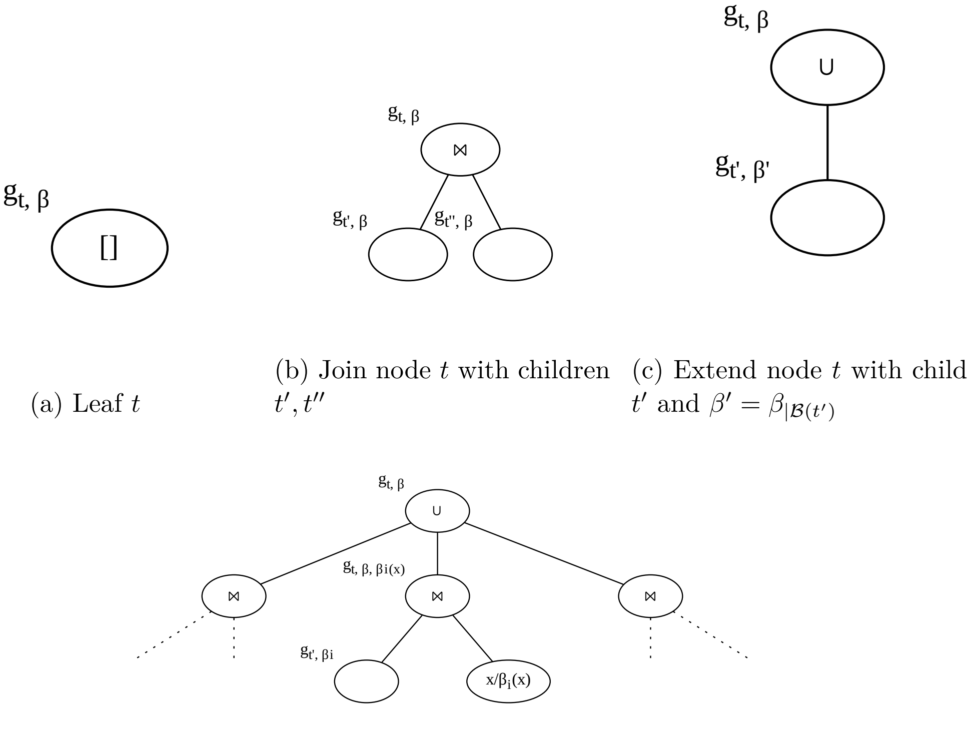
The compilation is also illustrated in Figure 5.4. In the next section we will show that this circuit is a $\{\uplus, \times\}$-circuit that represents the answers of Q and moreover that it is informed on any bag of T.

5.5.2 Correctness of the compilation

In this section we will prove that a compiled circuit does indeed represent the answers of its input query and that it also informed on any bag of the input tree decomposition.

Throughout this section we fix a query Q, a database $\mathbb{D}$ and a tree decomposition T of Q. For brevity we denote $compile(Q, T, \mathbb{D})$ by G and we use the notations defined in Defintion 5.20 when referring to the gates of G. We also carry over the initial assumptions from the previous section. For brevity we denote $rel(G_g)$ by $rel(g)$.

We begin by proving the following lemma about the typing of the gates of the compiled circuit.



(a) Leaf t

(b) Join node t with children t', t''

(c) Extend node t with child t' and $\beta' = \beta_{|\mathcal{B}(t')}$

(d) Project node t with child t' and $\beta_i \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t')}[\beta]$

Figure 5.4: Gate created for a node t and $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t)}$

Lemma 5.21.
The circuit G is well-typed and for any $t \in T$ and $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t)}$, $Attr(g_{t,\beta}) = \mathsf{Attr}(\downarrow t) \setminus \mathcal{B}(t)$.

Proof. We show this by bottom-up induction on the nodes $t \in T$. In each case we fix a $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t)}$.

Base case: t is a leaf.

In this case $g_{t,\beta}$ is labelled with [] so $Attr(g_{t,\beta}) = \emptyset$. Observe that because t is a leaf then $\mathsf{Attr}(\downarrow t) \setminus \mathcal{B}(t) = \mathcal{B}(t) \setminus \mathcal{B}(t) = \emptyset$. Thus $Attr(g_{t,\beta}) = \mathsf{Attr}(\downarrow t) \setminus \mathcal{B}(t)$.

Inductive case 1: t is a extend node with a child t'.

Observe that $g_{t,\beta}$ is a $\cup$-gate with a single child $g_{t,\beta}$ so $Attr(g_{t,\beta}) = Attr(g_{t',\beta}) = \mathsf{Attr}(\downarrow t') \setminus \mathcal{B}(t')$ by induction. Moreover it is obviously well-typed.

Recall that t is a extend node so there is a variable x such that $\mathcal{B}(t) = \mathcal{B}(t') \uplus \{x\}$ and moreover that $x \notin \mathsf{Attr}(\downarrow t')$ by connectivity of T. It follows that $\mathsf{Attr}(\downarrow t) \setminus \mathcal{B}(t) = (\mathsf{Attr}(\downarrow t') \uplus \{x\}) \setminus (\mathcal{B}(t') \uplus \{x\} = \mathsf{Attr}(\downarrow t') \setminus \mathcal{B}(t')$. Thus $Attr(g_{t,\beta}) = \mathsf{Attr}(\downarrow t) \setminus \mathcal{B}(t)$.

Inductive case 2: t is a project node with a child t'.

Observe that t is a project node so there is a variable x such that $\mathcal{B}(t') = \mathcal{B}(t) \uplus \{x\}$. Recall that $g_{t,\beta}$ has a child $g_{t,\beta,\beta_i(x)}$ for any value $\beta_i \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t')}[\beta]$. Observe that $Attr(g_{t,\beta,\beta_i(x)}) = Attr(g_{t',\beta'}) \cup Attr([x/\beta_i(x)])$. By induction $Attr(g_{t',\beta'}) = \mathsf{Attr}(\downarrow t') \setminus \mathcal{B}(t')$ so $Attr(g_{t,\beta,\beta_i(x)}) = \mathsf{Attr}(\downarrow t') \setminus \mathcal{B}(t') \cup \{x\} = \mathsf{Attr}(\downarrow t) \setminus \mathcal{B}(t)$.

Thus $g_{t,\beta}$ is well-typed and $Attr(g_{t,\beta}) = \mathsf{Attr}(\downarrow t) \setminus \mathcal{B}(t)$.

Inductive case 3: t is a join node with children t' and t''.

