FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Query Answering Through Reformulation

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

European Master's Program in Computational Logic

eingereicht von

Yue Chen
Matrikelnummer 1128567

an der Fakultät für Informatik

der Technischen Universität Wien
Betreuung: Prof. Dr. Reinhard Pichler, Prof. Dr. Enrico Franconi
Mitwirkung: Ngo Phuong Nhung, MSc

Wien, 8. Jänner 2016

Yue Chen

Reinhard Pichler

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Query Answering Through Reformulation

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

European Master's Program in Computational Logic

by

Yue Chen
Registration Number 1128567

to the Faculty of Informatics

at the Vienna University of Technology
Advisor: Prof. Dr. Reinhard Pichler, Prof. Dr. Enrico Franconi
Assistance: Ngo Phuong Nhung, MSc

Vienna, 8th January, 2016

Yue Chen

Reinhard Pichler

Erklärung zur Verfassung der Arbeit

Yue Chen
Schrickgasse 18A/14, 1220 Vienna, Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 8. Jänner 2016

Yue Chen

Acknowledgements

I thank Professor Enrico Franconi of Free University of Bolzano for his generosity in sharing his inspiring thoughts with me, which encourages me to attempt, carefully but excitingly, the freedom outside $\mathcal{A}$ box and its accompanied complexity charts. He first taught me to withdraw from the honest toils in conjunctive queries and to harvest a just reward in first order logic. He has taught me to hope that by nothing more than logic and patience one can achieve the holy grail of AC^0 without scrutinizing the landscape of $DL\text{-}Lite$ families, moreover, he has taught me to hope stubbornly. Without these stubborn hope this study would not have been started. I would also like to thank Ngo Phuong Nhung for her good hearted grace in helping me through the technical difficulties at my first attempts to understand the material that I study, for the chance she has provided me to observe that rare combination of greatness of mind and goodness of heart, which so delights me, and for her perseverance in her sustaining integrity, though it has never occurred to me that she in any way be constrained by them, which makes it so much greater. I am also grateful to Professor Reinhard Pichler for his supervision at TU Vienna and the useful suggestions he made to my advantage. Finally, I thank all the humans that have worked hard to make the online distant communication possible, which allows our thoughts to travel freely without the inconvenient bodies.

Kurzfassung

Diese Arbeit untersucht verschiedene Lösungsansätze zur Anfragenbeantwortung mittels effizienter Pläne. Effiziente Pläne können mittels SQL Techniken implementiert werden, daher stellt diese Arbeit eine Applikation für Datenbank Technologien dar. Die Reformulierung einer Anfrage hilft dabei, Ausdrücke einer Anfrage auf vollständing zugänglichen Datenbank Tabellen zu limitieren, so dass solch ein Ausdruck einfach in einen effizienten Plan konvertiert werden kann. Alle hier präsentierten Zugänge funktionieren anhand ihrer eigenen Bedingungen. Sie teilen sich jedoch die fundamentale Strategie des Verbindens eines Beweises mit einem Plan. Der Beweis ist der einer Beantwortbarkeit, gleichbedeutend mit der Existenz einer Reformulierung der Anfrage in einem bestimmten fixierten Vokabular, und der Plan ist eine ausführbare Anfrage, die aus einem Beweis mittels Interpolationstechnik extrahiert wird.

Die Klassifizierung der Herangehensweisen kann auf dem sie in Angriff nehmenden Anfragebeantwortungsproblems, oder aber auf der Art des Beweises, dass sie ausnutzt, basiert werden. Die ersten drei Kapitel sind *grosso modo* dem Problem der ontologie-basierten Anfragebeantwortung gewidmet. Daher sind die Bedingungen, unter welche dieser Ansatz funktioniert, solche, die die Ontologie und ihre Beziehung zur Anfrage einschränken, und hierbei besonders das zum Einsatz kommende Datenmodells der Ontologie, eines $\mathcal{D}$ Box Datenmodells anstatt des in Beschreibungslogiken gebräuchlichen $\mathcal{A}$ Box Datenmodells, welches die Verwendung von Reformulierungstechniken anspornt. Die letzten beiden Kapitel beschäftigen sich mit einem geringfügig abweichenden Problem: Anfragebeantwortung mittels eingeschränkter Schnittstelle. Hier gibt es keine Ontologie, vielmehr Einschränkungen des Datenzugriffs und der üblichen Integritätsbedingungen. So ist es essenziell immer noch ein Problem des Beantwortens einer Anfrage angesichts vorhandener Beschränkungen. Die Technik zum Lösen dieses Problems wurde bereits vor langer Zeit in Untersuchungen über die Prädikatenlogik in Form von Präservierungs- und Interpolationstheoremen eingeführt. Deren Anwendung in der Anfragebeantwortung hängt jedoch von der Verwendung

dieser Techniken zur Reformulierung der ursprünglichen Anfrage in eine andere Anfrage, welche vorhandene Datenbanktechniken handhabbarer nutzt.

Daher übergreifend aller Herangehensweisen, das Ziel der Reformulierung ist immer eine ausführbare Anfrage, unabhängig davon, ob sich die Ausführbarkeit in einer sicheren Bereichsanfrage manifestiert, oder aber in einem Plan einer relationalen Algebra. Am Ende von Kapitel Fünf präsentiere ich einen Algorithmus zum Extrahieren einer Selektions-Projektions-Join-Anfrage anhand eines vorwärts verketteten Beweises basierend auf Tupel-generierender Abhängigkeiten als Beschränkungen. Obgleich der Beweis hier mittels einer Chase Folge repräsentiert wird, ist es immer noch ein Implikationsbeweis, und der Plan wird trotzdem noch mittels Interpolation extrahiert.

Abstract

This thesis is a study on different approaches towards answering query via efficient plans. Efficient plans can be implemented by SQL technique, hence the application to database technology. Reformulation of query helps to limit the query expression to fully accessible database tables so that such an expression can be easily translated into an efficient plan. The approaches presented here function under their own conditions. However, they share a fundamental strategy of connecting proof with plan. The proof is an answerability proof, signifying the existence of a reformulation of the query in a certain fixed vocabulary, and the plan is an executable query extracted from the proof by interpolation technique.

The classification of approaches can be based on the query answering problem that they address, or it can be based on the kind of proof they exploit. Roughly speaking, the first three chapters are devoted to the problem of ontology based query answering. Therefore the conditions under which the approach functions are those constraining the ontology and its relation to the query, especially the data model employed by the ontology, a $\mathcal{D}$ box data model instead of the more description-logic-conventional $\mathcal{A}$ box data model, which motivates the use of reformulation technique. The last two chapters can be viewed as addressing a slightly different problem at hand: query answering over restricted interface. There is no ontology here, rather restrictions on data access and the usual integrity constraints. So it is essentially still the problem of answering query in face of constraints. The technique for solving this problem has been introduced long time ago in the study of predicate logic in the form of preservation and interpolation theorems. However, its application in answering queries depends on the use of these techniques to reformulate the original query into some other query that is more manageable using database techniques. Therefore across the approaches the goal of reformulation is always an executable query, whether this executability reveals itself in a safe range executable query or a relational algebra plan. At the end of chapter five I present an algorithm for extracting a Select-Project-Join-plan from a forward chaining proof given tuple generating dependencies as

constraints. Though the proof here is represented by a chase sequence, it is still an entailment proof and the plan is still extracted by interpolation.

Contents

CHAPTER 1

Introduction

Query answering over relational database has been well studied for decades, whereas query answering with constraints over database is a relatively new subject. When the constraints are viewed from the perspective of ontology, it is a subject that straddles the common ground between description logic and ontology on the one hand, and query answering on the other. However, there is another way of dealing with database constraints which stays more within the traditional database technology of answering queries. It aims at using the constraints to reformulate the query so that its rewritten form can be translated into a plan that renders itself implementable in SQL. This idea of reformulation through constraints has been investigated not just from the ontological point of view, but also from interests in other constraints that we often encounter in practical database applications, such as restricted interface of databases. This thesis offers some understanding into the idea of reformulation through constraints, along with its advantages and limits.

1.1 Active Domain Semantics and Domain Independence

Conventional semantics of relational calculus has quantified variables ranging over all elements of the underlying domain, i.e., **Dom**. However, in the classical definition of first order query semantics, the natural semantics of first order theory is generalized to allow explicit specification of the underlying domain. The semantics thus generated for first order query ϕ is a relativized instance [Abiteboul et al., 1994] over database signature schema **R**, which is a pair (**d**, **I**), where **I**

is a database instance over **R**, and $adom$(**I**, ϕ) $\subseteq$ **d** $\subseteq$ **Dom**. As usual, $adom$(**I**, ϕ) denotes $adom$(**I**) $\cup$ $adom$(ϕ). $adom$(**I**) is the set of all constants occurring in **I**, and $adom$(ϕ) is defined analogously. A relational calculus query ϕ is interpretable over (**d**, **I**) if $adom(\phi) \subseteq$ **d**. If v is a valuation over *free*(ϕ) with range contained in **d**, then **I** satisfies ϕ for v relative to **d**, denoted $\mathbf{I} \models_{\mathbf{d}} \phi[v]$. The recursive definition along the structure of ϕ is defined as in the usual way for relational calculus.

Let $Q = \{(x_1, \ldots, x_n) \mid \phi\}$ be a relational calculus query over schema **R**, and let (**d**, **I**) be a relativized instance over **R**. Then the answer of Q over the instance **I** of **R** is denoted as follows:

$$Q_{\mathbf{d}}(\mathbf{I}) = \{v(x_1, \ldots, x_n) \mid \mathbf{I} \models_{\mathbf{d}} \phi[v]\} \ .$$

The queries under discussion here are all interpretable over relativized instance (**d**, **I**). Given $adom(\mathbf{I}, \phi) \subseteq \mathbf{d} \subseteq \mathbf{Dom}$, the natural (unrestricted) interpretation of Q on **I** is denoted $Q_{\mathbf{Dom}}$(**I**); and the active domain interpretation of Q on **I** is denoted $Q_{adom(Q,\mathbf{I})}$(**I**), with shorthand Q_{adom}(**I**).

The active domain semantics as defined above is the general assumption associated with database queries. However, for reasons that shall unfold later, an alternative approach to the semantics of relational calculus query is domain independence with standard name assumption. Without further specification, we are talking about relational calculus query. A query Q is domain independent if for each instance **I**, and **d**, **d**′ $\subseteq$ **Dom**, $Q_{\mathbf{d}}(\mathbf{I}) = Q_{\mathbf{d}'}(\mathbf{I})$.