By induction $Attr(g_{t',\beta}) = \mathsf{Attr}(\downarrow t') \setminus \mathcal{B}(t')$ and $Attr(g_{t'',\beta}) = \mathsf{Attr}(\downarrow t'') \setminus \mathcal{B}(t'')$. By definition $\mathsf{Attr}(\downarrow t) = \mathsf{Attr}(\downarrow t') \cup \mathsf{Attr}(\downarrow t'') \cup \mathcal{B}(t)$. However recall that $\mathcal{B}(t') = \mathcal{B}(t'') = \mathcal{B}(t)$ because T is normalized so $\mathcal{B}(t) = \mathcal{B}(t') \subseteq \mathsf{Attr}(\downarrow t')$ thus $\mathsf{Attr}(\downarrow t) = \mathsf{Attr}(\downarrow t') \cup \mathsf{Attr}(\downarrow t'')$.

Finally $Attr(g_{t,\beta}) = Attr(g_{t',\beta}) \cup Attr(g_{t'',\beta}) = \mathsf{Attr}(\downarrow t') \setminus \mathcal{B}(t') \cup \mathsf{Attr}(\downarrow t'') \setminus \mathcal{B}(t'')$ which we can rearrange as $\mathsf{Attr}(\downarrow t') \cup \mathsf{Attr}(\downarrow t'') \uplus \mathcal{B}(t)$. Thus $Attr(g_{t,\beta}) = \mathsf{Attr}(\downarrow t) \setminus \mathcal{B}(t)$.

Observe that through this induction we have also checked that any $\cup$-gate in G is well-typed so G is well-typed.

□

We then prove that the gates of G are deterministic and decomposable.

Lemma 5.22.
The circuit G is a $\{\uplus, \times\}$-circuit.

Proof. We show that we only introduce deterministic $\cup$-gates and decomposable $\bowtie$-gates when constructing $g_{t,\beta}$ for a node $t \in T$ and $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t)}$.

Case 1: t is a leaf

In this case $g_{t,\beta}$ is also a leaf so there is nothing to verify.

Case 2: t is a *extend* node with child t'.

It is clear that $g_{t,\beta}$ is decomposable as it only has a single child.

Case 3: t is a *project* node with child t'.

Observe that, for any child $g_{t,\beta,\beta_i(x)}$ of $g_{t,\beta}$, any $\tau \in rel(g_{t,\beta,\beta_i(x)})$ satisfies $\tau(x) = \beta_i(x)$ by construction. Thus $rel(g_{t,\beta,\beta_i(x)})$ and $rel(g_{t,\beta,\beta_j(x)})$ are pairwise disjoint for any $i \neq j$ so $g_{t,\beta}$ is deterministic.

Now consider a $g_{t,\beta,\beta_i(x)}$ gate introduced as the child of a $\cup$-gate $g_{t,\beta}$ introduced for a *project* node. By definition there is a variable x such that $\mathcal{B}(t') = \mathcal{B}(t) \uplus \{x\}$. Thus by Lemma 5.21, $x \notin Attr(g_{t',\beta\bowtie[x/d_i]}) = \mathsf{Attr}(\downarrow t') \setminus \mathcal{B}(t')$ so $g_{t,\beta,\beta_i(x)}$ is decomposable.

Case 4: t is a *join* node with children t' and t''.

By Lemma 5.21 we know that $Attr(g_{t',\beta}) = \mathsf{Attr}(\downarrow t') \setminus \mathcal{B}(t')$ and $Attr(g_{t'',\beta}) = \mathsf{Attr}(\downarrow t'') \setminus \mathcal{B}(t'')$. Observe that for any variable x if $x \in \mathsf{Attr}(\downarrow t') \cap \mathsf{Attr}(\downarrow t'')$ then $x \in \mathcal{B}(t') = \mathcal{B}(t'')$ by connectivity of T. Thus $Attr(g_{t',\beta}) \cap Attr(g_{t'',\beta}) = \mathsf{Attr}(\downarrow t') \setminus \mathcal{B}(t') \cap \mathsf{Attr}(\downarrow t'') \setminus \mathcal{B}(t'') = \emptyset$ so $g_{t,\beta}$ is decomposable.

□

We now prove an intermediate lemma that we will use to tie the next proof to some properties from Chapter 4.

Lemma 5.23.
Let X and $Y \subseteq X$ be two sets of variables and D be the domain. Given a set of variable assignments $A \subseteq D^X$ and an assignment $\beta : Y \to D$, $A_{|X}[\beta] = A[\beta]_{|X}$.

Proof. We show this by double inclusion.

For left-to-right inclusion, let $\alpha \in A_{|X}[\beta]$. By definition there exists an $\alpha' \in A[\beta]$ such that $\alpha'_{|X} = \alpha$. Thus $\alpha'_{|X} = \alpha \in A[\beta]_{|X}$.

For right-to-left inclusion, let $\alpha \in A[\beta]_{|X}$. Observe that $A[\beta] \subseteq A$ so $\alpha \in A_{|X}$ Moreover $\alpha_{|Y} = \beta$ by definition so $\alpha \in A_{|X}[\beta]$.

□

We now show that G encodes the relation $ans^{\mathbb{D}}(Q)$.

Lemma 5.24.
The circuit G represents the answers of Q that is,

$$rel(G) = ans^{\mathbb{D}}(Q).$$

Proof. We show by bottom-up induction on the nodes of T that for any $t \in T$ and $\beta \in \mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(t)}$, $\mathit{rel}(g_{t,\beta}) = \mathit{ans}^{\mathbb{D}}(Q)[\beta]_{|\mathsf{Attr}(\downarrow t)\setminus\mathcal{B}(t)}$.

First of all observe that the lemma follows from applying the induction at the root of the decomposition tree. Indeed, $\mathit{rel}(G) = \mathit{rel}(G_{g_{r,[]}})$ which is equal to $\mathit{ans}^{\mathbb{D}}(Q)[[]]_{|\mathsf{Attr}(\downarrow r)\setminus\mathcal{B}(r)} = \mathit{ans}^{\mathbb{D}}(Q)$ by induction.

Base case: t is a leaf.

By definition $\mathit{rel}(g_{t,\beta}) = \{[]\}$. Observe that since we assumed that $\mathit{ans}^{\mathbb{D}}(Q) \neq \emptyset$ then it follows that $\mathit{ans}^{\mathbb{D}}(Q)[\beta]_{|\mathsf{Attr}(\downarrow t)\setminus\mathcal{B}(t)} = \mathit{ans}^{\mathbb{D}}(Q)[\beta]_{|\emptyset} = \{[]\}$. Thus $\mathit{rel}(g_{t,\beta}) = \mathit{ans}^{\mathbb{D}}(Q)[\beta]_{|\mathsf{Attr}(\downarrow t)\setminus\mathcal{B}(t)}$.

For the inductive case, we will show by induction that $\mathit{rel}(g_{t,\beta}) = \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta]_{|\mathsf{Attr}(\downarrow t)\setminus\mathcal{B}(t)}$ in order to be able to use lemmas from Chapter 4. Observe that the equivalence of this equality with the induction hypothesis easily follows from Lemma 5.23.

Inductive case 1: t is a extend node with a child t'.

Let $\beta' = \beta_{|\mathcal{B}(t')}$. By definition $\mathit{rel}(g_{t,\beta}) = \mathit{rel}(g_{t',\beta'})$ so by induction $\mathit{rel}(g_{t,\beta}) = \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t')}[\beta']_{|\mathsf{Attr}(\downarrow t')\setminus\mathcal{B}(t')}$. By Lemma 4.21 we know that $\mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t')}[\beta'] = \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta]_{|\mathsf{Attr}(\downarrow t')}$ from which it follows that $\mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t')}[\beta']_{|\mathsf{Attr}(\downarrow t')\setminus\mathcal{B}(t')} = \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta]_{|\mathsf{Attr}(\downarrow t')}{}_{|\mathsf{Attr}(\downarrow t')\setminus\mathcal{B}(t')}$. Observe that $\mathsf{Attr}(\downarrow t') \setminus \mathcal{B}(t') \subseteq \mathsf{Attr}(\downarrow t')$ so $\mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta]_{|\mathsf{Attr}(\downarrow t')}{}_{|\mathsf{Attr}(\downarrow t')\setminus\mathcal{B}(t')} = \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta]_{|\mathsf{Attr}(\downarrow t')\setminus\mathcal{B}(t')}$. Thus it follows from the two previous equalities that $\mathit{rel}(g_{t,\beta}) = \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta]_{|\mathsf{Attr}(\downarrow t')\setminus\mathcal{B}(t')}$.

Finally because t is an extend node then $\mathsf{Attr}(\downarrow t) \setminus \mathcal{B}(t) = \mathsf{Attr}(\downarrow t') \setminus \mathcal{B}(t')$ so $\mathit{rel}(g_{t,\beta}) = \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta]_{|\mathsf{Attr}(\downarrow t)\setminus\mathcal{B}(t)}$.

Inductive case 2: t is a project node with a child t'.

Let x be the variable such that $\mathcal{B}(t') = \mathcal{B}(t) \uplus \{x\}$.

By definition $\mathit{rel}(g_{t,\beta}) = \biguplus_{\beta' \in \mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(t')}[\beta]} \mathit{rel}(g_{\beta'(x)})$ which we can further expand to $\biguplus_{\beta' \in \mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(t')}[\beta]} \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t')}[\beta']_{|\mathsf{Attr}(\downarrow t')\setminus\mathcal{B}(t')} \times \{[x/\beta'(x)]\}$. Given $\beta' \in \mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(t')}[\beta]$, observe that when we compute the Cartesian product of $\mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t')}[\beta']_{|\mathsf{Attr}(\downarrow t')\setminus\mathcal{B}(t')}$ with $\{[x/\beta'(x)]\}$ we are actually restoring the value of x that was projected away and so we obtain $\mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t')}[\beta']_{|\mathsf{Attr}(\downarrow t')\setminus\mathcal{B}(t)}$. Moreover, $\mathsf{Attr}(\downarrow t') = \mathsf{Attr}(\downarrow t)$ because t is a project node so this disjoint union is equal to $\biguplus_{\beta' \in \mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(t')}[\beta]} \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta']_{|\mathsf{Attr}(\downarrow t)\setminus\mathcal{B}(t)} = (\biguplus_{\beta' \in \mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(t')}[\beta]} \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t')}[\beta'])_{|\mathsf{Attr}(\downarrow t)\setminus\mathcal{B}(t)}$. Finally by Lemma 4.19, $(\biguplus_{\beta' \in \mathit{ans}^{\mathbb{D}}(Q)_{|\mathcal{B}(t')}[\beta]} \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t')}[\beta']) = \mathit{ans}^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta]$.

Thus $rel(g_{t,\beta}) = \biguplus_{\beta' \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t')}[\beta]} ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow\beta')}[t]_{|\mathsf{Attr}(\downarrow\beta')\setminus\mathcal{B}(\beta')} = ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow\beta)}[t]_{|\mathsf{Attr}(\downarrow\beta)\setminus\mathcal{B}(\beta)}$.

Inductive case 3: t is a join node with children t' and t''.

By definition $rel(g_{t,\beta}) = rel(g_{t',\beta}) \times rel(g_{t'',\beta})$ so by induction $rel(g_{t,\beta}) = ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t')}[\beta]_{|\mathsf{Attr}(\downarrow t')\setminus\mathcal{B}(t')} \times ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t'')}[\beta]_{|\mathsf{Attr}(\downarrow t'')\setminus\mathcal{B}(t'')}$.

By Lemma 4.22 we know that $ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t')}[\beta] \bowtie ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t'')}[\beta] = ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta]$ from which it follows that $ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t')}[\beta]_{|\mathsf{Attr}(\downarrow t)\setminus\mathcal{B}(t)} \bowtie ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t'')}[\beta]_{|\mathsf{Attr}(\downarrow t)\setminus\mathcal{B}(t)} = ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta]_{|\mathsf{Attr}(\downarrow t)\setminus\mathcal{B}(t)}$.

Thus $rel(g_{t,\beta}) = ans^{\mathbb{D}}(Q)_{|\mathsf{Attr}(\downarrow t)}[\beta]_{|\mathsf{Attr}(\downarrow t)\setminus\mathcal{B}(t)}$.

□

Finally we show that G is informed on any bag of T.

Lemma 5.25.
For any node $t \in T$, G is informed on $\mathcal{B}(t)$.

Proof. In this proof, given a gate g of G, we denote by $rel(G, g)$ the set of tuples of $rel(G)$ such that their proof-tree contains g.

We will show by top-down induction on the nodes of T that $rel(G, g_{t,\beta}) = rel(G)[\beta]$ for any $\beta \in rel(G)_{|\mathcal{B}(t)}$.

First of all observe that the lemma will indeed follow from this induction. Let $edgs_{\mathcal{B}(t)}(\beta) = \mathsf{Out}(g_{t,\beta})$. Observe that $rel(G, g_{t,\beta}) = \biguplus_{o\in\mathsf{Out}(g_{t,\beta})} rel(G, o)$ so it follows by the induction hypothesis that $rel(G)[\beta] = \biguplus_{o\in edgs_{\mathcal{B}(t)}(\beta)} rel(G, o)$ thus G is informed on $\mathcal{B}(t)$.