By Codd's equivalence theorem, we can prove that the domain independent calculus and the calculus under active domain semantics are equivalent in their expressive power. That is, for every domain independent calculus query there is an equivalent query in the domain calculus under the active domain semantics. However, there is one point worthy of remark. Domain calculus covers all the queries under discussion here, where variables range over the underlying domain of values, i.e., **d**. Though $adom$(**I**, Q) $\subseteq$ **d**, it is not guaranteed that for an arbitrary element a in $adom$(Q), we have $\mathbf{I}(a) = a$. This might cause problems when we use domain independent queries such as $Q = \{x \mid x = a\}$. x is guarded by a, therefore Q is domain independent, that is, Q(**I**) = Q_{adom}(**I**). However, if we do not assume standard name, then it is possible that Q(**I**) = b where b satisfies the condition that $b \in$ **Dom**, $b \neq a$ and $b \notin adom(\mathbf{I}, Q)$. In this case, we just have to let $\mathbf{I}(a) = b$, then the answer of Q under **I**, i.e., b, is in the active domain no more. Hence, to achieve active domain semantics from domain independence, we must also have standard name assumption (SNA). This being said, and given that we

are mostly dealing with domain independent queries, we can, without essential loss, use natural interpretation of first order logic with SNA, instead of classical domain calculus semantics.

1.2 $\mathcal{A}$ Box or $\mathcal{D}$ Box: A Question with Ontology

Now we take ourselves to the less classical question of answering queries over relational databases with ontology. As we know, ontology, the meaningful ones especially, often contains predicate vocabularies that do not exist in the underlying database. Therefore the first question we encounter in handling query answering with ontology is to ask, how we may model the data so as to make our query understandable in it? In fact, ontology based query answering makes us query the modeled data, not the original information lying in the base tables. In face of this question, there are two streams of thought. They can simply be differentiated by the way they model their data with respect to the original database. To put it in a nutshell, one models it incompletely, and the other completely. This means, in the incomplete data model or sound view, as it is sometimes called, for any database predicate, its interpretation in the data model is *contained* in its original interpretation, i.e., that in the original database. On the other hand, the complete data model, or exact view correspondingly, has as interpretation of every database predicate exactly the *same* set of tuples as in its original interpretation. A slightly more formal definition is given below in the name of $\mathcal{A}$ box and $\mathcal{D}$ box for the two kinds of data model.

Ontology imposes constraints on the data we use to answer queries. Given a database instance $\mathbf{I}$ with constraints, there are two ways in which we can answer a query Q based on materialized views $\mathcal{V}$, depending on whether the view instance I_V is generated from $\mathbf{I}$ on open or closed world assumption. If we perform query answering with constraints on open world assumption, then $I_V \subseteq \mathcal{V}(\mathbf{I})$; and if, on the other hand, our query answering is based on closed world assumption, then $I_V = \mathcal{V}(\mathbf{I})$.

In the case of ontology based query answering, there are two ways of defining ontology, i.e., as combination of $\mathcal{KB}$ with $\mathcal{A}$ box such that $\mathcal{O} = \langle \mathcal{KB}, \mathcal{A} \rangle$, or as combination of $\mathcal{KB}$ with $\mathcal{D}$ box, such as in $\mathcal{O} = \langle \mathcal{KB}, \mathcal{D} \rangle$. Both $\mathcal{A}$ box and $\mathcal{D}$ box model data from the original database in the same way that materialized view does. Corresponding to query answering based on materialized views, $\mathcal{A}$ box and $\mathcal{D}$ box also represent the two assumptions about looking at the database information from the perspective of *view*. It is only that here we view the database in the eyes of ontology. The view instance $I_{\mathcal{A}}$ in $\mathcal{A}$ box is based on open world

assumption, whereas the view instance $I_{\mathcal{D}}$ in $\mathcal{D}$ box is built on closed world assumption.

However, the view instance is *not* equivalent with either $\mathcal{A}$ box or $\mathcal{D}$ box. In fact it is only a part of our ontology. This is because both $\mathcal{A}$ box and $\mathcal{D}$ box consist of predicates and individual constants that are not in the original database, as the data model is supposed to represent facts with respect to a *theory* of the database, not raw, primitive data, hence the name ontology, signifying unto us the mystical union between metaphysics and physics. A tuple of individual constants $\bar{c}$ from $\mathcal{A}$ box (or correspondingly $\mathcal{D}$ box) answers the query Q over ontology $\mathcal{O}$ iff $\bar{c}$ is a *certain answer* to Q with respect to all models of $\mathcal{KB}$, regardless of what the domain is, a notion that shall later be explained formally.

Therefore it makes a difference to query answering over materialized views whether the view definition $\mathcal{V}$ gives us complete or incomplete knowledge about the database instance $\mathbf{I}$. In the context of ontology based query answering, it makes a difference whether we use $\mathcal{A}$ box or $\mathcal{D}$ box to model our data. When $\bar{c}$ is a certain answer via $\mathcal{A}$ box, it must be an answer via $\mathcal{D}$ box, but not vice versa. A result that can hardly occasion any wonder though, since we see the actual database through our data model, and it is obviously easier to get a certain answer when the model gives us complete knowledge about the underlying database.

Thus the two streams of thought lead to quite different methodologies in dealing with the complexity of query answering. The most important difference lies in the fact that, for ontology that uses $\mathcal{A}$ box, reformulation of query is irrelevant, whereas if one chooses to use $\mathcal{D}$ box, query reformulation is essential.

CHAPTER 2

A Project on Exact Reformulation

2.1 The Importance of Being Exactly Reformulated

Both $\mathcal{A}$ box and $\mathcal{D}$ box worldview will eventually boil down to interpretation. Whether an interpretation models an $\mathcal{A}$ box or a $\mathcal{D}$ box reveals itself by the way it interprets plain database predicates. Since both $\mathcal{A}$ box and $\mathcal{D}$ box is a set of assertions of the form $A(a)$ and $R(a, b)$, where A is a concept, and R a role in the signature σ. An interpretation that models an $\mathcal{A}$ box interprets every $\mathcal{A}$ box individual as itself, and keeps the extension of concepts and roles exactly as they are in the $\mathcal{A}$ box. The rest of the predicates in the first order language are randomly interpreted. The same holds for $\mathcal{D}$ box. Therefore from now on I shall only speak of $\mathcal{A}$ box and $\mathcal{D}$ box as data models, or perspectives, which guide the building of interpretation that models them, since interpretation is what we really need for query answering. Though the common goal of both streams of thought is to answer query with ontology efficiently, they have different subgoals. One important division between them is the question of reformulating the query. This is out of the question with an $\mathcal{A}$ box model, for it is too expansive. Since $\mathcal{A}$ box models the actual data incompletely, the answer to a query via this data model must be a certain answer, that is, an answer regardless of interpretation. For this reason, we must consider all the *legal* interpretations. A database instance **I** is legal for an ontology $\mathcal{KB}$ if there exists a model $\mathcal{I}$ of $\mathcal{KB}$ embedding **I**. Then $\mathcal{I}$ as a database embedding interpretation is legal for $\mathcal{KB}$. There are two ways of embedding a database in an interpretation, I name them here soundly embedding and embedding *per se* just to distinguish them from each other. An interpretation $\mathcal{I}$ soundly embeds a database instance **I** if for every $a \in adom$(**I**), $\mathcal{I}(a) = \{a\}$, and for every database predicate P, $(c_1, \cdots, c_n) \in \mathcal{I}(P)$ if $P(c_1, \cdots, c_n) \in$ **I**. The

set of all such interpretations $\mathcal{I}$ that soundly embeds the database instance $\mathbf{I}$ is denoted as $SE(\mathbf{I})$. Let $(x_1, \cdots, x_n)$ be the set of free variables of Q, the set of certain answers of a Q over $\mathbf{I}$ under $\mathcal{KB}$ is thus defined:

$$\{v(x_1, \ldots, x_n) \mid \forall \mathcal{I} \in M(\mathcal{KB}) \cap SE(\mathbf{I}) \ : \ \mathcal{I}, v \ \models \ Q(x_1, \ldots, x_n)\} \ .$$

As we can see, when we use $\mathcal{A}$ box as our data model, the process of extracting certain answers must take into consideration *all* legal models of the ontology, thus its evaluation is a problem of validity, more specifically, validity of ground atoms. The answer to one model may not be the same as that to another, by virtue of which that answer can not be a certain answer. For example, given $\mathcal{KB} = \{\forall x(\mathit{TomCat}(x) \vee \mathit{Human}(x) \vee \mathit{Rest}(x))\}$, the query $Q(x) = \mathit{TomCat}(x) \wedge \forall y(\mathit{Human}(y) \rightarrow (x \neq y))$ over the database $\mathbf{I} = \{\mathit{TomCat}(a), \mathit{Human}(b)\}$ with the active domain $\{a, b\}$ has $\emptyset$ as the certain answer. This is because although there exist some legal interpretations, in this case, $\mathcal{I}$ such that $\mathcal{I} \in SE(\mathbf{I})$, that satisfy the query, such as when $\mathcal{I}(\mathit{TomCat}) = \{a\}$ and $\mathcal{I}(\mathit{Human}) = \{b\}$, matching one to one with that of $\mathbf{I}$; there also exist other legal interpretations such as $\mathcal{I} = \{\mathit{TomCat}(a), \mathit{Human}(a), \mathit{TomCat}(b), \mathit{Human}(b), \ldots\}$ for which $\mathcal{I} \not\models Q(x)$ for any x in the domain. Since query answering with $\mathcal{A}$ box is a matter of entailment, the central concern becomes that of reducing the complexity of ontology and query themselves so as to make the entailment manageable. This is why relational calculus queries are rarely considered with $\mathcal{A}$ box, even the domain independent ones, such as the query given in the above example. In other words, complexity is a tradeoff between expressive power and efficiency.