Base case: t is the root of T

Recall that $\mathcal{B}(t) = \emptyset$ so $ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t)} = \{[]\}$. By definition $g_{t,[]}$ is the root of G so $rel(g_{t,[]}) = rel(G) = rel(G)[[]]$.

For the inductive cases we fix a $\beta \in rel(G)_{|\mathcal{B}(t)}$. We distinguish cases based on the type of the parent of t.

Inductive case 1: t is the child of an extend node t'

By construction, because t' is an extend node, the parents of $g_{t,\beta}$ are all the gates $g_{t',\beta'}$ such that $\beta' \in rel(G)[\beta]$ and that $g_{t,\beta}$ is the only child of each of these gates as shown in Figure 5.4c. Thus it follows that $rel(G, g_{t,\beta}) = \biguplus_{\beta'\in rel(G)[\beta]} rel(G, g_{t',\beta'})$ which is equal to $\biguplus_{\beta'\in rel(G)[\beta]} rel(G)[\beta']$ by induction.

Finally by Lemma 4.19 $rel(G, g_{t,\beta}) = rel(G)[\beta]$.

Inductive case 2: t is the child of a project node t'

Let $\beta' = \beta_{|\mathcal{B}(t')} \in rel(G)_{|\mathcal{B}(t')}$. By construction, because t' is a project node, $g_{t,\beta}$ is a child of $g_{t',\beta',\beta(x)}$ which is a child of $g_{t',\beta'}$ as shown in Figure 5.4d. We prove the induction hypothesis by double inclusion.

Let $\tau \in rel(G)[\beta]$. Observe that $rel(G)[\beta] \subseteq rel(G)[\beta']$ so $\tau \in rel(G, g_{t',\beta'})$ by induction. Moreover, because $\tau(x) = \beta(x)$, it follows that $pt_G(\tau)$ contains the leaf labelled with $[x/\beta(x)]$ thus $\tau \in rel(G, g_{t',\beta',\beta(x)})$. Finally, because $o(g_{t',\beta',\beta(x)}) = \times$ it follows that $\tau \in rel(G, g_{t,\beta})$. Thus $rel(G)[\beta] \subseteq rel(G, g_{t,\beta})$.

Now let $\tau \in rel(G, g_{t,\beta})$. Observe that, by the structure of G, $\tau \in rel(G, g_{t',\beta'})$ so by induction $\tau \in rel(G)[\beta']$. Moreover, because $o(g_{t',\beta',\beta(x)}) = \times$, it follows that $pt_G(\tau)$ contains the leaf labelled with $[x/\beta(x)]$ so $\tau(x) = \beta(x)$ so $\tau \in rel(G)[\beta]$. Thus $rel(G, g_{t,\beta}) \subseteq rel(G)[\beta]$.

Finally $rel(G, g_{t,\beta}) = rel(G)[\beta]$ by double inclusion.

Inductive case 3: t is the child of a join node t'

By construction $g_{t,\beta}$ has a single parent $g_{t',\beta}$ as shown in Figure 5.4b. Observe that, because $o(g_{t',\beta}) = \times$ then $rel(G, g_{t,\beta}) = rel(G, g_{t',\beta})$.

Thus $rel(G, g_{t,\beta}) = rel(G)[\beta]$ by induction.

□

Finally the following proposition easily follows frome Lemma 5.24 and Lemma 5.25.

Proposition 5.26.
Let L be a closed $LP(CQ_{qf})_{proj}$ with tree decomposition $\mathcal{T}$.

The collection of circuits $(compile(Q, T_Q, \mathbb{D}))_{Q \in cq(L)}$ is informed on L.

5.5.3 Link to the T-factorized compilation

Let L be a closed $LP(CQ_{qf})_{proj}$ model with tree decomposition $\mathcal{T}$. We have shown in Theorem 4.10 that the T-factorized interpretation of L is equivalent to its natural interpretation. We can now show that, following the previous sections, we are able to compile an informed circuit for any $Q \in cq(L)$ and that the C-factorized interpretation of L is equivalent to its natural interpretation. More interestingly, we will also show that the size of the C-factorized interpretation is similar to that of the T-factorized interpretation.

We begin by showing that compiling a query Q with a yields a concise circuit.

Lemma 5.27.
Let Q be a conjunctive query with a tree decomposition T of width k and $\mathbb{D}$ be a database.

The circuit $G = compile(Q, T, \mathbb{D})$ has $O(|T||\mathbb{D}|^k)$ edges.

Proof. Recall that the construction of G is shown in Figure 5.4.

In order to bound the number of edges of G we consider each type of node t in T. Observe that, by Lemma 2.5, there are at most $|\mathbb{D}|^k$ tuples in $ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t)}$.

Case 1: t is a leaf

By construction no edges are introduced for t.

Case 2: t is a join-node

By construction there are 2 new edges for each $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t)}$ so there are $2|\mathbb{D}|^k$ edges for t.

Case 3: t is a extend-node

By construction there is one new edge for each $\beta \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t)}$ so there are $|\mathbb{D}|^k$ edges for t.

Case 4: t is a project-node

Let t' be the child of t, by construction there are 3 new edges for each $\beta' \in ans^{\mathbb{D}}(Q)_{|\mathcal{B}(t')}$ so there are $3|\mathbb{D}|^k$ edges for t.

□

Theorem 5.28.
Let L be a closed $LP(CQ_{qf})_{proj}$ model with a tree decomposition $\mathcal{T}$ of width k and $\mathbb{D}$ be a database.

The C-factorized interpretation $\rho^{\mathcal{G}}(L)$ with $\mathcal{G} = (compile(Q, T_Q, \mathbb{D}))_{Q \in cq(L)}$ verifies that:

- *$opt(\rho^{\mathcal{G}}(L)) = opt(\langle L \rangle^{\mathbb{D}})$,*
- *$\rho^{\mathcal{G}}(L)$ has $O(|\mathcal{T}| \cdot |\mathbb{D}|^k)$ variables,*
- *$opt(\rho^{\mathcal{G}}(L))$ can be computed in $O(|\rho^{\mathcal{G}}(L)|_b \cdot |\mathcal{T}| \cdot |\mathbb{D}|^{k \cdot \ell})$ in combined complexity with $|\rho^{\mathcal{G}}(L)|_b = O((|L| + |\mathcal{T}|) \cdot |\mathbb{D}|^k \cdot \log(|\mathbb{D}|^k))$,*
- *$opt(\rho^{\mathcal{G}}(L))$ can be computed in $O(|\mathbb{D}|^{k \cdot (\ell+1)} \cdot \log(\mathbb{D}))$ in data complexity.*

Proof. We know by Proposition 5.26 that $\mathcal{G}$ is informed on L so it follows from Theorem 5.10 that the first item of this theorem holds.