In the $\mathcal{D}$ box data model, the extensions of all database predicates exactly match that of the original database, the answer to query over the database is preserved under the ontology, hence it is also referred to as *exact answers*. It is almost a dual notion of certain answer, i.e., what is in the complement of this set is certainly not an answer. However, here we only consider interpretations that embed, not soundly embed, the database, which is the other way of embedding a database in an interpretation as we have earlier introduced. An interpretation $\mathcal{I}$ embeds a database instance $\mathbf{I}$ if for every $a \in \mathit{adom}(\mathbf{I})$, $\mathcal{I}(a) = \{a\}$ and for every database predicate P, $(c_1, \cdots, c_n) \in \mathcal{I}(P)$ *if and only if* $P(c_1, \cdots, c_n) \in \mathbf{I}$. The set of all such interpretations $\mathcal{I}$ that embeds the database instance $\mathbf{I}$ is denoted as $E(\mathbf{I})$. Clearly $E(\mathbf{I}) \subseteq SE(\mathbf{I})$. The notion of legal database instance being still unchanged, the set of exact answers of a Q over $\mathbf{I}$ under $\mathcal{KB}$ is thus defined:

$$\{v(x_1, \cdots, x_n) \mid \forall \mathcal{I} \in M(\mathcal{KB}) \cap E(\mathbf{I}) \ : \ \mathcal{I}, v \ \models \ Q(x_1, \cdots, x_n)\} \ .$$

It is obvious from the definition that exact answers contain certain answers

due to the fact that $E(\mathbf{I}) \subseteq SE(\mathbf{I})$. If we use $\mathcal{D}$ box as the data model, given exactly the same example as mentioned above, $\{x/a\}$ is an *exact answer*. The answer is exact in the sense that if $\{x/a\}$ is not an answer, then it is not, indeed it cannot be, the answer of *any* legal interpretation $\mathcal{I}$ embedding the database, in this case, $\mathcal{I} \in E(\mathbf{I})$. Compared with $\mathcal{A}$ box, those answers to Q that are not certain answers might still be the exact answers with respect to all legal interpretation that embeds, not soundly embeds the database, that is, for every database predicate P, $(c_1, \cdots, c_n) \in \mathcal{I}(P)$ *if and only if* $P(c_1, \cdots, c_n) \in \mathbf{I}$. $\{x/a\}$ is one such answer that is preserved by fixing the interpretation of the database predicates, i.e., for all legal interpretations $\mathcal{I}$ that embed $\mathbf{I}$, we have $\mathcal{I}(TomCat) = \{a\}$ and $\mathcal{I}(Human) = \{b\}$, therefore it is an exact answer, it is not a *certain* answer only because it is not the answer under all interpretations that soundly embed $\mathbf{I}$, that is with $\mathcal{I}(TomCat) \supseteq \{a\}$ and $\mathcal{I}(Human) \supseteq \{b\}$.

What is less obvious but is equally natural as a consequence of using embedding instead of soundly embedding is that it is now desirable to reduce Q to a query which uses only the original database predicates, so that its answer depends only upon the fixed extension of them, which never wavers in the changing world of ontology models. If we use $\mathcal{A}$ box as the knowledge base to compute the answer to Q, the extension of database predicates in the $\mathcal{A}$ box is not certain, changing from one legal interpretation to another, so long as it contains the relevant predicate extension in the original database, therefore our knowledge with respect to the database predicate concepts is incomplete. Given this characteristic of $\mathcal{A}$ box modeling, even if there is a query reformulation in terms of the plain database predicates, since we can't deduce their interpretations in the actual database from the $\mathcal{A}$ box facts, it is of no use, as we are looking for certain answers anyway. However, with $\mathcal{D}$ box which contains the exact view of the underlying database, then the predicates in the $\mathcal{D}$ box have exactly the same extension as those in the actual database. Hence if there exists a domain independent reformulation in terms of the database predicates, then the query evaluation problem can be reduced to that of SQL evaluation on plain databases. Therefore, when we take the $\mathcal{D}$ box or exact view perspective, the query reformulation problem in terms of plain database predicates becomes essential.

2.2 Much Ado About Model Reduction

Although $\mathcal{D}$ box, as $\mathcal{A}$ box, might contain vocabulary that is beyond that of the original database predicates, if the interpretation of the database predicates $\mathsf{P}_{\mathcal{DB}}$ are fixed, and Q is implicitly determined by $\mathsf{P}_{\mathcal{DB}}$, then there is an explicit definition, or reformulation, of Q in terms of the predicates of $\mathsf{P}_{\mathcal{DB}}$. This being the case, if $\mathcal{KB}$

and Q are both domain independent, then the reformulation of Q is domain independent, and its answers are directly determined by the interpretation of $\mathtt{P}_{\mathcal{DB}}$. Thereby we can reduce query answering over ontology to query reformulation and query answering over plain database, a model checking problem. The notions of implicit determination, explicit definition and reformulation all have their formal counterparts. The detailed explanation of these concepts and the process of reducing ontology based query answering to model checking is the subject that shall occupy the next few chapters. First we must sort out the prerequisites for this reduction. It is a reduction from many models to one model, or from entailment to model checking. There are three things to consider: standard name assumption, domain independence of query with respect to ontology and finite controllability. I will come back to finite controllability. SNA has been discussed in the first chapter, now we turn to domain independence of query with respect to ontology and query reformulation. The ontology itself is always assumed to be domain independent.

Modeling data in $\mathcal{D}$ box makes it possible to reduce multiple models into one model, because it keeps the interpretation of $\mathtt{P}_{\mathcal{DB}}$ fixed. However, this possibility is eventually based on the domain independence of the reformulation $\widetilde{Q}$ of Q in terms of the predicates in $\mathtt{P}_{\mathcal{DB}}$, which again depends upon the domain independence of Q with respect to $\mathcal{KB}$. It has been proved in [Borgida et al., 2010] that given a domain independent query Q and a set of domain independent constraints in $\mathcal{KB}$, if Q is implicitly determined by the database predicates under $\mathcal{KB}$, then the reformulated query $\widetilde{Q}$ in terms of the database predicates is domain independent. Later in [Franconi et al., 2013], the "if" in the claim is strengthened into an "if and only if;" in the mean while, the condition of the "only if" is weakened so that the original query Q only has to be domain independent with respect to $\mathcal{KB}$. That is to say, Q itself does not have to be domain independent, it only requires that it is implicitly determined by the database predicates $\mathtt{P}_{\mathcal{DB}}$ and is domain independent under all $\mathcal{KB}$ models. However, we lose the "only if" if we have $\mathcal{KB}$ to be not domain independent. For example, given an ontology $\mathcal{KB} = \{\forall x(TomCat(x) \vee Lioness(x))\}$, a database instance $\mathbf{I} = \{Lioness(a)\}$ and the query $Q = \neg TomCat(x)$, Q is not safe but domain independent for all the models of $\mathcal{KB}$, besides, it is implicitly determined by $\mathtt{P}_{\mathcal{DB}}$; however, it has a domain independent reformulation $\widetilde{Q} = Lioness(x)$. Hence the failure of the "only if" direction. In finding exact answers, we assume that $\mathcal{KB}$ is consistent and non-tautological, and that there exists at least one interpretation that is legal for $\mathcal{KB}$. When $\mathcal{KB}$ is not domain independent, it restrains the domain of legal interpretations. Continuous with the example, if we have as database instance $\mathbf{I} = \{Lioness(a), TomCat(b)\}$, then there is only one legal interpretation $\mathcal{I}$

for the ontology, with its domain restricted to exactly $\mathbf{I}(TomCat) \cup \mathbf{I}(Lioness)$, for it is the only model of $\mathcal{KB}$ that embeds $\mathbf{I}$. So long as we extend the domain, there will be element in the domain that is neither in $TomCat^{\mathcal{I}}$ nor in $Lioness^{\mathcal{I}}$, since according to $\mathcal{D}$ box model, $TomCat^{\mathcal{I}} \equiv \mathbf{I}(TomCat)$ and $Lioness^{\mathcal{I}} \equiv \mathbf{I}(Lioness)$, therefore $\mathcal{I}$ becomes an illegal interpretation for $\mathcal{KB}$. Of course if we use $\mathcal{A}$ box model with $TomCat^{\mathcal{I}} \supseteq \mathbf{I}(TomCat)$ and $Lioness^{\mathcal{I}} \supseteq \mathbf{I}(Lioness)$, then there are infinitely many models that are legal for the ontology, and the answer to Q would be $\{x/a\}$, which is the certain answer to all such models. But then at the same time $\mathcal{A}$ box model would make it redundant for us to be concerned with perserving query answers with respect to plain database. This is contrary to the intention of $\mathcal{A}$ box model, which breaks from the problems of the plain underlying database, such as incompleteness et cetera, and lifts the query answering to a different level with only a sketch of the original database left in it. On the other hand, to the ontology based query answering with respect to a $\mathcal{D}$ box model, this result is essential, exactly because the reformulation makes it possible to preserve query answers under the original database, without which the $\mathcal{D}$ box representation would lose its advantage of reducing entailment to SQL efficiency, and the technique that applies to plain database query answering would not apply. However, it is still an open question as to what might still be the necessary and sufficient condition with respect to the query Q and its relation to $\mathcal{KB}$ in order for the reformulation to be domain independent, when we allow $\mathcal{KB}$ to be not domain independent. All we know so far is some piecemeal understanding represented by examples, such as if $\mathcal{KB}$ is not domain independent, even if Q is domain independent with respect to all $\mathcal{KB}$ models, the reformulation can still not be guaranteed to be domain independent. The example is trivial, such as when $\mathcal{KB} = \{\forall x A(x)\}$, $\mathtt{P}_{\mathcal{DB}} = \{A\}$, and also $Q = \forall x A(x)$, this naturally makes the boolean query Q implicitly determined by $\mathtt{P}_{\mathcal{DB}}$ and domain independent with respect to all models of $\mathcal{KB}$, namely, to constantly generate the answer $\{\}$. The reformulation is itself, but not domain independent. If, on the other hand, $\mathcal{KB}$ is domain independent, i.e., every sentence in it is domain independent, but Q, though by itself is not domain independent, is domain independent under all the models of $\mathcal{KB}$, we do not lose the "if and only if" condition. Of course, if Q is domain independent and the predicates of Q are all in $\mathtt{P}_{\mathcal{DB}}$, then its reformulation is itself, and domain independent, regardless of $\mathcal{KB}$. To point it out is not to trivialize the question, but to turn to the more interesting point that when Q is not domain independent and is not entirely written in $\mathtt{P}_{\mathcal{DB}}$, it cannot be guaranteed to have a domain independent reformulation without reference to $\mathcal{KB}$.

2.3 Determinacy: From Implicit to Explicit

Implicit and explicit determinacy, or implicit determination and explicit definition, have been so far mentioned without thorough discussion. Now it is time to provide a formal definition. For the sake of convenience, I from now on refer to them as implicit and explicit determinacy. A set of predicates, denoted as views $\mathbf{V} = \{V_1, V_2, \dots\}$, which determine answers to a query Q can be formulated in first order model theory as implicit determinacy, which in its original context is the question of whether we can define a relation symbol in first order logic (FOL) in terms of other relation symbols that are also in FOL. It is the well known Beth definability [Fitting, 1990]. To put it in database context, it is the idea that the views $\mathbf{V}$ provide enough information to uniquely determine the answer of Q. The precise formulation dates back to Tarski [1935]. Given a relation symbol R, a set of relation symbols $\mathbf{R} = \{R_1, R_2, \dots\}$, and a theory Δ, which is a finite set of sentences in the signature of $R \cup \mathbf{V} \cup \mathbf{R}$, we say Δ implicitly defines R in terms of $\mathbf{V}$ iff

$$\forall R, R', \mathbf{V}, \mathbf{R}, \mathbf{R}'(A(R; \mathbf{V}; \mathbf{R}) \wedge A(R'; \mathbf{V}; \mathbf{R}') \rightarrow R \equiv R') \ ,$$

where $\mathbf{V} = \{V_1, V_2, \dots\}$, $\mathbf{R} = \{R_1, R_2, \dots\}$, and $\mathbf{R}' = \{R'_1, R'_2, \dots\}$. $A(R; \mathbf{V}; \mathbf{R})$ denotes the conjunction of all sentences in the theory Δ. The two theories, $A(R; \mathbf{V}; \mathbf{R})$ and $A(R'; \mathbf{V}; \mathbf{R}')$ differ from each other only in in terms of relation symbols outside of the views.

Explicit determinacy, on the other hand, answers the question of whether for a given relation symbol R, there exists an FOL formula ϕ built in the signature of $\mathbf{V}$ that is equivalent with R under Δ. The move from implicit to explicit determinacy gives rise to the original use of Craig interpolation. Many proof systems have been used for interpolation. Among the most commonly used is biased tableau method [Fitting, 1990], which I shall later come back to on the subject of ontology based query answering. To generalize Craig interpolation from FOL to an arbitrary language $\mathcal{L}$, given that Δ implicitly defines R in terms of $\mathbf{V}$, where Δ, R, $\mathbf{V}$ are all in the language of $\mathcal{L}$, if there exists such a formula ϕ which rewrites R using only relation symbols from $\mathbf{V}$, then we say $\mathcal{L}$ is complete for definition and R is explicitly defined by Δ in terms of $\mathbf{V}$. Craig interpolation generates ϕ.

Beth proves completeness for FOL definition in [Beth, 1956]. Concerning database applications the two questions are thoroughly investigated for the first time by [Nash et al., 2010]. They provided a study of many logical systems, fragments of FOL mostly, in search of an answer to the question of definition completeness. The essential question is, what is the smallest expansion of a definition

complete logical system $\mathcal{L}$ in which definition completeness is preserved. This part of history is mentioned here because the question of implicit and explicit determinacy bears significance for ontology based query answering on exact view definition or exact reformulation of a query Q in terms of database predicates. The existence of an exact reformulation of a query Q when Q and $\mathcal{KB}$ are both expressed in FOL coincides with definition completeness of FOL. It has been demonstrated that FOL is the smallest reformulation language that guarantees definition completeness [Nash et al., 2010].

CHAPTER 3

Proof to Plan

3.1 A Little History to Begin With

For historical reasons, implicit and explicit determinacy have their information theoretic correspondents, namely, view determinacy and view-based rewriting. Completeness for definition is also completeness for view-based rewriting. A query Q is determined by a set of views $\mathbf{V}$ if and only if for any two database instances D_1 and D_2, if $\mathbf{V}(D_1) = \mathbf{V}(D_2)$, then $Q(D_1) = Q(D_2)$. Such is a semantic view of the intuitive notion that it is sufficient for a set of views to determine the answer to a given query, in spite of the difference between the underlying database instances. A corresponding syntactic understanding focuses on finding the exact reformulation of Q in a certain language $\mathcal{L}$, usually a fragment of FOL, such that the predicates of $\mathbf{V}$ are the only primitive symbols in the reformulation, thus generating a new, reformulated query $\widetilde{Q}$ such that Q and $\widetilde{Q}$ are logically equivalent under a set of database constraints Σ, i.e., $Q \equiv_\Sigma \widetilde{Q}$. Given a query Q and a set of views $\mathbf{V}$ in $\mathcal{L}$, even when there is no view determinacy for Q in terms of $\mathbf{V}$, there can still be a rewriting of Q in $\mathcal{L}$ such that the rewritten query is contained in, instead of equivalent with, Q, i.e., not an exact but a contained rewriting of Q. Indeed the search and research for *sound view* determinacy was actually earlier than that for *exact view* determinacy, there the focus was on maximally contained query rewriting [Duschka and Genesereth, 1997]. Sound view determinacy is view determinacy as defined shortly above where the view in question is sound view, likewise for exact view determinacy, two notions as were mentioned in section 1.2.

On the one hand is determinacy, and on the other, rewriting, regardless of whether they are views $\mathbf{V}$ or database predicates $\mathtt{P}_{\mathcal{DB}}$ under discussion. The syntactic and semantic are two sides of any interesting query language when it comes to query answering, and furthermore, they are linked. As mentioned at the end of the last chapter, the study of the relationship between these two meta-linguistic properties of a given language $\mathcal{L}$ gives rise to many results of definition incompleteness [Nash et al., 2010]. The various names that have been adopted to describe the link between the syntactic and semantic understanding of query determinacy reflect the different attitudes towards the problem that had been so far taken in history ranging from the early study of mathematical logic in the nineteen-fifties, which was not even concerned with query determinacy, rather with the definability of a first order formula, to the forefront of ontology based data access (OBDA) in database theory. Here we assume definition completeness so that there is a way from query determinacy to query rewriting, since the main concern, the ultimate goal here is still to find an efficient way to answer query under constraints.

3.2 Streams of Ideas

Query answering in practice faces many constraints. So far the many facets of the question under investigation can be combined and reformulated as looking for efficient plans that generate complete answers of a query in the presence of integrity and other constraints. These other constraints can be internal access constraints on database relations, or it can be ontology with which the query shares a language that goes beyond the database predicates. We focus on three things: the plan must be efficient, the answers must be complete, and the constraints must be respected. To solve this many layered problem, there are several streams of ideas. One major stream is inspired by the early 1950s' study of first order logic with its method of preservation and interpolation, dating back to the work of Tarski and Craig [Fitting, 1990]. In subsequent texts I shall refer to this stream of idea as the interpolation method. The other major stream answers query by using constraints to chase and back chase, but this method does not apply generally to arbitrary first order constraints, rather it works particularly well with a certain guarded fragment of integrity constraints called tuple generating dependencies (TGD). From now on I shall refer to this stream of idea as the chase method. The work on OBDA with $\mathcal{D}$ box, the subject that occupies the early chapters, can be roughly shovelled into the first stream, as it uses interpolation and biased tableaux method to find exact answers, but it deserves some special attention because ontology is no usual database schema constraints.

Still the two major streams can in general be discussed under the suggestive name of *proof to plan*, because the goal is to generate a plan that gives complete answers to a query via proof of its answerability. Their difference lies in the kind of proof they exploit. The answerability of the query generates an entailment which can be proved valid with respect to all constraints, and by constructing interpolation of the entailment, we formulate a plan, i.e., the interpolant. What characterizes the method is that though different proofs can be generated using vastly different kinds of constraints, including ontology, the proofs they generate are answerability proofs. Franconi et al. [2013] proposes a proof that guarantees implicit determinacy, which says that the query Q written in the language of ontology $\mathcal{KB}$ has its answer determined by the source database regardless of the interpretation of other relation symbols, since the worldview assumed here is $\mathcal{D}$ box data model. This proof guarantees that the query Q has a reformulation in terms of database predicates that is also generated by interpolation. This amounts to the guarantee of Q being answerable with the original database. Benedikt et al. [2014] proposes various answerability proofs with respect to different schema access restriction axioms. A detailed discussion of various constraints and plans that are generated by the answerability proof with respect to those constraints shall be included in the following discussion of interpolation method. On the other hand, the chase method directly generates a plan from a forward chaining proof sequence via chasing. I shall also provide a detailed discussion of the chase method in comparison with the interpolation method.

If a plan is generated, we would further like to know whether there is such a method that guarantees that while it generates plan from proof, it generates a *good* plan. But what does "good" mean in this context? In the results presented below, there are two flavors of good plan. Strictly speaking, they are not comparable. If we take access restrictions into consideration, then a good plan is an executable rewriting of the query such that all the access patterns of the schema are respected. There is an algorithm for generating such a plan [Benedikt et al., 2014, Deutsch et al., 2007]. However, if we do not consider access restrictions at all, supposing all database relations are perfectly accessible, then a good plan is a safe range rewriting, we need to guarantee that the plan can be translated into a nice SQL statement so that its execution can be carried out with low complexity [Franconi et al., 2013].

CHAPTER 4

Interpolation Method

4.1 Ontology, Database, Exact Reformulation

In the case of ontology based query answering, the whole of ontology, a finite set of first order sentences, can be regarded as a set of constraints in first order logic, therefore the interpolation method applies. Franconi et al. [2013] has laid out in detail the process of constructing first an answerability proof, and then extract exact reformulation from it. Given an ontology $\mathcal{KB}$, a first order query Q that is domain independent with respect to $\mathcal{KB}$, and a set of database predicates $\mathtt{P}_{\mathcal{DB}}$, an answerability proof guarantees that the answers of Q can be completely determined by the interpretation of predicates in $\mathtt{P}_{\mathcal{DB}}$. The theoretical foundation of which can be found in sections 2.1 and 2.2. Here all database predicates are assumed to be perfectly accessible, therefore the determination of the answers of Q by the interpretation of database predicates in $\mathtt{P}_{\mathcal{DB}}$ is equivalent with implicit determinacy of Q by $\mathtt{P}_{\mathcal{DB}}$. Applying Tarski's definition of implicit determinacy as we mentioned in section 2.3, given a relation symbol R, a set of relation symbols $\mathbf{R} = \{R_1, R_2, \dots\}$, and a theory $\mathcal{KB}$ which is a finite set of sentences in the signature of $R \cup \mathtt{P}_{\mathcal{DB}} \cup \mathbf{R}$, we say $\mathcal{KB}$ implicitly defines R in terms of $\mathtt{P}_{\mathcal{DB}}$ iff

$$\forall R, R', \mathtt{P}_{\mathcal{DB}}, \mathbf{R}, \mathbf{R}'(A(R; \mathtt{P}_{\mathcal{DB}}; \mathbf{R}) \wedge A(R'; \mathtt{P}_{\mathcal{DB}}; \mathbf{R}') \rightarrow R \equiv R') \ ,$$

where $\mathtt{P}_{\mathcal{DB}} = \{P_1, P_2, \dots\}$, $\mathbf{R} = \{R_1, R_2, \dots\}$, and $\mathbf{R}' = \{R'_1, R'_2, \dots\}$. $A(R; \mathtt{P}_{\mathcal{DB}}; \mathbf{R})$ denotes the conjunction of all sentences in the theory $\mathcal{KB}$. The two theories, $A(R; \mathtt{P}_{\mathcal{DB}}; \mathbf{R})$ and $A(R'; \mathtt{P}_{\mathcal{DB}}; \mathbf{R}')$ differ from each other only in in terms of relation symbols outside of the set of database predicates $\mathtt{P}_{\mathcal{DB}}$, i.e., those in $\mathbf{R}$, as

$\mathbf{R} \cap \mathtt{P}_{\mathcal{DB}} = \emptyset$. The set of views in the original definition is now the whole $\mathtt{P}_{\mathcal{DB}}$, because there is no access restrictions.