The second item follows from Lemma 5.27. Indeed the number of variables of $\rho^{\mathcal{G}}(L)$ is equal to the number of edges of $\mathcal{G}$ by definition.

The combined complexity follows from Theorem 5.10 with $|\mathcal{G}| = O(|\mathcal{T}| \cdot |\mathbb{D}|^k)$ and $I_{\mathcal{G}} = |\mathbb{D}|^k$. Indeed recall that in this case we defined $edgs_X(\beta)$ to be $\mathsf{Out}(g_{t,\beta})$. Observe that $\mathsf{Out}(g_{t,\beta})$ only contains more than one edge when t is the child of an extendt'. The gate $g_{t,\beta}$ has a parent $g_{t',\beta'}$ for each extension β' of β. There are at most $|\mathbb{D}|^k$ such extensions so $I = |\mathbb{D}|^k$.

Finally the data complexity easily follows by considering $|L|$ and $|\mathcal{T}|$ to be constants. □

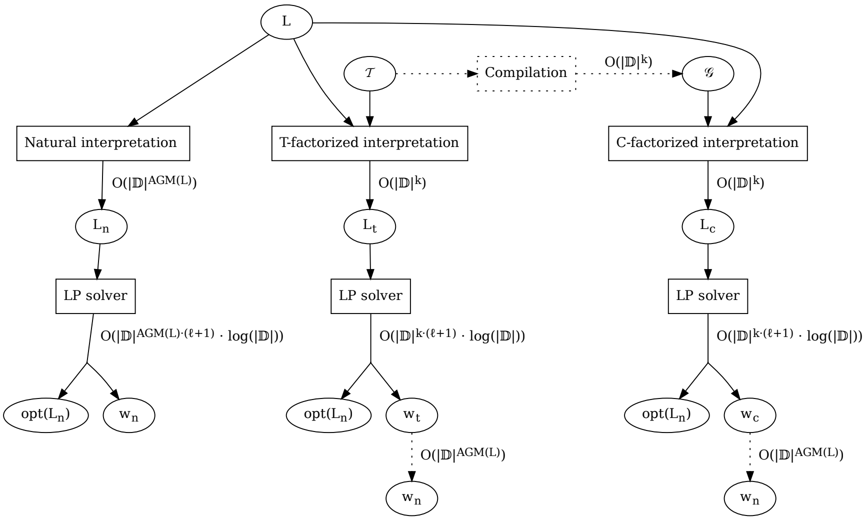

L
$\mathcal{T}$
Compilation
$O(|\mathbb{D}|^{k})$
$\mathcal{G}$
Natural interpretation
T-factorized interpretation
C-factorized interpretation
$O(|\mathbb{D}|^{AGM(L)})$
$O(|\mathbb{D}|^{k})$
$O(|\mathbb{D}|^{k})$
L_n
L_t
L_c
LP solver
LP solver
LP solver
$O(|\mathbb{D}|^{AGM(L)\cdot(\ell+1)} \cdot \log(|\mathbb{D}|))$
$O(|\mathbb{D}|^{k\cdot(\ell+1)} \cdot \log(|\mathbb{D}|))$
$O(|\mathbb{D}|^{k\cdot(\ell+1)} \cdot \log(|\mathbb{D}|))$
$opt(L_n)$
w_n
$opt(L_n)$
w_t
$opt(L_n)$
w_c
$O(|\mathbb{D}|^{AGM(L)})$
$O(|\mathbb{D}|^{AGM(L)})$
w_n
w_n

Figure 5.5: Summary and data complexity of the solving process of a closed $LP(CQ)$ model L

Observe that it follows from this theorem that the C-factorized interpretation covers every possible use of the T-factorized interpretation. The C-factorized interpretation is actually more general as it can be applied in wider situations on which the T-factorized could not be applied, we will see an example of such a situation in Section 6.2.

5.6 Conclusion

In this chapter we introduced relational circuits and $\{\uplus, \times\}$-circuits as means to efficiently encode relations. We then defined a succinct C-factorized interpretation of informed closed $LP(CQ)$ models based on $\{\uplus, \times\}$-circuits and proved the correctness of this interpretation. Finally we recaptured the result of the previous chapter by showing that we were able to compile the answer sets of the queries of $LP(CQ)_{proj}$ programs into succinct $\{\uplus, \times\}$-circuits.

In Figure 5.5, we give a summary of the solving process with the three interpretations we defined in this thesis as well as their compared performances in data complexity by considering $|L|$ and $|\mathcal{T}|$ to be constants. On the left-hand side we recall the natural and T-factorized interpretation processes we already presented in Figure 4.4. On the right-hand side we represent the C-factorized interpretation of L to L_c using a collection of circuits $\mathcal{G}$ which may be obtained by a compilation based on tree decompositions $\mathcal{T}$of width k. It then yields the optimal value and an optimal solution w_c of L_c from which we can reconstruct an optimal solution w_n of the natural interpretation L_n.

Chapter 6

Extensions and limitations

Contents

In this section we present a few interesting ideas to extend our results in two different areas as well as their (current) limitations. We first present a few extensions from an optimization problem point of view, then we consider enriching the query language on which the $LP(CQ)$ language is based.

6.1 Going beyond linear programs

Throughout this thesis we have specifically targeted linear programs with non negative real values. In this section we identify three variations of this setting, namely lifting the restriction to non negative values, solving programs with integer values and the relaxing the linearity of the programs.

In the following sections, we either describe how our results can be extended to lift to these variations or leave them as open problems.

6.1.1 Allowing variables to take negative values

A severe restriction throughout this thesis is that we only consider linear program variables that take non-negative values. This restriction is necessary to prove the weighting correspondences from which the equivalence of the factorized interpretations with the natural semantics follows. More specifically, parts of our proofs rely on the fact that the total weight of a subset of variables is 0 iff the individual weight of each variable is 0. In this section we will thus discuss the problem of solving $LP(CQ)$ models in $\mathbb{R}$ rather than in $\mathbb{R}^+$. For clarity we introduce a new notation $opt_S(L)$ to denote the optimal value of L when solved with values in S. Observe that, while we have implicitly denoted $opt_{\mathbb{R}^+}(L)$ by $opt(L)$ throughout this thesis, we are now interested in computing $opt_{\mathbb{R}}(L)$.