This theorem of implicit determinacy can be then extended from a relation symbol R to a first order logic formula Q such that a query Q is implicitly determined by a database with predicates $\mathtt{P}_{\mathcal{DB}}$ under the ontology $\mathcal{KB}$ iff

$$\mathcal{KB} \cup \widetilde{\mathcal{KB}} \models \forall \overline{x}(Q(\overline{x}) \equiv \widetilde{Q}(\overline{x}))$$

where $\widetilde{Q}$ is the formula obtained from Q by uniformly replacing every occurrence of each non-database predicate R with a new predicate $\widetilde{R}$ that does not occur in $\mathtt{P}_{\mathcal{DB}}$. And $\widetilde{\mathcal{KB}}$ is likewise obtained from $\mathcal{KB}$.

Given the answerability proof, namely, implicit determinacy, we obtain by normal logical deduction the following tautology:

$$((\bigwedge \mathcal{KB} \wedge Q(\overline{x}/\overline{c})) \rightarrow (\bigwedge \widetilde{\mathcal{KB}} \rightarrow \widetilde{Q}(\overline{x}/\overline{c}))$$

$\overline{c}$ is a set of distinct new constants $\{c_1, \ldots, c_n\}$ outside of $\mathcal{KB}$ and Q, assigned to the set of free variables $\{x_1, \ldots, x_n\}$ (shorthand $\overline{x}$) in Q. Given that the interpolant extracted from the tautology is the sentence $\hat{Q}(\overline{c})$, meaning that all the free variables in the original $Q(\overline{x})$ has been grounded using constants $\overline{c}$ for the sake of interpolant extraction, then $\hat{Q}(\overline{c}/\overline{x})$, a first order formula with its free variables substituted back, is the exact reformulation of Q under $\mathcal{KB}$ over $\mathtt{P}_{\mathcal{DB}}$. From now on, I use the shorthand $\hat{Q}$ to denote the possibly open formula $\hat{Q}(\overline{c}/\overline{x})$. The interpolation method used here is Craig interpolation via tableaux method, following the well-known mechanism of biased tableaux specified in [Fitting, 1990]. Very simply speaking, it consists of two steps: first, construct a biased closed tableaux corresponding to the contradiction formula which is the negation of the tautology; second, compute interpolant from leaves to root following the interpolation rules. Both the tableaux and interpolation rules are given in the Appendix A and B, respectively.

Since the relations in $\mathtt{P}_{\mathcal{DB}}$ are all accessible without cost, once the interpolant is extracted, we have got a plan to execute the query. However, to guarantee that it is a good plan in the sense that it can be evaluated using SQL techniques, we need to check a few things. This part of theory has been laid out in section 2.2. Given the research so far, we should check whether $\mathcal{KB}$ is domain independent, and whether Q is domain independent with respect to $\mathcal{KB}$, if the answer is yes to both these questions, then from the implicit determinacy of Q under $\mathcal{KB}$ from $\mathtt{P}_{\mathcal{DB}}$ we can draw the conclusion that we have got a good plan, i.e., the exact

reformulation $\hat{Q}$, as it satisfies the conditions for being a *domain independent* query expressed in terms of $\mathtt{P}_{\mathcal{DB}}$. Since domain is of no consequence now, it is natural to think of using the active domain. Hence to execute it means merely to check the validity of it against one model, i.e., the database embedding $\mathcal{KB}$ model restricted to the active domain signatures. There is only one such model, therefore, such model checking techniques as we use for evaluation of SQL applies. Formally, let $rng(v)$ denote the range of assignment v, $\mathcal{C}$ denote the set of database constants, and $\sigma(\hat{Q})$ denote the signature of $\hat{Q}$, the following equality between the two sets of answers holds:

$$\begin{aligned}&\{v(x_1,\ldots,x_n) \mid \forall \mathcal{I} \in M(\mathcal{KB}) \cap E(\mathbf{I}) : \mathcal{I}, v \models Q(x_1,\ldots,x_n)\}\\ =&\left\{v(x_1,\ldots,x_n) \mid \begin{array}{l} rng(v) \subseteq adom(\sigma(\hat{Q}),\mathbf{I}) \text{ and}\\ \forall \mathcal{I} = \langle \mathcal{C}, \cdot^{\mathcal{I}}\rangle \in E(\mathbf{I}) : \mathcal{I}|_{\mathtt{P}_{\mathcal{DB}}\cup\mathcal{C}}, v \models \hat{Q}(x_1,\ldots,x_n)\end{array}\right\}\end{aligned}$$

What this equality suggests is the final goal of ontology based query answering with $\mathcal{D}$ box data model, namely, the reduction from entailment problem to model checking problem. $\mathcal{I}|_{\mathtt{P}_{\mathcal{DB}}\cup\mathcal{C}}$ denotes the interpretation $\mathcal{I}$ restricted to a smaller signature, i.e., the interpretation with the same domain and the same interpretation function but defined only for predicates in $\mathtt{P}_{\mathcal{DB}}$ and constants in $\mathcal{C}$. Such a $\mathcal{I}|_{\mathtt{P}_{\mathcal{DB}}\cup\mathcal{C}}$ as it is so defined is unique, and all the quantified variables in $\hat{Q}$ is guaranteed to be range-restricted. To deal with the still floating non-range-restricted free variables in $\hat{Q}$, Franconi et al. [2013] introduced an active domain predicate for each such free variable x in Q, i.e.,

$$\begin{aligned}Adom_Q(x) := \bigvee\nolimits_{P\in\mathtt{P}_Q}\Big(\exists z_1,\ldots,z_{AR(P)-1} \bigvee\nolimits_{c\in\mathcal{C}_Q}(x=c)\\ \vee (P(x,z_1,\ldots,z_{AR(P)-1}) \vee\cdots\vee P(z_1,\ldots,z_{AR(P)-1},x))\Big)\ ,\end{aligned}$$

where $\mathtt{P}_Q$ denotes predicates in Q and $\mathcal{C}_Q$ denotes constants in Q. The purpose of defining this new predicate is to guard the free variables that do not appear in any positive predicates, therefore are not yet put into safe range. Then it follows from this definition that

$$\mathcal{KB} \models \forall \overline{x}\ (Q(x_1,\ldots,x_n) \rightarrow Adom_Q(x_1)\wedge\cdots\wedge Adom_Q(x_n))\ .$$

As $\hat{Q}(x_1,\ldots,x_n)$ is the interpolant with free variables $x_1,\ldots,x_n$, combining the above logical deduction with interpolant extraction, we have

$$\mathcal{KB} \models \forall \overline{x}\ (Q(x_1,\ldots,x_n) \rightarrow \hat{Q}(x_1,\ldots,x_n)\wedge Adom_{\hat{Q}(x_1)}\wedge\cdots\wedge Adom_{\hat{Q}(x_n)})\ .$$

Let $\hat{Q}\wedge Adom$ denote $\hat{Q}(x_1,\ldots,x_n)\wedge Adom_{\hat{Q}(x_1)}\wedge\cdots\wedge Adom_{\hat{Q}(x_n)}$, it is obvious that $\hat{Q}\wedge Adom$ is ground safe-range, meaning that its grounding is safe-range.

Since $\hat{Q}$ is an exact reformulation of Q, this means that the answers of $\hat{Q}$ exactly coincides with that of Q. We now have a safe-range exact reformulation of Q using only plain database predicates, hence a good plan.

4.2 Finite Controllability

It is obvious from the definition of ontology based query answering that we must first obtain legal interpretations before we can answer Q. Such a legal interpretation is a model of $\mathcal{KB}$ that embeds the database instance $\mathbf{I}$. As our $\mathcal{KB}$ is simply a set of first order sentences that are domain independent, as it happens, for some $\mathcal{KB}$, the entailment of implicit determinacy holds only in finite models of $\mathcal{KB}$; and for others only in infinite models of $\mathcal{KB}$. In the case of the former, we have to write

$$\mathcal{KB} \cup \widetilde{\mathcal{KB}} \models_{fin} \forall \overline{x}(Q(\overline{x}) \leftrightarrow \widetilde{Q}(\overline{x})) \ ,$$

while

$$\mathcal{KB} \cup \widetilde{\mathcal{KB}} \not\models \forall \overline{x}(Q(\overline{x}) \leftrightarrow \widetilde{Q}(\overline{x}))$$

in order to express implicit determinacy. The example 1 given in [Franconi et al., 2013] provides a case where $\forall \overline{x}(Q(\overline{x}) \leftrightarrow \widetilde{Q}(\overline{x}))$ is impossible to be deduced from $\mathcal{KB} \cup \widetilde{\mathcal{KB}}$ over models with infinite domain, that is to say, infinite domain models that satisfies $\mathcal{KB}$ can not validate the sentence $\forall \overline{x}(Q(\overline{x}) \leftrightarrow \widetilde{Q}(\overline{x}))$. On the other side, the set of Peano axioms for defining the set of natural numbers $\mathbb{N}$ using constant symbol 0 and a unary function symbol S does not admit finite models.

Therefore finite controllability condition is important, because if we were only to consider the finite models of $\mathcal{KB}$, we might lose both Beth and Craig theorem validity. It has been demonstrated in [Hájek, 1977] that due to finiteness of models, failure of Beth definability and failure of Craig interpolation will follow. In [Hájek, 1977] failure of the two essential theorems are thus formally expressed:

Failure of Beth Definability. There is a first order L-sentence, which defines a unary predicate P implicitly but not explicitly.

Failure of Craig Interpolation. Let $L = \{<, c\}$. There are an $L \cup \{P\}$-sentence ϕ_1 and an $L \cup \{Q\}$-sentence ϕ_2 so that $\phi_1 \models \phi_2$ but no L-sentence satisfies both $\phi_1 \models \theta$ and $\theta \models \phi_2$.

Besides, the complexity of checking Beth definability with finite models is an open problem. When Beth definability and Craig interpolation fail, so does the task of implicit and explicit determinacy. It defeats the object of reformulating a query under ontology in order to answer it independent of the ontology. Therefore [Franconi et al., 2013] assumes a fragment of first order logic with property of finite controllability, such that implicit and explicit determinacy under models with infinite interpretation coincides with that under models with finite interpretation with respect to such database predicates in $\mathtt{P}_{\mathcal{DB}}$. As such, we avoid the problems created by finite and infinite models.

4.3 Plans in Fragments of Relational Algebra

Ontology, considered as a set of constraints over the underlying database, concentrates on the relations between database predicates from a conceptual point of view, describing the ontological picture of a logically possible world. However, when we consider query answering in practical situations, the underlying tables have logical properties of their own that must be taken into account. Apart from the familiar primary key, foreign key, functional dependencies and other dependencies, one salient feature that keeps coming up in practice is the accessibility of tables. It often happens that extracting answers from a single table is impossible without the help of other accessible tables, which in turn depends on other tables for their accessibility, so on and so forth, thus forming a dependency graph on the accessibility conditions required for answering a query. Though constraints are different, the central methodology is still interpolation based on entailment. The entailment is an answerability proof, as was discussed in section 3.2, which guarantees the existence of a plan to answer the query. And the extraction of such a plan is also done by interpolation, following the blueprint of proof to plan. In the case of these arbitrary first order logic constraints, when the complexity of contraints goes beyond tuple generating dependencies, which shall be detailed in later discussions on chase method, the existence of a plan is still implicated by the answerability proof, which is an entailment holding between two first order logic formulas under a set of rules that encode integrity and other database constraints. Benedikt et al. [2014] has shown that the choice of axioms that encode access restrictions on database predicates can make a difference in the answerability proof, and consequently generate plans that are built on different languages, each using a subset of relational algebra operators. The plans so constructed are given such various names as RA-plans, USPJ-plans, and in between, USPJ$^{\neg}$-plans. Here the characters U, S, P, J indicates the relational algebraic operators that get used in the building of algebraic expressions in the commands of the plan, namely, union, select, project and join. RA-plans use the whole set of operators

in relational algebra. USPJ$^{\neg}$-plans use positive RA operators with a restricted difference operator E $-$ E′, that is, the difference operator only applies in taking output tuples of an algebraic expression E and subtracting out from them tuples that are in some accessible relation R.

It is worth remarking that at the end of chapter 3, a good plan has been taken to be an executable rewriting of the query such that all the access patterns of the schema are respected. It has been shown in [Benedikt et al., 2014] that there is a linear time procedure which converts an executable boolean first order query into an RA-plan. And if the executable boolean query is existential, the result is an USPJ$^{\neg}$-plan. If it is existential without inequalities, it can be converted into an EUSPJ$^{\neg}$-plan, which is an USPJ$^{\neg}$-plan with further restriction that no inequality is allowed to be used in select and join. If it is positively existential, the result is an USPJ-plan. By the same token, if no inequality is allowed in the query, then it is also not allowed in the select and join of the plan; therefore if the query is also positive existential, then the result is an EUSPJ-plan.

4.4 Answerability Proof With Access Constraints

As is the custom of interpolation method, we first introduce an answerability proof, which suggests the existence of a plan, and then use interpolation to extract the plan out of the entailment under constraints. Since the constraints here are not ontology, rather properties of database schema, especially those concerning access restrictions, to introduce the answerability proof, we first have to axiomatize the properties of access restrictions, a technique that first appears in the work of Duschka et al. [2000] and Deutsch et al. [2007]. Given schema S_0 with access restrictions, the rules that encode these restrictions are defined not directly on S_0, rather on a new schema built upon S_0, called Accessible Schema for S_0, denoted as $\mathsf{AcSch}(S_0)$, which has no access restriction. The new schema $\mathsf{AcSch}(S_0)$ contains all the constants and relation symbols of S_0, besides, it also contains for each relation symbol R two more copies, one denoted by $\mathsf{Accessed}R$, the other by $\mathsf{InferredAcc}R$. $\mathsf{AcSch}(S_0)$ also has a unary relation $\mathsf{accessible}(x)$. The constraints of $\mathsf{AcSch}(S_0)$ are of three kinds:

1. Original integrity constraints of S_0;
2. Substitution instance of original integrity constraints of S_0, with each R substituted by $\mathsf{InferredAcc}R$;
3. Three new axioms
   a) $\forall \overline{x}(\mathsf{Accessed}R(\overline{x}) \rightarrow \mathsf{accessible}(x_i))$;

b) $\forall \overline{x}(\mathsf{accessible}(x_{j_1}) \wedge \cdots \wedge \mathsf{accessible}(x_{j_m}) \wedge R(\overline{x}) \rightarrow \mathsf{Accessed}R(\overline{x}))$;

c) $\forall \overline{x}(\mathsf{Accessed}R(\overline{x}) \rightarrow \mathsf{InferredAcc}R(\overline{x}))$.

where $\overline{x}$ is again shorthand for $(x_1, \ldots, x_n)$ and $\{x_{j_1}, \ldots, x_{j_m}\}$ is a collection of positions of R, called input positions, the access of which is a gateway for R to become accessible. Benedikt et al. [2014] has shown that

Theorem 1 ([Benedikt et al., 2014])**.** For any conjunctive query Q and schema S_0 with first order integrity constraints and access restrictions, there is a USPJ-plan for Q if and only if Q entails $\mathsf{InferredAcc}Q$ with respect to $\mathsf{AcSch}(S_0)$. $\mathsf{InferredAcc}Q$ is obtained from Q by replacing every occurrence of R in Q by $\mathsf{Accessed}R$ for every relation symbol R and adding a conjunct $\mathsf{accessible}(x)$ for every free variable in Q.

As was mentioned in section 4.3, the choice of axioms that encode access restrictions on database predicates can make a difference in the answerability proof, and consequently generate plans that are built on different languages, each using a subset of relational algebra operators. While leaving everything in the axiomatization so far unchanged, if we add two more rules to $\mathsf{AcSch}(S_0)$:

(1) $\forall \overline{x}(\mathsf{Accessed}R(\overline{x}) \rightarrow R(\overline{x}))$;

(2) $\forall \overline{x}(\mathsf{accessible}(x_{j_1}) \wedge \cdots \wedge \mathsf{accessible}(x_{j_m}) \wedge \mathsf{InferredAcc}R(\overline{x}) \rightarrow \mathsf{Accessed}R(\overline{x}))$;

the resulting axiomatization is a new schema $\mathsf{AcSch}^{\leftrightarrow}(S_0)$, and following the same interpolation method as in theorem 1, the plan we now obtain from the answerability proof under the new schema $\mathsf{AcSch}^{\leftrightarrow}(S_0)$ is an RA-plan, i.e

Theorem 2 ([Benedikt et al., 2014])**.** For any relational query Q and schema S_0 with first order integrity constraints and access restrictions, there is a RA-plan for Q if and only if Q entails $\mathsf{InferredAcc}Q$ with respect to $\mathsf{AcSch}^{\leftrightarrow}(S_0)$.

So we have a variant of theorem 1. Along this line of investigation, if we add a different rule to $\mathsf{AcSch}(S_0)$ such as

$$\forall \overline{x}(\mathsf{accessible}(x_1) \wedge \cdots \wedge \mathsf{accessible}(x_n) \wedge \neg R(\overline{x}) \rightarrow \mathsf{InferredAcc}R(\overline{x}))$$

still leaving everything else unchanged, we again obtain a new schema $\mathsf{AcSch}^{\neg}(S_0)$, still using interpolation method from answerability proof under the new schema, the plan is now an USPJ$^{\neg}$-plan.

Theorem 3 ([Benedikt et al., 2014])**.** For any relational algebra query Q and schema S_0 with first order integrity constraints and access restrictions, there is a USPJ$^{\neg}$-plan for Q if and only if Q entails $\mathsf{InferredAcc}Q$ with respect to $\mathsf{AcSch}^{\neg}(S_0)$.

The method used for extracting plan in all three cases is access interpolation, which is an extension of Craig interpolation theorem. Strictly speaking, what we obtain from interpolation is not a plan for Q, rather an executable first order query ϕ that is equivalent to Q over all the instances that satisfy the constraints of the respective schema.

ϕ is an executable first order query in a schema S if and only if every binding pattern in $\mathsf{BindPatt}(\phi)$ is compatible with a method in S. To fully comprehend these statements, we need the following definitions:

Method. A method mt_R for a relation R symbol in S_0 is a pair (R, M), where M is the set of input positions (a set of natural numbers) in R such that $\mathsf{accessible}(x_i)$ for all $i \in M$ is necessary for the access of relation R in S_0. Every relation symbol in S_0 associates with it a set of methods.

Binding Pattern. A binding pattern is a pair (R, N), where R is a relation symbol and N is a set of positions in R. For a certain relation symbol R, its binding pattern is compatible with a method (R, M) if $N \subseteq M$.

Recursive Definition. The binding pattern of a first order logic formula ϕ is a set of binding patterns, denoted by $\mathsf{BindPatt}(\phi)$, each corresponds to a relation symbol occurring in ϕ. It is recursively defined as follows:

- $\mathsf{BindPatt}(\top) = \mathsf{BindPatt}(x = y) = \varnothing$;
- $\mathsf{BindPatt}(R(x_1, \ldots, x_n)) = \{(R, \{1, \ldots, n\})\}$;
- $\mathsf{BindPatt}(\neg\phi) = \mathsf{BindPatt}(\phi)$;
- $\mathsf{BindPatt}(\phi \wedge \psi) = \mathsf{BindPatt}(\phi) \cup \mathsf{BindPatt}(\psi)$;
- $\mathsf{BindPatt}(\phi \vee \psi) = \mathsf{BindPatt}(\phi) \cup \mathsf{BindPatt}(\psi)$;
- $\mathsf{BindPatt}(\exists \overline{x} R(\overline{x}) \wedge \phi) = \mathsf{BindPatt}(\phi) \cup \{(R, \{i \mid x_i \notin \overline{x}\})\}$;
- $\mathsf{BindPatt}(\forall \overline{x} R(\overline{x}) \rightarrow \phi) = \mathsf{BindPatt}(\phi) \cup \{(R, \{i \mid x_i \notin \overline{x}\})\}$;

The technique used to extract the interpolant is standard Craig interpolation technique with a twist on binding patterns. The interpolant thus obtained is

called access interpolant. Let ϕ and ψ be first order sentences. ϕ entails ψ. Then there exists a first order sentence ϕ' such that

1. ϕ entails ϕ' and ϕ' entails ψ.
2. A relation symbol R occurs positively (negatively) in ϕ' only if R occurs positively (negatively) in both ϕ and ψ.
3. A constant symbol occurs in ϕ' only if it occurs in both ϕ and ψ.
4. If $\mathsf{BindPatt}(\phi)$ and $\mathsf{BindPatt}(\psi)$ are defined, then $\mathsf{BindPatt}(\phi') \subseteq \mathsf{BindPatt}(\phi) \cup \mathsf{BindPatt}(\psi)$.

ϕ' is the access interpolant. This access interpolant obtained from the answerability proof of Q, i.e., the proof that Q entails $\mathsf{InferredAcc}Q$, is an executable query ϕ that is logically equivalent with Q for all instances that satisfy the constraints of S_0. Once it is obtained, it can be converted into a plan PL in linear time. However, the structure of ϕ depends on the schema in which Q entails $\mathsf{InferredAcc}Q$. Corresponding to the three theorems introduced above, when the entailment proof is conducted in $\mathsf{AcSch}^{\leftrightarrow}(S_0)$, the resulting access interpolant ϕ is first order, then PL is an RA-plan. When the proof is done in $\mathsf{AcSch}^{\neg}(S_0)$, every relation symbol in ϕ is only existentially quantified, then PL is an USPJ$^{\neg}$-plan. And when the entailment is valid with respect to $\mathsf{AcSch}(S_0)$, every relation symbol in ϕ is positive and only existentially quantified, then PL is an USPJ-plan.