In linear programming it is well known that a linear program with real values can be equivalently rewritten to a standard form with non-negative values. Given a linear program L, the idea is to introduce two new variables ξ_x^+ and ξ_x^- such that $\xi_x^+ \geq 0$, $\xi_x^- \geq 0$ for any variable x of L then replace every occurrence of x with the difference $(\xi_x^+ - \xi_x^-)$ yielding a new linear program L'. It is well known that $opt_{\mathbb{R}}(L) = opt_{\mathbb{R}^+}(L')$. Moreover, given an optimal solution $w' : var(L') \to \mathbb{R}^+$ it is easy to reconstruct an optimal solution $w : var(L) \to \mathbb{R}$ by computing the difference $w(x) = w'(\xi_x^+) - w'(\xi_x^-)$.

It turns out that this idea can easily be lifted to $LP(CQ)$ models without losing the ability to use the factorized interpretations. First we define the standard form of a $LP(CQ)$ model as follows:

Definition 6.1.
Let L be a closed $LP(CQ)$ model. For each query $Q \in cq(L)$ let $Q^- = Q \wedge true$.

We denote the standard form of L by $\mathsf{sf}(L)$, it is the closed $LP(CQ)$ model obtained by replacing every $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ in L with $\mathbf{weight}_{\mathbf{x}:Q'}(Q) - \mathbf{weight}_{\mathbf{x}:Q'}(Q^-)$.

Observe that in this definition, each Q^- is semantically equivalent but syntactically different from its associated query Q.

We now show that solving $\mathsf{sf}(L)$ in $\mathbb{R}^+$ is equivalent to solving L in R.

Lemma 6.2.
Given a closed $LP(CQ)$ model L, $opt_{\mathbb{R}^+}(\langle \mathsf{sf}(L) \rangle^{\mathbb{D}}) = opt_{\mathbb{R}}(\langle L \rangle^{\mathbb{D}})$.

Sketch of proof. By definition, when computing $\langle \mathsf{sf}(L) \rangle^{\mathbb{D}}$, the instantiation of $\mathbf{weight}_{\mathbf{x}:Q'}(Q) - \mathbf{weight}_{\mathbf{x}:Q'}(Q^-)$ is $\sum_{\tau \in ans_{\mathbf{x}}^{\mathbb{D}}(Q \wedge Q')} \theta_Q^\tau - \sum_{\tau \in ans_{\mathbf{x}}^{\mathbb{D}}(Q^- \wedge Q')} \theta_{Q^-}^\tau$ which can be rewritten as $\sum_{\tau \in ans_{\mathbf{x}}^{\mathbb{D}}(Q \wedge Q')} (\theta_Q^\tau - \theta_{Q^-}^\tau)$ since $ans^{\mathbb{D}}(Q) = ans^{\mathbb{D}}(Q^-)$. Observe that because Q and Q^- are syntactically different then θ_Q^τ and $\theta_{Q^-}^\tau$ are distinct variables that can take the roles of $\xi_{\theta_Q^\tau}^+$ and $\xi_{\theta_Q^\tau}^-$.

Thus $\langle \mathsf{sf}(L) \rangle^{\mathbb{D}}$ is the standard form of $\langle L \rangle^{\mathbb{D}}$ so the lemma holds. □

Moreover this approach can be used to efficiently solve L on $\mathbb{R}$. We formalize this for the T-factorized interpretation, it is then straightforward to see that the C-factorized interpretation can be used for similar results.

Lemma 6.3.
Let L be a $LP(CQ_{qf})_{proj}$ model, $\mathcal{T}$ a decomposition of L of width k and $\mathbb{D}$ a database.

One can compute $opt_{\mathbb{R}}(\langle L \rangle^{\mathbb{D}})$ in $O(|\mathbb{D}|^{\ell \cdot \mathsf{fhtw}(L)})$ with $\ell < 2.37286$.

Proof. Observe that $\mathsf{sf}(L)$ is also a $LP(CQ_{qf})_{proj}$ model since we can use the decomposition $T_Q \in \mathcal{T}$ for each duplicated query Q^-. Thus by Theorem 4.10, $opt_{\mathbb{R}^+}(\rho^{\mathcal{T},\mathbb{D}}(\mathsf{sf}(L))) = opt_{\mathbb{R}^+}(\langle \mathsf{sf}(L) \rangle^{\mathbb{D}})$. Finally, by Lemma 6.2, $opt_{\mathbb{R}^+}(\rho^{\mathcal{T},\mathbb{D}}(\mathsf{sf}(L))) = opt_{\mathbb{R}}(\langle L \rangle^{\mathbb{D}})$.

For the complexity, observe that $\rho^{\mathcal{T},\mathbb{D}}(\mathsf{sf}(L))$ has twice as many variables as $\rho^{\mathcal{T},\mathbb{D}}(L)$ so $\rho^{\mathcal{T},\mathbb{D}}(\mathsf{sf}(L))$ can be solved in $O(2|\mathbb{D}|^{\ell \cdot \mathsf{fhtw}(L)})$. □

Observe that Lemma 4.13 can be lifted to this setting by computing the differences $\theta_Q^\tau - \theta_{Q^-}^\tau$.

6.1.2 Solving programs on integer values

While we have considered linear program variables to take (positive) real values, in some settings it makes more sense to restrict the variables to integer values in order to reason about discrete objects. Such programs are know as *Integer Linear Programs (ILPs)* or *Mixed Integer Programs (MIPs)* if there are both integer and real variables.

Since an ILP only differs from a LP in the domain of its variables, the $LP(CQ)$ language can directly be used to describe ILPs. Indeed, we can simply compute $\langle L\rangle^{\mathbb{D}}$ then hand it to an ILP solver to obtain $opt_{\mathbb{N}}(\langle L\rangle^{\mathbb{D}})$ (the technique from the previous section can be used if one is interested in $opt_{\mathbb{Z}}(\langle L\rangle^{\mathbb{D}})$).

Unfortunately the T-factorized and C-factorized interpretations cannot be directly used to compute $opt_{\mathbb{Z}}(\langle L\rangle^{\mathbb{D}})$ more efficiently. Indeed recall that half of the proof of the correctness of both these interpretations relies on reconstructing a solution of $\langle L\rangle^{\mathbb{D}}$ with the $\Pi(.)$ function that makes use of divisions and would thus often yield real values rather than the expected integer values.

We leave finding a succinct interpretation of $LP(CQ)$ models that can be equivalently solved with integer values as an open problem.

6.1.3 Relaxing linearity

Finally, while linear programs cover a wide array of applications, they are only a subset of optimization problems and it may be useful to relax the linearity of the programs.