CHAPTER 5

Chase Method

5.1 Chase Proof

We define chase only with respect to embedded dependencies of the form

$$\mathrm{IC}(\mathcal{L}) := \{\forall \overline{x}(\mathrm{U} \rightarrow \mathrm{V}) \mid \mathrm{U}, \mathrm{V} \in \mathcal{L}\}$$

where $\mathrm{IC}(\mathcal{L})$ denotes the set of inclusion constraints over $\mathcal{L}$, which is the language of conjunctive queries with equality atoms. $\overline{x}$ is the set of free variables. By a sequence of logical steps, every embedded dependency rule in conjunctive queries (CQ) can be normalized into a set of constraints in the following form

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

referred to as tuple generating dependencies (TGD). Guarded TGD is a subclass of TGD, which requires the free variables in the antecedent to be all *guarded* by conjunction with an atom. The chase and backchase method in the work of [Deutsch et al., 2006, 1999] set the foundation of the methodology to be introduced here: a method of using constraints to chase the query.

It has been shown in [Benedikt et al., 2014] that with TGD constraints, we can exploit chase proof, a proof via chase sequence, to construct a plan. Given a set of dependencies Σ, a chase sequence consists of a sequence of database instances, starting with the canonical instance $F_i : 1 \leq i \leq n$, where F_{i+1} is obtained from F_i by some rule in Σ. The set of facts in each F_i is a chase configuration. And each rule firing generates a set of new facts. A homomorphism of a query Q' into the configuration of a chase sequence is called a match for Q'. The notion of

proof via chase sequence is as follows: for two conjunctive queries Q and Q' with the same free variables, and a set of TGD constraints Σ, Q entails Q' if and only if there is a chase sequence with respect to Σ beginning with the canonical database of Q that ends with a configuration which has a match for Q', i.e., there exists a homomorphism of Q' into the configuration. The free variables of Q' are mapped by the homomorphism to the variable constants in the canonical database corresponding to the free variables of Q. The canonical database of Q is the database instance $\mathbf{I}$ such that its elements are either constants or variable constants, the latter are obtained from the variables of Q, with each variable in Q corresponding to a variable constants in $\mathbf{I}$, and for each $(x_1, \ldots, x_n) \in R^{\mathbf{I}}$, there is an atom $R(x_1, \ldots, x_n)$ in Q. A *candidate matching* for the firing of a rule δ on F_i is a tuple of constants from F_i that violates δ. Therefore the firing of the rule δ adds new constants called *chase constants* that form new facts in F_{i+1}. Intuitively speaking, the chase constants are there to rectify the violation signified by the candidate matching.

We know from theorem 1 in section 4.3 that given accessible schema $\mathsf{AcSch}(S_0)$ obtained from S_0 and its rules, for any conjunctive query Q and schema S_0 with first order integrity constraints and access restrictions, there is a USPJ-plan for Q iff Q entails $\mathsf{InferredAcc}Q$ with respect to $\mathsf{AcSch}(S_0)$. When the rules are in TGD form (the integrity constraints and access restrictions are in guarded TGD form already), we can extract a USPJ-plan directly from the chase sequence, and the proof via the chase sequence is a forward chaining proof, from the canonical database of Q to an instance F_n with a match for $\mathsf{InferredAcc}Q$. Later I shall introduce an algorithm that performs the task of plan extraction from chase proof with a focus on plan cost.

If we look from the perspective of plan cost, the kind of rule firings we use in the forward chaining proof fall into two categories: on the one hand original integrity constraints and its copies, and on the other accessibility axiom. Both generate chase constants, but only the latter amounts to relation access, which is the source of cost. Therefore we calculate the cost of plan by the number of firings of the accessibility rule, i.e.,

$$\begin{aligned}\forall \overline{x}(&\mathsf{accessible}(c_{j_1}) \wedge \cdots \wedge \mathsf{accessible}(c_{j_m}) \wedge R(c_1, \cdots, c_n) \\ &\rightarrow \mathsf{Accessed}R(c_1, \cdots, c_n))\end{aligned}$$

On i-th step of the sequence, we keep track of the accessible elements of F_i by a temporary table T_i, which starts with $\emptyset$ and expands as more accessibility rules are fired. In the following algorithm, F_i is a state of the given canonical database instance F in its evolution through rule firing.

For all the plans that appear in this discussion, the structure of the plan is made of relational algebra expressions and access command in the form of Comms($F, R(\bar{c}),$ mt).

Algorithm 1 Building Plan from Chase

```
1: procedure CHASE(database F, constraints Σ)
2:     for σ ∈ Σ do
3:         if F contains a candidate match for σ then
4:             Fire σ in F
5:             Add new σ-facts to F
6:         end if
7:     end for
8:     return F
9: end procedure
```

```
1: procedure PLAN(Q, InferredAccQ, Σ)
2:     plan = ∅
3:     F is canonical database of Q
4:     C = ∅
5:     T_o = ∅
6:     while F has no match for InferredAccQ do
7:         for σ ∈ Σ_AC do
8:             F = CHASE(F, Σ_IN)
9:             F = CHASE(F, {σ})
10:            let T be the table for AccessedR(c_1, ··· , c_n) generated by σ
11:            if T_o is ∅ then
12:                T_o = T
13:            end if
14:            C = C ∪ const(F)
15:            Select a method mt_R for R
16:            Take c_{j_i} from the subset C' of C such that each c_{j_i} ∈ C' corresponds
               to an attribute of T
17:            Map element of C' into input position of mt_R
18:            Add access command Comms(F, R(c̄), mt) to plan
19:            T := T_o ⋈ T
20:            T_o = T
21:        end for
22:    end while
23:    return plan
24: end procedure
```

Σ_{IN} is the set of integrity constraints. For strategic reasons, it is better to exhaust the use of all integrity constraints before using access command for a given set of facts, i.e., for each F_i. In this way the numbers of times we use the access command can be reduced to the minimum. C is the set of chase constants accumulated by rule firings. T is a temporary table whose attributes are accessible elements at each step of rule firing. The access command $\mathsf{Comms}(F, R(\overline{c}), \mathsf{mt})$ uses a method mt_R to access R, take from the subset C' of C such c_{j_i} that each $c_{j_i} \in C'$ corresponds to an attribute of T, and then map them into the input position of mt_R. In plain language, this is the action of taking accessible constants from storage and using them to access a relation. $T := T_o \bowtie T$ symbolizes the generation of T at step i through join with T at step i-1. Therefore T records at each state of F all its accessible elements, such is the invariant of the while loop. Benedikt et al. [2014] has pointed out that termination of chase on $\mathsf{AcSch}(S_0)$ is guaranteed by many restricted classes of the accessible schema, and has formulated soundness and completeness of the technique, where completeness is restricted to finite instances.

5.2 In Comparison with Non-Chase Proof

The essential step of the above algorithm is the two chase steps and the access command. The chase steps change the set of facts and might ultimately result in a match for $\mathsf{InferredAcc}Q$, whereas the access command builds the plan and renders relations accessible. In section 4.4, we mentioned that the constraints of $\mathsf{AcSch}(S_0)$ are of three kinds:

1. Original integrity constraints of S_0;
2. Substitution instance of original integrity constraints of S_0, with each R substituted by $\mathsf{InferredAcc}R$.
3. Three new axioms
   a) $\forall\overline{x}(\mathsf{Accessed}R(\overline{x}) \rightarrow \mathsf{accessible}(x_i))$;
   b) $\forall\overline{x}(\mathsf{accessible}(x_{j_1}) \wedge \cdots \wedge \mathsf{accessible}(x_{j_m}) \wedge R(\overline{x}) \rightarrow \mathsf{Accessed}R(\overline{x}))$;
   c) $\forall\overline{x}(\mathsf{Accessed}R(\overline{x}) \rightarrow \mathsf{InferredAcc}R(\overline{x}))$.

The first two type of rules fire in Chase(F, Σ) and the three rules in the third type together mark the use of access command $\mathsf{Comms}(F, R(\overline{c}), \mathsf{mt})$. Rule (3a) generates C' out of C whose elements corresponds to attributes of T, rule (3b) maps the elements to input positions of R according to an access method mt_R and rule (3c) generate $\mathsf{InferredAcc}R$ so that we can apply rules in the second

type again. Benedikt et al. [2014] has indicated that for a conjunctive query Q, every chase sequence proving $\mathsf{InferredAcc}Q$ from Q with respect to constraints of $\mathsf{AcSch}(S_0)$ generates a USPJ-plan that completely answers Q, which can be translated to an executable query ϕ that is logically equivalent with Q for all instances that satisfy the constraints of S_0, thus an access interpolant. Similarly for RA-plan and USPJ$^{\neg}$ with respect to $\mathsf{AcSch}^{\leftrightarrow}(S_0)$ and $\mathsf{AcSch}^{\neg}(S_0)$, provided that Q is a conjunctive query. Each step i is associated with an intermediate chase configuration F_i and a partial plan PL_i. $\mathsf{Accessed}F_i$ denotes the conjunctive query formed by taking the conjunction of all facts of the form $\mathsf{InferredAccR}(\overline{c})$ in F_i and turning them into an existentially quantified conjunction of facts $\mathsf{R}(\overline{w})$, changing the chase constants c that satisfy $\mathsf{accessible}(c)$ to free variables and other chase constants to existentially quantified variables. Let $T_i(\mathbf{I})$ be the instance of table T_i produced by PL_i when run on an instance $\mathbf{I}$ of schema S_0, Benedikt et al. [2014] has shown that firstly, if Q returns a non-empty result on $\mathbf{I}$, then $T_i(\mathbf{I})$ is non-empty; secondly, $T_i(\mathbf{I})$ is a subset of $\mathsf{Accessed}F_i(\mathbf{I})$; and lastly, when F_i has a match for $\mathsf{InferredAcc}Q$, then $\mathsf{Accessed}F_i$ entails Q. At the last step, when F_i has a match for $\mathsf{InferredAcc}Q$, $T_i(\mathbf{I})$ is finally equivalent with $\mathsf{Accessed}F_i(\mathbf{I})$. By soundness, $T_i(\mathbf{I})$ at the last step generates only answers for Q, therefore $T_i(\mathbf{I})$ is a subset of $Q(\mathbf{I})$. Then it is as if PL_i interpolates between $\mathsf{Accessed}F_i$ and Q.