Consider for instance the task assignment example from Example 3.2. We could minimize the squares of the assignments of the employees in order to obtain more balance between the workloads of the employees as follows:

Example 6.4 (Updated task assignment model)**.**

$$\begin{array}{rl} \textbf{\textit{minimize}} & \displaystyle\sum_{(e'):\exists s'.Skills(e',s')} \mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign})^2 \\ \textbf{\textit{subject to}} & \forall(t',d'){:}\exists s'.Tasks(t',s',d').\mathbf{weight}_{(t,s,e,d):t\doteq t'}(Q_{assign}) \geq \mathbf{num}(d') \\ \wedge & \forall(e'){:}\exists s'.Skills(e',s').\mathbf{weight}_{(t,s,e,d):e\doteq e'}(Q_{assign}) \leq 40. \end{array}$$

It is possible to extend the core idea of the $LP(CQ)$ language to generic optimization problems by considering models of the form

$$\begin{array}{rl} \textbf{minimize} & f(\mathbf{x}) \\ \textbf{subject to} & \bigwedge_{i=1}^{m} g_i(\mathbf{y}_i) \leq 0 \\ & \bigwedge_{j=1}^{n} h_j(\mathbf{z}_j) = 0 \end{array}$$

where $\mathbf{x}, \mathbf{y}_i$ and $\mathbf{z}_j$ are vectors of **weight** operators.

It is easy to see that one can compute the natural interpretation of such models, and that the only limitation is the existence of an algorithm to solve the resulting optimization problem.

Moreover it turns out that, if applicable, the T-factorized and C-factorized interpretations of such models are still equivalent with their natural semantics. Intuitively the base building blocks of this relaxed language are still the **weight** operators so the framework defined in Section 3.6 translates quite easily to this new setting.

Indeed it is straightforward that we can lift Lemma 3.15 to this setting as follows:

Lemma 6.5.
Let f be a function and $\vec{\mathbf{w}}$ be a vector of **weight** *operators.*

If $w_1(Inst_1(\mathbf{w})) = w_2(Inst_2(\mathbf{w}))$ for any $\mathbf{w} \in \vec{\mathbf{w}}$ then $[\![Inst_1(f(\vec{\mathbf{w}}))]\!]_{w_1} = [\![Inst_2(f(\vec{\mathbf{w}}))]\!]_{w_2}$.

It then follows that we can also lift Proposition 3.17 to this setting thus the T-factorized interpretation and C-factorized interpretation can indeed be used to solve these models.

For example we remark that the model L of Example 6.4 describes a *least squares* problem, so a factorized interpretation could be handed to a solver with least squares capabilities in order to compute $opt(\langle L \rangle^{\mathbb{D}})$ more efficiently by reducing the number of variables to be handled by the solver.

6.2 Going beyond conjunctive queries

We now discuss how to extend the $LP(CQ)$ language by lifting the restriction to conjunctive queries. Observe that since we will always consider queries Q that describe a relation $ans^{\mathbb{D}}(Q)$, the natural interpretation will easily be translateable to these new types of queries. On the other-hand, since the tree decomposition based algorithms are specifically tied to conjunctive queries, the T-factorized interpretation will cease working with any of these extensions. Thus we will mostly discuss the impact of these changes on the C-factorized interpretation and see that the additional abstraction sometimes allows it to be more general than the T-factorized interpretation. While there are a lot of different ways to extend conjunctive queries up to first order queries, we only present two specifically interesting extensions here.

UCQs

First we consider a common extension by adding disjunctions $\vee$ to the syntax of CQ_Σ. This allows us to describe a class of queries commonly referred to as *Unions of Conjunctive Queries* (UCQs) from which we can define the $LP(UCQ)$ language of linear programs with UCQs in a straightforward manner.

Let $Q = R(x) \vee S(y)$ be a UCQ and $\mathbb{D}$ be a database of domain $\{0, 1, 2\}$ with the following tables:

$R^{\mathbb{D}}$	x
	0

$S^{\mathbb{D}}$	y
	1

Observe that Q indeed defines a relation $ans^{\mathbb{D}}(Q) = ans^{\mathbb{D}}_{fv(Q)}(R(x)) \cup ans^{\mathbb{D}}_{fv(Q)}(S(y))$ as follows so the natural interpretation can be seamlessly ported to the $LP(UCQ)$ language.

$ans^{\mathbb{D}}(Q)$	x	y
	0	0
	0	1
	0	2
	1	1
	2	1

Observe that if we consider a UCQ $Q = Q_1 \vee \cdots \vee Q_i \vee \cdots \vee Q_n$ then $ans^{\mathbb{D}}(Q) = ans^{\mathbb{D}}(Q_1) \cup \cdots \cup ans^{\mathbb{D}}(Q_i) \cup \cdots \cup ans^{\mathbb{D}}(Q_n)$. Thus a first idea to port the C-factorized interpretation to the $LP(UCQ)$ language would be to compile each individual query Q_i into a $\{\uplus, \times\}$-Circuit as presented in Section 5.5.1 then consider the union of these circuits. However these circuits are not guaranteed to describe disjoint relations so the full circuit that describes Q would not necessarily be a $\{\uplus, \times\}$-Circuit itself. Thus the C-factorized interpretation we presented in this thesis would not directly work for UCQs, we leave finding some form of C-factorized interpretation that doesn't rely on determinism as an open question.

CQs with negations

We now consider the extension of conjunctive queries to *CQs with negations* (CQ_{neg}) where we allow atoms to be negated where a negated atom represents the complement of the relation in the domain.

For example, let $Q = R(x) \wedge \neg S(y)$ be a conjunctive query and $\mathbb{D}$ be a database of domain $\{0, 1, 2\}$ with the following tables:

$R^{\mathbb{D}}$	x
	0

$S^{\mathbb{D}}$	y
	1

The query Q describes the following relation, from which if follows that the natural interpretation can also be seamlessly ported to the $LP(CQ_{neg})$ language of linear programs with conjunctive queries with negations.

$ans^{\mathbb{D}}(Q)$	x	y
	0	0
	0	2

Interestingly, the result from [Cap17] could be adapted to compile some CQ_{neg} queries to $\{\uplus, \times\}$-circuits as follows:

Theorem 6.6.
Given a β-acyclic CQ_{neg} query Q and database $\mathbb{D}$, there is a $\{\uplus, \times\}$-circuit G of size $O(|D| \cdot |Q|)$ such that $rel(G) = ans^{\mathbb{D}}(Q)$.