5.3 Query Answering Under Constraints

From logic programming point of view, a database instance is a collection of facts. However, without TGD rules, we cannot determine the candidate match for a certain rule using only database facts. Then the chase method is handicapped. On the other hand, when we are allowed to use it, the chase method has the advantage of avoiding the expensive tableaux technique for extracting plan from proof and construct the plan directly from the chase sequence, as is shown by the algorithm. Naturally it does not apply to ontology based query answering, the central subject that occupies the first three chapters. If we choose to use $\mathcal{D}$ box as our data model in answering query under ontology, then it is unavoidable to reformulate the query through the technique of preservation and interpolation via tableaux.

The entailment proof might take different forms, but the technique of moving from proof to plan still lies at the foundation of all the approaches presented. This is because the problem addressed, regardless of its specific form, is basically the problem of answering queries under constraints, be the constraints ontology, access restrictions, or integrity constraints. Therefore the main methodology here to solve this problem is to construct a proof which preserves a solution to the

query in terms of a completely accessible schema, by accessing, by deduction, even by ontology. This solution is brought to light by interpolation. In this sense these different approaches may be reunited.

APPENDIX A

Tableaux Rules

Let C be the set of all constants in the input formulas of the tableau. C^{par} extends C with an infinite set of new constants. A constant is new if it does not occur anywhere in the tableau. With these notations, we have the following rules:

- Propositional rules

Negation rules			$\alpha-$rule	$\beta-$rule
$\dfrac{X(\neg\neg\varphi)}{X(\varphi)}$	$\dfrac{X(\neg\top)}{X(\bot)}$	$\dfrac{X(\neg\bot)}{X(\top)}$	$\dfrac{X(\varphi_1 \wedge \varphi_2)}{\begin{array}{c}X(\varphi_1)\\X(\varphi_2)\end{array}}$	$\dfrac{X(\neg(\neg\varphi_1 \wedge \neg\varphi_2))}{X(\varphi_1) \quad \mid \quad X(\varphi_2)}$

- First order rules

$\gamma-$rule	$\sigma-$rule
$\dfrac{X(\forall x.\varphi)}{X(\varphi(t))}$ for any $t \in C^{par}$	$\dfrac{X(\exists x.\varphi)}{X(\varphi(c))}$ for a new constant c

- Equality rules

reflexivity rule	replacement rule
$\dfrac{X(\varphi)}{X(t=t)}$ $t \in C^{par}$ occurs in φ	$\dfrac{\begin{array}{c}X(t=u)\\Y(\varphi(t))\end{array}}{Y(\varphi(u))}$

APPENDIX B

Interpolation Rules

Given a closed biased tableau, the interpolant is computed by applying *interpolant rules*. An interpolant rule is written as $S \xrightarrow{int} I$, where I is a formula and

$$S = \{L(\phi_1), L(\phi_2), \ldots, L(\phi_n), R(\psi_1), R(\psi_2), \ldots, R(\psi_m)\} \ .$$

- Rules for closed branches

  r1. $S \cup \{L(\varphi), L(\neg\varphi)\} \xrightarrow{int} \bot$ r2. $S \cup \{R(\varphi), R(\neg\varphi)\} \xrightarrow{int} \top$

  r3. $S \cup \{L(\bot)\} \xrightarrow{int} \bot$ r4. $S \cup \{R(\bot)\} \xrightarrow{int} \top$

  r5. $S \cup \{L(\varphi), R(\neg\varphi)\} \xrightarrow{int} \varphi$ r6. $S \cup \{R(\varphi), L(\neg\varphi)\} \xrightarrow{int} \neg\varphi$

- Rules for propositional cases

  p1. $\dfrac{S \cup \{X(\varphi)\} \xrightarrow{int} I}{S \cup \{X(\neg\neg\varphi)\} \xrightarrow{int} I}$ p2. $\dfrac{S \cup \{X(\top)\} \xrightarrow{int} I}{S \cup \{X(\neg\bot)\} \xrightarrow{int} I}$

  p3. $\dfrac{S \cup \{X(\bot)\} \xrightarrow{int} I}{S \cup \{X(\neg\top)\} \xrightarrow{int} I}$ p4. $\dfrac{S \cup \{X(\varphi_1), X(\varphi_2)\} \xrightarrow{int} I}{S \cup \{X(\varphi_1 \wedge \varphi_2)\} \xrightarrow{int} I}$

p5. $$\frac{S \cup \{L(\varphi_1)\} \xrightarrow{int} I_1 \; S \cup \{L(\varphi_2)\} \xrightarrow{int} I_2}{S \cup \{L(\neg(\neg\varphi_1 \wedge \neg\varphi_2))\} \xrightarrow{int} I_1 \vee I_2}$$

p6. $$\frac{S \cup \{R(\varphi_1)\} \xrightarrow{int} I_1 \; S \cup \{R(\varphi_2)\} \xrightarrow{int} I_2}{S \cup \{R(\neg(\neg\varphi_1 \wedge \neg\varphi_2))\} \xrightarrow{int} I_1 \wedge I_2}$$

- Rules for first order cases

f1. $$\frac{S \cup \{X(\varphi(p))\} \xrightarrow{int} I}{S \cup \{X(\exists x.\varphi(x))\} \xrightarrow{int} I}$$ where p is a parameter that does not occur in S or φ

f2. $$\frac{S \cup \{L(\varphi(c))\} \xrightarrow{int} I}{S \cup \{L(\forall x.\varphi(x))\} \xrightarrow{int} I}$$ if c occurs in $\{\varphi_1, ..., \varphi_n\}$

f3. $$\frac{S \cup \{R(\varphi(c))\} \xrightarrow{int} I}{S \cup \{R(\forall x.\varphi(x))\} \xrightarrow{int} I}$$ if c occurs in $\{\psi_1, ..., \psi_m\}$

f4. $$\frac{S \cup \{L(\varphi(c))\} \xrightarrow{int} I}{S \cup \{L(\forall x.\varphi(x))\} \xrightarrow{int} \forall x.I[c/x]}$$ if c does not occur in $\{\varphi_1, ..., \varphi_n\}$

f5. $$\frac{S \cup \{R(\varphi(c))\} \xrightarrow{int} I}{S \cup \{R(\forall x.\varphi(x))\} \xrightarrow{int} \exists x.I[c/x]}$$ if c does not occur in $\{\psi_1, ..., \psi_m\}$

- Rules for equality cases

e1. $$\frac{S \cup \{X(\varphi(p)), X(t = t)\} \xrightarrow{int} I}{S \cup \{X(\varphi(p))\} \xrightarrow{int} I}$$

e2. $$\frac{S \cup \{X(\varphi(u)), X(t = u)\} \xrightarrow{int} I}{S \cup \{X(\varphi(t)), X(t = u)\} \xrightarrow{int} I}$$

e3. $$\frac{S \cup \{L(\varphi(u)), R(t = u)\} \xrightarrow{int} I}{S \cup \{L(\varphi(t)), R(t = u)\} \xrightarrow{int} t = u \to I}$$ if u occurs in $\varphi(t), \psi_1, ..., \psi_m$

e4. $$\frac{S \cup \{R(\varphi(u)), L(t = u)\} \xrightarrow{int} I}{S \cup \{R(\varphi(t)), L(t = u)\} \xrightarrow{int} t = u \wedge I}$$ if u occurs in $\varphi(t), \psi_1, ..., \psi_m$

e5. $$\frac{S \cup \{L(\varphi(u)), R(t = u)\} \xrightarrow{int} I}{S \cup \{L(\varphi(t)), R(t = u)\} \xrightarrow{int} I[u/t]}$$ if u does not occur in $\varphi(t), \psi_1, ..., \psi_m$

e6. $$\frac{S \cup \{R(\varphi(u)), L(t = u)\} \xrightarrow{int} I}{S \cup \{R(\varphi(t)), L(t = u)\} \xrightarrow{int} I[u/t]}$$ if u does not occur in $\varphi(t), \psi_1, ..., \psi_m$

List of Algorithms

Bibliography

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases: The Logical Level*. Addison Wesley, 1994.

Michael Benedikt, Balder ten Cate, and Efthymia Tsamoura. Generating low-cost plans from proofs. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 200–211. ACM, 2014. ISBN 978-1-4503-2375-8. doi: 10.1145/2594538.2594550. URL `http://dl.acm.org/citation.cfm?id=2594538`.

E. W. Beth. On Padoa's Method in the Theory of Definition. *Journal of Symbolic Logic*, 21(2):194–195, 1956.

Alexander Borgida, Jos de Bruijn, Enrico Franconi, Inanç Seylan, Umberto Straccia, David Toman, and Grant E. Weddell. On Finding Query Rewritings under Expressive Constraints. In Sonia Bergamaschi, Stefano Lodi, Riccardo Martoglia, and Claudio Sartori, editors, *Proceedings of the Eighteenth Italian Symposium on Advanced Database Systems, SEBD 2010, Rimini, Italy, June 20-23, 2010*, pages 426–437. Esculapio Editore, 2010. ISBN 978-88-7488-369-1.

Alin Deutsch, Lucian Popa, and Val Tannen. Physical Data Independence, Constraints, and Optimization with Universal Plans. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 459–470. Morgan Kaufmann, 1999. ISBN 1-55860-615-7. URL `http://www.vldb.org/conf/1999/P44.pdf`.

Alin Deutsch, Lucian Popa, and Val Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006. doi: 10.1145/1121995.1122010.

Alin Deutsch, Bertram Ludäscher, and Alan Nash. Rewriting queries using views with access patterns under integrity constraints. *Theoretical Computer Science*, 371(3):200–226, 2007. doi: 10.1016/j.tcs.2006.11.008.

Oliver M. Duschka and Michael R. Genesereth. Answering Recursive Queries Using Views. In Alberto O. Mendelzon and Z. Meral Özsoyoglu, editors, *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona, USA*, pages 109–116. ACM Press, 1997. ISBN 0-89791-910-6. doi: 10.1145/263661.263674. URL `http://dl.acm.org/citation.cfm?id=263661`.

Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. Recursive query plans for data integration. *The Journal of Logic Programming*, 43(1):49–73, 2000. doi: 10.1016/S0743-1066(99)00025-4.

Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 1990. doi: 10.1007/978-1-4684-0357-2.

Enrico Franconi, Volha Kerhet, and Nhung Ngo. Exact Query Reformulation over Databases with First-order and Description Logics Ontologies. *Journal on Artificial Intelligence Research*, 48:885–922, 2013. doi: 10.1613/jair.4058.

P. Hájek. Generalized quantifiers and finite sets. *Prace Nauk. Inst. Mat. Politech. Wroclaw Ser. Konfer*, 14(1):91–104, 1977.

Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Transactions on Database Systems*, 35(3), 2010. doi: 10.1145/1806907.1806913.

Alfred Tarski. Einige methodologische Untersuchungen über die Definierbarkeit der Begriffe. *Erkenntnis*, 5:pp. 80–100, 1935.