While finding an informed compilation algorithm for CQ_{neg} queries would be more involved, recall that any $\{\uplus, \times\}$-circuit is necessarily informed on single variables. Thus the C-factorized interpretation could be applied to $LP(CQ_{neg})$ models whose **weight** operators are of the form $\mathbf{weight}_{\mathbf{x}:x \doteq d}(Q)$ with $x \in \mathbf{x}$ and d a domain value.

Observe that it is possible to rewrite a UCQ into a union of disjoint queries by using negations to make sure no answer is selected twice. While this opens another approach to handle UCQs, the rewritten query could grow to be much larger than the original query which would lead to decreased efficiency. Moreover, this approach would fall prey to a loss of expressiveness because of the restricted informedness mentioned above.

Chapter 7
Conclusion

In this thesis we have defined the $LP(CQ)$ language to model linear programs that reason about the answers of conjunctive queries without materializing the answer sets of these conjunctive queries beforehand. Because solving $LP(CQ)$ models is hard in general, we presented the T-factorized interpretation to efficiently solve some closed $LP(CQ)$ models using hypertree decompositions of the queries that appear in the models. We then generalized this approach with the C-factorized interpretation that relies on $\{\uplus, \times\}$-circuits that succinctly encode the answer sets of the queries of the models. Finally we presented some extensions that take our results further.

We now list a few open questions that follow from this thesis, some of which were already mentioned in the previous chapter.

As mentioned in the previous chapter, it would be interesting to investigate whether it is possible to efficiently compute $opt_{\mathbb{Z}}(\langle L\rangle^{\mathbb{D}})$ for a fragment of $LP(CQ)$ models L given a database $\mathbb{D}$.

In the previous chapter we discussed the extension of the $LP(CQ)$ language to the $LP(UCQ)$ language and left the question of efficiently solving $LP(UCQ)$ models as an open problem. A first lead to solve this question would be to try and adpating the C-factorized interpretation to work with circuits that are not not deterministic.

While we have focused on efficiently solving *closed* $LP(CQ)$ models throughout this thesis, trying to close $LP(CQ)$ models more efficiently would also be an interesting line of research.

Finally it would be interesting to implement the $LP(CQ)$ language, possibly as an extension of an existing modeling language, as well as our T-factorized and C-factorized solving algorithm to see how they would fare on practical problems.

Bibliography

[AGM13] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013. (Cited on page 12.)

[AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021. (Cited on page 13.)

[BDG07] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *International Workshop on Computer Science Logic*, pages 208–222. Springer, 2007. (Cited on pages 4, 5 and 64.)

[Cap17] Florent Capelli. Understanding the complexity of #sat using knowledge compilation. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–10. IEEE Computer Society, 2017. (Cited on page 95.)

[CCNR22] Florent Capelli, Nicolas Crosetti, Joachim Niehren, and Jan Ramon. Linear programs with conjunctive queries. In Dan Olteanu and Nils Vortmeier, editors, *25th International Conference on Database Theory, ICDT 2022, March 29 to April 1, 2022, Edinburgh, UK (Virtual Conference)*, volume 220 of *LIPIcs*, pages 5:1–5:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on page 4.)

[CLS21] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021. (Cited on page 13.)

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 77–90, New York, NY, USA, 1977. ACM. (Cited on page 36.)

[CM07] Marco Cadoli and Toni Mancini. Combining relational algebra, sql, constraint modelling, and local search. *Theory and Practice of Logic Programming*, 7(1-2):37–65, 2007. (Cited on page 3.)

[Dar01] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, July 2001. (Cited on pages 65 and 66.)

[DM02] Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002. (Cited on pages 65 and 66.)

[FGK90] Robert Fourer, David M Gay, and Brian W Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990. (Cited on page 2.)

[GLS99] Georg Gottlob, Nicola Leone, and Francesco Scarcello. On tractable queries and constraints. In *International Conference on Database and Expert Systems Applications*, pages 1–15. Springer, 1999. (Cited on page 4.)

[GLS02] G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable Queries. *Journal of Computer and System Sciences*, 64(3):579–627, May 2002. arXiv: cs/9812022. (Cited on page 4.)

[GM14] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms (TALG)*, 11(1):4, 2014. (Cited on page 11.)

[Gro06] Martin Grohe. The structure of tractable constraint satisfaction problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 58–72. Springer, 2006. (Cited on page 4.)

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Comb.*, 4(4):373–396, 1984. (Cited on pages 4 and 13.)

[KLMT15] Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. Compiling constraint networks into multivalued decomposable decision graphs. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 332–338. AAAI Press, 2015. (Cited on page 66.)

[Klo94] Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994. (Cited on page 12.)

[KPT13] Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. Efficient querying of inconsistent databases with binary integer programming. *Proceedings of the VLDB Endowment*, 6(6):397–408, April 2013. (Cited on page 3.)

[Lib13] Leonid Libkin. *Elements of finite model theory.* Springer Science & Business Media, 2013. (Cited on page 11.)

[MS12] Alexandra Meliou and Dan Suciu. Tiresias: The database oracle for how-to queries. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 337–348, New York, NY, USA, 2012. ACM. (Cited on page 3.)

[NSB$^+$07] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007. (Cited on page 2.)

[Olt16] Dan Olteanu. Factorized databases: A knowledge compilation perspective. In *AAAI Workshop: Beyond NP*, 2016. (Cited on pages 41, 65 and 66.)

[OZ12] Dan Olteanu and Jakub Závodnỳ. Factorised representations of query results: size bounds and readability. In *Proceedings of the 15th International Conference on Database Theory*, pages 285–298. ACM, 2012. (Cited on pages 5, 41 and 64.)

[OZ15a] Dan Olteanu and Jakub Závodnỳ. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):1–44, 2015. (Cited on page 5.)

[OZ15b] Dan Olteanu and Jakub Závodný. Size Bounds for Factorised Representations of Query Results. *ACM Transactions on Database Systems*, 40(1):1–44, March 2015. (Cited on pages 64 and 80.)

[PD08] Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI*, pages 517–522, 2008. (Cited on page 65.)

[PS13] Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 79:984–1001, September 2013. (Cited on pages 4 and 64.)

[SOC16] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, pages 3–18. ACM, 2016. (Cited on page 64.)

[ŠP16] Laurynas Šikšnys and Torben Bach Pedersen. SolveDB: Integrating optimization problem solvers into SQL databases. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, page 14. ACM, 2016. (Cited on page 3.)

[Vel14] Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, pages 96–106. OpenProceedings.org, 2014. (Cited on pages 11 and 36.)

[Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data*

Bases - Volume 7, VLDB '81, pages 82–94. VLDB Endowment, 1981. (Cited on pages 4 and 41.)